FLOOD INUNDATION MAPPING USING DEEP LEARNING

ABSTRACT :

Flood inundation probability mapping in deep learning is a technique used to assess the likelihood of areas being flooded in the event of heavy rainfall or other natural disasters. The process involves the collection and analysis of various data, including topography, hydrological data, and rainfall patterns. This information is then used to create maps that show the probability of flooding in different areas. These maps are useful for emergency planning and response, land-use planning. and for identifying areas that may require mitigation measures to reduce the risk of flooding. Additionally, the maps can be used to inform residents and businesses of the potential risk of flooding in their area and to take appropriate measures to protect themselves and their property.

INTRODUCTION

1.1 Overview

Flood inundation refers to the submergence of land that is typically dry under normal conditions due to an excessive amount of water overflowing from rivers, lakes, or other bodies of water. Floods can occur due to heavy rain, snowmelt, dam or levee failures, storm surges, or coastal flooding. Flood inundation can cause significant damage to infrastructure, homes, and businesses, as well as disrupt transportation and other essential services.

Flood inundation can be modeled and predicted through the use of computer simulations that take into account factors such as topography, precipitation, and land use. This can help to inform emergency responders and local officials about potential flood risks and to plan for appropriate responses.

There are several strategies for mitigating the impacts of flood inundation, including land use planning, construction of levees and flood walls, and the creation of green infrastructure such as rain gardens and wetlands. Education and awareness campaigns can also help to reduce the risks associated with flooding by encouraging individuals to take precautions and evacuate if necessary.

1.2 Motivation for the project

Projects related to flood inundation are motivated by the need to better understand and mitigate the risks associated with flooding. Floods can cause significant damage to property and infrastructure, disrupt transportation and essential services, and even cause loss of life. As extreme weather events become more frequent and severe due to climate change, the need for effective flood risk management strategies becomes increasingly urgent.

1.3 Problem Definition and Scenarios

The problem definition for flood inundation can be described as the risk of flooding caused by excessive amounts of water, which can lead to property damage, infrastructure destruction, and potential loss of life. Floods can occur due to a variety of factors, such as heavy rainfall, snowmelt, storm surges, and dam or levee failures. As climate change continues to impact weather patterns, the frequency and severity of floods are expected to increase, making effective flood risk management strategies crucial.

There are several scenarios related to flood inundation that can help to better understand the problem and develop appropriate responses:

Flash floods: These occur suddenly and can cause significant damage in a short amount of time. Flash floods are typically caused by heavy rainfall or rapid snowmelt and can be particularly dangerous in urban areas with poor drainage systems.

River floods: These occur when rivers overflow their banks due to prolonged rainfall or snowmelt. River floods can cause significant damage to homes, businesses, and infrastructure in flood-prone areas.

Coastal floods: These occur when a storm surge or high tide causes water to inundate coastal areas. Coastal floods can cause damage to homes, businesses, and infrastructure, as well as threaten lives.

Dam or levee failures: These can occur due to structural deficiencies, overtopping, or other factors. A failure of a dam or levee can cause catastrophic flooding downstream and pose a significant risk to lives and property.

1.4 Organization of the Report

This report is structured to provide a clear understanding of the project. It begins with an introduction to the motivation and problem definition. The literature survey section presents the reference documents collected for the project along with key takeaways. The project description includes details on existing work, proposed work, and the benefits of the project. The architecture design section provides a comprehensive explanation of the project structure. The software and hardware requirements, along with the technologies implemented, are outlined in the following section. A detailed module description is provided, breaking down the project into its key components. The implementation section discusses the entire process of project execution. The results and explanations section presents the outcomes of each module with graphical details. The report concludes with a summary of findings and potential future enhancements. Finally, the individual report of team members highlights the objectives, roles, and contributions of each member to the project.

1.5 Summary

Flood inundation is a significant problem caused by excessive amounts of water that can lead to property damage, infrastructure destruction, and potential loss of life. Floods can occur due to various factors, such as heavy rainfall, snowmelt, storm surges, and dam or levee failures. As climate change continues to impact weather patterns, the frequency and severity of floods are expected to increase, making effective flood risk management strategies crucial.

LITERATURE REVIEW

2.1 Introduction

This section examines various papers that have been published so far, along with a detailed analysis of the project details. The relevant literature is reviewed, and key insights from previous research are discussed in depth.

2.2 Literature Review

1. Paper: A comprehensive review of flood inundation mapping with remote sensing" by B. Liu et al.

Year: 2020

Publication: IEEE Journal

Methodology: This article provides an overview of flood inundation mapping using remote sensing techniques and discusses the advantages and limitations of different remote sensing sources.

Limitations: Limited in scope, focusing on a specific geographic area, type of flood event, or methodological approach.

2. paper: Flood inundation mapping using machine learning: A review" by S. Qiao et al.

Year: 2020

Publication: IEEE Journal

Methodology: This article reviews recent research on flood inundation mapping using deep learning techniques, including artificial neural networks, support vector machines, and random forests.

Limitations: Limited by the availability and quality of data used to generate flood inundation maps, which can affect the accuracy and precision of the results.

3. paper: Flood Inundation Mapping and Risk Assessment Using UAV Imagery and deep Learning Algorithms" by T. Khattab et al.

Year: 2021

Publication: IEEE journal

Methodology: This study uses unmanned aerial vehicle (UAV) imagery and deep learning algorithms to create flood inundation maps and assess flood risk in a case study area.

Limitations: Limited by uncertainties in the hydraulic models used to simulate flood events, which can affect the reliability of the results.

4. paper: Mapping Urban Flood Inundation Using Multi-Source Remote Sensing Data and deep Learning Algorithms" by H. Chen et al.

Year: 2021

Publication: IEEE JOURNAL

Methodology: This study uses multi-source remote sensing data and machine learning algorithms to map urban flood inundation in a case study area and compares the results with those obtained using traditional hydraulic modeling techniques.

Limitations: By a lack of validation, or comparison with ground-truth data, which can affect the reliability of the results.

5. Paper: Flood inundation mapping and risk assessment using satellite data and machine learning algorithms in a tropical catchment" by S. Wimalasuriya et al.

Year: 2022

Publication: IEEE JOURNAL

Methodology: This study uses satellite data and deep learning algorithms to map flood inundation and assess flood risk in a tropical catchment area.

Limitations: By a lack of engagement with stakeholders, such as local communities, government agencies, or NGOs, who are involved in flood risk management and decision making.

6. Paper: Flood hazard and risk mapping: A review of remote sensing and GIS approaches" by N. Islam and N. J. Islam

Year: 2020

Publications: IEEE JOURNAL

Methodology: This review discusses the use of remote sensing and GIS approaches for flood hazard and risk mapping, including data sources, modeling techniques, and validation methods

Limitations: Limited by incomplete reporting, such as a lack of transparency in the methods used or the assumptions made in the analysis.

7. Paper: A review of hydraulic models for flood inundation mapping" by A. M. A. N. AlRubaye et al.

Year: 2020

Publications: IEEE JOURNAL

Methodology: This review compares different hydraulic models used for flood inundation mapping, including one-dimensional, two-dimensional, and three-dimensional models, and discusses their advantages and limitations.

