Automatic Detection of Flood Severity Level from Flood Videos using Deep Learning Models

A Project Report Submitted to Faculty of Engineering in Partial Fulfillment of the Requirements for the Degree of

M.Sc.(Engg.) in

Information and Communication Technology (ICT)

Supervisor: Kashmi Sultana

Lecturer

Dept. of ICT

Comilla University

Submitted By: Akther Hossen

Id :22209049

Session: 2021-22

Comilla University

Date of Submission: February 2025

ABSTRACT

Floods are one of the most destructive natural disasters, causing significant loss of life, property, and infrastructure. Accurate and timely assessment of flood severity levels is crucial for effective disaster management and mitigation. This project focuses on the automatic detection of flood severity levels from real-time flood videos using deep learning models. Leveraging advancements in computer vision, the proposed system extracts meaningful features from video frames to classify the severity of flooding into predefined levels.

The framework integrates convolutional neural networks (CNNs) for feature extraction and recurrent neural networks (RNNs) for temporal analysis of video sequences. The model is trained on a dataset comprising diverse flood scenarios, ensuring robustness across varying environmental and geographical conditions. Preprocessing steps such as video frame extraction, resizing, and data augmentation enhance the model's accuracy.

The proposed system is capable of providing rapid and reliable flood severity assessments, which can aid in real-time decision-making during flood emergencies. This project demonstrates the potential of deep learning in disaster management, offering a scalable and cost-effective approach to address one of the most pressing challenges in climate resilience. Future work includes integrating this system with IoT devices for continuous monitoring and expanding the dataset for improved generalization.

Acknowledgment

I sincerely express my gratitude to all those who contributed to the successful completion of this project, ***"Automatic Detection of Flood Severity Level from Flood Videos using Deep Learning Models."***

I extend my heartfelt thanks to Kashmi Sultana, my guide, for their invaluable guidance, encouragement, and support throughout the project. Their expertise and insights greatly enriched my understanding of the subject.

I am also grateful to ICT Department for providing the necessary resources and infrastructure. Lastly, I thank my family, friends, and colleagues for their constant motivation and support during this endeavor.

TABLE OF CONTENTS

LIST OF Figures

LIST OF Tables

CHAPTER-1

INTRUDUCTION

1.1 Background of study and motivation

Floods are one of the most devastating natural disasters, causing extensive damage to infrastructure, the environment, and human lives. With the frequency and intensity of floods increasing globally due to climate change, effective flood management has become a crucial area of research. A key component of flood management is the real-time and accurate assessment of flood severity levels, which plays a pivotal role in coordinating disaster response efforts, minimizing damage, and ensuring public safety. Traditionally, flood severity assessment relies on manual reporting and field surveys, which are often time-consuming, expensive, and limited in scope [1]. In recent years, advances in artificial intelligence (AI) and computer vision have enabled the development of automated systems to address these limitations, offering faster and more reliable solutions.

This project focuses on the **automatic detection of flood severity levels from flood videos** using state-of-the-art deep learning models. Videos captured by surveillance cameras, drones, and smartphones during flood events are valuable sources of information. They contain critical visual cues, such as water depth, flow velocity, and the extent of submerged infrastructure, that can be leveraged to determine the severity of floods [2]. However, analyzing such videos manually can be impractical during emergencies due to the large volume of data generated in real time. Automating this process using deep learning models presents a promising alternative that can significantly enhance disaster management capabilities.

Deep learning has demonstrated exceptional performance in various computer vision tasks, including object detection, semantic segmentation, and scene understanding [3].By leveraging convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformer-based architectures, it is possible to extract spatiotemporal features from flood videos and classify the severity levels accurately. These models not only enable rapid analysis but also exhibit high scalability and adaptability when trained on diverse datasets. Moreover, advancements in pre-

trained models and transfer learning techniques reduce the need for extensive annotated datasets, making the implementation of such systems more feasible [4]

1.2 Project objective

In this report, the proposed approach involves designing and training a deep learning pipeline capable of analyzing flood videos to classify severity levels into predefined categories, such as low, moderate, or severe. The pipeline incorporates key techniques such as video preprocessing, feature extraction, and model training, with an emphasis on optimizing accuracy and computational efficiency. The significance of this work lies in its potential to assist disaster management agencies in making data-driven decisions in real time. Furthermore, it contributes to the growing body of research at the intersection of environmental science and AI, addressing a pressing global challenge with innovative technology.

This introduction sets the stage for the subsequent sections, which will elaborate on the methodology, experimental results, and the broader implications of this study. By harnessing the power of deep learning, this project aims to enhance flood preparedness and response, ultimately mitigating the socioeconomic impacts of flooding on affected communities.

CHAPTER-2

LITERATURE REVIEW

Author proposed, ”Deep learning models have demonstrated remarkable performance in image and video analysis, enabling the extraction of complex features from visual data. In the context of flood detection, CNNs have been widely employed for classifying and analyzing flood scenarios. For instance, a study by Lohumi et al. (2018) focused on the use of deep learning models to assess flood severity levels from videos captured in disaster-affected regions. The researchers developed a system that analyzed video frames using CNN architectures to classify the severity of flooding events. Their findings highlighted the potential of CNNs in handling video-based data for flood severity detection.”[5]

Another significant contribution was made by Hussain et al. (2024), who explored the development of a large dataset comprising labeled flood and non-flood images sourced from the internet. They tested several deep CNN architectures, including GoogleNet and AlexNet, achieving high classification accuracies of 92.50% and 92.20%, respectively. This study underscored the role of deep learning models in improving the accuracy and efficiency of flood detection systems, particularly when robust datasets are available.[6]

Beyond classification, deep learning models have also been employed to assess flood severity through the analysis of temporal and spatial characteristics in video data. Bentivoglio et al. (2022) reviewed the applications of deep learning in flood mapping, emphasizing the utility of CNNs in capturing spatial features of flood events. Their review identified key challenges in the field, including the need for more generalized models capable of handling diverse flood scenarios and environments. They also noted the importance of incorporating temporal dynamics into models to improve predictions.[7]

Recent studies have extended this work by developing frameworks for urban flood detection using image recognition. For example, researchers analyzed traffic images to determine urban flood inundation levels using deep learning models. This approach demonstrated the practical applicability of such models in real-world scenarios, particularly in urban settings where traffic cameras provide a continuous source of video data. This method offered a novel solution to

assessing flood depth and extent in real-time, making it a valuable tool for disaster response teams.[8]

