Social Media Integration for Real-Time Disaster Management

Marri Bharadwaj
Computer Science & Engineering (CSE)
IIT Jodhpur
bharadwaj.4@iitj.ac.in

Dr. Chandana N
Center for Emerging Technologies for Sustainable Development (CETSD)
IIT Jodhpur
chandana@iitj.ac.in

***Abstract*—Early identification of disaster-related events on social media channels like Twitter considerably improves emergency response and risk reduction efforts. In this paper, I studied and compared the performance of two deep learning models, namely Long Short-Term Memory (LSTM) and Bidirectional Encoder Representations from Transformers (BERT), for carrying out binary classification of tweets as disaster-related or non-disaster-related. my LSTM model utilises FastText embeddings with gated memory cells to make full use of sequential information, and BERT relies on deep contextual embeddings enabled by attention. Despite the groundbreaking structure of BERT, I noted that my LSTM model performs better than BERT for short tweets, with an F1 value of 0.79 compared to BERT's value of 0.74. Most of this performance advantage arises due to the length of tweets being short, so that better training and optimisation could be achieved by the LSTM model. The study points out the possibility of using NLP-enabled deep learning models for real-time monitoring of disasters, with scalability of solutions amenable for deployment in emergency management systems.**

***Index Terms*—Disaster classification, LSTM, BERT, Natural Language Processing, Social Media Mining, Tweet Analysis, Emergency Response, Deep Learning, Real-Time Detection, FastText.**

I. Introduction

Disasters, both natural and human-made, threaten human life, infrastructure, and environmental stability. Climate change and urban vulnerabilities are increasing the frequency and intensity of disasters, leading to a growing need to deliver and assess rapid detection, communication, and response mechanisms. Conventional models for disaster management typically rely on slow messages and formal alerts that do not always communicate what is happening on the ground at the time.

Social media, especially Twitter, has emerged as a dynamic space where a person shares live updates, personal experiences, and requests for assistance. These types of user-generated insights can act as signals of an impending crisis or existing emergency situation. The challenge comes in filtering through the vast quantity of unstructured text data to locate relevant, credible information. [1]

This project seeks to assess and demonstrate that Natural Language Processing (NLP) techniques can be used to classify tweets as disaster related or not, thus exhibiting a way to contribute to real-time disaster portal monitoring efforts. I used deep learning models trained on annotated data to demonstrate how machine learning models can read human language in real-time and derive urgency and relevance from it.

The larger rationale behind this work is driven by public good. An effective tweet classification model may serve as a useful adjunct to aid emergency services, public health agencies, and humanitarian organisations in monitoring unfolding disasters, analysing their path, and coordinating urgent response to the disaster. It reflects the increased overlap of technology and society where machine learning is increasingly vital for humanitarian assistance and risk reduction. [2]

II. Data Collection

In order to build and test sound disaster detection model utilising NLP approaches, I assemble a data set comprised of differing social media posts specifically related to natural and man-made disasters. The data included social media posts most often collected from platforms including Twitter, Facebook, and Reddit with the express goal of including as much diversity in real-world context, informal language-styles, and reporting behaviour as possible.

The final data set comprises 7,613 posts, each of which is labeled as either disaster-related posts (target = 1) or non-disasters posts (target = 0). The posts covered a variety of disaster events, ranging from:

- *Natural disasters*: Earthquakes, hurricanes, floods, wildfires, tsunamis, tornadoes, blizzards, volcanic eruption, landslides, heat waves, droughts, and storms.
- *Man-made disasters*: terrorist attacks, industrial accidents, an oil spill, radiation emergency, explosion, transportation accidents.

The data set was collected in a CSV file and contained the following variables:

- *Text*: The post or tweet text.
- *Keyword*: A relevant keyword extracted from the post (when applicable).
- *Location*: The location in which post was created (often partial or missing).
- *Target*: A binary variable indicating if the post is about a real disaster (1) or not (0) - only available in the training data set.