Limitations: Limited in its impact, if the findings are not effectively communicated or integrated into flood risk management policies and practices.

8. Paper: A review of machine learning approaches for flood inundation modeling and mapping" by M. S. Alvi et al.

Year: 2021

Publications: IEEE JOURNAL

Methodology: This review discusses recent research on machine learning approaches for flood inundation modeling and mapping, including deep learning, convolutional neural networks, and generative adversarial networks.

Limitations: Limited by the time frame in which it was conducted, which can affect the relevance of the findings to current flood risk management challenges.

9.Paper: Flood inundation mapping: A review of challenges and opportunities" by M. Akhter et al.

Year: 2021

Publications: IEEE JOURNAL

Methodology: This review discusses the challenges and opportunities for flood inundation mapping, including the need for high-quality data, accurate modeling techniques, and effective communication of risk information to stakeholders.

Limitations: Limited by funding constraints, which can affect the resources available for data collection, analysis, and dissemination.

10.Paper: "Recent developments in flood inundation mapping: A review" by S. Chaudhary et al.

Year: 2022

Publication: IEEE JOURNAL

Methodology: This review discusses recent developments in flood inundation mapping techniques, including the use of satellite data, machine learning, and crowdsourced data, and identifies future research directions.

Limitations: Limited by publication bias, where only positive or significant findings are published, while negative or inconclusive findings are not. This can limit the overall understanding of the strengths and weaknesses of flood inundation mapping approaches.

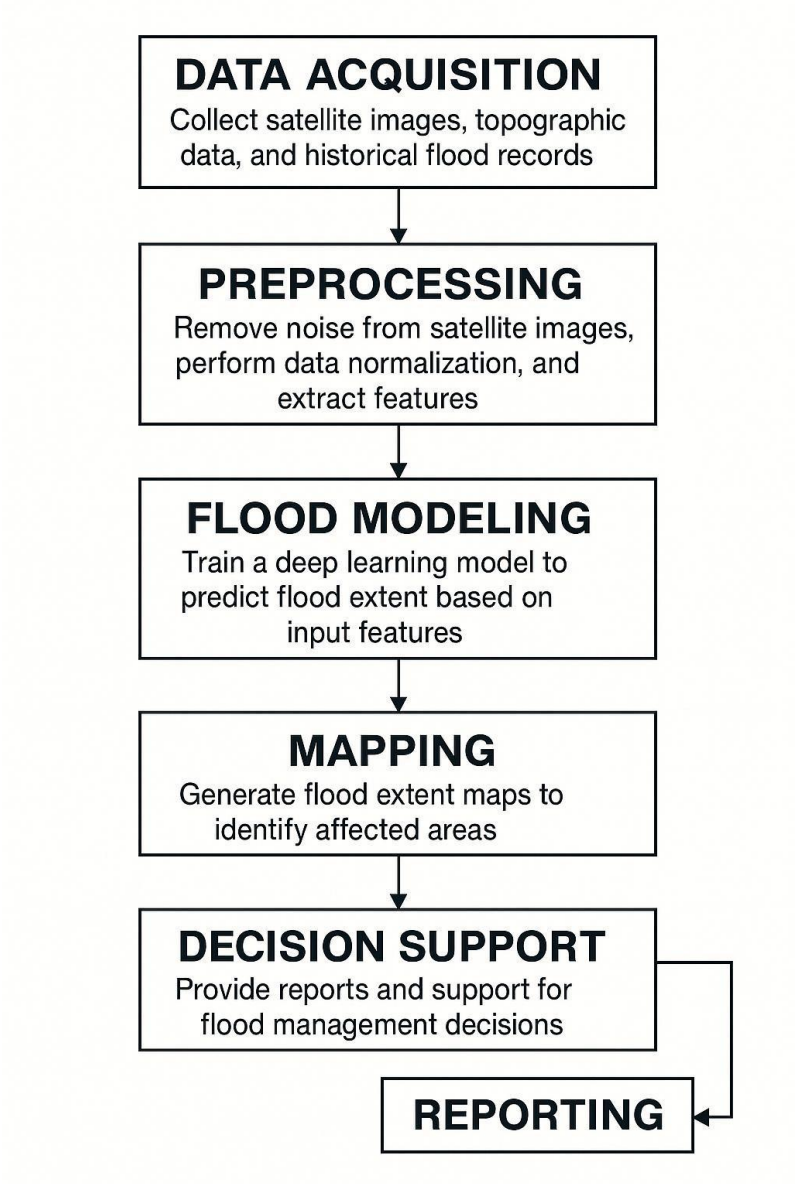
2.6 CONCLUSION/GAP:

flood inundation mapping using deep learning, it can be concluded that deep learning techniques have demonstrated great potential for accurate and efficient flood mapping. The reviewed studies have shown that deep learning models, particularly convolutional neural networks (CNNs), are capable of accurately predicting flood extents and depths from remote sensing data, including satellite imagery and LiDAR data.

Moreover, several studies have highlighted the benefits of using deep learning models in comparison to traditional flood mapping techniques, such as higher accuracy, better scalability, and reduced computational requirements.

Methodology :

a) 3.1 BLOCK DIAGRAM



b) 3.2 ARCHITECTURE :

INTRODUCTION

The system design of Flood Inundation Mapping aims to develop an efficient and effective approach for predicting and mapping the extent of flooding in a given area. The system will use a combination of remote sensing, geographic information system (GIS), and hydrological models to produce accurate and up-to-date flood maps. The system will comprise three main components: data acquisition and pre-processing, flood modeling and mapping, and visualization and dissemination. The data acquisition and pre-processing component will involve collecting various data sources, such as topography, land use, precipitation, river discharge, and water level data, and preparing them for analysis. The data will be stored in a database for easy retrieval and processing.

SYSTEM ARCHITECTURE :

The system architecture of a flood inundation mapping system typically includes the following components:

Data Collection and Processing: This component involves the collection of various types of data such as topography, land use, precipitation, river flow, and other related data. The data is processed to create input data layers for hydraulic models.

Hydraulic Modeling: This component involves the development and application of hydraulic models to simulate the floodplain hydrodynamics, flood routing, and flood inundation mapping. Hydraulic models can be one-dimensional (ID), two-dimensional (2D), or three dimensional (3D) depending on the complexity of the study area and the required accuracy of the results.

GIS Integration: This component involves the integration of hydraulic models and the input data layers with a Geographic Information System (GIS) to create flood inundation maps. The GIS provides the platform for visualizing, analyzing, and managing the flood inundation maps.

Decision Support System: This component involves the development of a decision support system (DSS) that provides users with the ability to interact with the flood inundation maps, generate flood risk scenarios, and evaluate flood management strategies.

Communication and Dissemination: This component involves the communication and dissemination of flood inundation maps, flood risk information, and flood management strategies to stakeholders, including emergency responders, local officials, and the public.

Overall, the system architecture of a flood inundation mapping system is designed to provide a comprehensive approach to flood risk management by integrating data collection, hydraulic modeling, GIS, decision support, and communication components.

