# Analyzing Social Media Images and Text for Disaster Response

## Problem Statement

Disaster response teams need timely and accurate information to prioritize resources and save lives. With the growing use of social media, platforms like Twitter and Instagram serve as valuable sources for real-time data on disasters. However, manually processing and filtering vast amounts of multimedia posts to identify critical disaster-related content is time-consuming and inefficient. The challenge lies in developing a system that can automatically process both images and text from social media posts to detect disaster-related information, ensuring faster response times and resource allocation.

## Problem Deliverables

The project aimed to deliver a comprehensive system capable of:

1. Detecting disaster-related objects and events in images.
2. Analyzing text posts to identify disaster-related information such as calls for help or reports of damage.
3. Integrating both image and text classification models into a web application that allows real-time processing of social media posts.
4. Ensuring the system can handle multiple requests simultaneously, offering seamless user interaction and scalability.

## Metrics to Calculate

The performance of both models was evaluated using several key metrics:

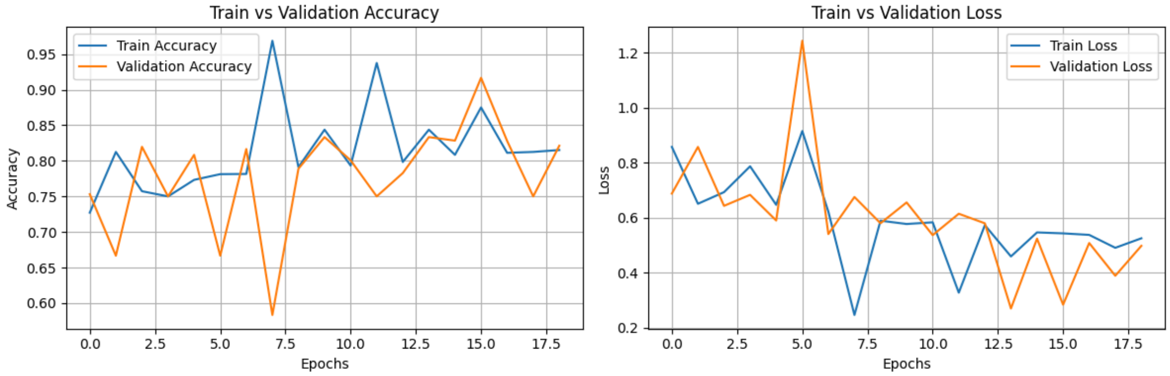
- **Accuracy:** To measure the proportion of correctly classified disaster-related content.
- **F1-Score:** To assess the model's ability to correctly identify disaster-related content without too many false positives or negatives.
- **Confusion Matrix:** For visualizing the classification performance across different disaster categories.
- **Loss Graphs:** To track the model's convergence and generalization to unseen data.

# Approach

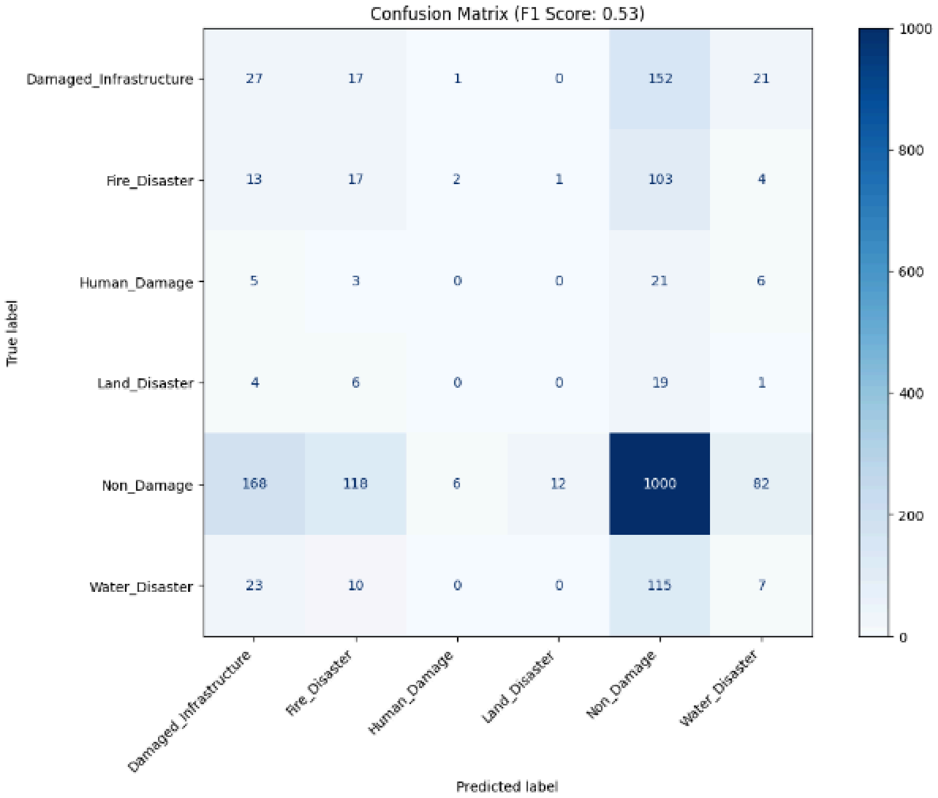
### Image-based Classification: Approach, Problems, and Results

For image classification, I used **ResNet50**, a popular deep learning model, through transfer learning. Given the limited time and the dataset's size, transfer learning allowed me to leverage a pre-trained network and fine-tune it for the disaster image dataset. The dataset posed challenges, such as images of various sizes, corrupt files, and the multi-class nature of the classification problem. To address these, I resized all images to a consistent size and normalized the pixel values by scaling them with 1/255. This preprocessing step helped standardize the input data for the ResNet50 model.