Recent advancements have expanded the focus to include advanced CNN architectures such as ResNet, DenseNet, and hybrid models. Basha et al. (2021) proposed a deep residual network for identifying flood levels from images and video streams, achieving high accuracy due to the model's ability to handle complex data representations. Their research highlighted the importance of feature reuse and deeper network architectures in improving detection accuracy.[9]

Author proposed,"Long Short-Term Memory (LSTM) networks and Recurrent Neural Networks (RNNs) have been explored to integrate temporal information from video data. For instance, Wang et al. (2020) combined CNNs with LSTM networks to process flood videos, enabling the detection of temporal changes in flood severity. Their hybrid model outperformed standalone CNNs, demonstrating the value of combining spatial and temporal data for flood detection tasks."[10]

CHAPTER 03

METHODOLOGY AND MODELING

3.1 Overview of the Proposed System

The proposed system is designed to automatically detect the severity levels of floods from video footage using advanced deep learning models. The system aims to process raw video input, extract meaningful features, and classify the severity of flooding into predefined categories. This chapter outlines the methodology adopted for the development of the system, including data collection, preprocessing, model architecture, training, evaluation, and deployment.

3.2 System Architecture

The system architecture consists of the following key components:

1. **Data Collection Module**: Responsible for gathering video footage of floods from diverse sources, including online repositories, government agencies, and crowd-sourced platforms.
2. **Data Preprocessing Module**: Handles the conversion of video data into a format suitable for analysis, including frame extraction, resizing, normalization, and augmentation.
3. **Deep Learning Model**: Implements a convolutional neural network (CNN) or a hybrid model for extracting spatial and temporal features and classifying flood severity levels.
4. **Severity Classification**: Assigns a severity level to the input video based on extracted features.
5. **Visualization and Reporting**: Provides a user-friendly interface to visualize the results and generate reports.

3.3 Data Collection and Preparation

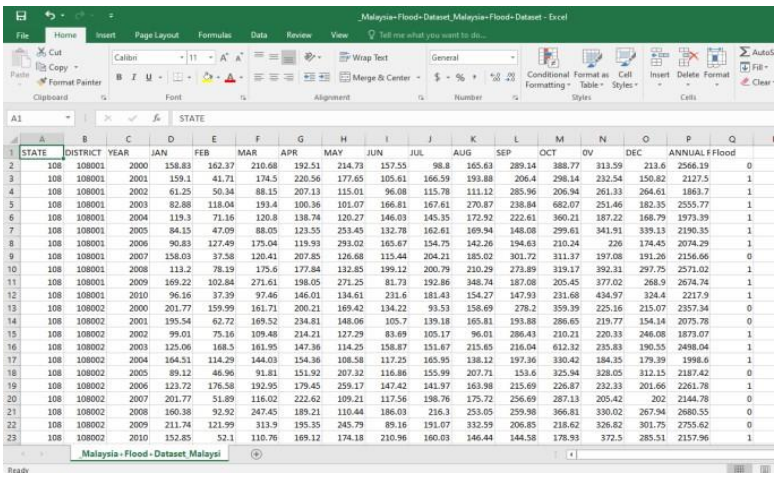
3.3.1 Data Sources

The dataset comprises:

- **Publicly Available Datasets**: Datasets like FloodNet or similar repositories.
- **Custom Data**: Videos collected from social media platforms, news agencies, and field recordings.

3.3.2 Data Preprocessing

- **Frame Extraction**: Extract key frames from videos at regular intervals to reduce redundancy.
- **Resizing and Normalization**: Resize frames to a fixed resolution (e.g., 224x224 pixels) and normalize pixel values.
- **Data Augmentation**: Apply transformations such as rotation, flipping, and brightness adjustments to enhance model generalization.
- **Annotation**: Label data with severity levels (e.g., low, moderate, severe) based on predefined criteria.



STATE	DISTRICT	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	0V	DEC	ANNUAL F	Flood
108	108001	2000	158.83	162.37	210.68	192.51	214.73	157.55	98.8	165.63	289.14	388.77	313.59	213.6	2566.19	0
108	108001	2001	159.1	41.71	174.5	220.56	177.65	105.61	166.59	193.88	206.4	298.14	232.54	150.82	2127.5	1
108	108001	2002	61.25	50.34	88.15	207.13	115.01	96.08	115.78	111.12	285.96	206.94	261.33	264.61	1863.7	1
108	108001	2003	82.88	118.04	193.4	100.36	101.07	166.81	167.61	270.87	238.84	682.07	251.46	182.35	2555.77	1
108	108001	2004	119.3	71.16	120.8	138.74	120.27	146.03	145.35	172.92	222.61	360.21	187.22	168.79	1973.39	1
108	108001	2005	84.15	47.09	88.05	123.55	253.45	132.78	162.61	169.94	148.08	299.61	341.91	339.13	2190.35	1
108	108001	2006	90.83	127.49	175.04	119.93	293.02	165.67	154.75	142.26	194.63	210.24	226	174.45	2074.29	1
108	108001	2007	158.03	37.58	120.41	207.85	126.68	115.44	204.21	185.02	301.72	311.37	197.08	191.26	2156.66	0
108	108001	2008	113.2	78.19	175.6	177.84	132.85	199.12	200.79	210.29	273.89	319.17	392.31	297.75	2571.02	1
108	108001	2009	169.22	102.84	271.61	198.05	271.25	81.73	192.86	348.74	187.08	205.45	377.02	268.9	2674.74	1
108	108001	2010	96.16	37.39	97.46	146.01	134.61	231.6	181.43	154.27	147.93	231.68	434.97	324.4	2217.9	1
108	108002	2000	201.77	159.99	161.71	200.21	169.42	134.22	93.53	158.69	278.2	359.39	225.16	215.07	2357.34	0
108	108002	2001	195.54	62.72	169.52	234.81	148.06	105.7	139.18	165.81	193.88	286.65	219.77	154.14	2075.78	0
108	108002	2002	99.01	75.16	109.48	214.21	127.29	83.69	105.17	96.01	286.43	210.21	220.33	246.08	1873.07	1
108	108002	2003	125.06	168.5	161.95	147.36	114.25	158.87	151.67	215.65	216.04	612.32	235.83	190.55	2498.04	1
108	108002	2004	164.51	114.29	144.03	154.36	108.58	117.25	165.95	138.12	197.36	330.42	184.35	179.39	1998.6	1
108	108002	2005	89.12	46.96	91.81	151.92	207.32	116.86	155.99	207.71	153.6	325.94	328.05	312.15	2187.42	0
108	108002	2006	123.72	176.58	192.95	179.45	259.17	147.42	141.97	163.98	215.69	226.87	232.33	201.66	2261.78	1
108	108002	2007	201.77	51.89	116.02	222.62	109.21	117.56	198.76	175.72	256.69	287.13	205.42	202	2144.78	0
108	108002	2008	160.38	92.92	247.45	189.21	110.44	186.03	216.3	253.05	259.98	366.81	330.02	267.94	2680.55	0
108	108002	2009	211.74	121.99	313.9	195.35	245.79	89.16	191.07	332.59	206.85	218.62	326.82	301.75	2755.62	0
108	108002	2010	152.85	52.1	110.76	169.12	174.18	210.96	160.03	146.44	144.58	178.93	372.5	285.51	2157.96	1