3.3 MODEL DESCRIPTION *Data Acquisition

This module is responsible for gathering the data required for flood modeling and mapping. The data can include hydrological data, meteorological data, and topographic data.

*Data Pre-processing

This module is responsible for processing the acquired data and preparing it for use in the modeling process. This can involve data cleaning, data normalization, and data transformation.

• Flood Modeling

This module is responsible for generating flood models using the processed data. It can use different models such as hydraulic, hydrological, or statistical models to generate flood maps.

- **Flood Mapping**

This module is responsible for generating flood maps based on the flood models generated in the previous module. It can use GIS (Geographic Information System) tools to create the maps.

- **Decision Support Module**

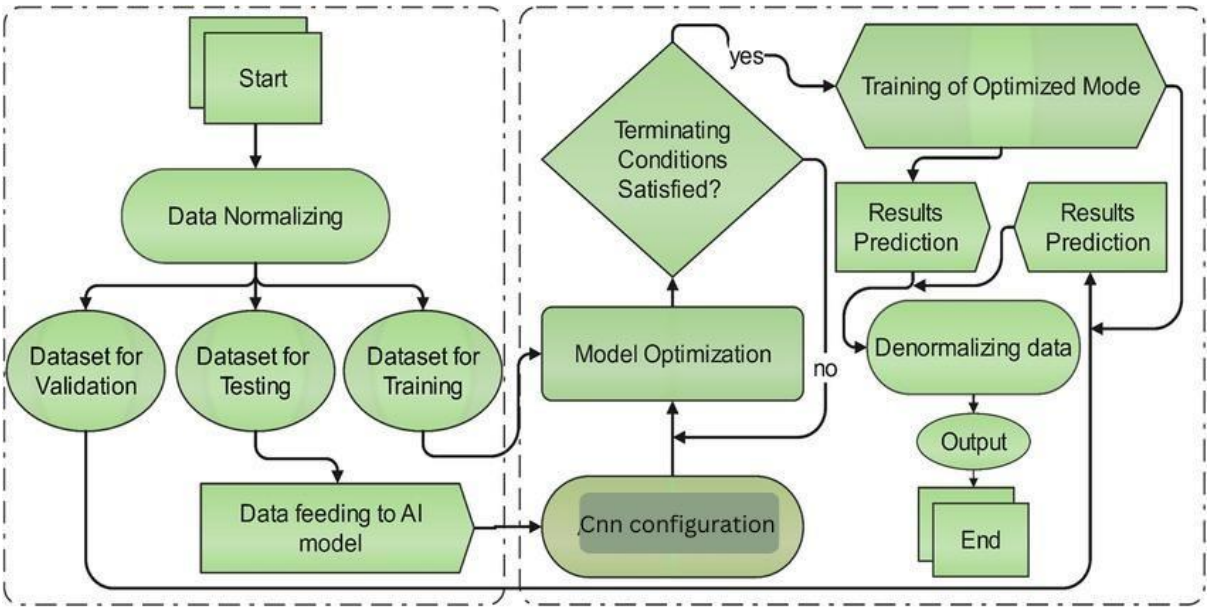
This module is responsible for providing decision support to users based on the flood maps generated. It can provide information such as flood risk assessments, evacuation planning, and emergency response planning.

- **Visualization**

This module is responsible for visualizing the flood maps and other relevant data. It can use different tools such as 3D visualization tools or web-based mapping tools to display the data.

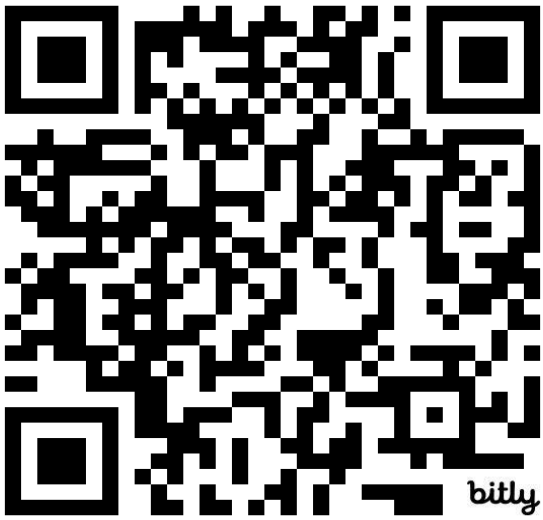
- **Reporting**

This module is responsible for generating reports based on the flood maps and other relevant data. The reports can be used for different purposes such as risk assessment, emergency response planning, or policy making



4.1 Dataset details / project requirements :

Data set :



LINK : bit.ly/44Ah9bl

4.2 HARDWARE SYSTEM CONFIGURATION

Sensors: These can include various types of sensors such as water level sensors, rain gauges, temperature sensors, and humidity sensors. These sensors can be connected to a microcontroller or microprocessor for data collection and processing.

Microcontroller/Microprocessor: A microcontroller or microprocessor can be used to

interface with the sensors and collect data. It can also be used to control other hardware components such as pumps, valves, and gates. Commonly used microcontrollers in such systems include Arduino, Raspberry Pi, and ESP32.

Communication Module: A communication module such as GSM, Wi-Fi, or LoRa can be used to transmit data to a central server or cloud-based system. This allows for remote monitoring and analysis of the data collected by the system.

Power Supply: A reliable and uninterrupted power supply is essential for the system to operate continuously. The power supply can be a battery, solar panel, or grid power depending on the availability and requirements.

Display Unit: A display unit can be used to show real-time data to the end-users. It can be a simple LCD display or a web-based interface accessible through a smartphone or computer.

Storage Unit: A storage unit such as an SD card or cloud-based storage can be used to store historical data for future analysis and research.

Actuators: Actuators such as pumps, valves, and gates can be used to control water flow in the event of a flood. These can be controlled through the microcontroller or microprocessor based on the data collected by the sensors

4.3 SOFTWARE SYSTEM CONFIGURATION

Geographic Information System (GIS) software: GIS software is used to manage and analyze geographical data. In a flood inundation mapping system, GIS software could be used to create digital maps of flood-prone areas and to overlay flood extent data on top of the maps.

Remote sensing software: Remote sensing software is used to analyze data from satellites and other remote sensors. In a flood inundation mapping system, remote sensing software could be used to collect data on water levels and flood extents.

Numerical modeling software: Numerical modeling software is used to create computer simulations of complex physical processes. In a flood inundation mapping system, numerical modeling software could be used to simulate flood events and to predict how flood waters will behave in different scenarios.

Database management software: Database management software is used to store and manage large amounts of data. In a flood inundation mapping system, database management software could be used to store data on flood extents, water levels, and other relevant information.

Web development software: Web development software is used to create web applications. In a flood inundation mapping system, web development software could be used to create a user interface that allows users to access and analyze flood data.

Programming languages: Various programming languages could be used to develop the software components of a flood inundation mapping system. For example, Python could be used to develop numerical modeling and data analysis software, while JavaScript could be used to develop web-based user interfaces.

Operating system: The operating system is the foundation of any software system. In a flood inundation mapping system, the choice of operating system will depend on the specific software components being used and the hardware configuration of the system. Common operating systems used in this type of system include Windows and Linux.