The model's architecture, designed for image recognition tasks, was fine-tuned with early stopping to prevent overfitting, stopping at **19 epochs** when validation accuracy plateaued. The model achieved a training accuracy of **81.47%** and a validation accuracy of **82.12%**. The loss scores were **0.5248** (training) and **0.4975** (validation), indicating that the model was learning well without overfitting. However, the F1 score of **0.53** suggested that the model faced difficulty in handling class imbalances, particularly when classifying minority disaster categories. To further optimize, techniques such as data augmentation or oversampling could be considered in future work to address these imbalances.



**Accuracy and Loss Graphs**



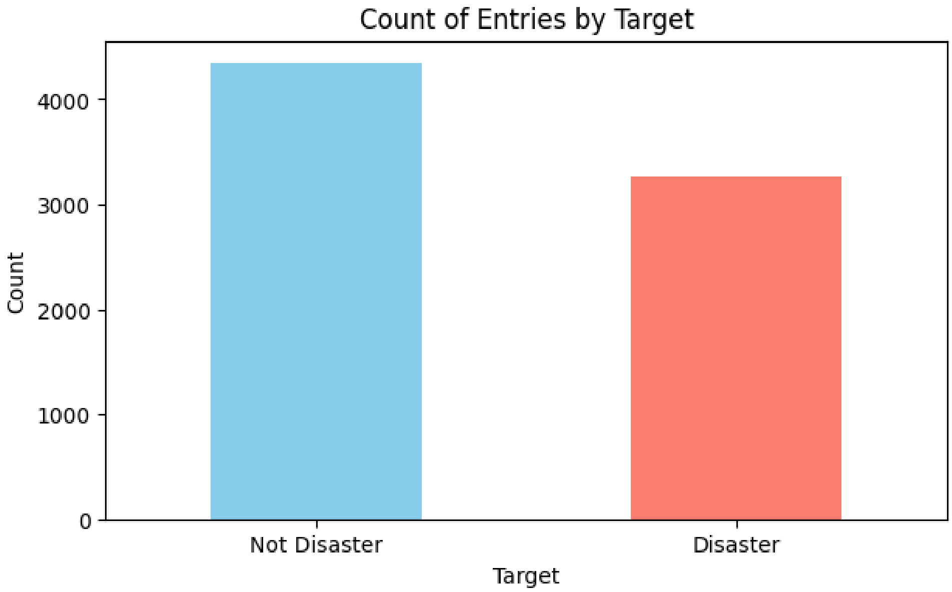
**Confusion Matrix for Disaster Image Model**

### Text-based Classification: Approach, Problems, and Results

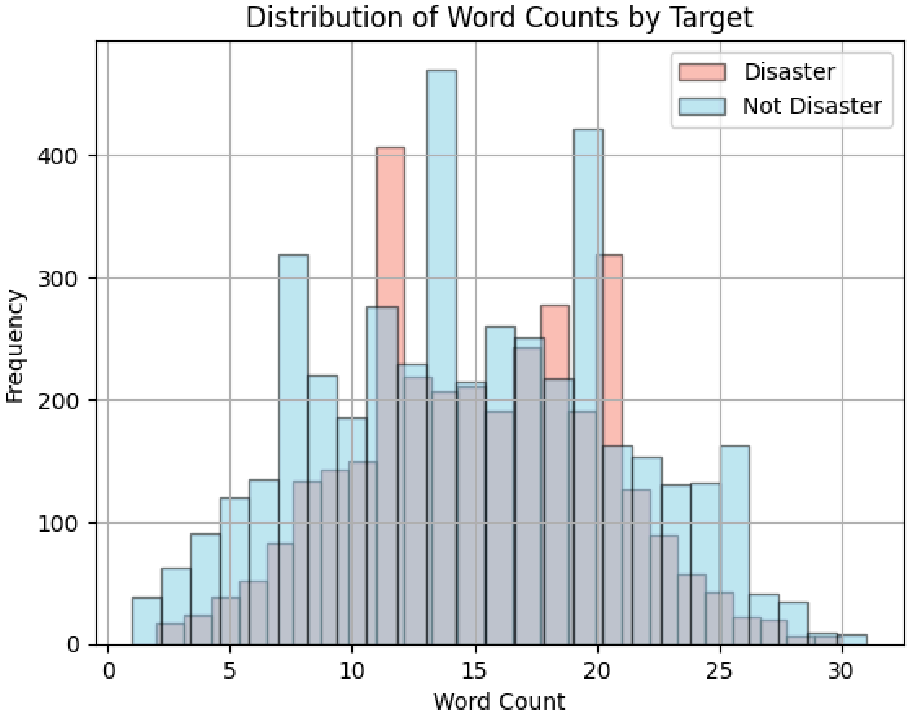
For the text classification task, I opted for a traditional deep learning approach using a **Conv1D neural network**. The Keras tokenizer was employed to transform the raw social media text into numerical sequences, which were then padded to a fixed length. Given the unstructured and noisy nature of social media data—where spelling, grammar, and abbreviations vary—I processed the data using tokenization and sequence padding techniques. The model was designed to handle binary classification, identifying whether the text pertained to a disaster or not, and trained using early stopping to prevent overfitting.