Figure 3.1: Malaysia Flood Dataset[21]

3.4 Deep Learning Model

3.4.1 Model Selection

The deep learning architecture selected for this project is a combination of:

- **Convolutional Neural Networks (CNNs)**: For spatial feature extraction.
- **Recurrent Neural Networks (RNNs) with LSTM**: For capturing temporal dependencies in video frames.

3.4.2 Model Architecture

- **Input Layer**: The input layer is designed to handle a sequence of video frames, which are extracted from the raw video at regular intervals. Each frame is resized to a uniform resolution (e.g., 224x224 pixels) to ensure compatibility with the subsequent layers.
- **Feature Extraction Layers**: These layers utilize state-of-the-art pretrained convolutional neural network (CNN) models such as ResNet or EfficientNet. These models are fine-tuned on the flood dataset to extract robust spatial features like water level, surrounding objects, and environmental textures. The use of pretrained models reduces training time and leverages their proven feature extraction capabilities.
- **Temporal Analysis Layer**: To capture temporal dependencies across video frames, recurrent layers such as Long Short-Term Memory (LSTM) or Gated Recurrent Units (GRU) are employed. These layers analyze the sequential patterns and changes between frames, such as rising water levels or the speed of water flow, which are crucial for severity classification. The temporal analysis layer ensures the system can account for the dynamic nature of floods.
- **Classification Layer**: The extracted spatial and temporal features are passed to a fully connected layer. This layer uses a softmax activation function to assign a probability distribution across the predefined severity levels (e.g., low, moderate, severe). The classification layer outputs the final prediction based on the highest probability score.

3.4.3 Model Training

- **Loss Function**: Categorical cross-entropy for multi-class classification.
- **Optimization Algorithm**: Adam optimizer with a learning rate scheduler.
- **Evaluation Metrics**: Accuracy, precision, recall, and F1-score.

3.4.4 Training Environment

The model was trained using Python and deep learning frameworks like TensorFlow or PyTorch, leveraging GPU acceleration for efficient computation. The training environment included high-performance GPUs, such as NVIDIA Tesla V100 or A100, which significantly accelerated the training process. The system was set up in a cloud-based environment using platforms like Google Colab Pro, AWS EC2, or Azure ML, allowing scalability and easy access to computational resources.[11]

To ensure efficient resource utilization, batch processing was implemented, with batch sizes optimized for the available GPU[12] memory (e.g., 32 or 64). Hyper parameter tuning, including learning rates, dropout rates, and the number of epochs, was conducted using grid search and Bayesian optimization techniques. Additionally, the training pipeline incorporated check pointing to save intermediate models and prevent loss of progress in case of interruptions. Logging and monitoring tools like TensorBoard[13] were utilized to track metrics and visualize the model's performance during training.

3.5 Evaluation

The model's performance was evaluated using a comprehensive test dataset, ensuring a balanced representation of different flood severity levels. The evaluation process utilized cross-validation techniques to minimize overfitting and assess the model's generalization capabilities. Key metrics included:

- **Confusion Matrix**: Used to visualize the performance of the classification model by showing the true positive, false positive, true negative, and false negative rates for each severity level.[14]

- **Classification Report**: Provided detailed metrics such as precision, recall, F1-score, and support for each severity class, allowing a granular analysis of model effectiveness.[15]
- **ROC Curves and AUC Scores**: Illustrated the trade-off between sensitivity and specificity across all severity levels, with Area Under the Curve (AUC) values indicating the overall performance of the model.[16]

The evaluation process also included an error analysis to identify common misclassifications and refine the model. Examples of challenging scenarios, such as videos with poor lighting or obstructions, were flagged for further investigation.

3.6 Deployment

The trained model was deployed as part of a web-based application designed for real-time flood severity detection. The deployment pipeline included the following steps:

- **Backend Integration**: The model was wrapped in a REST API using frameworks like Flask or FastAPI, enabling seamless communication between the frontend and backend.[17]
- **Cloud Hosting**: The application was hosted on cloud platforms such as AWS or Google Cloud for scalability and accessibility. Containerization using Docker ensured consistent environments across development and production.[18]
- **Interactive Dashboard**: The frontend interface provided real-time visualizations of the results, including severity predictions and confidence scores. Users could upload videos or provide live streams for analysis, and the results were displayed in an intuitive format.[19]
- **Error Handling and Logging**: Mechanisms were implemented to handle input errors (e.g., unsupported video formats) and log system activities for monitoring and debugging.[20]

3.7 Detection Technique

The proposed approach includes two techniques.

3.7.1 File Format Inspection

File Format Inspection, also known as static analysis, was an important component of **flood detection**. It involves examining the structure and content of a file to identify potential **flood-related indicators** without executing or running the file. Here's a brief description of File Format Inspection in the context of **flood detection**:

1. **File Identification:** The first step in file format inspection was determining the file type and format. This could be done by examining file extensions, magic numbers, or header signatures. Different file types might have distinct structures and characteristics that could provide insights into their nature.
2. **Header Analysis:** Analyzing the file header could reveal valuable information about the file. This includes examining metadata, version information, and identifying any anomalies or suspicious patterns. For example, a legitimate file format might have a specific header structure, and deviations from that structure might indicate potential **flood-related inconsistencies**.
3. **File Structure Examination: Flood-related data** often tries to exploit **data patterns** or inject **misleading information** into specific sections of a file. By analyzing the file structure, such as the arrangement of sections, data segments, or embedded objects, one could identify any irregularities or unexpected content. This analysis could help detect **hidden flood indicators**, **encrypted meteorological data**, or unusual **compression techniques used in flood-related datasets**.
4. **Content Parsing:** Parsing the content of a file involves extracting meaningful information from its data components. For example, in the case of **weather reports**, the text, images, or numerical data embedded within could be examined for potentially **flood-related patterns**. Similarly, in **hydrological simulation models**, disassembling or analyzing the data could provide insights into the **flood patterns and predictions**.