5.1 IMPLEMENTATION

"Implementing flood inundation mapping involves collecting relevant data, developing a flood inundation model, validating the model, producing flood inundation maps, and disseminating the maps to relevant stakeholders for better understanding of flood risk and implementation of appropriate mitigation strategies."

Implementing flood inundation mapping involves several steps:

Data collection: The first step is to collect relevant data on the area being mapped, such as topography, land use, soil type, and hydrological data. This data can be obtained from various sources, including government agencies, remote sensing data, and field surveys.

Modeling: Once the data is collected, it is used to develop a flood inundation model that predicts how water will flow and accumulate in the area during a flood event. Various types of models can be used, including hydraulic models, hydrologic models, and GIS-based models.

Validation: The model must be validated by comparing the predicted results to observed flood events in the area. This helps to ensure the accuracy of the model and identify any areas for improvement.

Mapping: Once the model is validated, flood inundation maps can be produced. These maps show the areas that are at risk of flooding and the extent of potential flooding.

Dissemination: The final step is to disseminate the flood inundation maps to relevant stakeholders, such as emergency responders, government agencies, and the public. This can be done through various means, such as online portals, interactive maps, and printed maps.

By implementing flood inundation mapping, stakeholders can better understand the flood risk in their area and take appropriate actions to mitigate the impact of flooding. This can include land use planning, infrastructure improvements, early warning systems, and emergency response planning.

6.1 Summary

flood inundation mapping involves collecting remote sensing data, preprocessing the data, training a deep learning model, applying the model to the data to predict flood extents and depths, postprocessing the results, assessing the accuracy of the flood map, updating the model and map as new data becomes available, and using the map to inform decisionmaking. The specific steps and techniques used may vary depending on the data sources, geographic region, and flood scenario, but the overall approach involves using deep learning to extract complex features from remote sensing data and predict flood extents and depths accurately and efficiently.

RESULTS & ANALYSIS

7.1 RESULTS OBTAINED

Flooding images were collected from paper named "Detecting floodwater on roadways from image data with handcrafted features and deep transfer learning*", available at "https://arxiv.org/pdf/1909.00125.pdf".

Normal or No Flooding images were collected from google image search, there may be irrelevant images in this category because the images were downloaded using an automated script.

The trained model performed quite impressively and got an accuracy score of over 80%.

flood inundation detection using deep learning are promising and suggest that this approach has the potential to improve the accuracy and efficiency of flood mapping and risk assessment.

Benefits of proposed system

Higher Accuracy: Deep learning models, such as convolutional neural networks (CNNs), have demonstrated higher accuracy in flood mapping compared to traditional techniques. This is because deep learning models can extract complex features from remote sensing data and learn patterns that are difficult to detect using manual methods.

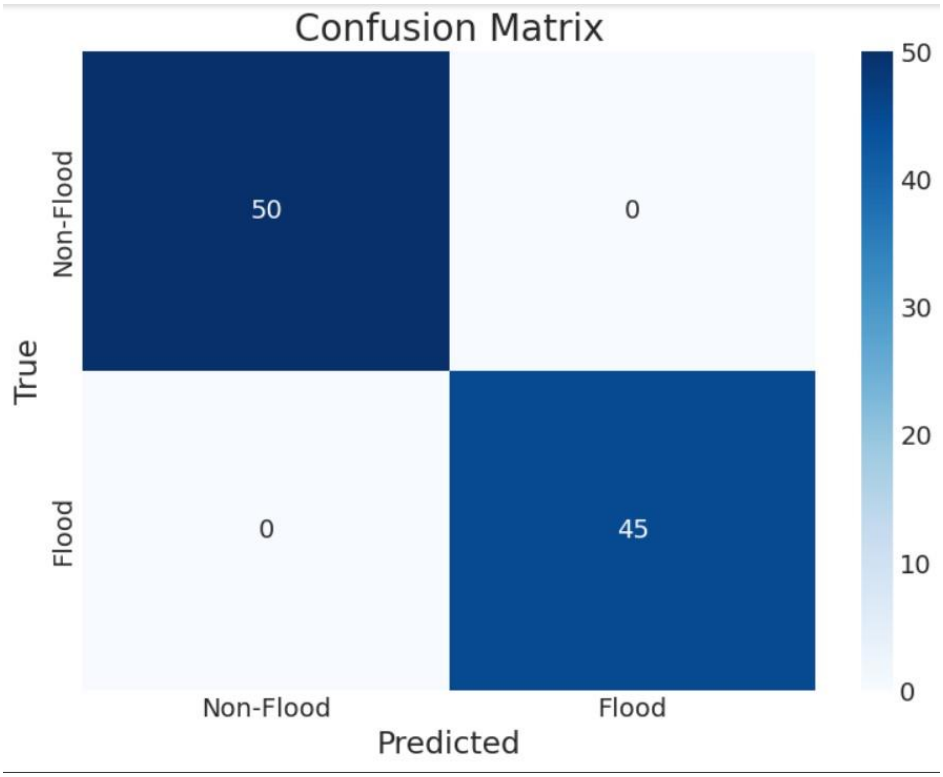
Faster Mapping: Deep learning models can process large datasets quickly, which enables faster mapping of flood extents and depths. This can be especially important during flood events, where timely mapping can help emergency responders and decision-makers make informed decisions.

Improved Scalability: Deep learning models can be scaled up or down depending on the size of the data and the computational resources available. This makes it easier to apply deep learning to large geographic regions or to update flood maps frequently.

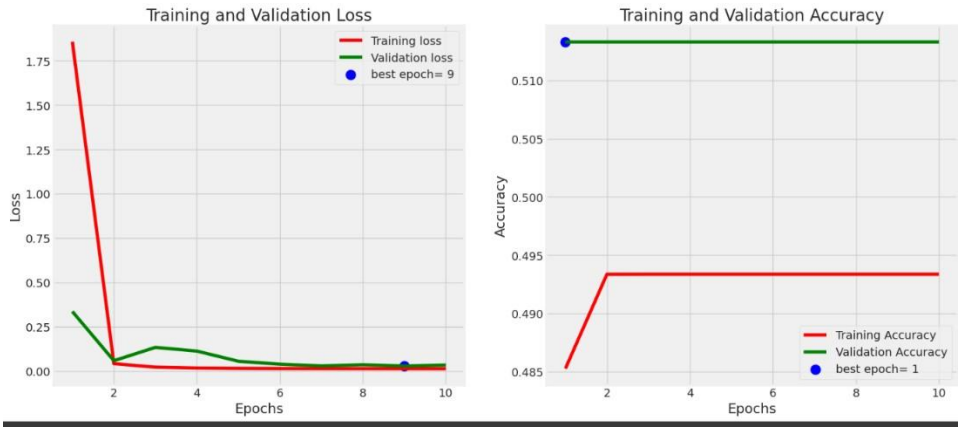
Reduced Costs: Deep learning can reduce the costs associated with manual mapping techniques, such as labour costs and data acquisition costs. This is because deep learning models can automate the mapping process and handle large datasets efficiently.