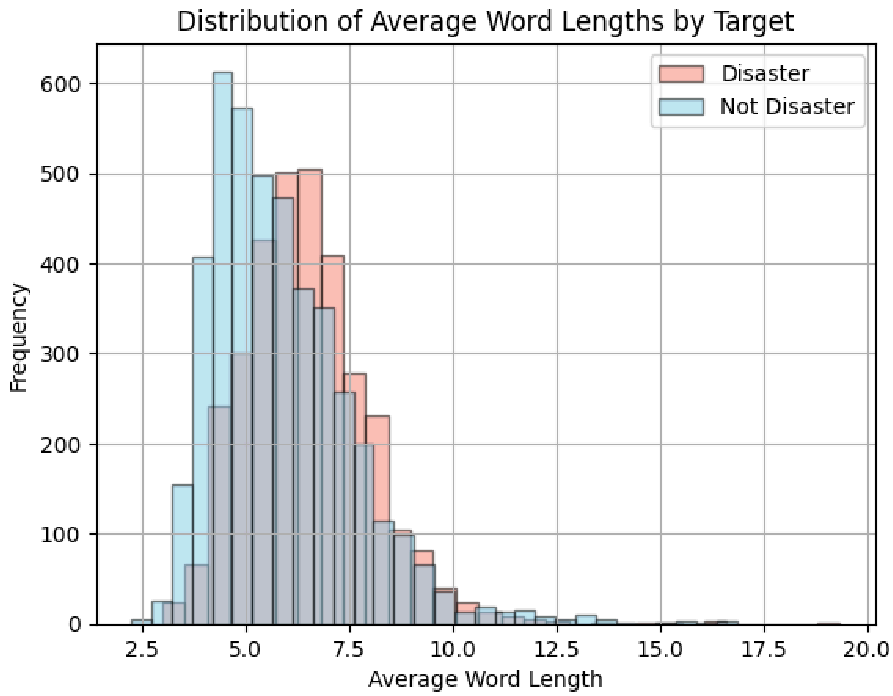
Before model training, we conducted extensive Exploratory Data Analysis (EDA) to gain insights into the dataset:

1. Analyzed the count distribution of location information and target variables.

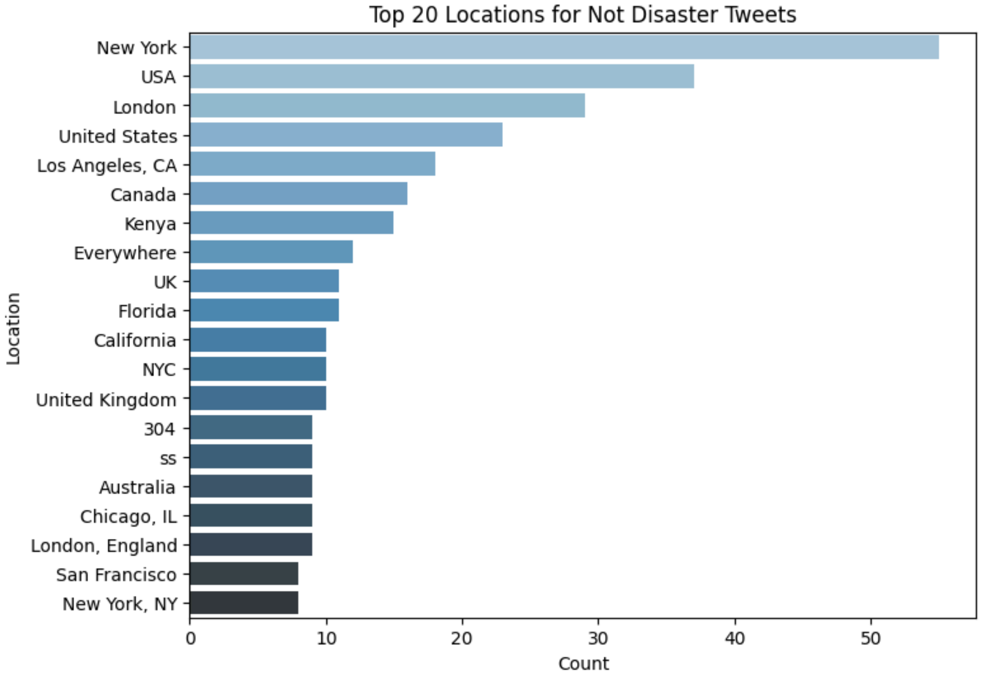


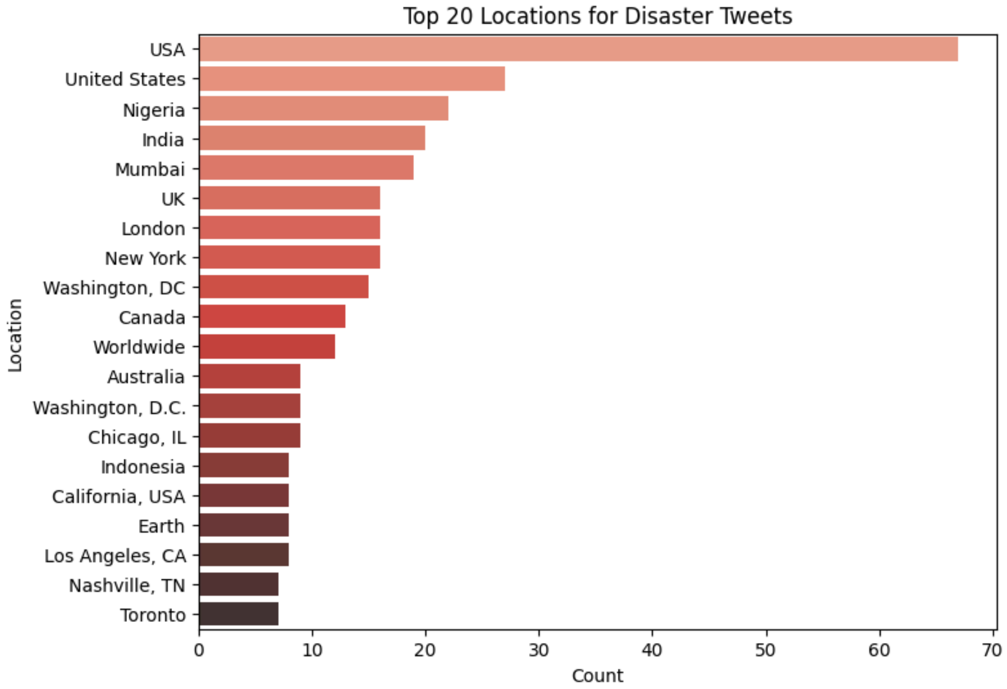
2. Examined the distribution of disaster vs. non-disaster tweets.