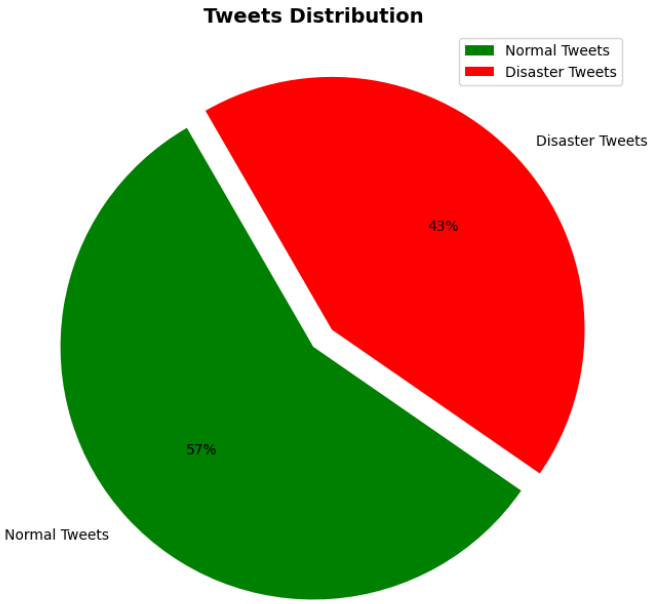

Fig. 1. Class Distribution

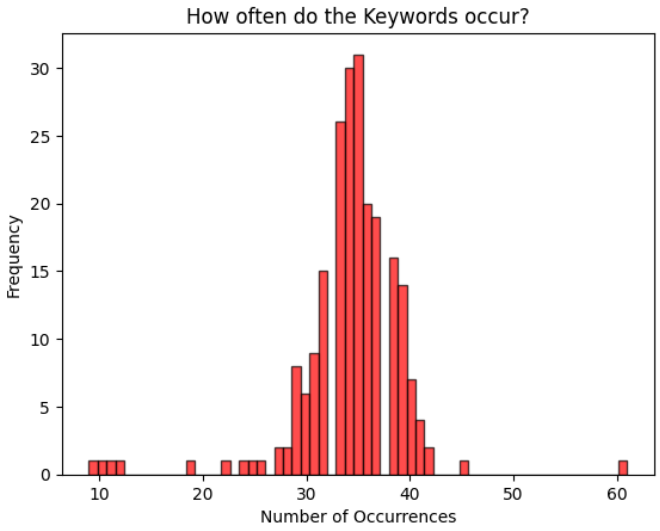

Fig. 2. Frequency Histogram

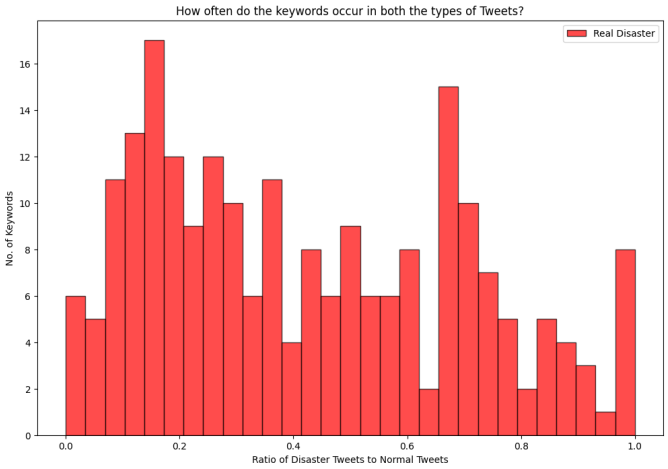

Fig. 3. Proportion of Posts across the Classes

A. Class Distribution

An initial analysis of the target labels revealed a fairly even split throughout the dataset with around 43% of the posts pointing to actual disasters and 57% pointing to non-disaster events. This balance is beneficial for training machine learning models as it minimises the likelihood of leaning towards any particular category.

Figure-1 visualises the class distribution.

B. Keyword Analysis

Of the 7,613 posts, only 61 did not have a keyword. The dataset comprises 222 unique keywords, many of which are multi-word keywords and correspond to certain disaster types.

A histogram of keyword frequencies (Figure-2) shows a long-tail distribution, with only a few keywords appearing frequently, while many keywords appeared a handful of times.

Also, Figure-3 exhibits the proportion of posts classified as a real disaster per keyword, with the high variance of proportions suggesting that keywords are good predictive features that correspond well with specific reports of real disasters.

C. Location Analysis

Due to a high level of inconsistency and noise in the location variable of the data set, including 2,533 missing cases and over 3,300 unique named locations, I conducted extensive data cleaning and assessment of this variable. Some examples of the issues to address are informal expressions found throughout the data, e.g., recorded locations of ”Earth” and ”Somewhere in the sky,” inconsistent naming of the location feature, or special characters. Data cleaning involved considerable efforts and included the following steps:

- Removing non-Latin characters.
- Lowercasing all text.
- Standardising country names, e.g., changed ”united states” and ”us” to ”usa”.
- Removing excess white space and formatting issues.

In the end, while the location variable had been cleaned, it was still very sparse and had high levels of ambiguity. In Figure-4 you can see that all but a few locations are unique or missing completely, and the distribution across classes is so uniform that it was irrelevant to move forward training a model. This aspect of the data will be omitted from any model training based on its low pertinence.

III. Methodology and Pipeline

A. Text Pre-processings and Word Embeddings

To refine the model’s capacity to comprehend and derive generalisations from textual data, I employed FastText embeddings to vectorise the words contained in it. FastText

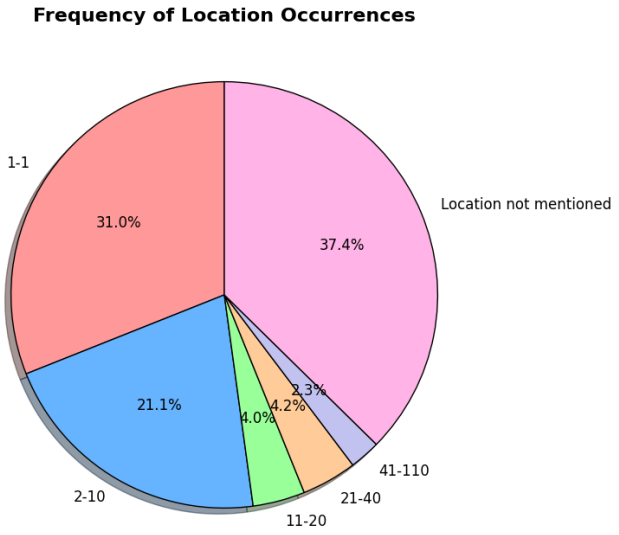

Fig. 4. Frequency of Location Occurences

embeddings differ from older embeddings like Word2Vec or GloVe, in that FastText utilises subword (character n-gram) information, which likewise permits it to generalise to out-of-vocabulary (OOV) words by building representations for new words. [3]

At first, the raw text data had comparatively low embedding coverage. FastText embeddings provided coverage of 51.5% of the vocabulary and 81.8% of the text. The limited embedding coverage was primarily due to tokens that were not able to be tokenised given the presence of irregular characters, symbols, and punctuation. To improve embedding coverage and generalisation in the embeddings, I executed a number of text pre-processing workflows:

- *Removal of URLs*: All hyperlinks were removed from the text using regular expressions, as these were predominantly noise, but also inconsequential for purposes of understanding content.
- *Expansion of Contractions*: Common contractions, like ”won't”, ”can't”, ”I'm”, and so forth, were enhanced to their full form (e.g., ”will not”, ”can not”, ”I am”) to improve the ability to match with the embedding vocabulary.
- *Lower-casing*: I then converted to lower case. This improved the coverage and reduced vocab size.
- *Removal of characters and symbols*: All characters that are not Latin (including punctuation and special characters) were removed using regular expressions.
- *Removing stopwords*: Common stopwords (e.g., ”the”, ”is”, ”and”) were removed to keep only the words that hold meaning.

After preprocessing the above tasks, the vocabulary coverage reached 78% and the text coverage was improved to 93.8% based on embedding coverage checks. This was an important preprocessing task since I wanted to ensure that the majority of the words in my dataset had valid vectors assigned to them, which directly supports my model in terms of remembering meaning.

Crawl Embeddings have 0.780 vocabulary cover example and of 0.938 text cover example.

I also preprocessed the keyword column by replacing encoded spaces (%20) with regular spaces and then replacing missing values with the placeholder ”empty”. This preprocess allowed us to treat the keyword field in my model as an additional feature.

B. Input Preparation

The clean text data, and preprocessed text data was then shaped to feed it into my model for training. I were able to split it into the datasets of train-test (96-4 split) using scikit-learn's ”train-test-split method, which allowed us to keep a small sample for final analysis and evaluation, while providing enough data to train effectively.

How to Prepare Inputs for LSTM:

- *Tokenisation*:
  - The text column was tokenised utilising Tokenizer() and converted into sequences of integers.
  - The keyword column was tokenised separately and represented in matrix format.
- *Padding*: The token sequences were given padding to ensure that there will be uniform sizes of input across all data samples.
- *Meta-data Features*: Additional float-type metadata features (such as word-count, etc.) were concatenated with the keyword matrix to create a large set of float features.
- *Embedding Matrix*: I initialised the embedding matrix with FastText's vectors, such that all words I found in the vocabulary were given a corresponding 300-dimensional vector. Any words not found in FastText's vectors were given the zero vector. I ended up with 3,386 unknown words, which was a significant drop from the preprocessed state.

C. LSTM-Based Model with Attention

To create a strong neural baseline for the tweet-level classification task, I used a Bidirectional Long Short-Term Memory (BiLSTM) network with an attention mechanism. This architecture enabled the model to capture my sentiment classification task's forward and backward contextual dependencies in the text, while also being able to pay attention to the most useful piece of the tweet. [4]

1) Data Pre-processing: The preprocessing pipeline was as follows:

- Converting tweets to lowercase, removing URLs, hashtags, mentions, emojis, special characters, and those extra whitespaces/unnecessary spaces, etc.
- Tokenising the cleaned tweets with a torchtext's ”Field” object with a maximum and fixed number of tokens of 40.
- Padding/truncating sequences to ensure uniform input lengths.

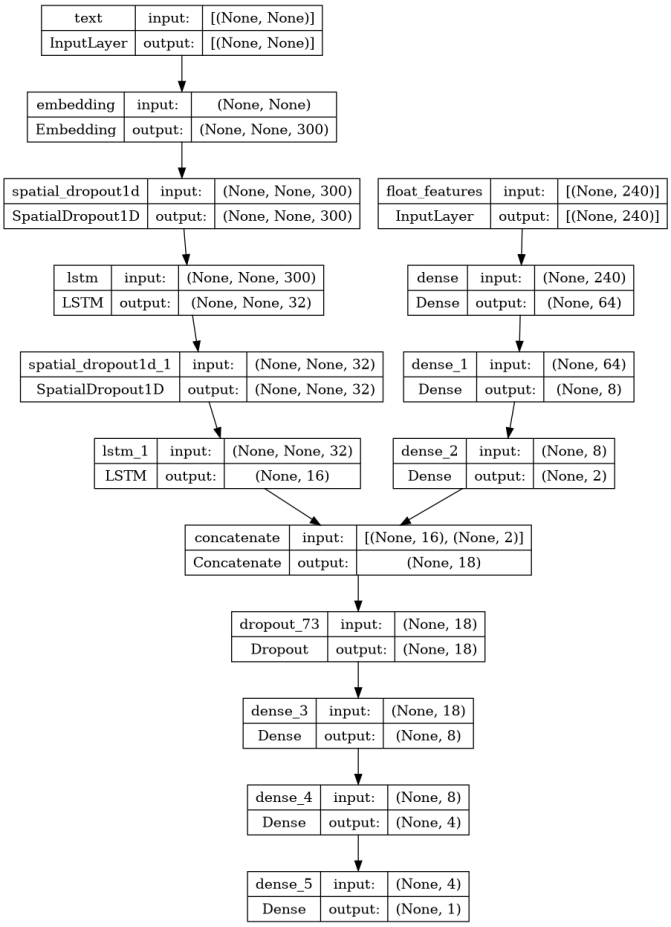

Fig. 5. LSTM Architecture

- Building a vocabulary from training data and initialising embedding using the pre-trained FastText word embeddings (300-dimensional).
- Tokens that were not found in the pre-trained word embeddings were given randomly initialised embeddings.

2) Model Architecture: The proposed architecture takes a modular approach:

- *Embedding Layer*: I initialised a FastText Embedding layer so the model can leverage embedding contextual richness with its subword structure.
- *Bidirectional LSTM*: A single layer BiLSTM was used, with a hidden layer output size of 128, which processes input in both forward and backward direction to access, and capture, the complete context.
- *Attention Mechanism*: A dot-product style attention computes weights across the hidden states of all time steps, forming a context vector as a weighted sum. This will allow the model to learn by assigning more attention to the relevant parts of the tweet, as determined by the attention weights.
- *Fully Connected Layer*: I took the attention-weighted context vector and applied a linear transformation followed by a sigmoid activation, producing the model output probability for a tweet being positive.

The ability of the BiLSTM and attention model to learn a dependency of relevance to an arbitrary sentiment not constrained to only the first words of the sequence is a powerful sentiment analysis approach.

I set the following training parameters:

- *Loss Function*: Binary Cross-Entropy (BCE)
- *Optimiser*: Adam with learning rate as 10^{-3}
- *Batch Size*: 32
- *Epochs*: 10
- *Validation Split*: 10% of training data
- *Device*: CUDA-enabled GPU for faster training

To alleviate overfitting, I monitored the validation accuracy, with early stopping based on patience over the best validation loss.

D. BERT Model Fine-Tuning

To enhance model efficacy further and to evaluate an alternative related to the LSTM-based architecture, I produced a fine-tuned BERT model. Bidirectional Encoder Representations from Transformers (BERT) has established state-of-the-art performance on a number of natural language processing (NLP) tasks including cases requiring contextual awareness. [5]

This implementation utilised the pre-trained bert-base-uncased model from HuggingFace's Transformers library. This implementation with its transformer-based embedding and pre-training on a large, unsupervised English corpus is capable of producing rich, contextual embeddings for each token in the input.

1) Data Tokenisation: I tokenised the raw text into input for the BERT model using bert's tokeniser:

- Each sentence was tokenised to subword tokens represented as a separate token.
- A [CLS] token was added to the beginning of each sentence and a [SEP] token added to the end of each sequence.
- The tokeniser also identified and produced attention masks and token types, which tell the model which tokens the model should give focus to and indicates the model how to read sentence pairs (not needed in this case but was trained to accommodate).

Each input was truncated or padded to a maximum input length of 128 tokens, that were then converted to and wrapped on PyTorch tensor datasets in order to batch and load the inputs into the model efficiently.

2) Model Architecture: I fine-tuned a BertForSequence-Classification model pre-trained for binary classification tasks:

- A classification head with a single linear layer and sigmoid activation for binary classification was added on top of the pooled [CLS] output produced by the BERT model.
- The model was trained using binary cross-entropy loss and optimised using the AdamW optimiser. The AdamW

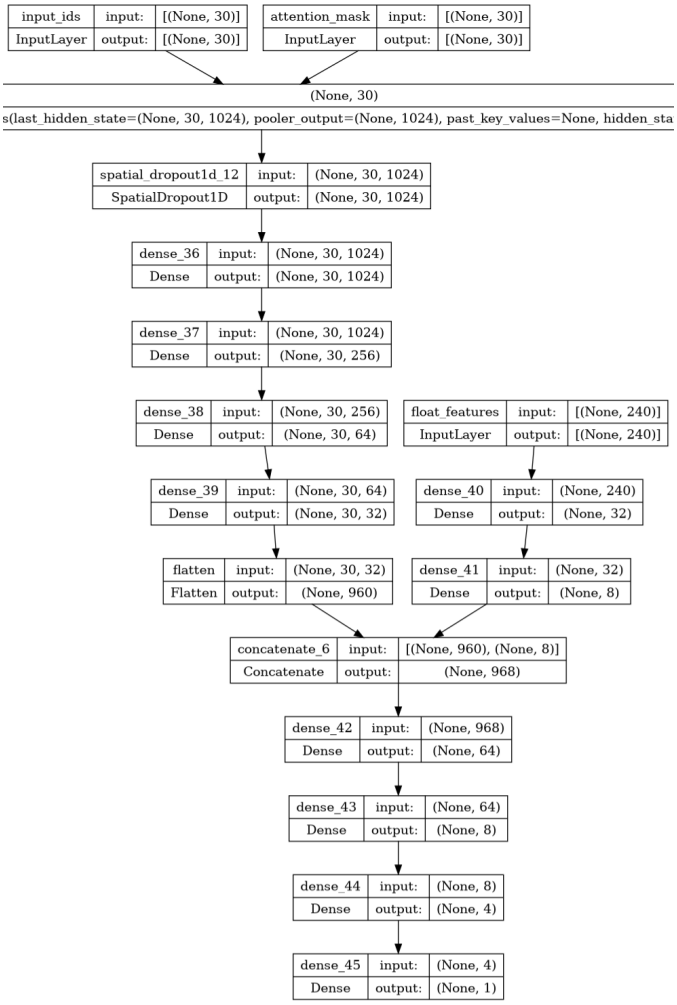

Fig. 6. BERT Architecture

optimiser is a commonly used training regime for transformer architectures.

- I used a learning rate scheduler during training with get-linear-schedule-with-warmup to improve convergence.

3) Training Strategy:

- I trained with a batch size of 16 for 4 epochs.
- To track overfitting and generalisation, I monitored the validation performance at the end of each epoch.
- The model was trained in the GPU environment using CUDA to speed convergence.

Predictions were made during inference by using torch.sigmoid(logits) to generate probability scores, which were subsequently thresholded to produce binary labels.

IV. Results and Comparison

The findings show that the LSTM model does a good job of capturing sequential dependencies of the text and generalises reasonably well to unseen data, providing a good balance of precision and recall.

I conducted hyperparameter tuning (learning rate and warmup proportion), and visualised the results to determine how hyperparameters affect performance metrics. I also put together training and validation curves which demonstrate that the model converged steadily over epochs in terms of accuracy, precision, recall, and F1 score, showing no signs of overfitting.

TABLE I
Comparison of LSTM and BERT Models

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
LSTM	80.33	78.87	78.87	78.87
BERT	75.08	72.00	76.06	73.98

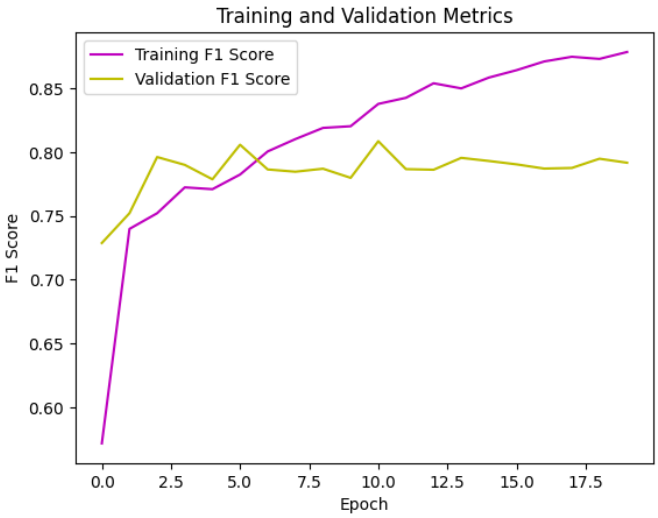

Fig. 7. Fluctuations in LSTM Training

Although BERT, which is based on the transformer architecture, is widely considered state-of-the-art for many NLP tasks, its performance in this case was a tick lower than the LSTM model. This was as the result of the nature of the dataset that favors a sequence-based modeling approach better suited for the LSTM type of modeling or the fine-tuning, owing to relatively smaller sizes of the tweets I was able to collect.

A. Comparitive Insights

- In terms of the majority of metric outputs, the LSTM model performed (in)significantly better than BERT. This is especially true with (highest levels) for precision and F1 score, which are the most meaningful metrics for disaster-related tasks using binary classification.
- Although, BERT performed better with recall, which is advantageous to applications delivering greater specificity to minimize false negatives; this is important to note as it is crucial during practical implementations to reduce false negatives.
- BERT average losses remain greater than LSTM suggesting BERT's predictions were of less projection. This could be attributed to the small size of the tweets I collected.

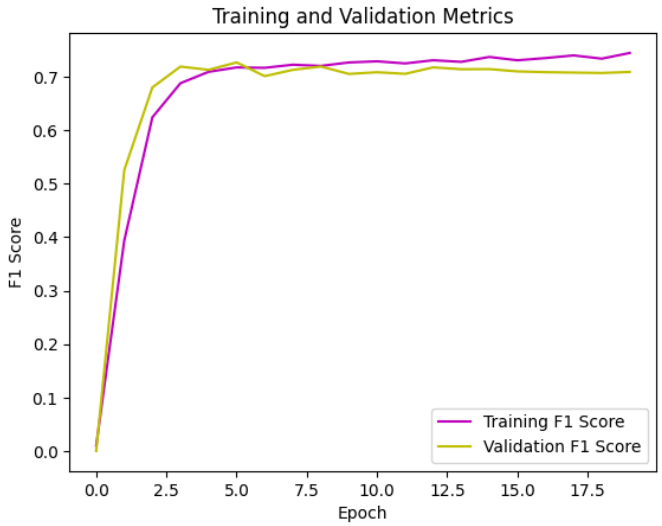

Fig. 8. Consistent BERT Training

V. Applications and Impact

The suggested system is a significant potential approach to real-time disaster detection, for extreme social media, especially Twitter, is both decentralised and rapid in its mode of communication—people will often tweet about experiencing a disaster minutes after it has occurred—sometimes before news agencies, or even governmental agencies, have had time to activate. Thus, if a user, for example, tweets about an earthquake, a BERT-based model can classify that tweet as being disaster-related (or not) with a near-zero response time so that it can be used as an early-warning system in situational awareness repositories in emergency management systems.

Furthermore, this approach could be incorporated into emergency response systems using real-time APIs connected to Twitter's streaming output. The model could monitor, continually and in real-time, classify, and flag tweeting about potential disasters to authorities in the field. If geolocated information is available, this can bolster the effectiveness of the model by identifying the general area impacted.

The consequences of such a system are broad-based, such as:

- *Public Health*: Early detection of a disaster may lead to an expedited response time in raising medical teams or medical supplies, if appropriate, particularly in health-related disasters or crises such as epidemics, or natural disasters where injuries are a potential outcome of the disaster.
- *Crisis Response Teams*: By classifying disaster tweets (as an example), this may prioritise resource allocation, by identifying and tracking a disaster as it unfolds.
- *General Public*: Social media or dashboards with alerts can serve timely notifications for the general public or individuals to provide them the opportunity of acting on recommendations for self-protective behaviour such as evacuating or sheltering.
- *Data Collection for Policy Officials*: Trends from tweets associated with disasters can be a component of methodical strategy for action and future resilience.

VI. Conclusion

In this project, I constructed and analysed a tweet classification model using LSTM and BERT, a platform that can assist researchers in differentiating disaster-associated and non-disaster-associated messages. The research was conducted by iteratively developing, fine-tuning LSTM and BERT, and measuring performance by standard evaluation metrics of accuracy, precision, recall, and F1-score. Overall, both showed good performance on measuring semantic patterns from social media posts.

Interestingly, in the series of experiments I conducted, I found that a LSTM-based model slightly outperformed BERT on the collected data set. It could be explained by the relatively short length and simple structure of the tweets found in my twitter data set. As a lighter and easier to train machine learning model, LSTMs were able to successfully model a semantic pattern for sentiment without the necessity for the deeper context modelling provided by BERT. Accordingly, in actual applications where tweet posts are more diverse, embedded slang and require deeper context, BERT is expected to outperform machine learning deep models due to its strong capacity for language representation.

Despite the promise of the results found with BERT, limitations exist. Specifically, BERT is still a computationally expensive model to train and requires extensive GPU and time. The computational cost poses a substantial barrier to real time or more broadly deployed applications without the appropriate infrastructure.

In future endeavours, I hope to achieve the following:

- Incorporate more diverse and robust datasets with a broader range of disaster types and linguistic styles.
- Extend the operation of my system to handle multilingual tweets, thus improving applicability worldwide.
- Include geolocation metadata to improve real-time disaster mapping and localised alerts.
- Investigate model distillation or lightweight transformers for better inference speed and real-time deployment.

References

[1] M. Imran, C. Castillo, F. Diaz, and S. Vieweg, "Processing social media messages in mass emergency: A survey," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 1–38, 2015. https://doi.org/10.1145/2771588

[2] S. Vieweg et al., "Microblogging during two natural hazards events: What Twitter may contribute to situational awareness," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2010, pp. 1079–1088. https://doi.org/10.1145/1753326.1753486

[3] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proc. EMNLP*, 2014, pp. 1532–1543. https://nlp.stanford.edu/pubs/glove.pdf

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. https://doi.org/10.1162/neco.1997.9.8.1735

[5] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint*, 2018. https://arxiv.org/abs/1810.04805