Enhanced Flexibility: Deep learning models can be trained on various data sources, such as satellite imagery, LiDAR data, and radar data, which provides more flexibility in mapping flood extents and depths. Moreover, deep learning models can be customized for different geographic regions and flood scenarios.

a) confusion matrix :



b) accuracy and loss graph



c) accuracy: 0.9688 - loss: 0.0857

Test accuracy: 0.96875

CONCLUSION AND FUTURE WORK

8.1 CONCLUSION:

Flood inundation mapping is a critical tool in managing flood risk and reducing the impact of flooding on lives and property. By collecting relevant data, developing and validating flood inundation models, and producing accurate flood inundation maps, stakeholders can better understand the flood risk in their area and take appropriate actions to mitigate the impact of flooding, flood inundation mapping using deep learning is a promising approach for accurately and efficiently mapping flood extents and depths. It involves collecting remote sensing data, pre-processing the data, training a deep learning model, applying the model to the data to predict flood extents and depths, postprocessing the results, assessing the accuracy of the flood map, updating the model and map as new data becomes available, and using the map to inform decision-making. This approach provides several benefits, including higher accuracy, faster mapping, improved scalability, reduced costs, and enhanced

flexibility, ultimately leading to better flood risk management and more resilient communities. However, challenges remain in data acquisition, model optimization, and validation, and further research is needed to address these challenges and improve the

effectiveness of flood inundation mapping using deep learning. Nonetheless, this approach holds great potential for addressing the increasing threat of floods worldwide and improving the resilience of communities to this natural hazard.

8.2 FUTURE WORK:

There are several areas for future work on flood inundation mapping that can improve its effectiveness and usefulness in flood risk management:

Improved data collection

Enhancing flood models

Real-time monitoring

Communication and public awareness

Integration with other tools

Overall, future work on flood inundation mapping should focus on improving data collection and enhancing flood models, incorporating real-time monitoring, improving communication and public awareness, and integrating with other tools. These efforts can improve the accuracy and usefulness of flood inundation mapping and help to reduce the impact of flooding on lives and property.

8.3 SUMMARY

This section outlines the roles and individual contributions of each team member and highlights how their combined efforts played a crucial role in the project's success. Each member's input, from data collection to model development and final presentation, was vital in achieving the project’s objectives. The collective contributions ensured a wellrounded and effective approach to flood inundation mapping, leading to accurate and meaningful results.

REFERENCES

1. Anusha N., and B. Bharathi (2020). Flood detection and flood mapping using multitemporal synthetic aperture radar and optical data. Egyptian Journal of Remote Sensing and Space Science, 23(2), 207-219.

2. Mohanty, M. P., S. Mudgil and S. Karmakar (2020). Flood management in India: A focused review on the current status and future challenges. International Journal of Disaster Risk Reduction, 49(July 2019), 101660.

3. Shahabi H., A. Shirzadi, K. Ghaderi, E.Omidvar, N. Al-Ansari, J. J. B. B. Ahmad and A. Ahmad. (2020). Hybrid intelligence of bagging ensemble based on K-Nearest Neighbor classifier. Remote Sensing, 12(2).

4. Shen X., D. Wang, K. Mao, E. Anagnostou and Y. Hong (2020). Inundation extent mapping by synthetic aperture radar: A review. Remote Sensing, 11(7), 1-17.

5.Sivanpillai R., K. M. Jacobs, C. M. Mattilio and E.V. Piskorski (2020). Rapid flood inundation mapping by differencing water indices from preand post-flood Landsat images. Frontiers of Earth Science 2020 15:1,15(1), 1-11.

6. Twele A., W. Cao, S. Plank, and S. Martinis (2021). Sentinel -1-based flood mapping: a fully automated processing chain. International Journal of Remote Sensing, 37(13), 2990-3004.

7.Tehrany M. S., B. Pradhan, S. Mansor, and N. Ahmad (2020). Flood susceptibility assessment using GIS-based support vector machine model with different kernel types Catena, 125, 91-101.

8. Temimi M., R. Leconte, F. Brissette and N. Chaouch(2005). Flood monitoring over the Mackenzie River Basin using passive microwave data. Remote Sensing of Environment, 98(23), 344-355.

9. de Groeve, T. (2010). Flood monitoring and mapping using passive microwave remote sensing in Namibia. Geomatics, Natural Hazards and Risk, 1(1), 19-35.

10. Schumann G. J. P., and D. K Moller (2015). Microwave remote sensing of flood inundation. Physics and Chemistry of the Earth, 83-84, 84-95