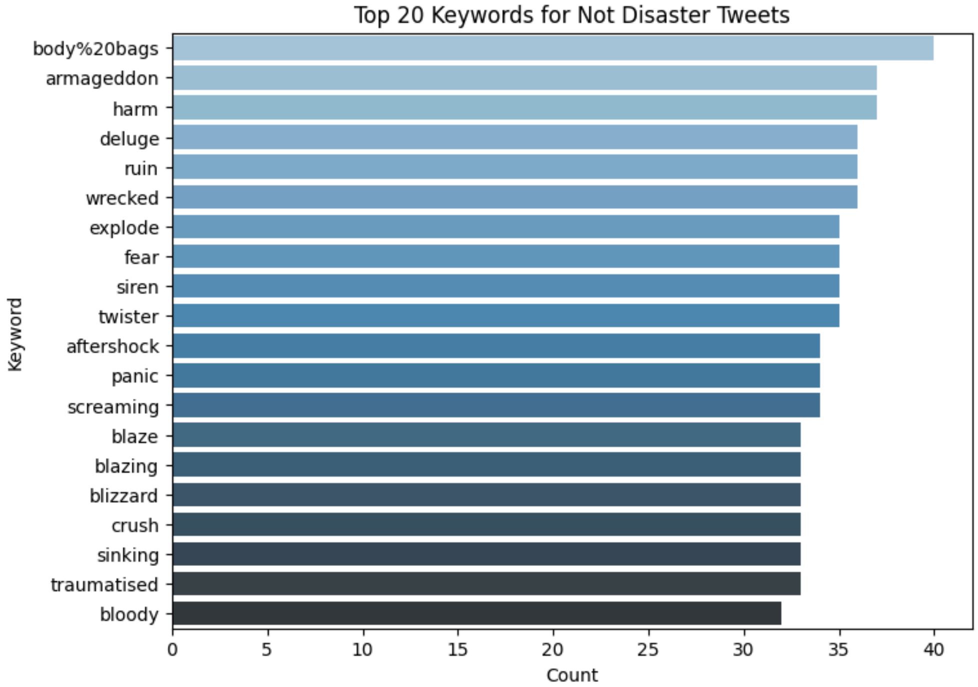


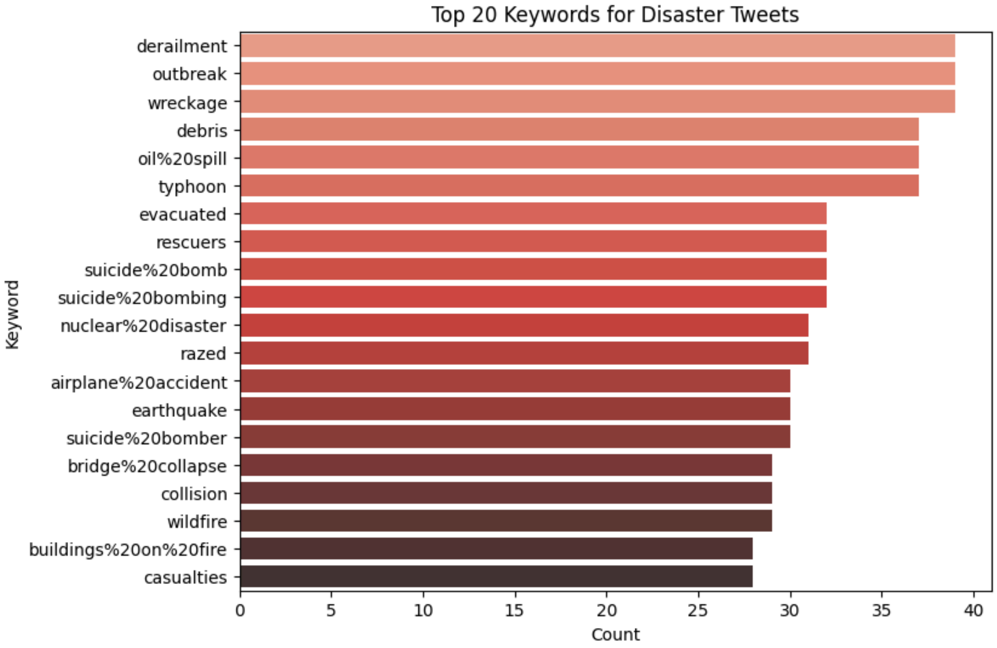
3. Identified the top 20 locations for both disaster and non-disaster tweets, providing geographical context to the data.





4. Analyzed commonly used hashtags, which helped in understanding trending topics and potential disaster-related keywords.



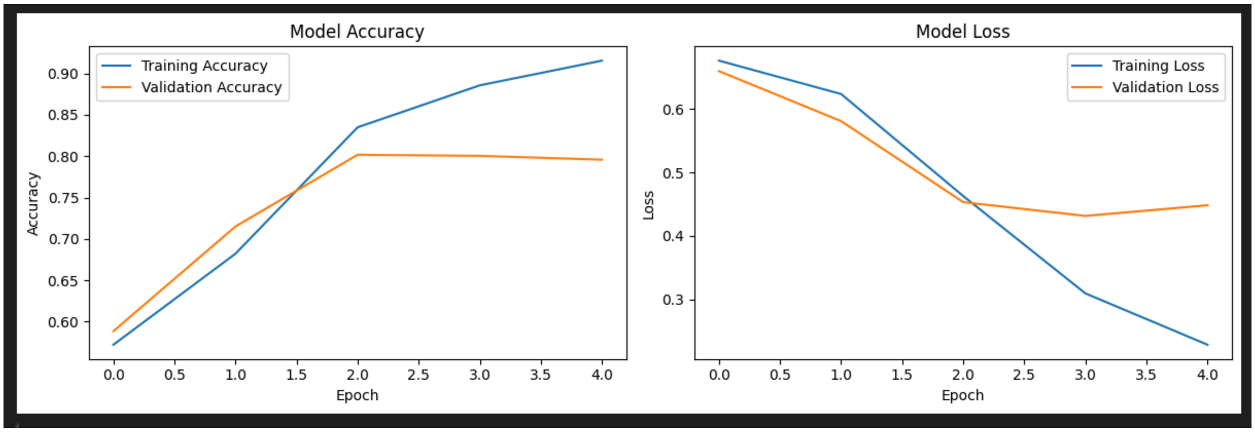


This EDA process was crucial in understanding the data characteristics and informed our preprocessing and modeling decisions.

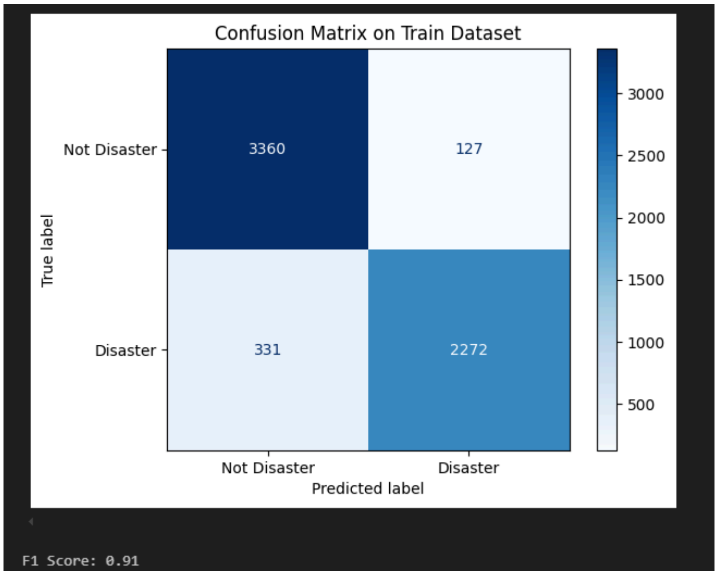
To optimize the performance, I again applied transfer learning with early stopping, halting training after **5 epochs** when validation accuracy began to stabilize. The model achieved a training accuracy of **91.63%** and a validation accuracy of **79.58%**, with an F1 score of **0.91** for the training set and **0.75** for the validation set.

These results indicate that the model was effective at identifying disaster-related text content, showing a good balance between precision and recall. The high training accuracy suggests that the model learned the patterns in the training data well, while the lower but still respectable validation accuracy indicates some generalization to unseen data. The F1 scores, particularly the validation F1 score of 0.75, demonstrate that the model maintains a good balance between precision and recall on new data.

