The LNM Institute of Information Technology

TEXT EMOTION ANALYSIS

BTP Report

Submitted by:

Mohit Jain (21UCS131)

Anurag Singh(21UCC020)

Himanshu Gautam(21UCS240)

Supervisor:

Dr. Durga Prasad Mishra

Co-supervisor:

Dr. Navneet garg

TABLE OF CONTENTS:

ACKNOWLEDGEMENT:

We extend our heartfelt gratitude to Dr. Durga Prasad Mishra and Dr. Navneet Garg for their invaluable guidance, unwavering support, and insightful feedback throughout the duration of our BTech project on "Text Emotion Analysis."

Their combined expertise, encouragement, and dedication have been instrumental in shaping our research into a comprehensive study. Their constructive criticism and encouragement propelled us to strive for excellence and explore new horizons in this fascinating field.

We are deeply thankful to them for generously sharing their knowledge, wisdom, and resources, which significantly enriched this project and enhanced our understanding of the subject matter.

HIMANSHU GAUTAM(21ucs240)

MOHIT JAIN(21ucs131)

ANURAG SINGH(21ucc020)

ABSTRACT:

Emotional recognition has emerged as a crucial area of research, offering valuable insights across various domains. Emotions manifest in diverse forms, including speech, facial expressions, written text, and gestures. Detecting emotions in textual documents presents a content-based classification challenge, drawing upon principles from Natural Language Processing (NLP) and deep learning. This study proposes the use of deep learning-assisted semantic text analysis (DLSTA) for human emotion detection leveraging big data. Emotion detection from textual sources leverages NLP concepts, with word embeddings playing a pivotal role. Word embeddings are widely employed in NLP tasks such as machine translation, sentiment analysis, and question answering, enhancing learning-based methods by incorporating semantic and syntactic features of the text. To verify the performance of the proposed model, it is compared with previous studies. The optimal value is brought into the model contrast experiment. In the experiment, the accuracy rate, recall rate and F value of this method are all greater than 91%, which is the highest among the models. It effectively improves the accuracy of text sentiment analysis, and has high research and practical value.

MOTIVATION:

Understanding the emotions conveyed through text is more than just a technological feat—it's a gateway to unlocking deeper human insights and enhancing user experiences in countless applications. Whether it's understanding customer sentiment, gauging public opinion, or even monitoring mental health trends, the ability to accurately analyze text emotion has far-reaching implications. By delving into this research, we're not just advancing the frontier of artificial intelligence; we're also empowering businesses, organizations, and individuals to make more informed decisions, foster stronger connections, and ultimately, enrich lives.

PROBLEM STATEMENT:

The exponential growth of digital content in the form of social media posts, product reviews, news articles, and online conversations has led to an increasing need for accurate and efficient methods of understanding and analyzing human emotions expressed in text. However, the inherent complexities of language, including sarcasm, ambiguity, and cultural context, pose significant challenges to traditional approaches of emotion analysis. Consequently, there is a pressing demand for advanced techniques that can effectively capture the nuanced and multifaceted nature of emotions within textual data.

Create a deep learning-based text emotion analysis model to accurately detect and classify emotions within textual data. The project encompasses data collection, model selection, optimization, and deployment for real-world applications. Emphasizing precision and versatility, it aims to discern subtle emotional nuances across various contexts with high reliability.

I. INTRODUCTION:

Emotions are an integral part of the personality of every individual and an integral part of human life. We know many different definitions of emotions. They are most often defined as “a complex pattern of reactions, including experiential, behavioral and physiological elements.” Many times, they are confused with feelings or moods. They are the way in which individuals cope with matters or situations that they find personally significant. Emotion can also be characterized as a conscious mental reaction (such as anger or fear) subjectively experienced as a strong feeling usually focused on a specific object, which is often accompanied by physiological and behavioral changes in the human organism (American Psychological Association – APA, 2023).

Text Classification (TC) refers to the systematic categorization of textual data into distinct groups based on their inherent properties and characteristics. Through automated analysis, TC effectively examined text and reliably assigned pre-established categories. This procedure is of utmost importance for facilitating the processing and extraction of valuable information from raw and unstructured textual data [1,2]. TC encompasses three distinct system types: rule-based, machine-learning based, and hybrid approaches [3, 4]. Rule-based systems employ a collection of predefined rules to effectively categorize text into organized groups. On the other hand, machine learning-based systems rely on prior observations and patterns to classify text. Hybrid systems, as the name implies, combine elements of rule-based and machine learning-based systems, utilizing both trained classifiers to improve classification accuracy and efficacy.

Many researchers have dedicated their efforts to enhancing the efficacy of Natural Language Processing (NLP) tasks through the development of Deep Learning (DL) frameworks [5, 6]. Notably, CNN, Long Short-Term Memory (LSTM), and Bidirectional LSTM (Bi-LSTM) techniques attained an accuracy rate of 77.4%, with CNN exhibiting the most favorable average performance [7]. These advancements in DL frameworks have contributed significantly to the improved performance and accuracy of NLP tasks in various applications. Researchers continue to develop hybrid DL models to achieve better results.

Combining CNN and BiLSTM in one architecture enables more comprehensive and in-depth data processing. CNN helps extract spatial features from the data, while BiLSTM helps extract temporal or contextual features [8]. Researchers have implemented hybrid CNN-BiLSTM models [9, 10], and using Word2Vec, GloVe, and fasttext. They used an embedding size of 300, Softmax activation, dropout, and recurrent dropout rates of 0.5 and 0.4, an SGD optimizer, 50 epochs, a filter size of 512, a kernel size of 3, embedding, CNN max pooling, BiLSTM, and dense layers, resulting in an accuracy of 82% [11].

Through rigorous experimentation and comparative evaluation on benchmark datasets, we endeavor to provide valuable insights into the strengths, limitations, and optimal configurations of CNN and BiLSTM-based models for text emotion analysis. By shedding light on the most effective techniques and methodologies, this research aims to contribute towards the advancement of emotion-aware applications and systems, fostering more nuanced and empathetic interactions in the digital realm.

Despite these advancements, challenges remain, including handling long inputs, unnecessary convolution operations for NLP, and data loss with increasing data size. The subsequent section of this article elaborates on the proposed methods, including data preprocessing, feature extraction, and model architecture. Experimental outcomes and dataset specifics are discussed in detail, followed by conclusions and insights into future research directions.

II. OBJECTIVE:

The primary objective of this report is to explore and evaluate state-of-the-art methods and techniques for text emotion analysis, with a focus on achieving accurate and robust emotion detection and classification in diverse textual contexts. Specifically, the objectives include:

Investigating the effectiveness of deep learning architectures, such as Convolutional Neural Networks (CNN) and Bidirectional Long Short-Term Memory networks (BiLSTM), in capturing semantic information and temporal dependencies within textual data for emotion analysis.

Conducting a comparative analysis of various embedding techniques, including Word2Vec, GloVe, and BERT, to determine their impact on the performance of text emotion analysis models.

Exploring different preprocessing techniques and methodologies, such as data augmentation, feature engineering, and ensemble learning, to enhance the performance and generalization capabilities of emotion analysis models.

Evaluating the performance of the developed models on benchmark datasets and real-world text data, considering metrics such as accuracy, precision, recall, and F1-score.

Providing insights and recommendations for the selection of optimal techniques and methodologies for text emotion analysis, considering the trade-offs between accuracy, computational complexity, and scalability.

By addressing these objectives, this research aims to contribute towards advancing the state-of-the-art in text emotion analysis, enabling the development of more effective and empathetic applications and systems in various domains, including social media analytics, customer feedback analysis, and personalized recommendation systems.

III. LITERATURE REVIEW:

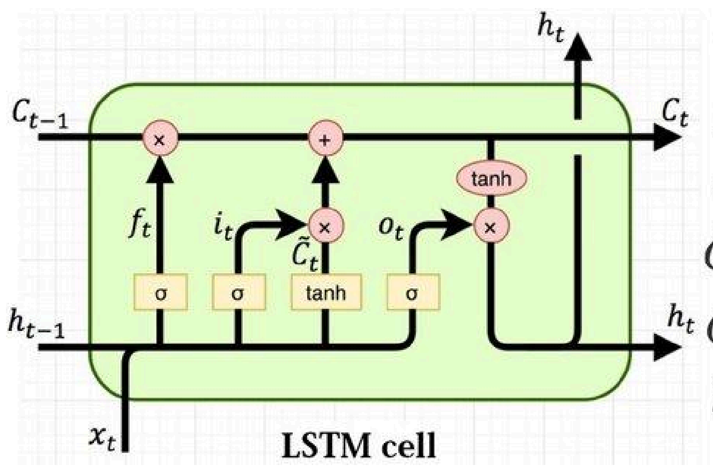
Previous studies related to text classification and sentiment analysis have been undertaken, and they adhere to the process of classifying text sentiment. These studies include:

- The study aimed to enhance text classification accuracy using a Bi-LSTM model that combines Word2Vec, CNN, and attention mechanisms [11]. The approach combined multiple effective text classification techniques: Word2Vec for richer word vectors, CNN for local feature extraction, and attention mechanisms for weight assignment to significant sections of the text. The model used parameters such as a skip gram embedding size of 300, a 0.2 dropout rate, batch size of 128, and the Adam optimizer. It achieved an average accuracy of 87.4%, with the highest F1-Score (90.1%) observed on a 13k-instance dataset. This study demonstrated the synergistic integration of these models enhances text classification accuracy. Note that having access to more extensive training data and further refinement time benefits the model.
- In a recent study, Salur and Aydin [9] introduced an innovative hybrid deep-learning model for sentiment classification. This model addresses dataset complexity and imbalances by synergistically combining CNN for local textual feature extraction and LSTM-based RNN for long term contextual information. To enhance feature representations, the model includes character embedding and pre-trained embedding (Word2Vec, GloVe, FastText). Using tweets related to a Turkish GSM operator, the results showed that a single BiLSTM model achieved 80.44% accuracy with FastText embedding. However, the hybrid CNN-BiLSTM model, which incorporates character embedding and FastText, outperformed it with 82.14% accuracy. These results emphasize the superiority of the proposed hybrid model for sentiment classification, although it is specifically tailored to Turkish, Arabic, and Lithuanian languages.

- In a study by Naqvi et al. [10], sentiment analysis in Urdu text was addressed using deep learning. Their approach utilizes deep learning to enhance Urdu sentiment analysis accuracy, combining CNN for local feature extraction and RNN with LSTM for long-term context understanding. Four embeddings, Samar, CoNLL, pretrained, and self trained Fast Text, were evaluated. The BiLSTM ATT model achieved the highest accuracy at 77.9%, while LSTM reached the highest precision (85.16%) with Samar embedding. These results highlight the effectiveness of their deep learning methods in Urdu sentiment analysis. Further advancement potential exists through exploring alternative methods. This section reinforces the background by providing evidence and discussing trends in the field. It should encompass all relevant studies and supporting evidence.

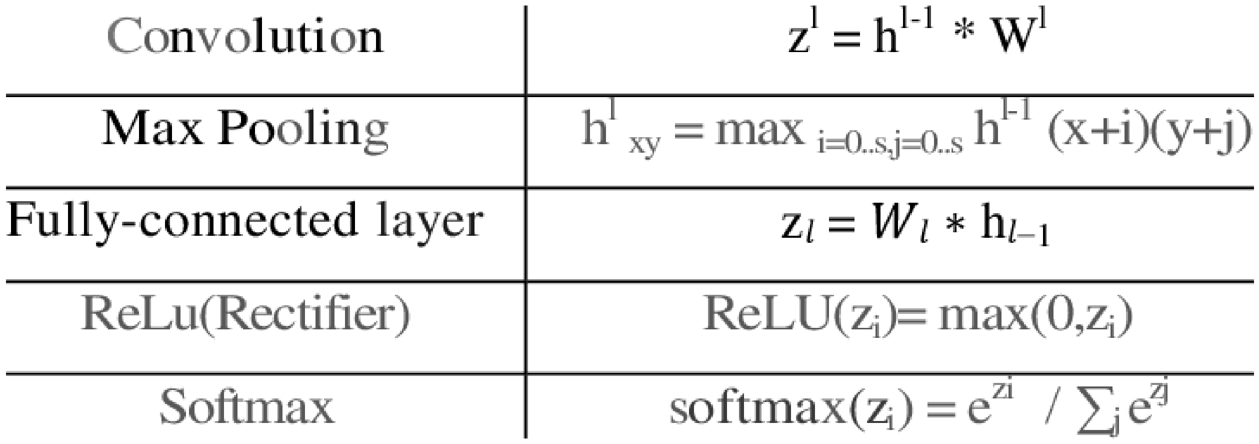
Formulas :

LSTM:



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

CNN:

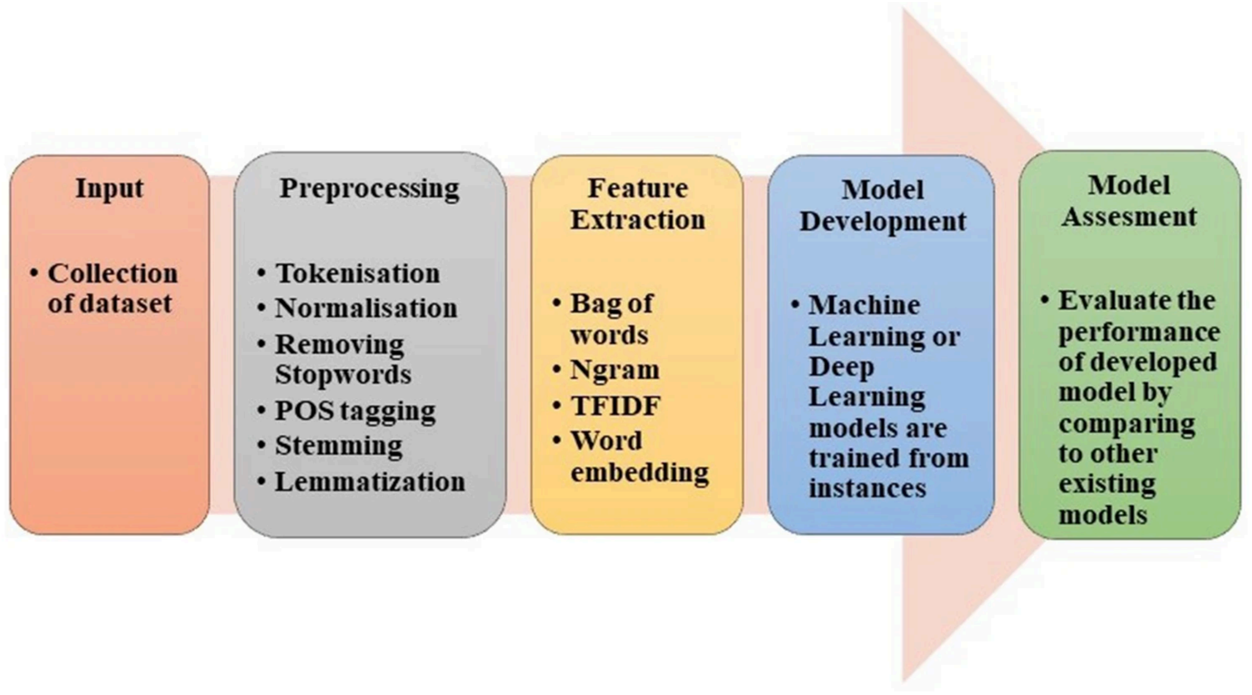


Convolution	$z^l = h^{l-1} * W^l$
Max Pooling	$h^l_{xy} = \max_{i=0..s,j=0..s} h^{l-1}(x+i)(y+j)$
Fully-connected layer	$z_l = W_l * h_{l-1}$
ReLu(Rectifier)	$\mathrm{ReLU}(z_i) = \max(0, z_i)$
Softmax	$\mathrm{softmax}(z_i) = e^{z_i} / \sum_j e^{z_j}$

A hybrid CNN-BiLSTM model was developed using different feature extraction techniques and kernels for the WAG dataset. This model is an innovative approach because it combines the advantages of both the techniques. The CNN is effective in its ability to extract local features from data, whereas the BiLSTM model excels in handling long-term dependencies and leveraging both past and future information to improve prediction accuracy. To accommodate varying data lengths, the inclusion of padding techniques adds value to the model’s capability. The incorporation of the Rectified Linear Unit (ReLU) and Softmax activation functions further enhanced the flexibility of data processing. With an embedding size of 300, the model could extract more intricate and complex features from the input data.

IV. MATERIALS AND METHODS

This study aims to enhance the performance of the hybrid CNN BiLSTM model, specifically for sentiment analysis in the TC domain. We evaluated this hybrid model thoroughly by considering the advantages and potentials of both the CNN and BiLSTM architectures. The methodology we adopted for this study focuses on crucial aspects, such as selecting appropriate data, performing meticulous labeling, developing robust feature vectors, and formulating the hybrid CNN BiLSTM approach. All these aspects contribute to a more precise and accurate sentiment analysis solution.



A. DATASET:

Twitter Dataset-1:
Dataset was combined from daily dialog, isear, and emotion-stimulus to create a balanced dataset with 5 labels: joy, sad, anger, fear, and neutral. The texts mainly consist of short messages and dialog utterances.

Test-https://github.com/Anu-vibes/text-emotion-analysis/blob/main/data_test.csv
Train-https://github.com/Anu-vibes/text-emotion-analysis/blob/main/data_train.csv
links: http://yanran.li/dailydialog.html ,
https://www.site.uottawa.ca/~diana/resources/emotion_stimulus_data/
https://paperswithcode.com/dataset/isear

Dataset-2:

https://huggingface.co/datasets/dair-ai/emotion/viewer
An example looks as follows.

```
{
  "text": "im feeling quite sad and sorry for myself but ill snap out of it soon",
  "label": 0
}
```

The data fields are:

- text: a string feature.
- label: a classification label, with possible values including sadness (0), joy (1), love (2), anger (3), fear (4), surprise (5).

The dataset has 2 configurations:

- split: with a total of 20_000 examples split into train, validation and split
- unsplit: with a total of 416_809 examples in a single train split

B. LABELING:

- **Categorizing Labels:**The original emotion labels ('joy', 'fear', 'anger', 'sadness', 'neutral') are mapped to integer labels using a predefined encoding dictionary (encoding). Each emotion label is associated with a unique integer value. This categorization simplifies the representation of emotion labels, making them suitable for feeding into machine learning models.
- **Integer Labels:** The emotion labels in the training and testing datasets (data_train.Emotion and data_test.Emotion) are replaced with their corresponding integer labels using list comprehension. This transformation converts the emotion labels from their original string format to integer format, facilitating numerical computation and model training.
- **One-Hot Encoding:** After converting the emotion labels to integer format, they are further encoded using one-hot encoding. One-hot encoding is a binary representation where each integer label is represented as a binary vector with a single '1' value at the index corresponding to the label's integer value and '0' values elsewhere. This encoding scheme ensures that the model interprets the emotion labels as categorical variables rather than ordinal variables, preventing unintended ordinal relationships between the labels.
- By categorizing and encoding the emotion labels, we prepare them for consumption by the neural network model during training. This encoding scheme enables the model to effectively learn the mapping between input text data and corresponding emotion categories, facilitating accurate emotion classification.

C. PREPROCESSING:

Text preprocessing is a vital component of text classification, encompassing a range of techniques to prepare and transform text data for analysis. This process holds significant importance across diverse domains and languages . Human-transferred text data needs to be in a machine-readable format for further analysis . Preprocessing is done to eliminate issues that may interfere with the results in the subsequent data processing steps. The data used in the classification process is not always in an ideal condition. Preprocessing is necessary for labeled data before proceeding to the next stages.
The present study encompasses several essential pre -processing steps to enhance the quality and suitability of the data.

1. **Data Cleaning**:

- Removing Hashtags and Usernames: This step eliminates hashtags (e.g., #example) and usernames (e.g., @user) from the text data, as they typically do not contribute to the emotion analysis task and might add noise to the data.
- Removing Punctuation: Punctuation marks like commas, periods, and exclamation points are removed from the text, as they do not typically convey emotion-related information and might interfere with subsequent processing steps.
- Removing Numbers: Numerical digits are removed from the text data since they are often irrelevant to emotion analysis tasks and might not contribute meaningfully to the analysis.
- Lowercasing: All words are converted to lowercase to ensure consistency in the text data. This step prevents the model from treating the same word with different cases (e.g., "Hello" vs. "hello") as different entities.
- Removing Stopwords: Commonly occurring words (stopwords) like "the," "is," and "and" are removed from the text data as they often do not carry significant meaning in emotion analysis tasks.

- Stemming: Words are reduced to their root form (stem) to normalize variations of the same word. For example, "running" and "ran" are both stemmed to "run," reducing the dimensionality of the feature space and improving generalization.
- Lemmatization: Similar to stemming, lemmatization reduces words to their base or dictionary form (lemma). This process ensures that different inflections of the same word are treated as a single entity, further improving the consistency and interpretability of the text data.

2. **Tokenization**:

- Tokenization is the process of converting text into a sequence of tokens or words.
- Tokenizer Initialization: We initialize a Tokenizer object without specifying any parameters. This object will be used to tokenize the text data.
- Fitting on Texts: We call the fit_on_texts method of the Tokenizer object and pass the entire corpus of texts (texts) as input. This method analyzes the texts and creates a vocabulary index based on the frequency of words in the corpus.
- Texts to Sequences: We use the texts_to_sequences method of the Tokenizer object to convert the text data (texts_train and texts_test) into sequences of integer indices. Each word in the texts is replaced by its corresponding index in the vocabulary created during the fitting step.
- Word Index: We retrieve the word index from the Tokenizer object using the word_index attribute. This dictionary maps each word to its corresponding integer index in the vocabulary.
- Vocabulary Size: The size of the vocabulary is calculated by adding 1 to the length of the word index. This addition accounts for the reserved index 0, which is typically used for padding sequences. The vocabulary size represents the total number of unique words in the corpus.

3. Padding:

- Padding is a crucial preprocessing step in natural language processing, especially when working with neural networks. It ensures that all input sequences have the same length, which is necessary for feeding the data into the model for training or inference.
- Padding Sequences: We use the pad_sequences function to pad the sequences of integer indices (sequence_train and sequence_test) to a fixed length (max_seq_len).
- Fixed Input Length: By specifying maxlen = max_seq_len, we enforce that all sequences are padded or truncated to have the same length. This fixed input length is essential for feeding the data into the neural network model, as most deep learning frameworks expect input tensors of uniform shape.
- Padding with Zeros: The default behavior of pad_sequences is to pad sequences with zeros at the beginning (pre-padding) to achieve the desired length. This zero-padding ensures that shorter sequences are extended to match the length of the longest sequence in the dataset.
- Handling Longest Input: In this case, the maximum sequence length (max_seq_len) is set to accommodate the longest input sequence in the dataset, which is approximately (500,200) words. This choice ensures that no information is lost during padding, as all input sequences are preserved at their original lengths or padded to match the maximum length.
- Overall, padding ensures uniformity in input data size, facilitating efficient batch processing and computation within the neural network model. It enables the model to learn meaningful patterns from the text data while handling variable-length inputs effectively.

D. FEATURE EXTRACTION:

In the field of NLP, a significant hurdle is developing a model capable of understanding the hierarchical representation of sentences in text data. This challenge arises primarily in the context of classification tasks and the extraction of relevant features [12]. Feature extraction involves capturing the characteristic features or attributes of a particular shape and then carrying out an analysis of the captured feature values. The process of feature extraction involves reducing the dimensionality of the output data to make it more manageable and enable efficient processing. By carefully selecting and/or combining variables into characteristics, the amount of data to be processed is effectively reduced.
In this study, we employed two popular techniques for word embedding: GloVe (Global Vectors for Word Representation) and Word2Vec.

GloVe embeddings are pre-trained word vectors that capture global word co-occurrence statistics in large text corpora. These embeddings encode semantic relationships between words based on their distributional patterns in the corpus. We utilized pre-trained GloVe embeddings, which have been trained on massive text corpora such as Wikipedia and Common Crawl, to obtain dense vector representations for words in our dataset. By leveraging GloVe embeddings, we aimed to capture the semantic similarities and contextual information present in the textual data, thereby enhancing the performance of our text emotion analysis models.

Word2Vec is another popular technique for word embedding, known for its ability to capture semantic relationships between words based on their contextual usage in the text. We employed the Word2Vec algorithm to train word embeddings specific to our dataset, learning vector representations that capture the semantic meaning of words based on their local context within the text. By training Word2Vec embeddings on our dataset, we aimed to capture domain-specific semantics and contextual nuances relevant to text emotion analysis.

E. MODEL

The dataset, which has been extracted and labeled with sentiment, was tested using several algorithms such as Neural Network, LSTM, BiLSTM, and CNN.Before the testing phase began, a series of pre-processing steps were carried out to ensure the cleanliness and suitability of the data. These steps included eliminating numbers ,hashtags, URLs, emojis and punctuation. Once the data was effectively cleaned, feature extraction began,which included tokenization and encoding techniques.
The data were then divided into specific subsets through data partitioning and subjected to filling procedures to standardize the sequence lengths for further analysis and modeling. The test results were compared with the developed hybrid model.
Firstly we decided to see how a simple BiLSTM model performs on the datasets we obtained.The BiLSTM model can be summarized as follows.

Model: "sequential"

Layers	**Output Shape**	**Parameters**
Embedding Layer	(None, 200, 100)	7530300
BiLSTM Layer	(None, 256)	234496
Dense Layer	(None, 6)	1542

The development of the hybrid CNN-BiLSTM model commenced with the labeling phase, pre-processing, and feature extraction, as previously described. We employed an embedding approach to represent text as low-dimensional numeric vectors with a padding size set at 300,which differs from previous approaches as given.

Model: "sequential"

Layers	**Output Shape**	**Parameter**
Embedding Layer	(None, 200, 100)	7530300
Convolution1D Layer	(None, 200, 64)	19264
Pooling Layer	(None, 100, 64)	0
BiLSTM Layer	(None, 256)	197632
Dense	(None, 6)	1542

And also one more model as follows:

Model: "sequential"

Layers	**Output Shape**	**Parameter**
Embedding Layer	(None, 200, 100)	7514600
Convolution1D Layer	(None, 198, 200)	60200
BiLSTM Layer	(None, 198, 128)	135680
Dropout	(None, 198,128)	0
BiLSTM Layer	(None, 128)	98816
Dense	(None, 50)	6450
Dense	(None, 50)	2550
Flatten	(None, 50)	0
Dense	(None, 100)	5100
Dense	(None, 6)	606

We compared this model's architecture with prior research, as documented in Table below. In each study, the model architecture was tailored to specific data characteristics. Jang et al. [11] utilized a hybrid CNN BiLSTM with Word2vec Skip Gram to process reviews of clothing and camera products. Salur and Aydin [9] employed tweets from Turkish GSM operator users using a model comprising CNN BiLSTM and character + fast text. Soumya and Pramod [13] explored sentiment in Malayalam Tweets using a model consisting of CNN BiLSTM and CNN LSTM.This hybrid model combines CNN layers for text feature extraction with BiLSTM layers to comprehend word context in the text. CNN, equipped with filters and kernels,identifies essential patterns within the text, whereas BiLSTM operates bidirectionally to understand word relationships. Non-linearity and the prevention of overfitting are introduced through the use of activation functions, specifically ReLU, and the incorporation of dropout mechanisms that randomly deactivate neurons during training.

We mitigated overfitting by adding a dropout layer following the initial BiLSTM layer. The dropout function introduces random neuron deactivation during training, and with a dropout rate set at 0, it ensures that active neurons prevent the model from closely fitting the training data, ultimately enhancing its generalization. Additionally, adding a dense layer with fewer units when paired with BiLSTM reduces the model's capacity to further prevent Overfitting.

The combination of CNN and BiLSTM within a single architecture enables more comprehensive and in-depth processing of the data. CNN helps in extracting spatial features while BiLSTM helps in extracting temporal or contextual features [8]. The hybrid CNN-BiLSTM model begins with the labeling, preprocessing and feature extraction phases as explained previously.

The model is designed with embedding to represent the text as low-dimensional numerical vectors using padding with a size of 300. This embedding differs from the approaches used by [7,11,9,13] as shown below.

MODEL	**EMBEDDING**	**F1-SCORE**	**ACCURACY**
CNN LSTM CNN BiLSTM[7]	POS Tagging, sentiment vector	81.7%	77.4%
CNN BiLSTM[11]	Word2vec+ Skip Gram	88.0%	87.6%
CNN BiLSTM[9]	Carakter +Fastext	89.0%	82.1%
CNN BiLSTM CNN LSTM[13]	Sentiment Tagging	75.0%	85.5%

V.RESULT AND OBSERVATION

MODEL	DATASET	EMBEDDING	ACCURACY
ML(SVM)	1	Tf-idf vectorizer	72.71%
ML(SVM)	2	Tf-idf vectorizer	87.11%
BiLSTM	1	word2vec(wikipedia file)	72.68%
BiLSTM	2	Glove (twitter)	92.26%
CNN-LSTM(1)	1	word2vec(wikipedia file)	73.36%
CNN-LSTM(1)	2	Glove (twitter)	93.46%
CNN-LSTM(2)	1	word2vec(wikipedia file)	73.16%
CNN-LSTM(2)	2	Glove (twitter)	94%
BERT	1	Wordpiece	82.67%
BERT	2	Wordpiece	97%

CNN+BIlSTM[2] dataset 2 :

Emotion	precision	recall	f1-score
anger	0.94	0.95	0.94
fear	0.94	0.86	0.90
joy	1.00	0.91	0.95
love	0.77	1.00	0.87
sadness	0.98	0.97	0.97
surprise	0.75	0.99	0.85
accuracy			0.94
macro avg	0.89	0.95	0.92
weighted avg	0.95	0.94	0.94

Model Output For Any Input Text

```
Message: ['I saw a tiger in a house when i am alone']
Probabilities for each emotion in given message:

sadness: 33.21581482887268
joy: 1.5119134448468685
love: 0.06470673251897097
anger: 8.698828518390656
fear: 56.042128801345825
surprise: 0.4666124004870653

predicted: fear
```

VI.Conclusion

In terms of NLP observations, the model effectively leverages word embeddings such as GloVe and Word2Vec to capture semantic relationships and contextual nuances within the text data. This aids in accurately interpreting emotions across different contexts and domains. The success of the model showcases the potential of deep learning-based approaches for real-world emotion detection applications, paving the way for more empathetic and sophisticated NLP systems.

We successfully demonstrate the effectiveness of a hybrid CNN-BiLSTM model for text emotion analysis, particularly in the domain of sentiment analysis. The model, combining convolutional and bidirectional long short-term memory networks, exhibits high performance across various datasets. It captures both spatial and temporal features of the text, leading to precise emotion classification with accuracy rates up to 94% using Glove embeddings. The model excels in accurately classifying emotions including joy, sadness, anger, fear , disgust, neutral and surprise.

Future Work

- Integrating text with other modalities, such as audio, video, and facial expressions, to provide a more comprehensive understanding of emotional states.
- Exploring the use of transformer-based models, attention mechanisms, and other deep learning techniques to further improve emotion analysis performance.
- Developing models that can effectively analyze emotional content across different languages and cultural contexts.
- Collecting and annotating larger, more diverse datasets to capture the richness and complexity of human emotional expression.

References

[1] M. P. Akhter, Z. Jiangbin, I. R. Naqvi, M. Abdelmajeed, A.Mehmood, and M. T. Sadiq, “Document-level text classification using single-layer multisize filters convolutional neural network,” IEEE Access, vol. 8, no. Ml, pp. 42689–42707, 2020. doi: 10.1109/ACCESS.2020.2976744

[2] A. Wahdan, S. Hantoobi, S. A. Salloum, and K. Shaalan, “A systematic review of text classification research based on deep learning models in Arabic language,” Int. J. Electr. Comput. Eng., vol. 10, no. 6, pp. 6629–6643, 2020. doi: 10.11591/IJECE.V10I6.PP6629-6643

[3] W. Fang, H. Luo, S. Xu, P. E. D. Love, Z. Lu, and C. Ye, “Automated text classification of near-misses from safety reports: An improved deep learning approach,” Adv. Eng. Informatics, vol. 44, no. March 2019, 101060, 2020. doi: 10.1016/j.aei.2020.101060

[4] Z. Liu, C. Lu, H. Huang, S. Lyu, and Z. Tao, “Hierarchical Multi- granularity attention- based hybrid neural network for text classification,” IEEE Access, vol. 8, pp. 149362–149371, 2020. doi: 10.1109/ACCESS.2020.3016727

[5] Q. Li et al., “A survey on text classification: From shallow to deep learning,” IEEE Trans. NEURAL NETWORKS Learn. Syst., vol. 31, no. 11, pp. 1–21, 2020.

[6] F. Zaman, M. Shardlow, S. Hassan, and N. Radi, “HTSS : A novel hybrid text summarisation and simplification architecture,” Inf. Process. Manag., vol. 57, no. 6, 102351, 2020. doi: 10.1016/j.ipm.2020.102351

[7] K. Pasupa, T. Seneewong, and N. Ayutthaya, “Thai sentiment analysis with deep learning techniques: A comparative study based on word embedding , POS-tag , and sentic features,” Sustain. Cities Soc., vol. 50, no. 7, 101615, 2019. doi: 10.1016/j.scs.2019.101615

[8] S. Kumar, C. Akhilesh, K. Vijay, and B. Semwal, “A multibranch CNN-BiLSTM model for human activity recognition using wearable sensor data,”

Vis. Comput., vol. 38, no. 12, pp. 4095–4109, 2021. doi: 10.1007/s00371-021-02283-3

[9] M. U. Salur and I. Aydin, “A novel hybrid deep learning model for sentiment classification,” IEEE Access, vol. 8, pp. 58080–58093,2020. doi: 10.1109/ACCESS.2020.2982538

[10] U. Naqvi, A. Majid, and S. A. L. I. Abbas, “UTSA : Urdu text sentiment analysis using deep learning methods,” IEEE Access, vol. 9, pp. 114085–114094, 2021. doi: 10.1109/ACCESS.2021.3104308

[11] B. Jang, M. Kim, G. Harerimana, S. Kang, and J. W. Kim, “Applied sciences Bi-LSTM Model to increase accuracy in text classification:Combining Word2vec CNN and attention mechanism,” Appl. Sci.,vol. 10, no. 17, 5841, 2020.

[12] A. U. Rehman, A. K. Malik, B. Raza, and W. Ali, “A hybrid CNN-LSTM model for improving accuracy of movie reviews sentiment analysis,” Multimed. Tools Appl., vol. 78, no. 18, pp. 26597–26613,2019. doi: 10.1007/s11042-019-07788-7

[13] S. Soumya and K. V Pramod, “Hybrid deep learning approach for sentiment classification of malayalam tweets,” Int. J. Adv. Comput. Sci. Appl., vol. 13, no. 4, pp. 891–899, 202

Code For BiLSTM+CNN :

Libraries Used:

```
import pandas as pd
import numpy as np
import nltk
import string
nltk.download('punkt')
# text preprocessing
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('stopwords')
import re, sys, os, csv
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
# plots and metrics
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix

# preparing input to our model
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
#from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical

# keras layers
from keras.models import Sequential
from keras.layers import Embedding, Bidirectional, LSTM,GRU, Dense

# Number of labels: joy, anger, fear, sadness, neutral
num_classes = 6
```

Variable Instantiation

```
# Number of dimensions for word embedding
embed_num_dims = 100
# Max input length (max number of words)
max_seq_len = 200
```

Reading Dataset

```
data_train = pd.read_csv('/content/drive/MyDrive/train_data2.csv', encoding='utf-8')
data_test = pd.read_csv('/content/drive/MyDrive/test_data2.csv', encoding='utf-8')

# Assuming 'class_names' is a dictionary mapping integer labels to emotion strings
class_names = {0: 'sadness', 1: 'joy', 2: 'love', 3: 'anger', 4: 'fear',5:'surprise'}

# Convert integer labels to emotion strings
data_test['label'] = data_test['label'].map(class_names)
data_train['label'] = data_train['label'].map(class_names)

data = pd.concat([data_train, data_test], ignore_index=True)
```

Defining Train And Test Dataset

```
X_train = data_train.text
X_test = data_test.text
y_train = data_train.label
y_test = data_test.label
d = pd.concat([X_train, X_test], ignore_index=True)
```

Finding Max length Of Sentence in Corpus

```
tokenized_sentences = [sentence.split(" ") for sentence in d]
longest_sen = max(tokenized_sentences, key=len)
longest_sen_length = len(longest_sen)
shortest_sen = min(tokenized_sentences, key=len)
shortest_sen_length = len(shortest_sen)
print("Longest sentence:", longest_sen)
print("Length of longest sentence:", longest_sen_length)
print("Shortest sentence:", shortest_sen)
print("Length of shortest sentence:", shortest_sen_length)
```

```
Longest sentence: ['a', 'few', 'days', 'back', 'i', 'was', 'waiting', 'for', 'the', 'bus', 'at', 'the', 'bus', 'stop', 'before',
'getting', 'into', 'the', 'bus', 'i', 'had', 'prepared', 'the', 'exact', 'amount', 'of', 'coins', 'to', 'pay', 'for', 'the', 'bus',
'fair', 'and', 'when', 'i', 'got', 'into', 'the', 'bus', 'i', 'put', 'these', 'coins', 'into', 'the', 'box', 'meant', 'to', 'collect',
'the', 'bus', 'fair', 'i', 'thought', 'that', 'i', 'had', 'paid', 'and', 'wanted', 'to', 'get', 'inside', 'however', 'the', 'bus',
'driver', 'called', 'me', 'and', 'asked', 'me', 'in', 'an', 'impolite', 'way', 'if', 'the', 'coins', 'were', 'stuck', 'at', 'the',
'opening', 'of', 'the', 'box', 'he', 'had', 'not', 'seen', 'me', 'paying', 'and', 'there', 'wasnt', 'a', 'stack', 'of', 'coins',
'in', 'the', 'box', 'i', 'could', 'not', 'understand', 'this', 'and', 'the', 'driver', 'kept', 'questioning', 'me', 'he', 'made',
'me', 'feel', 'angry', 'and', 'at', 'last', 'i', 'inserted', 'a', 'dollar', 'coin', 'in', 'the', 'box', 'just', 'to', 'get', 'away',
'from', 'him', 'later', 'i', 'found', 'that', 'i', 'had', 'forgotten', 'a', 'few', 'coins', 'in', 'my', 'pocket', 'and', 'had',
'not', 'paid', 'enough', 'for', 'the', 'fair', 'the', 'first', 'time', 'after', 'i', 'had', 'entered', 'the', 'bus', 'i', 'could',
'still', 'hear', 'him', 'scolding', 'me', 'and', 'i', 'felt', 'disgusted']
Length of longest sentence: 178
Shortest sentence: ['earlier']
Length of shortest sentence: 1
```

Printing Dataset info

```
print(data.label.value_counts())
data.head(100)

joy         141067
sadness     121187
anger        57317
fear         47712
love         34554
surprise     14972
Name: label, dtype: int64
```

Data Cleaning

```
def clean_text(data):
   # remove hashtags and @usernames
   data = re.sub(r"(#[\d\w\.]+)", '', data)
   data = re.sub(r"(@[\d\w\.]+)", '', data)
   data = re.sub(r'[^\w\s]','', data)
   data = re.sub(r'\d','',data)
   # tokenization using nltk
   data = data.lower()
   data = word_tokenize(data)
   stop_words = set(stopwords.words('english'))
   data = [word for word in data if word not in stop_words]
  return data

texts = [' '.join(clean_text(text)) for text in data.text]
texts_train = [' '.join(clean_text(text)) for text in X_train]
texts_test = [' '.join(clean_text(text)) for text in X_test]
```

Tokenization and Padding

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
sequence_train = tokenizer.texts_to_sequences(texts_train)
sequence_test = tokenizer.texts_to_sequences(texts_test)
index_of_words = tokenizer.word_index

# vocab size is number of unique words + reserved 0 index for padding
vocab_size = len(index_of_words) + 1
print('Number of unique words: {}'.format(len(index_of_words)))
Number of unique words: 75145

X_train_pad = pad_sequences(sequence_train, maxlen = max_seq_len )
X_test_pad = pad_sequences(sequence_test, maxlen = max_seq_len )
X_train_pad
```

Encoding:

```
encoding = {
    'sadness':0,
    'joy':1,
    'love':2,
    'anger':3,
    'fear':4,
    'surprise':5
}

# Integer labels
y_train = [encoding[x] for x in data_train.label]
y_test = [encoding[x] for x in data_test.label]


y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train
```

Embedding

```
def create_embedding_matrix(filepath, word_index, embedding_dim):
  vocab_size = len(word_index) + 1  # Adding again 1 because of reserved 0 index
  embedding_matrix = np.zeros((vocab_size, embedding_dim))
  with open(filepath) as f:
      for line in f:
          word, *vector = line.split()
          if word in word_index:
              idx = word_index[word]
              embedding_matrix[idx] = np.array(
                  vector, dtype=np.float32)[:embedding_dim]
  return embedding_matrix

embedd_matrix = create_embedding_matrix('/content/drive/MyDrive/glove.twitter.27B.100d_wv.txt',
index_of_words, embed_num_dims)
embedd_matrix.shape

(75146, 100)
```

Finding Unseen Words

```
# Inspect unseen words
new_words = 0

for word in index_of_words:
  entry = embedd_matrix[index_of_words[word]]
  if all(v == 0 for v in entry):
      new_words = new_words + 1

print('Words found in wiki vocab: ' + str(len(index_of_words) - new_words))
print('New words found: ' + str(new_words))

Words found in wiki vocab: 55884
New words found: 19261
```

Embedding Layer

```
# Embedding layer before the actaul LSTM
embedd_layer = Embedding(vocab_size,
                        embed_num_dims,
                        input_length = max_seq_len,
                        weights = [embedd_matrix],
                        trainable=False)
```

Model Pipeline

```
# Convolution
from keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense,Dropout,Flatten
# Parameters
from keras.optimizers import SGD,Adam
from keras import regularizers
lstm_output_size = 128
bidirectional = True


# Embedding Layer, LSTM or biLSTM, Dense, softmax
model = Sequential()
model.add(embedd_layer)
model.add(Conv1D(200, kernel_size=3, activation = "relu"))
model.add(Bidirectional(LSTM(64, return_sequences=True)))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(64)))
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Flatten())
  #l2 regularizer
model.add(Dense(100,kernel_regularizer=regularizers.l2(0.01),activation="relu"))
model.add(Dense(6, activation='softmax'))
  #sgd= SGD(lr=0.0001,decay=1e-6,momentum=0.9,nesterov=True)
adam=Adam(learning_rate=0.0005,beta_1=0.9,beta_2=0.999,epsilon=1e-07,amsgrad=False)
model.compile(loss='categorical_crossentropy',optimizer=adam,metrics=['accuracy'])
model.summary()
```

Model Training

```
batch_size = 256
epochs = 5
hist = model.fit(X_train_pad, y_train,
            batch_size=batch_size,
            epochs=epochs,
            validation_data=(X_test_pad,y_test))
```

Plotting

```
#  "Accuracy"
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()


# "Loss"
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

Saving Model

```
# creates a HDF5 file 'my_model.h5'
model.save('models/CNN_biLSTM_dataset2.h5')
```

Loading Model

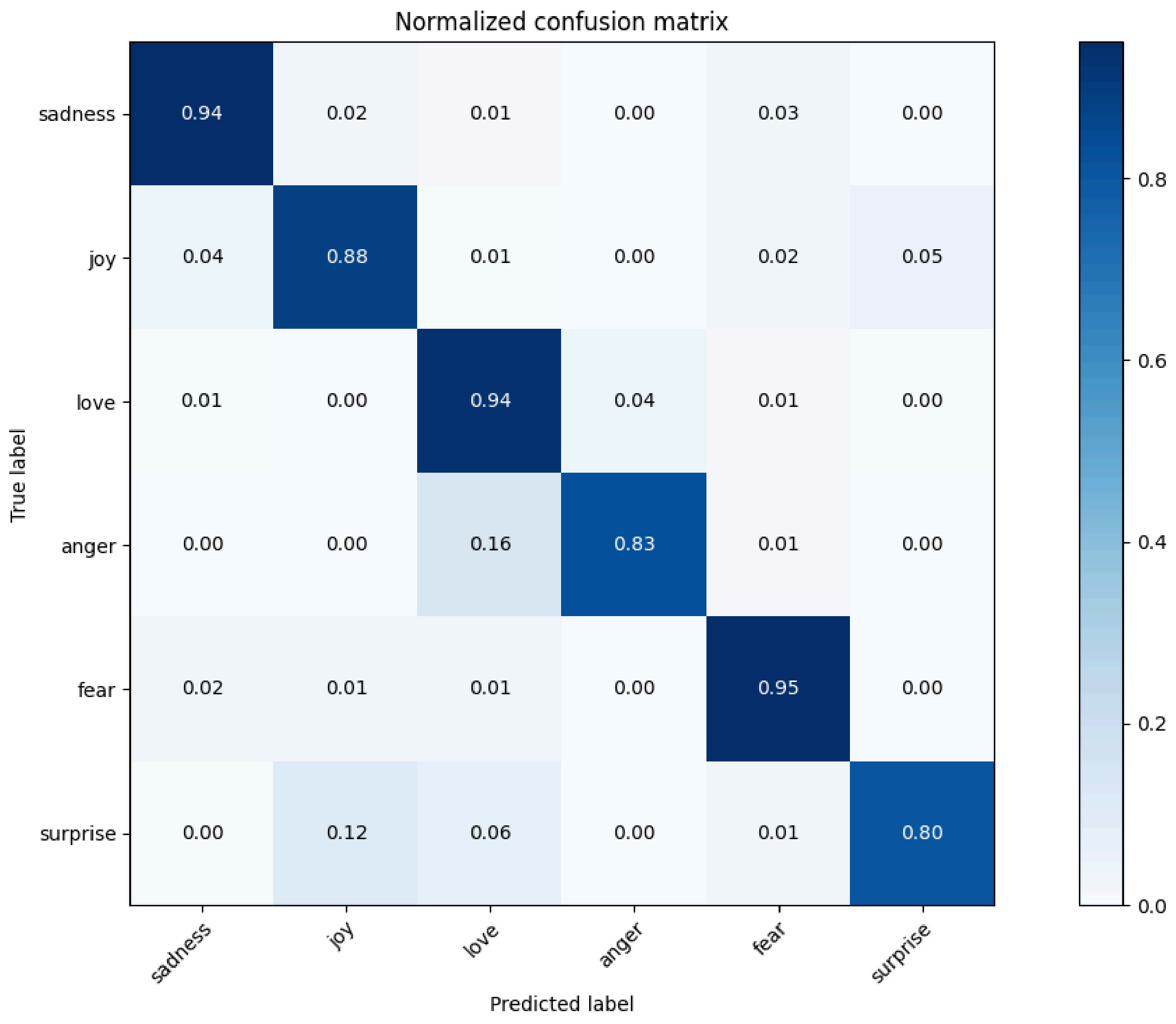
```
from keras.models import load_model
predictor = load_model('/content/drive/MyDrive/biLSTM_dataset2.h5')

predictions = predictor.predict(X_test_pad)
print("Accuracy: {:.2f}%".format(accuracy_score(data_test.label, predictions) * 100))
print("\nF1 Score: {:.2f}".format(f1_score(data_test.label, predictions, average='micro') * 100))
```

Classification report

```
from sklearn.metrics import classification_report
# Assuming predictions and data_test.label contain the predicted and true labels respectively
# Print classification report
print(classification_report(data_test.label, predictions))
```

Confusion Matrix

```
#plotting confusion matrix
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    '''
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    '''
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    fig, ax = plt.subplots()

    # Set size
    fig.set_size_inches(12.5, 7.5)
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    ax.grid(False)

    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


# Plot normalized confusion matrix
plot_confusion_matrix(data_test.label, predictions, classes=class_names, normalize=True,
title='Normalized confusion matrix')
plt.show()
```



Try Input

```
import time
message = ['When I meet you first time it was wonderful']
seq = tokenizer.texts_to_sequences(message)
padded = pad_sequences(seq, maxlen=max_seq_len)
start_time = time.time()
pred = predictor.predict(padded)
print('Message: ' + str(message))
print('predicted: {} ({:.2f} seconds)'.format(class_names[np.argmax(pred)], (time.time() - start_time)))
```

```
1/1 [==============================] - 0s 219ms/step
Message: ['When I meet you first time it was wonderful']
predicted: joy (0.50 seconds)
```