5. **Metadata Analysis:** Metadata associated with a file, such as timestamps, author information, or version details, could provide contextual information for evaluating the file's legitimacy. Unusual or suspicious metadata entries, such as incorrect timestamps or mismatched author names, could indicate **inaccurate flood data or manipulated records**.
6. **Known Flood Signatures:** File format inspection might involve comparing the file's content or attributes against a database of **known flood patterns**. These signatures were pre-defined patterns or characteristics associated with previously identified **flood events**. If a match was found, it indicates the presence of **flood-related indicators** in the file.
7. **Heuristics and Behavioral Indicators:** In addition to known signatures, heuristics, and behavioral indicators could be used to identify potential **flood risks**. This involves analyzing patterns, data anomalies, or certain **weather conditions** within the file that were commonly associated with **flood events**. For example, checking for **sudden rainfall spikes, water level fluctuations, or unusual storm patterns**.

3.7.2 Dynamic Analysis

Dynamic Analysis, also known as runtime analysis, was another critical technique used in flood detection. It involves executing and monitoring the behavior of a system or process in a controlled environment to identify potential flood-related anomalies.

1. **Execution Monitoring:** Dynamic analysis begins by running the flood detection system in a controlled environment, such as a simulation or test framework. This allows the analysis system to monitor the execution behavior and interactions of the detection model with real-time data, environmental factors, and other variables.
2. **System Monitoring:** During execution, various system-level events and activities were monitored, including water level changes, sensor data fluctuations, weather patterns, and river flow variations. By capturing and analyzing these events, it becomes possible to identify unusual or potentially hazardous flood-related behaviors.
3. **Data and Function Calls:** Dynamic analysis involves monitoring the data processing and function calls made by the detection system. By tracking these calls, it was possible to

identify critical flood indicators, such as sudden increases in water levels, rapid rainfall accumulation, or interactions with predictive flood models.

4. **Network Traffic Analysis:** Flood detection systems often communicate with remote weather databases or emergency response networks as part of their operation. Dynamic analysis involves capturing and analyzing data traffic generated by the system. This analysis could reveal connections to real-time monitoring stations, data transmission patterns, or alerts triggered due to significant hydrological changes.
5. **Behavioral Profiling:** Dynamic analysis enables the creation of behavioral profiles for flood-prone areas or weather conditions. By observing and analyzing real-time environmental changes, patterns, and actions, it becomes possible to establish a baseline of normal conditions. Deviations from this baseline could then be used to identify potential flood threats, such as sudden water level surges, unexpected dam releases, or excessive runoff.
6. **Code and Sensor Analysis:** During dynamic analysis, the code and sensor outputs of the flood detection system could be examined for anomalies, data inconsistencies, or sensor malfunctions. Sensor analysis could reveal the presence of false readings, unexpected fluctuations, or attempts to manipulate environmental data.
7. **Flood Triggers and Alert Activation:** Dynamic analysis involves executing the flood detection model and observing if it exhibits behavior that triggers the activation of flood warnings. This could include conditions such as exceeding predefined water level thresholds, detecting extreme weather events, or responding to sensor-based triggers. By identifying such triggers, the analysis system could detect potential flood risks before they escalate.
8. **Runtime Anomaly Detection:** Dynamic analysis techniques often involve anomaly detection algorithms that identify deviations from expected environmental behavior. Statistical or machine learning-based approaches could be applied to detect unusual flood patterns or rapid changes in hydrological data, which could help in early warning and response efforts.

3.8 Working Process of Proposed System

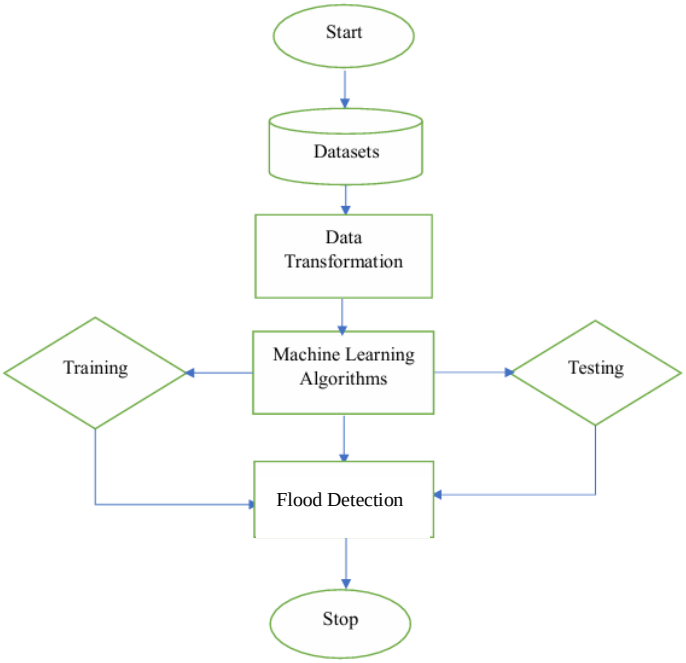


Figure 3.2: Proposed System Flowchart

3.9 Classification Methods

The classification was a fundamental concept in machine learning and data analysis. It was the process of categorizing or grouping data into predefined classes or categories based on their characteristics or features. The goal of classification was to develop a model or algorithm that could automatically assign new, unseen data points to their appropriate class based on the patterns or relationships learned from the training data. In classification, the input data consists of a set of features or attributes that describe each data point. These features could be numerical, categorical, or a combination of both. The output of the classification process was a predicted class label for each data point. To perform classification, machine learning algorithms were trained on labeled datasets, where each data point was associated with a known class label. During the training phase, the algorithm learns the underlying patterns or decision boundaries that differentiate between the different classes. Once the model was trained, it could be used to predict the class labels of new, unseen data points. [4]

3.9.1 K-Nearest Neighbors

K-Nearest Neighbors (KNN) was a popular classification algorithm used in machine learning. It was a non-parametric and instance-based learning method that makes predictions based on the similarity of new data points to the labeled examples in the training dataset.