However, the gap between training and validation performance suggests there's room for improvement in handling the noisy nature of social media data. In future iterations, strategies such as more extensive data cleaning, incorporating more diverse training data, or experimenting with advanced text preprocessing techniques could potentially bridge this gap and improve the model's performance on real-world, noisy text data.

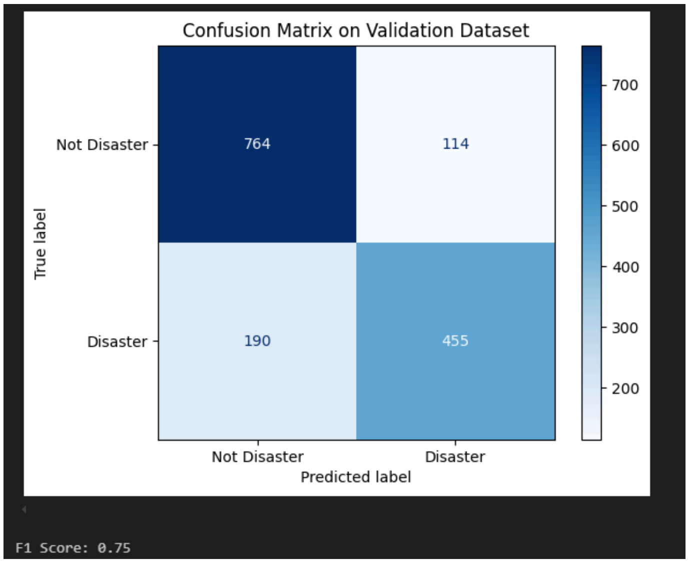


**Accuracy and Loss Graphs**



| | Not Disaster | Disaster |
|---|---|---|
| Not Disaster | 3360 | 127 |
| Disaster | 331 | 2272 |

```
F1 Score: 0.91
```



| | Not Disaster | Disaster |
|---|---|---|
| Not Disaster | 764 | 114 |
| Disaster | 190 | 455 |

```
F1 Score: 0.75
```

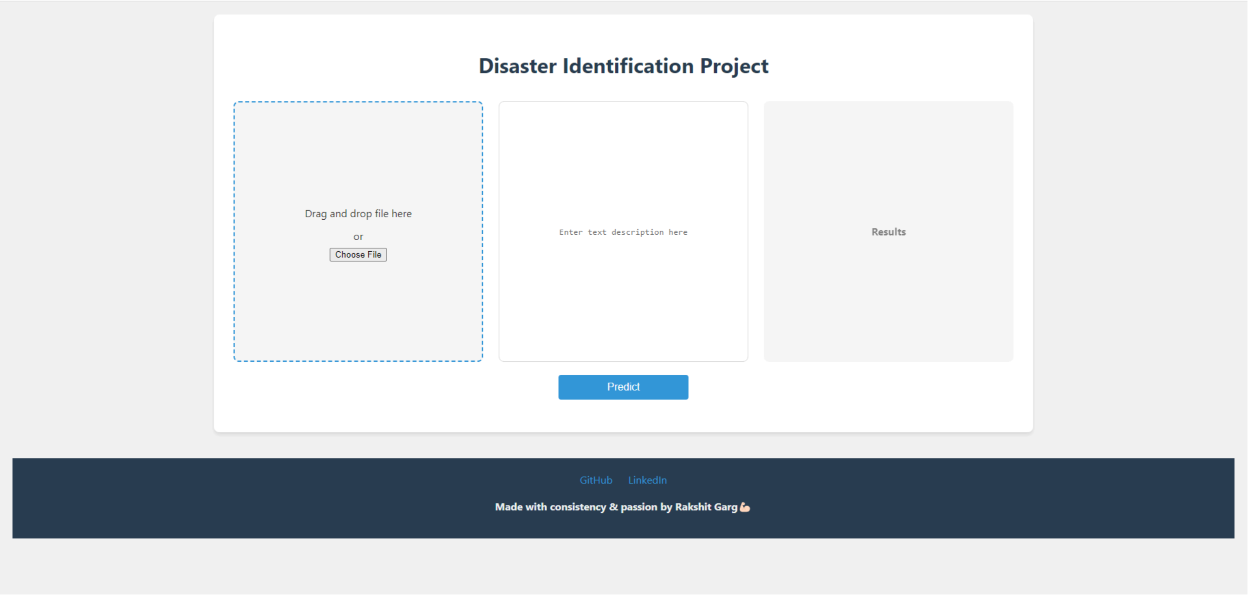
**Confusion Matrix for Disaster Text Model**

## Application Development

To make the system usable by disaster response teams, I developed a **Flask-based web application** that integrates both the image classification and text classification models. The web interface allows users to upload images or input social media text directly into the system. The backend, developed in Flask, handles the data preprocessing, forwards the input to the respective models, and displays the output in a user-friendly format.

The frontend was created using HTML and CSS, designed to be simple and accessible, making it easy for non-technical users to interact with the system. The main challenge during this phase was ensuring the smooth integration of both models and handling real-time user requests efficiently. By implementing an asynchronous system for handling model inference, the system was able to process multiple requests without delay.