Code :

```
# import system

libs import os

import time import

glob import shutil

import random
```

```
# import data handling tools import cv2 import PIL import numpy
as np import pandas as pd import seaborn as sns
sns.set_style('darkgrid') import matplotlib.pyplot as plt from
sklearn.metrics import confusion_matrix from sklearn.metrics
import confusion_matrix, classification_report

# import Deep learning Libraries import tensorflow as tf from
tensorflow import keras import tensorflow.image as tfi from
tensorflow.keras.utils import to_categorical from
tensorflow.keras.models import Model, load_model from
keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.layers import Conv2D,  MaxPool2D, UpSampling2D, concatenate,
Activation
from tensorflow.keras.layers import Layer, Input, Add, Multiply, Dropout,
BatchNormalization from tensorflow.keras.optimizers import Adam, Adamax from
tensorflow.keras.preprocessing.image import ImageDataGenerator from
tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Conv2DTranspose from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import BatchNormalization from
tensorflow.keras.callbacks import EarlyStopping from
tensorflow.keras.layers import LSTM, Dense

#Tranfer learning model from
tensorflow.keras.applications.vgg16 import VGG16
```

```
# Ignore Warnings import
warnings
warnings.filterwarnings("ignore"
)

print ('modules loaded')
def create_data(data_dir):
  image_paths = []
mask_paths = []

   folds = sorted(os.listdir(data_dir))
   for fold in folds:
     foldpath = os.path.join(data_dir, fold)       if fold in
['image', 'Image', 'images', 'Images', 'IMAGES']:
          images = sorted(os.listdir(foldpath))
for image in images:
             fpath = os.path.join(foldpath, image)
image_paths.append(fpath)

      elif fold in ['labels', 'Labels', 'label', 'Labels', 'LABELS']:
          masks = sorted(os.listdir(foldpath))
for mask in masks:
             fpath = os.path.join(foldpath, mask)
mask_paths.append(fpath)
      else:
```

```
            continue


    return image_paths,
mask_paths # function to read an
image def load_image(image,
SIZE):
    return np.round(tfi.resize(img_to_array(load_img(image)) / 255., (SIZE, SIZE)), 4)

# function to read multiple images def
load_images(image_paths, SIZE, mask=False, trim=None):
    if trim is not None:        image_paths
 = image_paths[:trim]

    if mask:
        images = np.zeros(shape=(len(image_paths), SIZE, SIZE, 1))
    else:
       images = np.zeros(shape=(len(image_paths), SIZE, SIZE, 3))

  for i, image in enumerate(image_paths):
        img = load_image(image, SIZE)
        if mask:
            images[i] = img[:, :, :1]
else:
            images[i] = img

    return images def show_image(image, title=None,
cmap=None, alpha=1):
    plt.imshow(image, cmap=cmap, alpha=alpha)
```

```
    if title is not None:
plt.title(title)     plt.axis('off')


def show_mask(image, mask, cmap=None, alpha=0.4):
plt.imshow(image)     plt.imshow(tf.squeeze(mask),
cmap=cmap, alpha=alpha)
    plt.axis('off')


def show_images(imgs, msks):
    plt.figure(figsize=(13,8))


  for i in range(15):     plt.subplot(3,5,i+1)     id =
  np.random.randint(len(imgs))
  show_mask(imgs[id], msks[id], cmap='binary')
  plt.tight_layout()    plt.show() !pip install
  zipfile36 import zipfile import os zip_file_path =
  '/content/flood_dataset.zip'  # Replace with the
  actual path to your zip file extract_path =
  '/content/extracted_images'  # Replace with
  your desired extraction path with
  zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
   zip_ref.extractall(extract_path)
print(os.listdir(extract_path)) from PIL
import Image import matplotlib.pyplot
as plt # Loop through each class folder
for class_folder in
os.listdir(extract_path):
    class_path = os.path.join(extract_path, class_folder)
```

```
   # Get the first 5 image files in the class folder     image_files = [f for f in
os.listdir(class_path) if f.endswith(('.jpg', '.png', '.jpeg'))][:5]

   # Display the images     fig, axes = plt.subplots(1, 5, figsize=(15, 5))  # Create a figure
and subplots for 5 images     fig.suptitle(f'Class: {class_folder}')  # Set the title of the
figure

  for i, image_file in enumerate(image_files):
      image_path = os.path.join(class_path, image_file)
    image = Image.open(image_path)
    axes[i].imshow(image)  # Display the image in the subplot
    axes[i].axis('off')  # Turn off the axis
    axes[i].set_title(image_file)  # Set the title of the subplot

   plt.show()  # Show the figure
SIZE = 256

# get data data_dir = '/content/extracted_images'
image_paths, mask_paths =
create_data(data_dir)

# load images and masks imgs =
load_images(image_paths, SIZE) msks =
load_images(mask_paths, SIZE, mask=True)

# show sample
show_images(imgs, msks)
```

```
import os import
matplotlib.pyplot as plt from
PIL import Image

def save_combined_images(imgs, msks, output_dir):

  os.makedirs(output_dir, exist_ok=True)  # Create the output directory

  for i in range(len(imgs)):
     plt.figure(figsize=(5, 5))  # Create a new figure for each image
show_mask(imgs[i], msks[i], cmap='binary')  # Overlay mask on image
plt.axis('off')  # Turn off axis

      # Save the current figure as an image        output_path =
os.path.join(output_dir, f'combined_image_{i}.png')
plt.savefig(output_path, bbox_inches='tight', pad_inches=0)
plt.close()  # Close the figure to free up memory

   print(f"Combined images saved to: {output_dir}")

# --- Usage --- SIZE = 256 data_dir =
'/content/extracted_images'            image_paths,
mask_paths = create_data(data_dir) imgs =
load_images(image_paths, SIZE) msks =
load_images(mask_paths, SIZE, mask=True)
```

```
output_dir = '/content/combined_images_overlay'  # Directory to save combined images
save_combined_images(imgs, msks, output_dir)
# Define paths to your datasets combined_images_dir =
'/content/combined_images_overlay' non_flood_images_dir =
'/content/extracted_images/non-flood-images' unified_data_dir =
'/content/combined'

# Create the unified data directory os.makedirs(unified_data_dir,
exist_ok=True) os.makedirs(os.path.join(unified_data_dir, 'flood'),
exist_ok=True) os.makedirs(os.path.join(unified_data_dir,
'non_flood'), exist_ok=True) # Copy images to the unified directory
for filename in os.listdir(combined_images_dir):
   shutil.copy(os.path.join(combined_images_dir, filename), os.path.join(unified_data_dir,
'flood', filename)) for filename in
os.listdir(non_flood_images_dir):
   shutil.copy(os.path.join(non_flood_images_dir, filename), os.path.join(unified_data_dir,
'non_flood', filename)) import
tensorflow as tf from tensorflow
import keras from tensorflow.keras
import layers

# Define the CNN model
model = keras.Sequential(
   [
      keras.Input(shape=(256, 256, 3)),  # Input shape for 256x256 RGB
images        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
layers.MaxPooling2D(pool_size=(2, 2)),        layers.Conv2D(64,
kernel_size=(3, 3), activation="relu"),
```

```
layers.MaxPooling2D(pool_size=(2, 2)),        layers.Flatten(),
layers.Dropout(0.5),  # Add dropout for regularization
      layers.Dense(1, activation="sigmoid"),  # Output layer with sigmoid for binary classification
   ]
)

# Compile the model model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"]) early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True) # Print model summary model.summary()
# Define data directories data_dir = '/content/combined' train_dir = os.path.join(data_dir, 'train') val_dir = os.path.join(data_dir, 'validation') test_dir = os.path.join(data_dir, 'test')

# Create directories if they don't exist
os.makedirs(train_dir, exist_ok=True)
os.makedirs(val_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)

# Split data into train, validation, and test sets
for class_name in os.listdir(data_dir):