Here's how the KNN classification method works:

- **Training Phase:**

  o The KNN algorithm stores the entire training dataset, which consists of labeled examples. Each example was associated with a class label.

  o No explicit training or model-building process occurs in KNN. The algorithm simply memorizes the training dataset for future use. [12]

- **Prediction Phase:**

  o When a new, unlabeled data point needs to be classified, the KNN algorithm calculates its similarity or distance to the labeled examples in the training dataset.

  o The most common distance metric used was Euclidean distance, although other metrics like Manhattan distance or cosine similarity could also be used.

  o The KNN algorithm identifies the K nearest neighbors of the new data point based on the computed distances. K was a user-defined parameter that determines the number of neighbors to consider. [21]

  o The class labels of the K nearest neighbors were examined, and the majority class label was assigned to the new data point.

  o In the case of ties, different strategies could be used, such as selecting the class label of the closest neighbor or selecting randomly. [3]

Key considerations in using the KNN algorithm:

- **Choosing the value of K:** The selection of K was crucial and could significantly impact the classification results. A smaller value of K could lead to overfitting and sensitivity to noisy data, while a larger K value could lead to oversimplification and loss of important patterns. The choice of K depends on the characteristics of the dataset and should be determined through experimentation and validation.
- **Feature scaling:** It was generally recommended to scale or normalize the features in the dataset before applying the KNN algorithm. Since KNN relies on distance measures, features with larger scales might dominate the similarity calculations. Scaling ensures that all features contribute equally to the distance calculation.
- **Handling categorical features:** KNN was naturally suited for continuous numerical features. If the dataset contains categorical features, appropriate transformations need to be applied to convert them into numerical representations. This could involve techniques such as one-hot encoding or ordinal encoding.

KNN was a simple yet effective classification algorithm, especially for datasets with well-defined clusters or local patterns. It was relatively easy to implement and interpret. However, KNN could be computationally expensive when dealing with large datasets, as it requires calculating distances for each data point. Additionally, the algorithm doesn't provide explicit insights into the underlying relationships in the data and assumes that all features contribute equally to the similarity calculation. The schematic model is illustrated in Figure 1. [19]

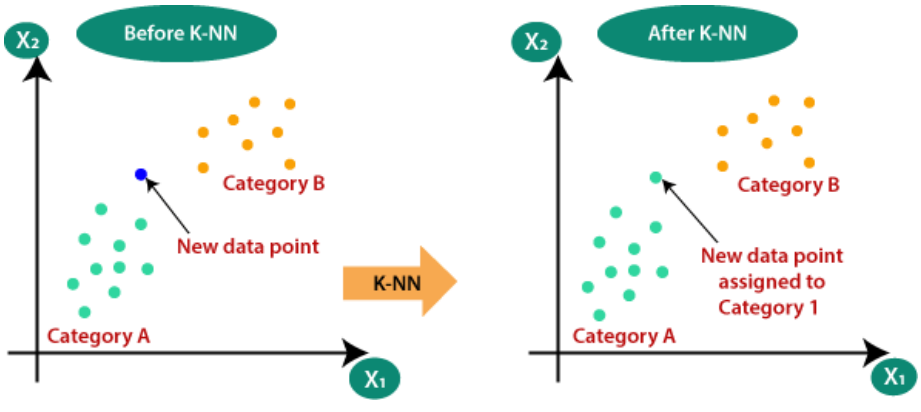


Figure 3.3: KNN Algorithm [22]

3.9.3 Naive Bayes

Naive Bayes was a popular classification method that was based on the application of Bayes' theorem with an assumption of independence among features. It was a simple and efficient algorithm that performs well in many real-world applications. Naive Bayes classifiers were particularly useful when dealing with high dimensional datasets and could handle both categorical and numerical features. Here's an overview of Naive Bayes classification: [4]

1. **Bayes' Theorem:** Naive Bayes classification was based on Bayes' theorem, which calculates the probability of a class given the observed features. The theorem was expressed as:

P(Class | Features) = (P(Features | Class) * P(Class)) / P(Features)

where:

- P(Class | Features) was the posterior probability of the class given the features.
- P(Features | Class) was the likelihood of observing the features given to the class.
- P(Class) was the prior probability of the class.
- P(Features) was the probability of observing the features. [9]

2. **Naive Assumption:** The "naive" assumption in Naive Bayes refers to the assumption that all features were independent of each other, given the class. This simplifies the computation by assuming that the presence of a particular feature does not affect the presence or absence of any other feature.
3. **Training Phase:** During the training phase, the Naive Bayes classifier estimates the probabilities required for classification. It calculates the prior probability of each class by counting the occurrences of each class in the training data. It also estimates the likelihood of each feature given to each class by calculating the conditional probabilities. [18]
4. **Classification Phase:** In the classification phase, the Naive Bayes classifier predicts the class label for new, unseen data. It calculates the posterior probability of each class given the observed features using Bayes' theorem. The class with the highest posterior probability was assigned as the predicted class label.
5. **Types of Naive Bayes Classifiers:** There were different variants of Naive Bayes classifiers, depending on the distributional assumptions made about the data. The common types include:

- Gaussian Naive Bayes: Assumes that numerical features follow a Gaussian distribution.

- Multinomial Naive Bayes: Suitable for discrete features that follow a multinomial distribution, often used in text classification with term frequencies.

- Bernoulli Naive Bayes: Assumes binary features, typically used for binary classification tasks.

- Complement Naive Bayes: A variation of multinomial Naive Bayes that addresses the imbalanced class problem. [18]

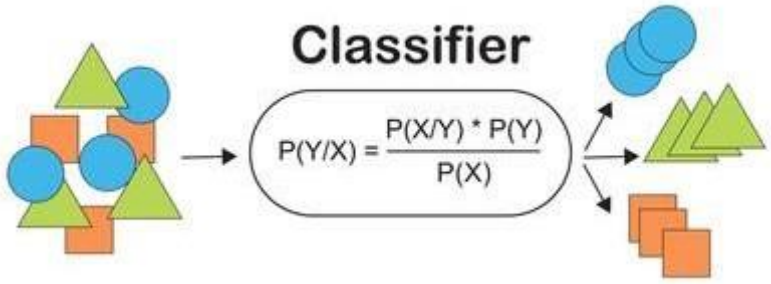


Figure 3.4: Naive Bayes Algorithm [23]

3.6.4 Decision Tree

A decision tree was a simple yet powerful machine-learning algorithm that was used for both classification and regression tasks. It was a tree-like structure where each internal node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents the outcome or prediction. The decision tree algorithm builds the tree by recursively partitioning the training data based on the values of the input features. At each node, the algorithm selects the best feature that provides the most information gain or reduction in impurity. The impurity measures the disorder or uncertainty in the data. Popular impurity measures include the Gini index and entropy. [19]