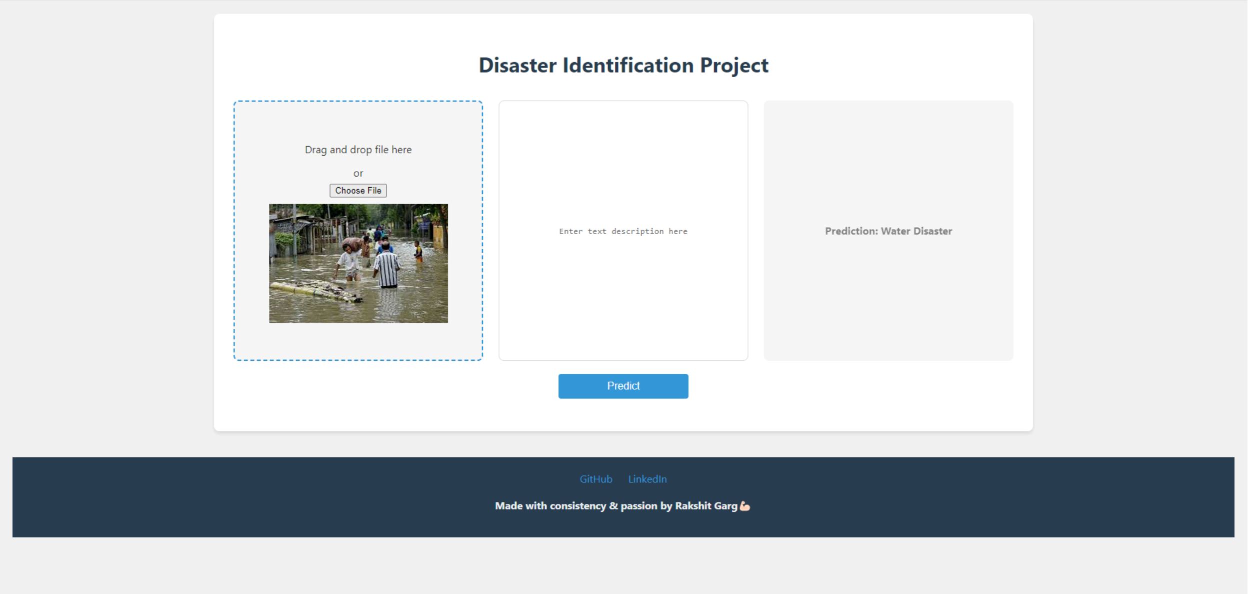
I've incorporated screenshots of both models giving their results:



Disaster Identification Project

Made with consistency & passion by Rakshit Garg 💪

**Home Page**

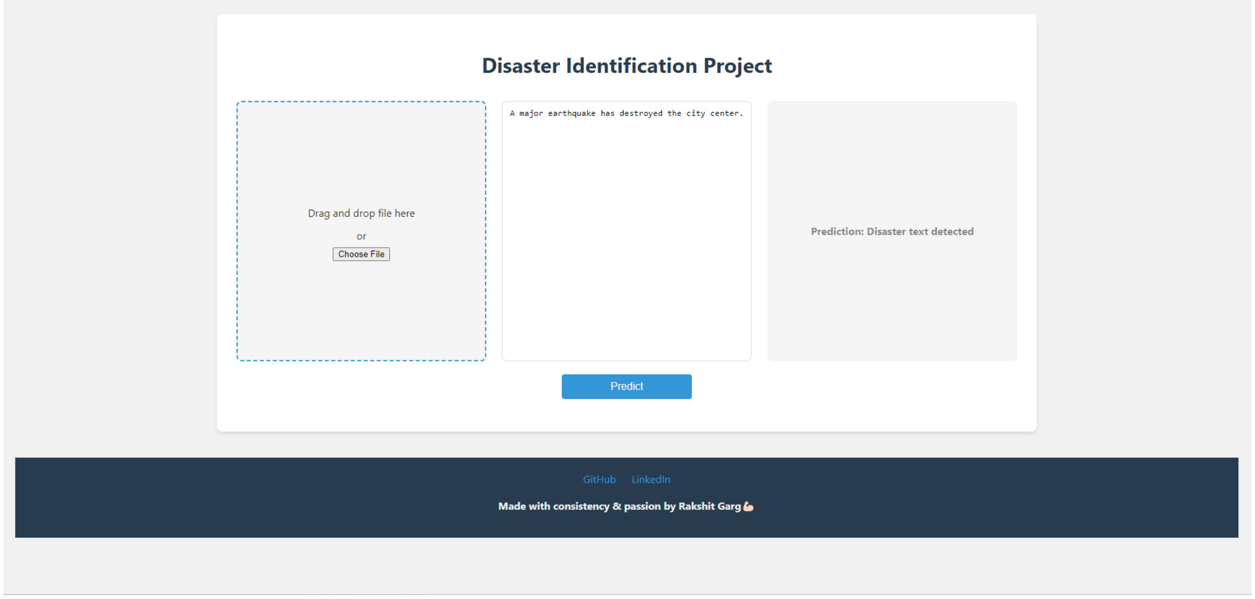


Disaster Identification Project

Prediction: Water Disaster

Made with consistency & passion by Rakshit Garg💪

**Results from Resnet Based Disaster Detection Image Model**



Disaster Identification Project

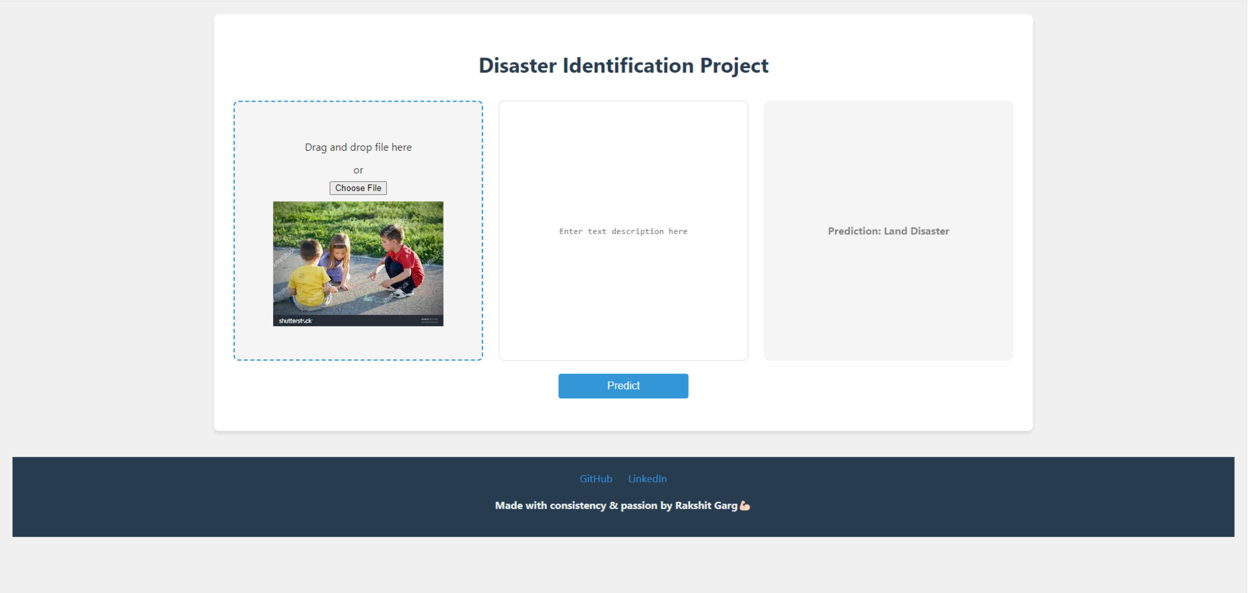
A major earthquake has destroyed the city center.

Prediction: Disaster text detected

Made with consistency & passion by Rakshit Garg💪

**Results from Conv1D NN Based Disaster Detection Text Model**

Additionally, I've included a fail case of the ResNet image model:



Disaster Identification Project

Prediction: Land Disaster

Made with consistency & passion by Rakshit Garg💪

**Resnet Model incorrectly identifying Children playing as Land Disaster**

This fail case demonstrates that while our model performs well in many scenarios, there is still room for improvement. The misclassification in this instance could be due to various factors such as limited training data for certain categories, complex or ambiguous image content, or potential biases in the training set. It's important to note that despite this limitation, the model still achieves good overall performance given the time constraints and dataset limitations we faced.

To address such cases in future iterations, we could consider:

1. Expanding the dataset with more diverse images, especially for categories that are currently underrepresented.
2. Implementing more advanced data augmentation techniques to increase the variety of training samples.
3. Exploring ensemble methods that combine multiple models or architectures to improve robustness.
4. Fine-tuning hyperparameters or experimenting with different pre-trained models as the base for transfer learning.

Despite these areas for potential improvement, the current implementation provides a solid foundation for disaster response image classification, offering valuable insights in many scenarios.

## Deployment

For deployment, I used **Gunicorn** in conjunction with **WSGI** to handle multiple concurrent requests and improve scalability. Gunicorn allows the web server to run multiple workers, which improves the system's performance in real-time, especially during peak loads. Additionally, I ensured that the application could be easily deployed in cloud environments, making it scalable and capable of serving a global disaster response team. Future improvements might include containerizing the application using Docker for smoother deployment and portability across different platforms.

## References and Future Improvements

In future iterations of this project, there are several improvements to consider:

1. **Image Classification Enhancements:** Balancing the dataset using augmentation techniques or oversampling to improve the F1 score and better handle minority classes.
2. **Text Classification Optimizations:** Incorporating additional text data from real-world social media platforms, and fine-tuning using DistilBERT further to handle noisy, unstructured text more effectively.
3. **Real-time Data Streaming:** Integrating APIs that pull real-time data from social media platforms like Twitter and Instagram to create a live feed of disaster-related posts.
4. **Deployment Improvements:** Containerizing the application using Docker for seamless cloud deployment and portability, and leveraging Kubernetes for dynamic scaling based on user demand.

References include publicly available datasets from Kaggle and CrisisNLP for training the models and documentation from Keras and Hugging Face for implementing the models.

1. https://www.kaggle.com/datasets/varpit94/disaster-images-dataset
2. https://www.kaggle.com/competitions/nlp-getting-started/data