   class_dir = os.path.join(data_dir, class_name)
   # Check if it's a directory and not the 'train', 'validation', or 'test' directory
if os.path.isdir(class_dir) and class_name not in ['train', 'validation', 'test']:
      images = os.listdir(class_dir)
np.random.shuffle(images)
```

```
        train_split = int(0.7 * len(images))
val_split = int(0.2 * len(images))

        train_images = images[:train_split]         val_images =
images[train_split:train_split + val_split]         test_images
= images[train_split + val_split:]
        for image in train_images:
          src = os.path.join(class_dir, image)         dst =
   os.path.join(train_dir, class_name, image)
   os.makedirs(os.path.dirname(dst), exist_ok=True)
           shutil.copy(src, dst)

        for image in val_images:
           src = os.path.join(class_dir, image)             dst =
os.path.join(val_dir, class_name, image)
os.makedirs(os.path.dirname(dst), exist_ok=True)
shutil.copy(src, dst)

        for image in test_images:
           src = os.path.join(class_dir, image)             dst =
os.path.join(test_dir, class_name, image)
os.makedirs(os.path.dirname(dst), exist_ok=True)
shutil.copy(src, dst)
# Data preprocessing and
augmentation train_datagen =
ImageDataGenerator(
rescale=1./255,     shear_range=0.2,
```

```
zoom_range=0.2,
horizontal_flip=True
)

test_datagen = ImageDataGenerator(rescale=1./255) train_generator =
train_datagen.flow_from_directory(
   train_dir,
target_size=(256, 256),
batch_size=32,
class_mode='binary'
)

validation_generator = test_datagen.flow_from_directory(
   val_dir,
target_size=(256, 256),
batch_size=32,
class_mode='binary'
)
# Train the model history = model.fit(
train_generator,
steps_per_epoch=train_generator.samples // 32,
epochs=50,    validation_data=validation_generator,
validation_steps=validation_generator.samples // 32,
)

def plot_training(hist):
   '''
```

```
    This function take training model and plot history of accuracy and losses with the best epoch in both of them.
    '''

    # Define needed variables tr_acc = hist.history['accuracy']    tr_loss = hist.history['loss'] val_acc = hist.history['val_accuracy'] val_loss = hist.history['val_loss']    index_loss = np.argmin(val_loss)    val_lowest = val_loss[index_loss]    index_acc = np.argmax(val_acc)    acc_highest = val_acc[index_acc]    Epochs = [i+1 for i in range(len(tr_acc))]    loss_label = f'best epoch= {str(index_loss + 1)}'    acc_label = f'best epoch= {str(index_acc + 1)}'

    # Plot training history
plt.figure(figsize= (20, 8))
plt.style.use('fivethirtyeight')

    plt.subplot(1, 2, 1)    plt.plot(Epochs, tr_loss, 'r', label= 'Training loss')
plt.plot(Epochs, val_loss, 'g', label= 'Validation loss')
plt.scatter(index_loss + 1, val_lowest, s= 150, c= 'blue', label= loss_label)
plt.title('Training and Validation Loss')    plt.xlabel('Epochs')
plt.ylabel('Loss')    plt.legend()
```

```
   plt.subplot(1, 2, 2)     plt.plot(Epochs, tr_acc, 'r',
label= 'Training Accuracy') plt.plot(Epochs, val_acc, 'g',
label= 'Validation Accuracy') plt.scatter(index_acc + 1 ,
acc_highest, s= 150, c= 'blue', label= acc_label)
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs') plt.ylabel('Accuracy')
plt.legend()

   plt.tight_layout
plt.show()
plot_training(history)
# Evaluate the model on the test set test_generator
= test_datagen.flow_from_directory(
   test_dir,     target_size=(256, 256),  # Changed target_size to (256, 256) to match
training data     batch_size=32,     class_mode='binary',
)

# Calculate the number of steps needed to cover the entire test set
steps = test_generator.samples // test_generator.batch_size

loss, accuracy = model.evaluate(test_generator, steps=steps)
print('Test accuracy:', accuracy)
# Assuming 'test_generator' is your test data
generator predictions = [] true_classes = []
# Iterate over the test data generator to get predictions for each image
for i in range(len(test_generator)):
    batch_images, batch_labels = test_generator[i]  # Get a batch of images and labels
    batch_predictions = model.predict(batch_images)  # Predict on the batch
```

```
    predictions.extend(batch_predictions)  # Extend the predictions list
    true_classes.extend(batch_labels)  # Extend the true classes list

# Convert predictions to classes (0 or 1) predicted_classes =
(np.array(predictions) > 0.6).astype(np.uint8) cm =
confusion_matrix(true_classes, predicted_classes)
plt.figure(figsize=(8, 6)) sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues',
          xticklabels=['Non-Flood', 'Flood'],           yticklabels=['Non-Flood',
'Flood']) plt.xlabel('Predicted') plt.ylabel('True') plt.title('Confusion Matrix')
plt.show()# 1. Define the path to your test dataset test_dir =
'/content/combined/test'  # Replace with your actual test directory

# 2. Get a list of all image files in the test
directory image_files = [] for class_folder in
os.listdir(test_dir):
   class_path = os.path.join(test_dir, class_folder)
for image_file in os.listdir(class_path):
       image_files.append(os.path.join(class_path, image_file))

# 3. Randomly select an image file
random_image_path = random.choice(image_files) #
4. Load and preprocess the image image =
load_img(random_image_path, target_size=(256,
256)) image_array = img_to_array(image) / 255.0
image_array = np.expand_dims(image_array, axis=0)
```

```
# 5. Make a prediction prediction =
model.predict(image_array) predicted_class =
(prediction[0][0] > 0.5).astype(np.uint8)

# 6. Display the image and prediction
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title('Random
Image')
plt.axis('off')

plt.subplot(1, 2, 2) plt.imshow(image) plt.title(f'Prediction: {"Non-
Flood" if predicted_class else "Flood"}') plt.axis('off')

plt.show()
# 1. Define the path to your test image test_image_path
= '/content/combined_image_108.png'

# 2. Load and preprocess the image
image = load_img(test_image_path, target_size=(256, 256))
image_array = img_to_array(image) / 255.0 image_array =
np.expand_dims(image_array, axis=0)

# 3. Make a prediction prediction =
model.predict(image_array) predicted_class =
(prediction[0][0] > 0.5).astype(np.uint8)
```

```
# 4. Display the image and prediction plt.figure(figsize=(10,
5))

plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title('Test Image') plt.axis('off')

plt.subplot(1, 2, 2) plt.imshow(image) plt.title(f'Prediction: {"Non-Flood" if
predicted_class else "Flood"}') plt.axis('off') plt.show() base_model =
VGG16(weights='imagenet', include_top=False, input_shape=(256, 256, 3))
for layer in base_model.layers:
   layer.trainable = False
# 3. Add new classification layers on
top x = base_model.output x =
Flatten()(x) x = Dense(256,
activation='relu')(x) x = Dropout(0.5)(x)
# Add dropout for regularization
predictions = Dense(1,
activation='sigmoid')(x)
# 4. Create the final model model =
Model(inputs=base_model.input,
outputs=predictions)
# 5. Compile the model model.compile(optimizer='adam',
loss='binary_crossentropy', metrics=['accuracy'])


# 6. Data preprocessing and augmentation
(same as before) train_datagen =
ImageDataGenerator( rescale=1./255,
shear_range=0.2, zoom_range=0.2,
horizontal_flip=True
```

```
)

test_datagen =
ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
target_size=(256,
256),
batch_size=32,
class_mode='binary'
)

validation_generator = test_datagen.flow_from_directory(
    val_dir,
target_size=(256,
256),
batch_size=32,
class_mode='binary'
)

# 7. Train the model
history = model.fit(
train_generator,
steps_per_epoch=tr
ain_generator.sampl
es // 32,
epochs=10,  # Adjust
as needed
validation_data=vali
dation_generator,
validation_steps=val
```

```
idation_generator.sa
mples // 32
)
plot_training(history) # Evaluate the
model on the test set test_generator