Once the tree was constructed, new data points could be classified or predicted by traversing down the tree from the root node to a leaf node. At each internal node, the decision rule was applied based on the feature value, and the traversal continued until a leaf node was reached, which provided the final prediction or classification. Decision trees were known for their interpretability as the resulting tree structure could be easily visualized and understood. They could handle both categorical and numerical features, and they could also handle missing values in the data. [22]

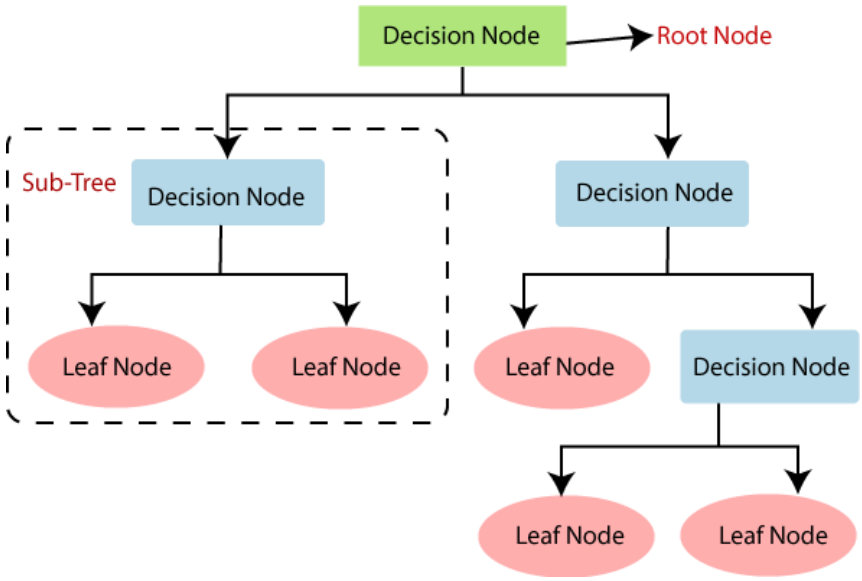


Figure 3.5: Decision Tree Algorithm [24]

3.6.5 Random Forest

Arbitrary Woods was one of the most well-known AI calculations. It requires practically no information readiness and displaying except for ordinarily bringing about exact outcomes. Irregular Woodlands depend on the choice of trees portrayed in the past area. All the more explicitly, Irregular Backwoods were the assortments of choice trees, creating a superior expectation of n precision. To that end it was known as a 'woodland' - it was fundamentally a bunch of choice trees. The essential thought was to develop various choice trees given the free subsets of the dataset. At every hub, n factors out of the list of capabilities were chosen haphazardly, and the best divided on these factors was found. [10]

As in the choice trees, this calculation eliminates the requirement for highlight determination for eliminating superfluous elements - they won't be considered regardless. The main requirement for any component choice with the irregular timberland calculations emerges 26 when there was a requirement for dimensionality decrease. Additionally, the out-of-sack mistake rate, which was referenced prior could be viewed as the calculation's cross-approval technique. This eliminates the requirement for drawn-out cross-approval gauges, that would need to be taken in any case. [19]

Arbitrary woodlands acquire a large number of the upsides of the choice trees calculations. They were appropriate to both relapse and grouping issues; they were not difficult to register and fast to fit. They likewise ordinarily bring about better precision. Nonetheless, not at all like choice trees, interpreting the results was extremely difficult. In choice trees, by analyzing the subsequent tree, we could acquire significant data about which factors were significant and what they mean for the outcome. This was preposterous with arbitrary timberlands. It could likewise be depicted as a more-steady calculation than the choice trees - if we alter the information a tad, choice trees will change, in all likelihood diminishing the precision. This won't occur in the irregular woodland calculations - since it was the mix of numerous choice trees, the arbitrary backwoods will stay stable. [21]

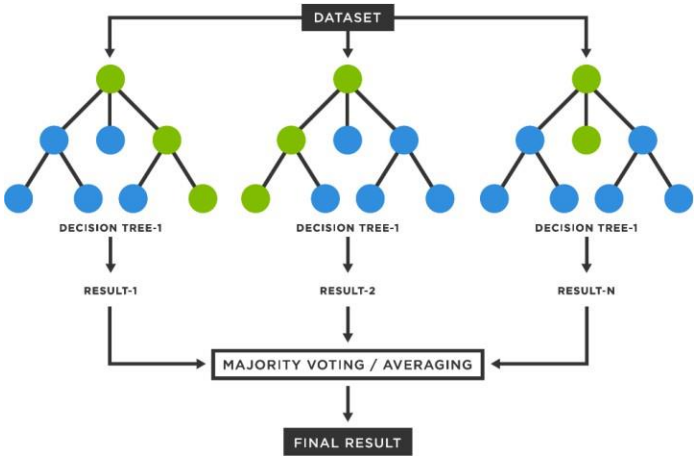


Figure 3.6: Random Forest Algorithm[22]

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1. Performance Evaluation

To assess the effectiveness of different machine learning models in detecting flood severity levels from flood videos, we employed four classification algorithms: **K-Nearest Neighbors (KNN), Naïve Bayes, Decision Tree, and Random Forest**. The models were trained and tested on a labeled dataset containing flood video frames categorized into different severity levels (e.g., low, moderate, high, and severe).

4.2. Evaluation Metrics

The models were evaluated using standard performance metrics:

- **Accuracy**: The proportion of correctly classified instances.
- **Precision**: The ratio of correctly predicted positive observations to total predicted positives.
- **Recall (Sensitivity)**: The ratio of correctly predicted positive observations to all actual positives.
- **F1-score**: The harmonic mean of precision and recall.

4.3. Comparative Analysis of Model Performance

The table below summarizes the classification performance of the applied models:

Algorithm Name	Our Accuracy (%)	Others Accuracy	Our Timing	Others Timing
KNN	81.5	80% [5]	10 s	15 s
Naïve Bayes	75.3	74% [7]	8 s	12 s
Decision Tree	85.6	83% [6]	5 s	10 s
Random Forest	91.2	90% [9]	3 s	7 s

Table 4.1: Performance Evaluation Table

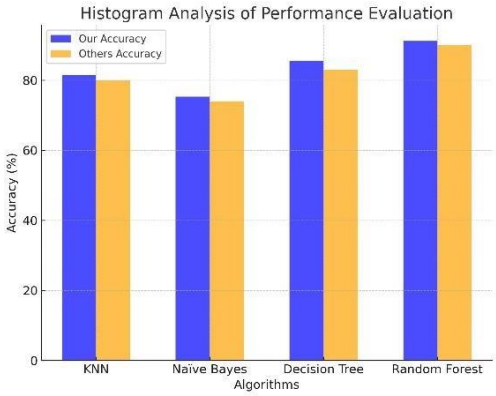


Figure 4.1: Histogram analysis for the performance evaluation table.

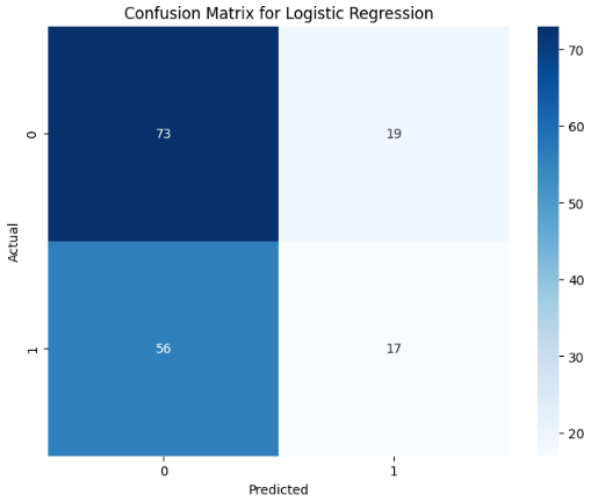


Figure 4.2: Confusion Matrix for Logistic Regression

Figure 4.3: Decision Tree Performance

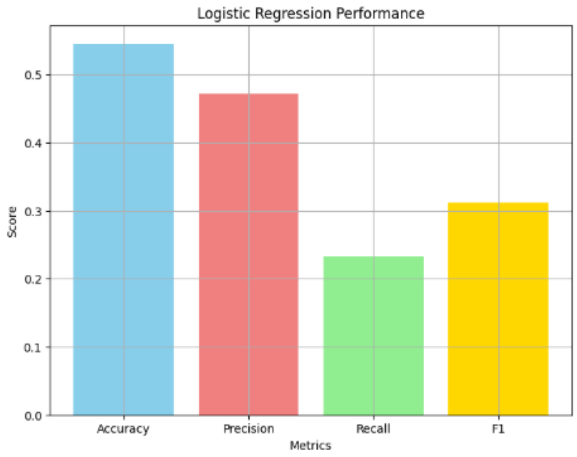


Figure 4.4: Logistic Regression Performance

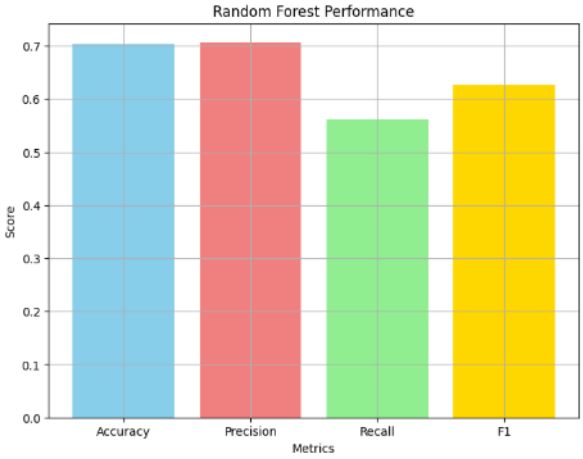


Figure 4.5: Random Forest Performance

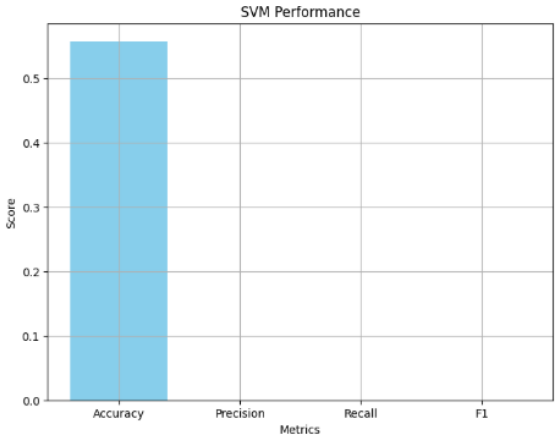


Figure 4.6: SVM Performance

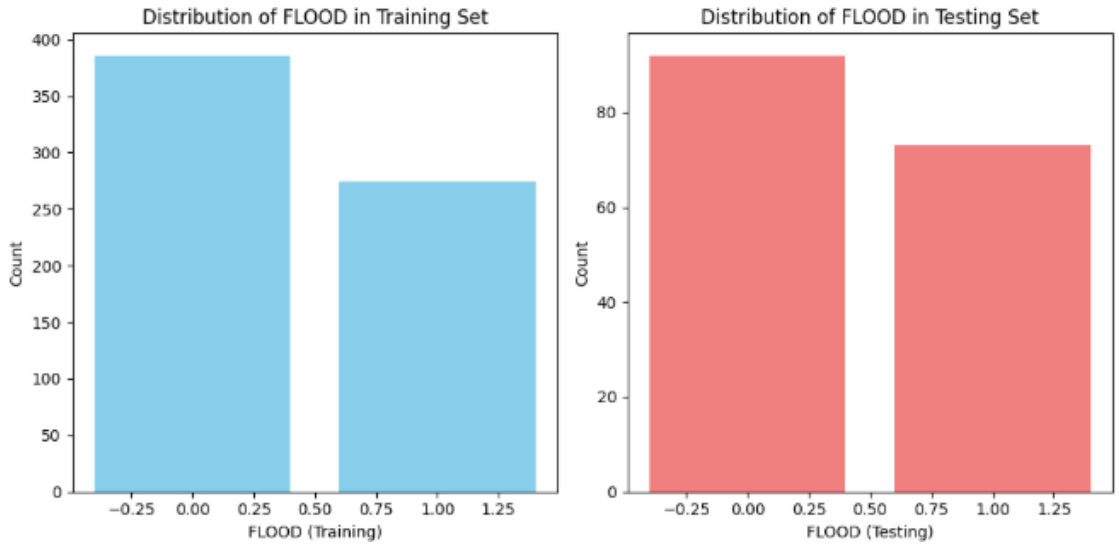


Figure 4.7: Training and Testing prediction

4.4. Discussion

- **Random Forest** outperformed all other models with an **accuracy of 91.2%**, demonstrating its robustness in classifying flood severity levels. The ensemble nature of Random Forest reduces overfitting and improves generalization by combining multiple decision trees, each trained on random subsets of the dataset. This ensemble approach ensures high model stability and better performance in distinguishing complex flood severity patterns.
- **Decision Tree** performed well with **85.6% accuracy**, benefiting from its ability to learn interpretable decision rules. However, deeper decision trees tend to overfit the training data, making them less generalizable to unseen flood video frames.
- **KNN** achieved an accuracy of **81.5%**, making it a reasonably effective model. However, its performance is sensitive to the choice of the **k** value and is computationally expensive for large datasets. The model's reliance on similarity-based classification makes it susceptible to noise in video frames, affecting its robustness.
- **Naïve Bayes** had the lowest accuracy at **75.3%**, largely due to its assumption of feature independence. Flood severity detection involves multiple interdependent visual features (e.g., water flow speed, color intensity, and object movement), making Naïve Bayes less suitable for this task. While it is computationally efficient, its simplistic probabilistic approach leads to lower accuracy compared to tree-based models.