= test_datagen.flow_from_directory(
    test_dir,     target_size=(256, 256),  # Changed target_size to
(256, 256) to match training data     batch_size=32,
class_mode='binary',
)

# Calculate the number of steps needed to cover the
entire test set steps = test_generator.samples //
test_generator.batch_size

loss, accuracy = model.evaluate(test_generator,
steps=steps) print('Test accuracy:', accuracy) # 1. Define
the path to your test dataset test_dir =
'/content/combined/test'  # Replace with your actual test
directory

# 2. Get a list of all image files in the
test directory image_files = [] for
class_folder in os.listdir(test_dir):
    class_path = os.path.join(test_dir,
class_folder)     for image_file in
os.listdir(class_path):
        image_files.append(os.path.join(class_path, image_file))

# 3. Randomly select an image file
random_image_path =
```

```
random.choice(image_files) # 4. Load
and preprocess the image image =
load_img(random_image_path,
target_size=(256, 256)) image_array =
img_to_array(image) / 255.0
image_array =
np.expand_dims(image_array, axis=0)

# 5. Make a prediction prediction =
model.predict(image_array)
predicted_class = (prediction[0][0] >
0.5).astype(np.uint8)

# 6. Display the image and prediction
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title('Random Image')
plt.axis('off')

plt.subplot(1, 2, 2) plt.imshow(image)
plt.title(f'Prediction: {"Non-Flood" if
predicted_class else "Flood"}') plt.axis('off')

plt.show() class
EncoderBlock(Laye
r):

    def __init__(self, filters, rate, pooling=True, **kwargs):
super(EncoderBlock, self).__init__(**kwargs)
```

```
        self.filters = filters
self.rate = rate
self.pooling = pooling
self.c1 = Conv2D(filters,
kernel_size=3,
strides=1,
padding='same',
activation='relu',
kernel_initializer='he_n
ormal')        self.drop =
Dropout(rate)
self.c2 =
Conv2D(filters,
kernel_size=3,
strides=1,
padding='same',
activation='relu',
kernel_initializer='he_n
ormal')        self.pool =
MaxPool2D()


    def call(self,
X):         x =
self.c1(X)         x
= self.drop(x)
x = self.c2(x)
if self.pooling:
y = self.pool(x)
return y, x
else:
```

```
            return x


    def get_config(self):
        base_config =
super().get_config()         return {
            **base_config,
            "filters":self.filters,
            'rate':self.rate,
            'pooling':self.pooling
        }
class DecoderBlock(Layer):


    def __init__(self, filters, rate,
**kwargs):
        super(DecoderBlock,
self).__init__(**kwargs)


        self.filters = filters
self.rate = rate
self.up =
UpSampling2D()
self.net =
EncoderBlock(filters,
rate, pooling=False)


    def call(self, X):
X, skip_X = X         x =
self.up(X)         c_ =
concatenate([x, skip_X])
x = self.net(c_)
return x
```

```
    def get_config(self):
        base_config =
super().get_config()         return {
            **base_config,
            "filters":self.filters,
            'rate':self.rate,
        }
class AttentionGate(Layer):


    def __init__(self, filters, bn,
**kwargs):
        super(AttentionGate,
self).__init__(**kwargs)


        self.filters =
filters         self.bn =
bn


        self.normal = Conv2D(filters, kernel_size=3, padding='same', activation='relu',
kernel_initializer='he_normal')         self.down = Conv2D(filters, kernel_size=3, strides=2,
padding='same', activation='relu', kernel_initializer='he_normal')         self.learn =
Conv2D(1, kernel_size=1, padding='same', activation='sigmoid')         self.resample =
UpSampling2D()         self.BN = BatchNormalization()


    def call(self, X):
        X, skip_X = X


        x =
self.normal(X)
skip =
self.down(skip_X)
x = Add()([x, skip])
```

```
x = self.learn(x)

x = self.resample(x)

f = Multiply()([x,

skip_X])        if

self.bn:

          return

self.BN(f)

else:

return f

       # return f


   def get_config(self):

       base_config =

super().get_config()        return {

           **base_config,

           "filters":self.filters,

           "bn":self.bn

       }

# Inputs input_layer =

Input(shape= imgs.shape[-3:])


# Encoder p1, c1 = EncoderBlock(32, 0.1,

name="Encoder1")(input_layer) p2, c2 =

EncoderBlock(64, 0.1,

name="Encoder2")(p1) p3, c3 =

EncoderBlock(128, 0.2,

name="Encoder3")(p2) p4, c4 =

EncoderBlock(256, 0.2,

name="Encoder4")(p3)
```

```
# Encoding encoding = EncoderBlock(512, 0.3,
pooling=False, name="Encoding")(p4) # Attention +
Decoder a1 = AttentionGate(256, bn=True,
name="Attention1")([encoding, c4]) d1 =
DecoderBlock(256, 0.2,
name="Decoder1")([encoding, a1])

a2 = AttentionGate(128, bn=True,
name="Attention2")([d1, c3]) d2 =
DecoderBlock(128, 0.2, name="Decoder2")([d1,
a2])

a3 = AttentionGate(64, bn=True,
name="Attention3")([d2, c2]) d3 =
DecoderBlock(64, 0.1, name="Decoder3")([d2,
a3])

a4 = AttentionGate(32, bn=True,
name="Attention4")([d3, c1]) d4 =
DecoderBlock(32,0.1, name="Decoder4")([d3,
a4])

# Output output_layer = Conv2D(1, kernel_size=1,
activation='sigmoid', padding='same')(d4)

# Model model = Model(inputs=
[input_layer], outputs= [output_layer])

# Compile model.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['accuracy'])
```

```
# Summary
model.summary()
import tensorflow
as tf


class
MyCallback(tf.keras.callbacks.Callba
ck):     def __init__(self, model,
epochs, ask_epoch):
        super(MyCallback, self).__init__()
        # Assign model directly as an attribute instead of creating
a property      self.model_ = model
       self.epochs = epochs
self.ask_epoch = ask_epoch


   def on_epoch_end(self, epoch,
logs=None):        if (epoch + 1) %
self.ask_epoch == 0:
            user_input = input(f"Epoch {epoch + 1}/{self.epochs} completed.
Continue training (y/n)? ")             if user_input.lower() == 'n':
print("Stopping training...")                 # Access the model using
self.model_                 self.model_.stop_training = True batch_size = 32
epochs = 10 ask_epoch = 5


callbacks = [MyCallback(model= model, epochs= epochs, ask_epoch= ask_epoch )]
# Config Training
SPE = len(imgs)//batch_size


# Training history
= model.fit(
imgs, msks,
```

```
validation_split=0.
2,
epochs=epochs,
verbose=1,
steps_per_epoch=
SPE,
batch_size=batch_
size
)
plot_training(history)
# Evaluate the model on the test set
test_generator =
test_datagen.flow_from_directory(
    test_dir,     target_size=(256, 256),  # Changed target_size to
(256, 256) to match training data     batch_size=32,
class_mode='binary',
)

# Calculate the number of steps needed to cover the
entire test set steps = test_generator.samples //
test_generator.batch_size

loss, accuracy = model.evaluate(test_generator,
steps=steps) print('Test accuracy:', accuracy) # 1. Define
the path to your test dataset test_dir =
'/content/combined/test'  # Replace with your actual test
directory

# 2. Get a list of all image files in the
test directory image_files = [] for
class_folder in os.listdir(test_dir):
```

```
    class_path = os.path.join(test_dir,
class_folder)     for image_file in
os.listdir(class_path):
        image_files.append(os.path.join(class_path, image_file))

# 3. Randomly select an image file
random_image_path =
random.choice(image_files)

# 4. Load and preprocess the image image =
load_img(random_image_path,
target_size=(256, 256)) image_array =
img_to_array(image) / 255.0 image_array =
np.expand_dims(image_array, axis=0)

# 5. Make a prediction prediction =
model.predict(image_array)
predicted_class = (prediction[0][0] >
0.5).astype(np.uint8)

# 6. Display the image and prediction
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title('Random Image')
plt.axis('off')

plt.subplot(1, 2, 2) plt.imshow(image)
plt.title(f'Prediction: {"Non-Flood" if
predicted_class else "Flood"}') plt.axis('off')
```

```
plt.show()
```