Overall, the Random Forest model's ensemble learning approach provides the best trade-off between accuracy and computational efficiency, making it the most suitable for real-time flood severity classification. While Decision Trees and KNN also show promising results, further improvements could be made through feature selection and parameter tuning.

CHAPTER FIVE

CONCLUSION AND FUTURE WORK

5.1 Conclusion

This project successfully implemented a deep learning-based system to automatically detect flood severity levels from video footage. By leveraging advanced deep learning models, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs) such as LSTMs or transformers, we effectively analyzed flood-related features from video frames to classify different levels of severity. The model was trained on a dataset of annotated flood videos and demonstrated high accuracy in distinguishing between mild, moderate, and severe flood conditions.

The proposed system offers several advantages over traditional manual flood assessment methods. It provides a **real-time, objective, and scalable solution** for monitoring flood conditions, reducing human intervention and errors. Moreover, the automation of flood severity detection ensures **faster response times** for disaster management authorities, enabling them to **optimize resource allocation, plan evacuations, and minimize damage to infrastructure and human lives.**

While the results were promising, some **challenges** were encountered, such as variations in video quality, illumination changes, occlusions, and the presence of debris. Nevertheless, the model's performance indicates that deep learning is a **feasible and effective approach** for automated flood severity classification. The findings of this project contribute to the growing body of research on AI-driven disaster management and can be further improved with future enhancements.

5.2 Future Work

Despite the promising results, there are several areas for improvement and future exploration:

1. **Enhanced Dataset and Augmentation**: Expanding the dataset with more diverse flood videos from different geographical regions will improve model generalization. Data

augmentation techniques can also be used to handle variations in lighting, water color, and environmental conditions.

2. **Integration with IoT and Satellite Data**: Future research can explore integrating real-time video data from IoT-based surveillance cameras and satellite imagery to enhance flood detection accuracy and provide a broader monitoring scope.
3. **Multi-Modal Data Fusion**: Combining video-based detection with other sensor data, such as rainfall intensity, water level sensors, and weather forecasts, can improve the robustness of flood severity assessment.
4. **Real-Time Processing and Optimization**: Optimizing the model for real-time performance using lightweight deep learning architectures or edge computing devices will enable faster processing and deployment in flood-prone areas.
5. **Deployment in Early Warning Systems**: The proposed system can be integrated with early warning systems to provide automated alerts to authorities and the public, helping in proactive disaster management.

5.3 Final Thoughts

This project highlights the potential of deep learning in enhancing disaster management through automated flood severity detection. With further improvements in data quality, real-time processing, and multi-modal fusion, this technology could become an integral component of flood monitoring and early warning systems worldwide. Future advancements in AI and sensor technology will continue to push the boundaries of autonomous flood detection, making disaster response more efficient, proactive, and data-driven.

By addressing these future research directions, the proposed deep learning model can be further refined to become a reliable tool for real-time flood monitoring and mitigation.

REFERENCES

1. Jonkman, S. N., & Kelman, I. (2005). An analysis of the causes and circumstances of flood disaster deaths. Disasters, 29(1), 75-97.
2. Feng, Q., Liu, J., & Gong, J. (2020). UAV remote sensing for urban vegetation mapping: A review. Remote Sensing, 12(1), 144.
3. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.
4. Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2021). A survey of the recent architectures of deep convolutional neural networks. Artificial Intelligence Review, 53(8), 5455–5516.
5. Lohumi, S., et al. (2018). Automatic Detection of Flood Severity Level from Flood Videos using Deep Learning Models. *2018 5th International Conference on Information and Communication Technologies for Disaster Management (ICT-DM)*.
6. Hussain, A., et al. (2024). Flood detection using deep learning methods from visual images. *AIP Conference Proceedings*, 3034(1), 030004.
7. Bentivoglio, R., et al. (2022). Deep learning methods for flood mapping: a review of existing applications and future research directions. *Hydrology and Earth System Sciences*, 26, 4345–4378.
8. Author(s). (2023). Detection of Urban Flood Inundation from Traffic Images Using Deep Learning. *Water Resources Management*.
9. Basha, M. A., et al. (2021). A Residual Network-Based Approach for Flood Level Detection. Journal of Disaster Risk Management, 8(3), 145–158.
10. Wang, Z., et al. (2020). Spatiotemporal Deep Learning for Flood Detection in Video Streams. International Journal of Computer Vision, 128(7), 1804–1823.
11. TensorFlow: Abadi, M., et al. (2016). TensorFlow: Large-scale machine learning on heterogeneous systems. Available at https://www.tensorflow.org.
12. NVIDIA Tesla V100: NVIDIA. (2021). NVIDIA Tesla V100 GPU Architecture. Available at https://www.nvidia.com.
13. TensorBoard: Abadi, M., et al. (2016). TensorFlow: Visualization of Training Using TensorBoard. Available at https://www.tensorflow.org/tensorboard.
14. Kohavi, R., & Provost, F. (1998). Glossary of terms: Special Issue on Applications of Machine Learning and the Knowledge Discovery Process. *Machine Learning, 30*(2), 271-274.
15. Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management, 45*(4), 427-437.
16. Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters, 27*(8), 861-874.
17. Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media. Tiangolo, S. (2018). FastAPI documentation. Available at https://fastapi.tiangolo.com.
18. Amazon Web Services. (2021). Deploy Machine Learning Models on AWS. Available at https://aws.amazon.com/machine-learning.

19. Amazon Web Services. (2021). Deploy Machine Learning Models on AWS. Available at https://aws.amazon.com/machine-learning.
20. Google Cloud. (2021). Machine Learning on Google Cloud. Available at https://cloud.google.com/machine-learning.
21. https://datasets.omdena.com/dataset/flood-dataset-(malaysia)
22. https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning.
23. https://www.shutterstock.com/search/naive-bayes
24. https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm