2020

Deep Neural Networks for Sentiment Analysis in Tweets with Emoticons

Mutharasu Narayanaperumal
Nova Southeastern University, nmarasu@gmail.com

DEEP NEURAL NETWORKS FOR SENTIMENT ANALYSIS IN TWEETS WITH EMOTICONS

by

Mutharasu Narayanaperumal

A dissertation submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy

in

Information Systems

College of Engineering and Computing

Nova Southeastern University

2020

We hereby certify that this dissertation, submitted by Mutharasu NarayanaPerumal conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

Sumitra Mukherjee — June 22, 2020

Sumitra Mukherjee, Ph.D.
Chairperson of Dissertation Committee

Date

June 22, 2020

Michael J. Laszlo Ph.D.
Dissertation Committee Member

Date

June 22, 2020

Francisco J. Mitropoulos, Ph.D.
Dissertation Committee Member

Date

Approved:

Meline Kevorkian — 06/22/2020

Meline Kevorkian, Ed.D.
Dean, College of Computing and Engineering

Date

College of Computing and Engineering
Nova Southeastern University

2020

Table of Contents

List of Tables

Tables

List of Figures

Figures

An Abstract of a Dissertation Submitted to Nova Southeastern University

in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

DEEP NEURAL NETWORKS FOR SENTIMENT ANALYSIS IN TWEETS WITH EMOTICONS

by

Mutharasu Narayanaperumal

May 2020

Businesses glean meaningful feedback in regard to products and services from social media posts in order to improve the quality of products and services, as well as to meet customer expectations. Sentiment analysis is increasingly being used to help businesses by assigning positive or negative polarity to such posts. Although methods currently exist to determine the polarity of sentiments, such methods are unreliable when posts contain terms that are not typically part of a standard dictionary used for sentiment analysis, such as slang and informal language. This dissertation has aimed to empirically investigate alternative methods to improve the classification accuracy of sentiments in such contexts. Specifically, it considers posts written in English that include emoticons.

The benchmark Sentiment140 English language datasets were used for evaluation and labeled tweets that included emoticons. Two types of deep neural networks–Convolution Neural Networks (CNN) and Long Short-Term Memory (LSTM) Networks–were used for classification since they have been demonstrated to produce the best results. All terms in the tweets were represented using the pre-trained embedding vectors word2vec, GloVe, and fastText. Baseline models were trained and tested using tweets with their emoticons removed. For each baseline model, a corresponding model was trained that included emoticons as inputs; in others, emoticons were replaced with English language. Accuracy, precision, recall, and F_1 scores of models using emoticons were compared to their corresponding baseline models that did not use emoticons.

Experiments are conducted on data with emoticons and emoticons removed for all the models. Our experiments showed that LSTM that uses an attention model with fastText embedding outperformed the linear models for identifying sentiment for the all datasets used. We also learned that when we replaced emoticons with English language, the sentiment classification accuracy improved. We therefore concluded that inclusion of emoticons as features achieves the highest accuracy in our research on sentiment classification.

Keywords: Sentiment analysis, deep Learning, emoticon, embedding, convolution neural network, long short-term memory, attention

Acknowledgements

Throughout my time in graduate school, many people have helped me, and I am so appreciative of their contributions. First, I would like to sincerely thank my advisor, Prof. Dr. Sumitra Mukherjee for his tremendous support of my PhD study and related research, as well as for his patience, motivation, and extensive knowledge. His direction helped me throughout my research and writing of this thesis. I would also like to thank to my committee members, Dr. Mitropoulos and Dr. Laszlo, for the guidance they offered and the clarity they gave me.

Several friends and colleagues have been supportive of my efforts to attain a PhD and I wish to thank them as well. Dr. Shyamala Srinivasan, one of my teachers in my master's degree who provided an emotional push in the initial stage, to move me from a point of indecision to commitment to my doctorate. Also, I would like to thank my manager Dr. Kaan Katircioglu, who provided encouragement at a key point in this journey when I was slowing down on my progress on the PhD journey.

Finally, I would like to show my sincerest gratitude to my parents and family members. Often, they used to ask and encourage me about what the graduation timeline was and that made me want to continue in the PhD program. Ultimately, I would like to thank my wife, Leela, who has supported me throughout this process. This dissertation would not have been possible without her warm love, continued patience, and endless support. I owe her a sincere debt of gratitude. And to my children Hritik, Anish and Anisha; they used to track my progress and helped make this happen.

1 Introduction

Sentiment analysis is valuable in social media monitoring as it allows one to gain insights into certain contexts or topics and is used in a diverse set of fields, such as marketing and advertising, social media, economics, and political science (Rosenthal et al., 2015). Social media platforms such as Twitter, Facebook, and YouTube are currently among the most popular venues for customers to debate and review new products and services in various markets (Poria, Cambria, Howard, Huang, & Hussain, 2016). However, the inherent chaotic nature of social media content poses severe challenges to the practical applications of sentiment analysis, such as extracting meaningful feedback for products or services, understanding product quality, or meeting customer expectations.

In recent years, deep neural networks have been shown to be effective for text classification (Kim, 2014). Convolution Neural Networks (CNN) were initially built for the computer vision domain. Subsequently, CNNs were explored for natural language processing purposes and achieved excellent results for text classification (Zhang & Wallace, 2017), modeling sentences (Kalchbrenner, Grefenstette, & Blunsom, 2014), sentiment analysis (dos Santos & Gatti, 2014). Long Short-Term Memory (LSTM), a deep learning model effective in analyzing long sequences, has been used to categorize emotions in natural language processing contexts (Cliché, 2017). These deep learning methods use word embeddings to learn the semantic relationships of words in order to improve model performance (Mikolov, Chen, Corrado, & Dean, 2013).

In this dissertation, we considered the significance of including emoticons in Twitter messages and followed an approach similar to Kim (2014), which involved the use of a

few different variations of CNN models. In addition, we evaluated a variety of LSTM models to identify an accurate deep learning model for sentiment analysis. We considered the emoticons of the input text and evaluated the importance of emoticons in sentiment analysis.

1.1 Problem Statement

A sentiment analysis technique was used in this study to classify the text data of Twitter posts into one of two sentiment polarities: POSITIVE, NEGATIVE. The sentiment analysis model produced different polarities for each tweet depending on whether it considered or ignored emoticons (Kiritchenko, Zhu, & Mohammad, 2014). Emoticons and product ratings are instances of emotional signals from customers that are connected to sentiments expressed in sentences or transcripts. The research we developed evaluated methods to improve the accuracy of sentiment classification by incorporating emoticons as features.

Recent research on polarity detection includes deep neural network techniques (CNN and LSTM), ensemble methods, and word embedding (Cambria, Poria, Gelbukh, & Thelwall, 2017). We considered the tweeter's text from the Sentiment140 dataset, using the deep learning techniques CNN and LSTM to investigate whether including emoticons could improve predictive accuracy.

2 Review of Literature

In this section, the existing research on Sentiment Analysis that uses deep learning networks and word embedding methods is discussed. Subsection 2.1 presents how CNNs utilize layers with convolution filters. Subsection 2.2 involves a discussion of LSTM, a subtype of recurrent neural networks, which has a memory that learns from context. Finally, word embedding is described in subsection 2.3.

2.1 Convolutional Neural Network (CNN)

The CNN is a deep neural network that utilizes layers with convolution filters that are applied to a set of features (Kim, 2014). Kim's (2014) research slightly changed the CNN architecture that was designed by Collobert et al. (2011). Collobert et al. (2011) considered the complete input sentences that would be passed through the lookup table layer in order to generate local features around each word of the sentence. Those features go through convolutional layers, which combines these features into a global feature vector that can then be fed to fully connected layers. The model proposed by Kim (2014) is shown in Figure 1.

Kim (2014) had a CNN model with one layer of convolution, in which the features were extracted from a small window of words instead of whole sentences. The input sentence of the model is represented as:

$$X_{1:n} = X_1 \parallel X_2 \parallel \cdots \parallel X_n$$

where n is the length of the sentence, $X_{1:n}$ refers to the combinations of the words $X_1, X_2, \ldots, X_n$, $\parallel$ is the concatenation operator, and $X_i \in R^k$ is the K-dimensional input word vector that corresponds to ith word in the sentence. Each sentence in the word vector should

have the same length and pad all the sequences to the maximum length of the sentences (n). A convolution operation uses a filter $w \in R^{hk}$ and a window of h words to generate a new feature. A feature c_i produced through a window of words $X_{i:i+h-1}$ in the sentence is represented as:

$$c_i = f(w.X_{i:i+h-1} + b)$$

where b is the bias term, f is a non-linear hyperbolic tangent function, and w is the filter.

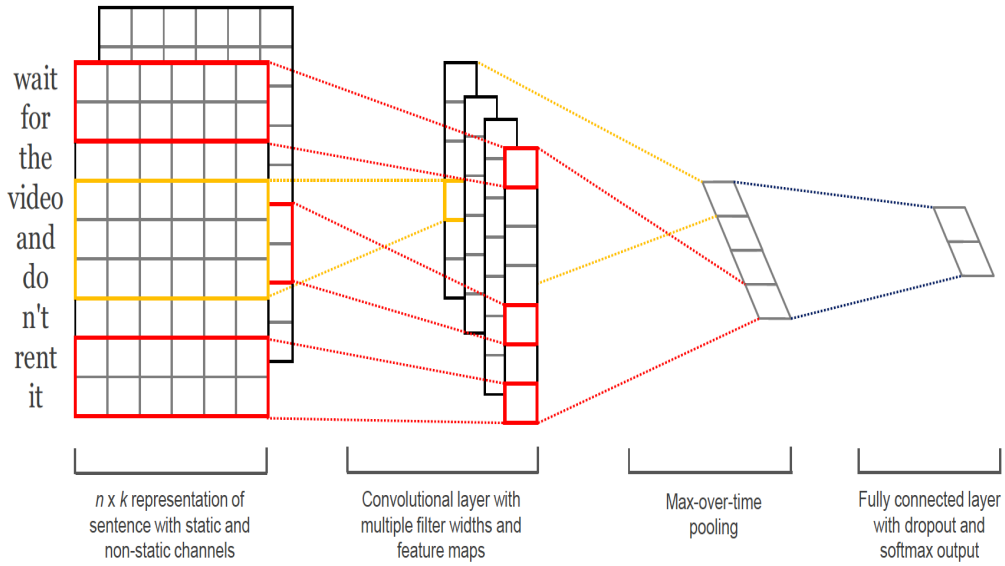

Figure 1. Kim's (2014) CNN model for two channels.

The convolution layer filter is applied to each possible window of words (h) in the input sentence to generate a feature map:

$$c = [c_1, c_2, \ldots, c_{n-h+1}]$$

where $c \in R^{n-h+1}$. It then collects the most important feature in each feature map that is extracted through a maximum over-time pooling operation i.e., $\hat{C} = \max\{c\}$. This operation ideally selects a feature that has the highest value on each feature map. For each filter in this model, a feature is produced. Kim's (2014) model utilized multiple filters with various window sizes of words to generate multiple features. These selected essential features form

the penultimate layer and are shifted to a fully connected softmax layer. Subsequently, the softmax layer produces the probability distribution over the labels.

Kim (2014) implemented dropout regularization on the penultimate layer with a constraint on l_2-norms of the weight vectors. Dropout regularization limits correlated hidden units by randomly dropping out during forward backpropagation operations. For instance, normal forward propagation gives,

$$y = w.z + b$$

where y is the output units, z is the penultimate layer $z = [\hat{c}_1, \hat{c}_2, \dots, \hat{c}_m]$, and m is the filters. When adopting the dropout layer, the previous forward propagation output becomes:

$$y = w.(z \odot r) + b$$

where $\odot$ is the element-wise multiplication operator and r is a 'masking' vector of Bernoulli random variables, with the probability of p being 1 ($r \in R^m$). Gradients used unmasked units for backpropagation.

Kim (2014) experimented with six different datasets including movie reviews, the Stanford Sentiment Treebank, a task-specific subjective dataset, and customer product reviews. The CNN model was trained on all six datasets and attuned to the hyperparameters to maximize classification accuracy. This model used "*relu*" as an activation function, [3,4,5] as filter windows size (h) along with a 100-feature map per window, 0.5 as a dropout rate, and l_2 as a regularization method, and 50 as a batch size. This CNN model initialized word vectors through pre-trained word2vec vectors that have been trained on 100 billion words from Google News, in addition to 300 dimensions used as a continuous bag-of-words mechanism.

Kim (2014) examined several variants of CNN models, such as the CNN-rand (base model), where all words are randomly initialized and updated during training. Within the CNN-static model where pre-trained vectors are defined by word2vec, all words that have been randomly initialized are retained as static and tuned to the model's parameters. The next model is CNN-non-static, which uses the same parameters as the CNN-static model, but only pre-trained vectors that are modified in each task. The CNN-multichannel model uses two sets of word vectors in a model and both are treated as 'channels.' Both channels' pre-trained vectors are initialized with word2vec. A filter is used on both channels, but gradients are used on just one of the channels to perform a backpropagation process. This model allows one set of vectors to remain static while another is being adjusted closely.

This research compared the models' performances using a confusion matrix, precision, and recall. Those metrics showed that the static vectors' model provided a better result than other CNN models. Kim's (2014) baseline model did not perform better than models with pre-trained vectors. A simple model with static vectors predicted sentiment classification extremely well. That demonstrated that the pre-training of word vectors is an important feature in deep neural network models for NLP. The author also commented that the multichannel model prevented overfitting problems better than a single channel model when the number of observations was small.

In CNNs, choosing the right model architecture and identifying the hyperparameters, such as filter size, and regularization parameters, are critical tasks. Zhang et al. (2015) performed a sensitivity analysis to determine the effect of architecture components on model performance using a single layer CNN that was similar to Kim's (2014) architecture. In this sensitivity experiment, initially tokenized sentences were converted into word

vectors using the pre-training word embedding models, including word2vec or GloVe. Zhang et al. (2015) compared the model's performance using nine different datasets–seven of them were used by Kim (2014) for sentence classification. To provide a point of reference for the CNN's performance, Zhang et al. (2015) classified the sentiment through a linear kernel support Vector Machine (SVM) and recorded the model's performance. This SVM model experimented with uni-gram, bi-gram, and a combination of uni-gram and bi-gram features. This model considered frequent 30k n-grams for all datasets for training and tuned the hyperparameters through cross-fold validation, which optimized the model's accuracy.

Zhang et al. (2015) built a CNN baseline model with hyperparameters, which had been used in Kim's (2014) work for evaluating performance. Performance of the model was measured through mean performance metrics via 10-fold cross validation (CV), in addition to randomly chosen dropout rates, filter sizes, feature maps, activation functions (from a list of including relu/tanh/Sigmoid/SoftPlus/Cube/tanh cube). The CV operation involved setting the pooling size as 1-max/k-max, the dropout rate from 0.0 to 0.9, and a random parameter weight. They repeated the CV experiment 100 times and recorded the model mean, minimum, and maximum average accuracy over each iteration. Afterwards, Zhang et al. (2015) measured performance with all datasets using different versions of the CNN models, as well using static and non-static word vectors.

Based on all of the experiments, they recommended the following hyperparameters for a sentiment analysis single layer CNN model: non-static word2vec or GloVe rather than one-hot word vectors, with a filter region size to be set between 1 and 10, as well as the number of feature maps for each filter region size to be between 100 and 600, the use of

'*relu*' and '*tanh*' as activation functions, including 1-max pooling, and a dropout rate higher than 0.5 for the regularization. The sensitivity analysis concluded that the CNN model hyperparameters would vary based on the dataset size.

In this dissertation, we used Kim's (2014) single channel CNN model as our base model along with Zhang et al.'s (2015) recommended hyperparameters. The input for this network was Twitter messages that were split into a sequence of words. Each word was mapped to an embedding vector, via one of the pre-trained word vectors: word2vec, GloVe, and fastText. The word vector was formed with $s \times d$ size, where s is the number of words in the tweet and d is the embedding dimension. We followed the padding logic that was used by Kim (2014), which produced the same matrix dimension for all tweets $X \in R^{S^1 \times d}$. In the tweet, some of the words may mislead the classification of certain phrases and marked the sentence as non-meaningful. For that, we added a dropout layer in the network that received embedding layer output and dropped random words to avoid overfitting.

Subsequently, we built the convolutional operation of the dropout layer output. Each convolution operation used a filtering matrix $w \in R^{c \times d}$, where c is the convolution size (i.e., the number of words). The convolutional operation is:

$$c_i = f(\sum_{j,k} w_{j,k} \left(X_{[i:i+h-1]}\right)_{j,k} + b)$$

where $b \in R$ is the bias term, and $f(x)$ is the activation function. The output of the convolution $c \in R^{S^1-h+1}$ is the concatenation of the convolution operator among all possible windows of words in a given tweet. Since we used different sizes of filters, each convolution produced tensors of different shapes, and so built a layer for each of them, and merged each convolution result into a single feature vector.

Next we applied a max-pooling operation in all convolutions, where $c_{max} = max(c)$. This operation extracted the most important feature from each convolution. The output of this operation returned the most important n-grams in the embedding vector for a better result on sentiment polarity. The max-pooling task collected all the c_{max} of each filter into one vector, $c_{max} \in R^m$, where m is the number of filters. This convolution layer used '*relu*' or '*tanh*' as an activation function. We also built a multichannel CNN model. A multichannel convolutional neural network for sentiment classification involved using multiple versions of the CNN model with various sized kernels from the embedding layer to the max-pooling task. The kernel size in a CNN defined the number of words to consider as the convolution was passed across the input tweet, providing a grouping parameter.

The final layer was the dropout layer, which is the most popular approach to regularizing a CNN and served to reduce overfitting issues. This layer prevented neurons from co-adapting and forced them to learn individually valuable features. The dropout layer passed the vector through a softmax layer in order to produce the expected sentiment analysis. This CNN model used '*categorical_crossentropy*' as a loss function and '*adagrad*' or '*adam*' as an optimizer. A categorical cross-entropy loss function compared the distribution of the prediction with the desired distribution as described here:

$$L(y, \hat{y}) = \sum_{j=0}^{m} \sum_{i=0}^{n} (y_{ij} \times log\ (\hat{y}))$$

where $\hat{y}$ is the predicted label. The optimizer '*adagrad,*' or the adaptive gradient, allowed the learning rate to adapt based on the parameter. It performed larger updates for infrequent features and smaller updates for frequently occurring features.

2.2 Long Short-Term Memory (LSTM)

A Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN)

model and has emerged as a powerful model in NLP applications that involve sequential data (Karpathy, Fei-Fei, & Li, 2016). The central principle of the LSTM model is that it can collect and retrieve information over long periods of time using its gating mechanisms. In NLP, LSTM uses its memory cells to remember long-range information and helps to maintain the message context. It makes use of standard stochastic gradient descent and truncated backpropagation through time. Karpathy et al. (2016) described how the LSTM resolved the difficulties of training RNN's backpropagation, which caused the gradients in an RNN to either explode or vanish.

The typical RNN recurrence form h_t^l is represented as:

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

where the hidden state vector is $h \in R^n$, W^l is the parameter matrix on each layer that has dimensions $[n \times 2n]$, tanh is applied element wise, $\mathrm{t} = 1..\mathrm{T}$ is the time, and $\mathrm{l} = 1..\mathrm{L}$ is the depth. W^l will change between layers and is shared the throughout the network. In RNN, the inputs from the layer below in depth (h_t^{l-1}) and before in time (h_{t-1}^l) are transformed and combined before being squashed by tanh (Karpathy et al., 2016).

LSTM models were mainly designed to mitigate the vanishing gradient problem. In addition to h_t^l, LSTM will maintain a memory vector c_t^l. In LSTM, each time-step uses explicit gating mechanisms for the read, write, or reset operations. The precise form of this model is delineated below:

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot tanh\left(c_t^l\right)$$

where W^l is a $[4n \times 2n]$ matrix, and the activation functions '*sigm'*, and '*tanh'* are implemented element wise. The three vectors $i, f, o \in R^n$ are represented as binary gates. i's primary function is to update each memory cell, whereas f resets to zero, and o maintains each hidden vector's local state. The activations of these gates depend on the sigmoid function that maintains the range between 0 and 1 in order to keep the model differentiable. The vector g $\in R^n$ keeps the value between -1 and 1 and is used to modify the memory contents additively. This additive operation is an essential feature of the LSTM model, because during backpropagation a sum operation assigns gradients. This allows gradients on the memory cells c to flow backward through time uninterrupted for long periods. In a LSTM, a network is required to maintain two vectors (c_t^l & h_t^l) at every point.

The standard LSTM cannot detect all the important context for sentiment classification. In order to address this issue, Wang, Huang, Zhu, and Zhao (2016) proposed a LSTM model with an attention mechanism that can capture the key part of a sentence for sentiment classification. The following diagrams differentiate the regular RNN versus the additive attention RNN (Wang et al., 2017). The attention layer yields an additional weight vector (α) for the LSTM that will be concatenated with the hidden layer vector (h) and return a weighted representation of sentence (r). That vector identifies the contribution of important elements in the final representation. The attention layer mechanism allows a LSTM to detect the most important part of a sentence when different aspects are present in the input dataset.

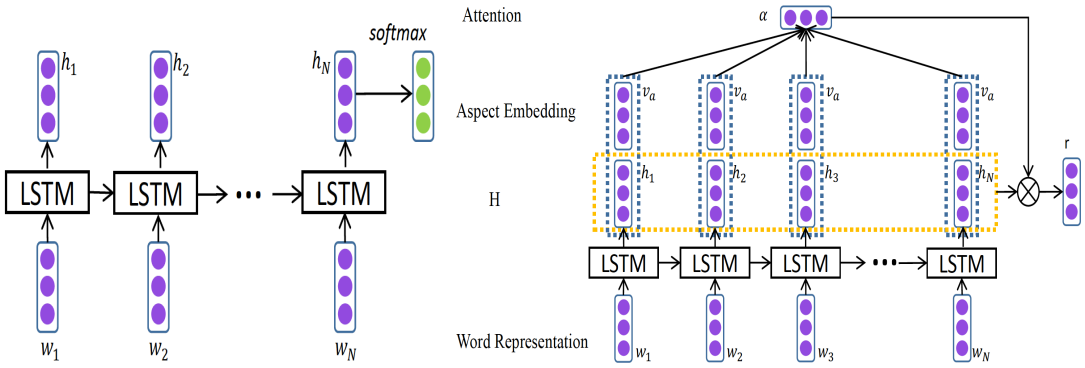

Figure 2. Left: Regular RNN model, Right: Attention RNN (Source: Wang et al., 2016)

Baziotis et al. (2017) presented two LSTM models for sentiment classification. The first LSTM model was designed to identify sentiment on the message-level. This model has a 2-layer Bidirectional LSTM, which was implemented with an attention mechanism for addressing the most informative words in the message. The second model was developed for topic-based sentiment analysis tasks. Baziotis et al. (2017) proposed a Siamese Bidirectional LSTM network with different attention logic than in the message-level deep learning network. For training these models, SemEval2017's subtasks A, B, C, D, and E dataset tweets were used.

2.2.1 2-layer Bidirectional LSTM

A 2-layer Bidirectional LSTM neural network model uses Twitter messages as input, in the form of a sequence of words. The input word vector X was converted into the low-dimensional vector space R^E through an embedding layer, where X is $(x_1, x_2, \dots, x_T)$, E is the size of the embedding layer, and T is the total words in a tweet. This model initializes the weights of the embedding layer with pre-trained word embedding vectors via GloVe and custom word vectors generated by 330M Twitter messages.

As illustrated in Figure 3, the study by Baziotis et al. (2017) used a Bidirectional LSTM (BiLSTM) to get word annotation summaries of the data from both directions. It

also stacked two layers of the BiLSTM to extract more important features from the tweets. The BiLSTM consists of a forward LSTM $\vec{f}$ that reads the text from x_1 to x_T and a backward LSTM $\overleftarrow{f}$ that reads the text from x_T to x_1. To do this final annotation for a given word, x_i, is achieved by concatenating both direction annotations i.e., $h_i = \overrightarrow{h_i} || \overleftarrow{h_i}$, $hi \in R^{2L}$, || in which L indicate the concatenation operation and the size of each LSTM, respectively.

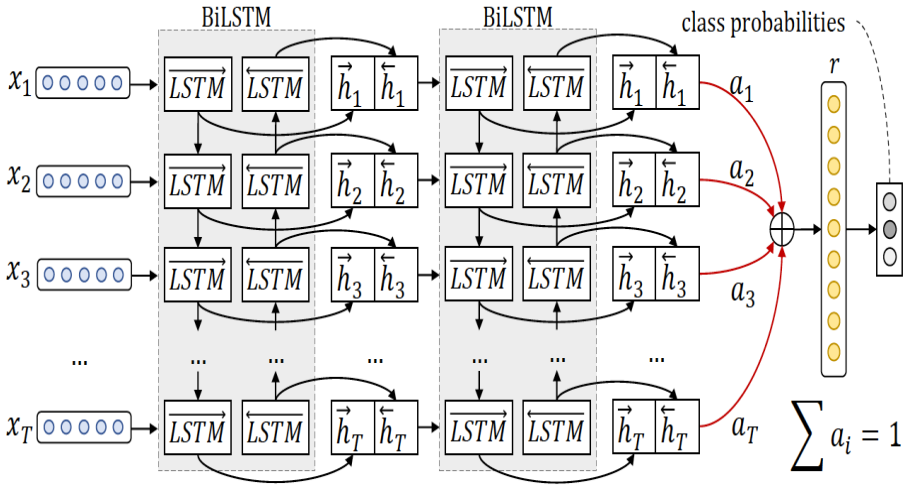

Figure 3. 2-layer Bidirectional LSTM (Source: Baziotis et al, 2017)

Normally, all the words in the tweet will not contribute to equally classifying the sentiment. This model used an attention mechanism to collect the importance of each word. This mechanism assigns a weight a_i to each word annotation, and consequently estimates the fixed representation of the whole message γ as the weighted sum of all word annotations.

$$e_i = tanh\,(W_h h_i + b_h)\,, e_i \in [-1,1]$$

$$a_i = \frac{exp\,(e_i)}{\sum_{t=1}^{T} exp\,(e_t)}, \sum_{i=1}^{T} a_i = 1$$

$$\gamma = \sum_{i=1}^{T} a_i h_i\,, \gamma \in R^{2L}$$

where W_h and b_h are the weights from the attention layer that will be high for the most essential words of a sentence. This model passes the feature vector γ for sentiment classification to a fully connected softmax layer in order to compute a probability distribution over all classes. This BiLSTM model used these as the hyperparameters, embedding the layer dimension as 300, the BiLSTM as 300, Gaussian noise σ as 0.2, dropout rate as 0.3 at the embedding layer, dropout rate as 0.25 at LSTM layer, and l_2 regularization of 0.0001 as the loss function.

2.2.2 Siamese Bidirectional LSTM

Baziotis et al. (2017) proposed another LSTM model as the Siamese Bidirectional LSTM (Figure 4), which has a different attention mechanism than the former one for a topic-based sentiment analysis task. This LSTM model takes two inputs from tweets (sequence of words) X^{tw} and topics (sequence of words) X^{to}, where $X^{tw} = (x_1^{tw}, x_2^{tw}, \dots x_{tw}^{tw})$, T_{tw} is the number of words in the tweet, $X^{to} = (x_1^{to}, x_2^{to}, \dots x_{to}^{to})$, and T_{to} is the number of words in the topic. This LSTM model projects all words to a low-dimensional vector space R^E using the pre-trained word embedding vector, where E is the size of the embedding layer.

For the topic-based sentiment analysis, the LSTM used a BiLSTM with shared weights to map the words of the tweet and the topic to the same vector space in order to make a meaningful comparison between the content of each (Baziotis et al., 2017). The BiLSTM generates annotations for the tweet H^{tw} and the topic H^{to}, where $H^{tw} = (h_1^{tw}, h_2^{tw}, \dots, h_{tw}^{tw})$ and $H^{to} = (h_1^{to}, h_2^{to}, \dots, h_{to}^{to})$ and each word annotation contains the concatenation of forward and backward layer annotation, that is defined as:

$$h_i^j = \overrightarrow{h_i^j} || \overleftarrow{h_i^j}, h_i^j \in R^{2L}, j \in \{tw, to\}$$

where || denotes concatenation operator, and size of the LSTM represents L.

Next, this network used a mean-Pooling layer over the annotation of the topic H^{to} for accumulating them to a single annotation. This layer produces a topic annotation,

$$\overline{h^{to}} = \frac{1}{T_{to}} \sum_{i=1}^{T_{to}} h_i^{to}$$

To get the final context-aware annotation for each word, the topic annotation $\overline{h^{to}}$ is concatenated to each word,

$$h_i = h_i^{tw} \parallel \overline{h^{to}}, h_i^j \in R^{4L}$$

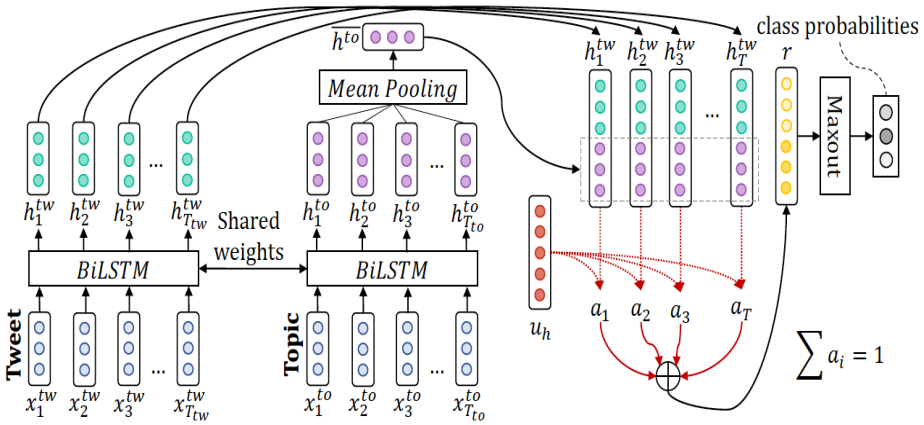

Figure 4. Siamese Bidirectional LSTM (Source: Baziotis et al, 2017)

The BiLSTM uses a context aware attention mechanism by adding the context vector u_h, to increase the contribution of words that would produce a better sentiment score for a given topic.

$$e_i = tanh(W_h h_i + b_h), e_i \in [-1, 1],$$

$$a_i = \frac{exp\,(e_i^T u_h)}{\sum_{t=1}^{T_{tw}} exp\,(e_i^T u_h)}, \sum_{t=1}^{T_{tw}} a_i = 1,$$

$$\gamma = \sum_{i=1}^{T} a_i h_i, \gamma \in R^{4L}$$

where W_h, b_h, and u_h are jointly learned weights. γ is passed through a dropout layer. This LSTM model used these as the hyperparameters: embedding layer dimension as 300, BiLSTM as 128, Gaussian noise σ as 0.2, dropout rate as 0.3 at the embedding layer, attention layer, dropout rate as 0.25 at LSTM layer, and l_2 regularization of 0.0001 at the loss function. Baziotis et al. (2017) used Gaussian noise at the embedding layer in both LSTM attention models to reduce the overfitting issues. They also added an l_2 regularization penalty to the loss function to discourage large weights and stop the training when validation loss value stopped declining.

In order to find the impact of the attention mechanisms, they evaluated the performance of each model, both with and without the attention layer. Research suggests that when the ability to store long-range information of a model is high with an additional attention layer, that model performs better at sentiment classification (Baziotis et al., 2017). In this dissertation, we performed the sentiment classification using an LSTM model with an attention layer, in addition to Kim's (2014) CNN model.

We additionally built the BiLSTM to be trained by the input tweets. These neural networks were constructed with sequential data by sharing their weights across the sequence. The input list of words was mapped to an embedding vector, via one of the pre-trained word vectors: word2vec, GloVe, and fastText. After the embedding layer, the sequence of words was passed through the BiLSTM to get to another hidden state. The hidden state h_t at time t was be computed as,

$$h_t = f(W_h \times x_t + U_h \times h_{t-1} + b_h)$$

where, x_t is the word embedding vector, weight matrix is $W_h \in R^{m \times d}$, and $U_h \in R^{m \times m}$, $b_h \in R^m$ is the bias term, and $f(x)$ is non-linear function (*tanh*).

In order to attribute all words in a tweet equally to people's understanding of the message context of a given tweet, we leveraged the use of an attention mechanism (Baziotis et al., 2017). In addition to the attention layer, a dense vector collected the weights of various word vectors, which is delineated in the following equations.

$$u_{ti} = tanh\,(Wh_{ti} + b)\,,$$

$$\alpha_{ti} = \frac{exp\,(u_{ti}^{T} u_w)}{\sum_{j=1}^{n} exp\,(u_{tj}^{T} u_w)},$$

$$s_t = \sum_{i=1}^{n} \alpha_{ti} h_{ti}$$

Specifically, t represents t^{th} tweet in the dataset, and n is the word count in a tweet. The h_{ti} represents the concatenation output of the BiLSTM layer. The term w represents the weight matrix of the neural network and b is the bias term of the multilayer perception (Baziotis et al., 2017). After that, we compared the u_{ti} and the word level context vector u_w using their similarity, and we measured the importance of each word in a tweet. The normalized importance weight α_{ti} was computed through a softmax function. When the weight is high for an α_{ti} that represents i^{th} word, it is the most important for sentiment classification. Finally, s_t represents a sentence vector, which is the weighted sum of word annotations.

The output of the attention layer was passed to a dense layer, and the activation functions for this layer were '*tanh*' or '*adam*'. After the dense layer, there was a dropout layer to avoid overfitting problems. The output of this dropout followed a dense layer of two hidden layers, using softmax as an activation function. That produced the sentiment label that belonged to one of the two classes.

2.3 Word Embedding

In natural language processing systems, words are expressed as discrete atomic symbols. As this encoding format treats each word independently, it does not provide additional information to the system when attempting to identify semantically similar words (Wehrmann, Becker, Cagnini, & Barros, 2017). Word embeddings are helpful because they encode both the syntactic and semantic information of words that should lie close in the embedded d-dimensional space. Word embeddings also consist of lists of words w in a d-dimensional space where semantically similar words are neighbors that could be generated using a dictionary.

Let $T \in \{w_1, w_2, \ldots, w_n\}$ be a text with n words, which will vary based on the text, and $f(w_i) = v_i$ be a mapping function that will map the word w_i into vector v_i. The word embedding space is defined by $T \in R^{n \times d}$. Figure 5 illustrates an example of the word-based representation of word embedding. Currently sentiment analysis classification tasks are performed through the network models such as CNN and LSTM, which process the text encoded as a sequence of word embeddings and produce a more promising result than raw word vectors (Kim, 2014).

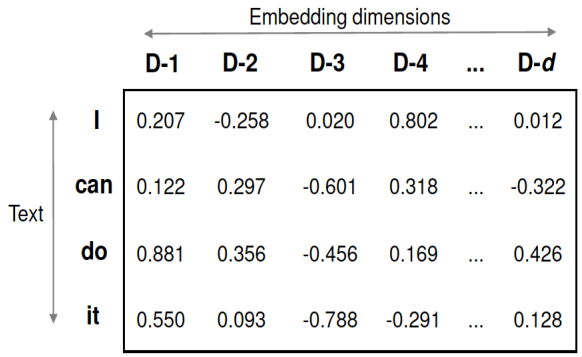

Figure 5. Word embedding for the words 'I can do it'. (Source: Wehrmann et al., 2017)

Word embedding methods are broadly classified into two categories, including

frequency-based and prediction-based embedding (Baroni et al., 2014). GloVe is an effective frequency-based tool that uses an unsupervised learning algorithm for obtaining vector representations for words (Baroni et al., 2014). word2vec is one of the computationally efficient predictive models for learning word embeddings from NLP. word2vec uses the Continuous Bag of Words (CBOW) or the Skip-gram model (Mikolov et al., 2014). Both of these models in word2vec are built with lightweight feedforward neural networks. FastText is an extension for word2vec that breaks words into several n-grams. In fastText, each word's word vector embedding will be the sum of all n-grams of that word. GloVe and fastText also supports CBOW and Skip-gram models for generating word embedding vectors.

2.3.1 Continuous Bag of Words (CBOW)

The CBOW model predicts target words from source context words i.e., in given a context, the goal is to know which word is the most likely to appear in it. In this model, all the words surrounding the target word when it is a target are fed into the networks and take the average of the extracted hidden layer. The input layer consists of a sequence of words $\{x_1, x_2, \dots, x_C\}$ as a one-hot encoding format, including C as the number of words, and V as the vocabulary size. The hidden layer is an N-dimensional vector. The input vectors are connected to the hidden layer through a $V \times N$ weight matrix w, and the hidden layer is attached to the output layer by a $N \times V$ weight matrix w'. Finally, the output layer has a list of words in the training dataset that is also one-hot encoded. The following figure describes the CBOW architecture.

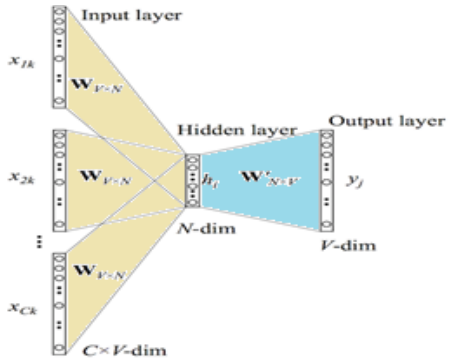

Figure 6. CBOW model

The output y_j is obtained by passing the input u_j through the softmax function.

$$y_j = p(w_{yj} \mid w_1, w_2, \dots, w_c)$$

$$= \frac{exp\ (u_j)}{\sum_{j=1}^{V} exp\ (u_j')}$$

In this equation, $u_j = {v'_{wj}}^T . h$, and h is the average of the input weighted by w, such that $\frac{1}{C} w . \sum_{i=1}^{C} x_i$ and ${v'_{wj}}^T$ is the j[th] column of w'. The weights w and w' are calculated via backpropagation.

2.3.2 Skip-gram

The Skip-gram model follows a similar topology to the CBOW model. This model's input is a target word, and the outputs are the lists of words surrounding the input word. Both input and output vectors are in the same dimension and in one-hot encoded format (Baroni et al., 2014). The Skip-Gram model architecture is presented in Figure 7. In this model, x represents the one-hot input vector size 1×N, which is connected to the hidden layer through an $V \times N$ weight matrix w, and the hidden layer is attached to the output layer by an $N \times V$ weight matrix w'. In the hidden layer, the i^{th} row represents the weights that correspond to the i^{th} word in the vocabulary vector V. The output one-hot vector C is 1 ×N.

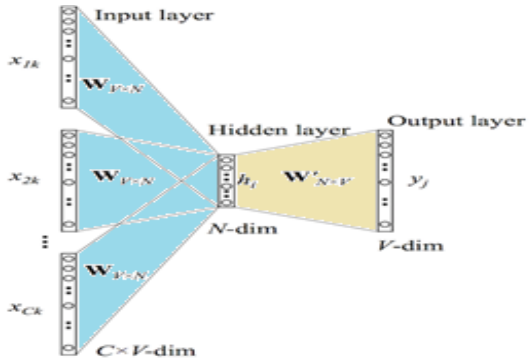

Figure 7. Skip-Gram model

The output of the j^{th} node of the c^{th} output word is obtained by passing the input u_{cj} through the soft-max function.

$$y_{cj} = p(w_{cj} = w_{0,c} | w_1)$$

$$= \frac{exp\ (u_{cj})}{\sum_{j=1}^{V} exp\ (u_j')}$$

In this equation, $u_{cj} = {v'_{wj}}^{T}.h$, h is the average of input weighted by w. i.e., $x^T w$, and ${v'_{wj}}^{T}$ is the j[th] column of the c^{th} output word. The weights w and w' are calculated with backpropagation and a stochastic gradient descent.

Since word embedding places the related words close together, it will be easy to interpret the tweet. In this research, this strategy was used to address all the errors in the tweets. In our study, we leveraged the list of pre-trained word embedding vectors including word2vec, GloVe, and fastText from open source code environments.

3 Methodology

The benchmark Sentiment140 English language dataset was used for evaluation in this dissertation. This dataset contained labeled tweets that included emoticons. Two types of deep neural networks – Convolution Neural Networks (CNN) and Long Short-Term Memory (LSTM) – were used for classification since they have been demonstrated to produce the best results. All terms in the tweets were represented using their word2vec or GloVe or fastText embeddings. Baseline models were trained and tested using tweets with their emoticons removed. For each baseline model, a corresponding model was trained that included emoticons as inputs. Alternate methods for obtaining embeddings for emoticons were explored. Accuracy, precision, recall, and F_1 scores of models using emoticons were compared to their corresponding baseline models that do not use emoticons.

The rest of this chapter is organized as follows: Section 3.1 summarizes the dataset. Section 3.2 describes text pre-processing procedures for the baseline models and the models with emoticons. Section 3.3 defines the evaluation measures used. Section 3.4 explains the embedding method using word2vec. The CNN model and the LSTM model are presented in sections 3.5 and 3.6, respectively. Section 3.7 outlines the method of comparing models trained with emoticons to the corresponding models trained without emoticons.

3.1 Data Set

In this research, we predicted the sentiment of twitter messages using the Sentiment140 English language dataset (Sentiment140, 2009). These Twitter datasets were limited to 140 characters of text and contained both transcripts and emoticons. The dataset

had six columns: polarity of the tweet, the id of the tweet, date of the tweet, the query of the user on api, the user that tweeted, and the text of the tweet. This study only included the polarity and text columns for classification. These datasets had 1.6 million rows and had been previously annotated with the three labels 4, 0, 2, which have been represented as positive, negative, and neutral, respectively (Go, Bhayani, & Huang, 2009). However, there was no sample data for the neutral label, so we do not consider that. Figure 8 illustrates the data distribution for the sentiment labels in this dataset. Both sentiment labels positive and negative were equally distributed in this dataset. Table 1 illustrates the same data for both labels from the dataset.

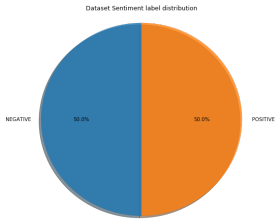

Figure 8. Sentiment label distribution – Raw dataset

Table 1.Sample data from the dataset

Sentiment	Text	Sentiment label
0	@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D	NEGATIVE
0	@tsarnick yay totally send me an e-mail! Cool I'm back at my appartment tomorrow so I'll have my laptop and my video software :)	NEGATIVE
0	@CaitlinOConnor i want tacos and margarhitas telll gay i say hello<3 Managed to save 50% The rest go out of bounds	NEGATIVE
4	editing my profile for the first time in forever ... Tommy's an uncle again ! <3	POSITIVE
4	@tsarnick yay totally send me an e-mail! Cool I'm back at my appartment tomorrow so I'll have my laptop and my video software :)	POSITIVE
4	@lauracowen hope you customised it with #ubuntuuk/#lugradio wallpapers and left podcasts on the desktop Never hurts to advertise ;)	POSITIVE

For this research, we considered only the rows that had at least one emoticon i.e., 131,523 rows were filtered from the sentiment140 dataset. This dataset had special characters that were transformed into an ascii format for understanding all the tweets clearly. The given dataset had hashtags, URLs, html tags, special characters, carriage return characters (\n), and line terminator characters (\t). We removed all these characters and did not consider them in our study. The filtered dataset's text column was copied into two more columns: one for a BASE_MODEL experiment and another one for a WITH_EMOTICON experiment. In the BASE_MODEL text column, all the emoticon characters were replaced with UNK. The sentiment140 dataset had the emoticons in the form of icons, instead of emojis, including: :-), :), :-], :], :-3, etc. Table 2 describes the list of ninety emoticons in the WITH_EMOTICON experiment text column alongside the number of occurrences across the tweets. Most of the tweets regularly used by most of the users had smiley faces, eyes crossed and tongue sticking out, thumbs up, and heart emoticons.

Table 2. Emoticons in the tweets with the number of occurrences

Emoticon	Occurs	Emoticon	Occurs	Emoticon	Occurs	Emoticon	Occurs	Emoticon	Occurs	Emoticon	Occurs
:/	74497	=D	871	:')	205	:c	74	=3	21	>:3	5
<3	13846	;D	785	:$	188	:b	73	:-}	21	:-]	5
xp	12011	:L	714	:>	180	:<	71	O-O	19	<pout>	2
;)	7516	:o	652	>_<	165	*)	70	:{	18	:-3	2
:3	4410	=/	522	;]	160	8D	63	v.v	14	o/\\o	1
xD	2848	=]	485	=p	146	8-)	54	>:/	14	^<_<	1
XD	1879	>.<	414	DX	122	:}	47	:(	13	><>	1
:p	1756	:\\	412	o_O	117	D:<	36	=L	12	=)	1
:|	1680	:-/	404	\\o/	103	>:O	35	*-)	11	;^)	1
:O	1659	:@	383	:X	103	o_o	34	^5	10	:D	1
":("	1464	XP	382	=\\	101	D=	34	:->	9		
:S	1398	</3	335	O_O	100	:^)	34	>;)	7		
d:	1355	0:3	317	O_o	87	%)	34	:)	7		
'D:'	1240	8)	299	:o)	83	>:[	28	:&	7		
';-)'	1122	:*	287	:-*	78	:-0	23	>:\\	6		
':]'	1041	:[	213	D;	76	D8	22	:#	6		

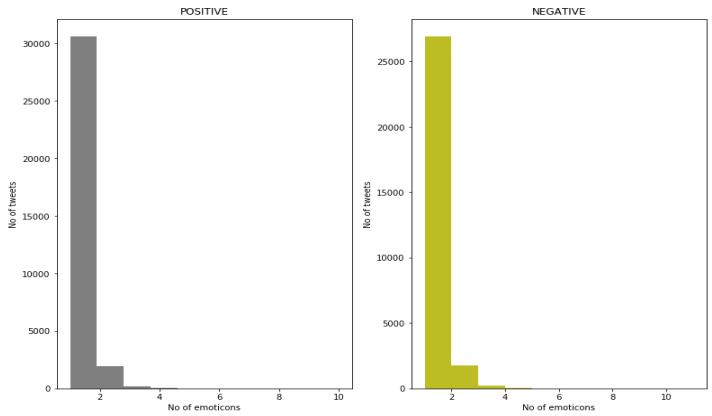

Figure 9. Number of emoticons distribution by Sentiment class

The WITH_EMOTICON text column were prepared using a list of English language emoticons mentioned in Wikipedia (Wikipedia, n.d.), in which each emoticon icon was replaced with the appropriate text. For instance, ':-)' was replaced with 'happy face,' ';-)' was replaced with 'happy face,' ':-(' was replaced with 'angry,' etc. Figure 9 delineates the number of emoticons distributed across the sentiment labels in the WITH_EMOTICON text column. More than 90% of tweets had one or two emoticons, and only a few tweets had more than two emoticons.

The number of rows in each of the sentiment labels POSITIVE and NEGATIVE in both datasets is reported in Table 3. Figure 10 explains the difference between the two datasets' tweets' length distribution in sentiment class. On the right side, the two graphs show the higher than default maximum tweet length of 140. Additionally, the WITH_EMOTICON dataset describes the minimum tweet length, but overall tweet sizes are greater than the default twitter message.

Table 3. No of rows in BASE_MODEL, and WITH_EMOTICON dataset by sentiment class

	POSITIVE	**NEGATIVE**
WITH_EMOTICON	78,529	52,994
BASE_MODEL	78,529	52,994

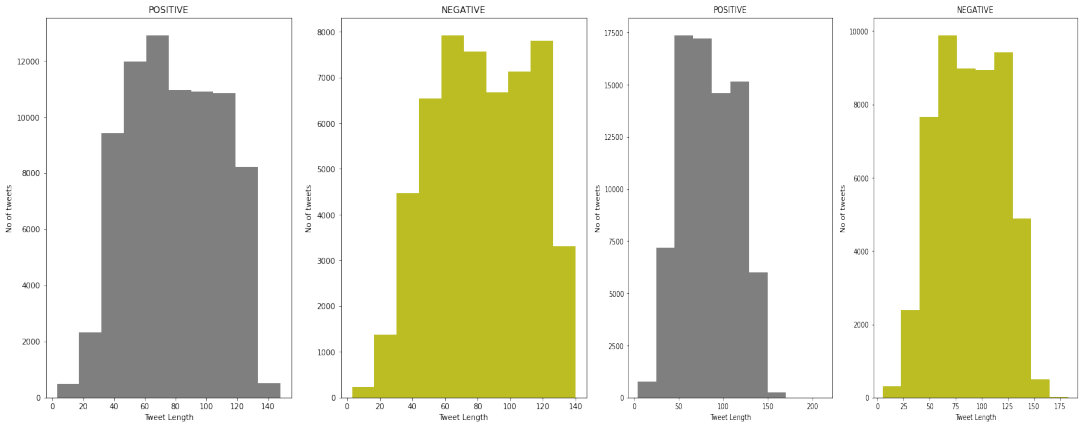

Figure 10. Tweets length distribution- Left: BASE_MODEL, Right: WITH_EMOTICON

Since new text was added in the WITH_EMOTICON text column, we expected an increase in the number of words in the raw dataset tweets. Figure 11 reports word length distribution in these two text columns, with the overall number of words in WITH_EMOTICON being larger than what is in the BASE_MODEL text column. We split a randomized sample method through the datasets, 80:10:10, into two sets of training test, and validation datasets. One of the training and test datasets had the BASE_MODEL text column and another set of training and test datasets had the WITH_EMOTICON text column. Both training datasets had 106,533 rows, validation datasets had 13,153 rows, and the test datasets had 11,837 rows.

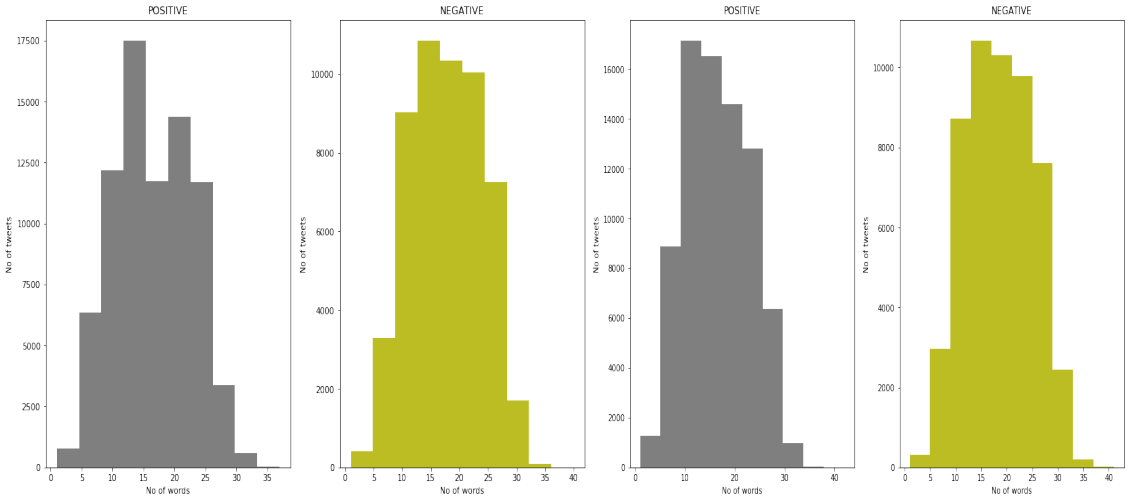

Figure 11. Word length distribution- Left: BASE_MODEL, Right: WITH_EMOTICON

3.2 Data preprocessing

The raw tweets had so many meaningless and unstructured data and repetitive words. Hence, each experiment followed a few preprocessing steps in order to make some changes to tweet transcripts so as to conduct further text analysis. In the previous section, we defined a few preprocessing steps to copy the filtered dataset text column into two further columns. Besides that, datasets went through the following preprocessing steps:

1) convert the text to lower case
2) removal of all nullable rows
3) removal of all additional empty spaces
4) removal of all numbers, alphanumeric and punctuations

We also removed and replaced all the stop words and stem words from the dataset using the NLTK (Natural Language Toolkit) English language dictionary, because such words have low predictive power (Rosenthal et al., 2017). For handling the stem and stop words, initially we split each tweet into a list of words using an NLTK tokenizer tool. If any of the tokens belonged to the NLTK stem's bag of words, the appropriate stem word from the NLTK stem was replaced for that token, and the token that was in NLTK corpus's stop words were removed from the tweet. Finally, we used NLTK wordnet lemmatization on the tweets, which linked words with similar meanings to one word. The lemmatization process used the bag of words from the tokenizer and mapped the appropriate word, based on the sentence. Subsequently, it excluded all the rows from the dataset where the tweet length was less than one.

3.3 Evaluation criteria

The primary evaluation criteria for this sentiment analysis compared the model

accuracy of the BASE_MODEL and the WITH_EMOTICON experiments. To evaluate the sentiment classification, model accuracy was measured using a confusion matrix, accuracy, precision, recall, and F_1 score metrics. The confusion matrix explains the distribution of predicted values across actual sentiment classes. Accuracy is the measure of all the correctly identified classes against the actual class. Precision is the fraction of the relevant instances among the predicted instances. Recall is the fraction of the desired instances that has been predicted over the total amount of desired instances. The F_1 score is defined as the weighted average of recall and precision. The following equations describe the accuracy (Acc), recall (Rec), precision (Pre), F_1, and other metrics from confusion matrix:

$$Acc_i = \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i},\ Acc = \frac{\sum_{i=0}^{n} TP_i + \sum_{i=0}^{n} TN_i}{\sum_{i=0}^{n} TP_i + \sum_{i=0}^{n} TN_i + \sum_{i=0}^{n} FP_i + \sum_{i=0}^{n} FN_i}$$

$$\text{Rec}_i = \frac{TP_i}{TP_i + FN_i}, Rec = \frac{\sum_{i=0}^{n} TP_i}{\sum_{i=0}^{n} TP_i + \sum_{i=0}^{n} FN_i}$$

$$\text{Pre}_i = \frac{TP_i}{TP_i + FP_i}, Pre = \frac{\sum_{i=0}^{n} TP_i}{\sum_{i=0}^{n} TP_i + \sum_{i=0}^{n} FP_i}$$

$$F_1 = \frac{2 \times Pre \times Rec}{Pre + Rec}$$

whereby Acc_i, Rec_i, and Pre_i are the accuracy, recall, and precision for the class i respectively, n is the number of classes in the sentiment classification, TP – True Positive, FP – False Positive, FN – False Negative, and TN – True Negative. TP_i is the number of examples in class i that are predicted in class i. TN_i is the number of examples NOT in class i that are NOT predicted in class i. FP_i is the number of examples NOT in class i that are predicted in class i. FN_i is the number of examples in class i that are NOT predicted class in i. In addition to these metrics, we will also evaluate the classification output using

a macro-averaging ($Macro_{avg}$) metric to determine the accuracy of the model. As the following equation illustrates, macro-averaging was the average of the precision and recall of the system on different sets. The evaluation process measured the impact of including emoticons on the sentiment classification task.

$$Macro\_Pre_{avg} = \frac{Pre_1 + \cdots + Pre_k}{k}$$

$$Macro\_Rec_{avg} = \frac{Rec_1 + \cdots + Rec_k}{k}$$

3.4 Base-line models

As Rosenthal et al. (2017) demonstrated, CNNs with multiple convolution operations are good for classifying sentiments. Baziotis et al. (2017) demonstrated how a BiLSTM with an attention layer yields better accuracy for sentiment classification. Within both experiments of our research, these models were employed to classify the sentiment as POSITIVE or NEGATIVE. Finally, the accuracy and performance of each method in the experiment was analyzed and the importance of emoticons in the sentiment analysis process was evaluated. The inclusion of emoticons resulted in a higher level of accuracy since positive or emotional words were included from the emoticons. Since Rosenthal et al. (2017) and Baziotis et al. (2017) used different datasets than our research dataset, we had to use a linear model as a base model for evaluating all the deep learning sentiment classification models.

3.5 Embedding Layer

Before building a sentiment analysis model, all the features contained in the tweets needed to be extracted and formed into a group of semantically related words through word

embedding. This study used a few different embedding approaches for our neutral network models, including custom word2vec embedding vectors, pre-trained word2vec embedding vectors, pre-trained GloVe embedding vectors, and pre-trained fastText embedding vectors. These embedding vectors were fed to an embedding layer.

3.5.1 Custom word2vec model

In order to construct word2vec embedding vectors through the given tweets, we used the Genism package from python. This API requires the input data to be in a list of sentences, with each sentence being a list of words meant for building the embedding vectors. Initially, the training dataset tweet messages were transformed into the format of a list of words. This model was defined by providing the number of dimensions (size), the maximum distance between a target word and words around the target word (window), it included or excluded terms based on their frequency (min_count), and the number of threads (workers). We determined the size of the vocabulary using the list of words in the training dataset's text column (vocabulary size). We then trained the word2vec model with the list of words, with a few iterations (epochs). Table 4 describes the list of parameters used in the word2vec model.

Table 4. Word2vec model hyperparameters

Parameter name	Value(s)
Dimensionality of the word vectors (size)	100-300
Maximum distance between the current and predicted word within the distance (window)	5-15
Total frequency threshold (min_count)	5-15
No of worker threads to train the model (workers)	8-10
Training iterations (epochs)	16-50
Maximum padding length	150-300

3.5.2 Pre-trained word2vec model

We used Google's public pre-trained word2vec embedding vector from an (, n.d.) available repository. It has 300 dimensions of pre-trained word vectors that have been trained using 100 billion words from a Google News dataset. This vector has the vocabulary size of Google news, is around 3 million words, and was transformed into a dictionary with a word as a key and coefficients as values for the embedding layer.

3.5.3 Pre-trained GloVe model

We used another pre-trained embedding GloVe (Global Vectors for Word representation) for our sentiment classification from the Stanford NLP public data repository. It is 200 dimensions of pre-trained word vectors (GloVe, n.d.) that were built using 2 billion tweets, 27 billion words, and has a vocabulary size of 1.2 million. This vector was transformed into a dictionary with a word as a key and coefficients as values.

3.5.4 Pre-train fastText model

This study tried one more pre-trained embedding of fastText for our sentiment classification from the Facebook opensource (FastText, n.d.) environment. It is 300 dimensions of pre-trained word vectors that were trained on Common Crawl and Wikipedia. These models were trained using CBOW, which has 300 dimensions, a window of size 5, character n-grams of length 5, position-weights, and 10 negatives. This vector was transformed into a dictionary with a word as a key and coefficients as values.

In our next step, we prepared the embedding layer through the above four embedding vectors. We leveraged the Keras Tokenizer function that could be fit onto the training dataset tweets (Keras, n.d.c.), could transform text to sequences consistently by calling the texts_to_sequences method on the Tokenizer class, and delivered access to the dictionary by the mapping of words to integers in a word_index attribute. Next, we created a matrix

of one embedding for each word in the training dataset text. We did that by enumerating all unique words in the Tokenizer.word_index function and identifying the embedding weight vector from the embedding vector, which was made in the initial proceedings.

Consequently, we created an embedding layer with the embedding matrix through the Keras embedding method (Keras, n.d.d.), which was then seeded with the specific word vectors' embedding weights. The output dimension was set based on the embedding vector dimension size. For instance, the pre-trained word2vec embedding layer output dimension was 300 since this embedding vector size was 300. The trainable attribute was set as false because we did not want it to update its learned word weights in this model. This vector representation of words was used as input data for the CNN and LSTM models, which was used, in turn, for sentiment classification.

3.6 CNN Model

For sentiment polarity, we implemented a model similar to Kim's (2014) CNN model. We made a few changes to Kim's (2014) CNN model by fine-tuning the parameters and building two types of CNNs: a simple CNN model and a multichannel CNN model. Each model worked with all the four embedding vectors; hence we had 8 different CNN variants of models for sentiment classification.

3.6.1 Simple CNN

We developed a sequential CNN model using a Keras API, as it has a linear stack of layers, which is described in Figure 12. The training and test dataset tweet messages were the same length across all the rows for the CNN model. The maximum length of the document was maintained through the max_length parameter. The original tweets' max_length value was 140 since the maximum length did not exceed 140 characters for

the BASE_MODEL experiment, while it was 200 for the WITH_EMOTICON experiment. The maximum length on these two datasets was controlled by the Keras Tokenizer and the pad_sequences functions. The Tokenizer vectorized the tweets and converted them into a sequence of integers and then we restricted the tokenizer to using only the topmost common vocabulary words. The number of topmost vocabulary words were managed through a hyperparameter in this model. The padding functionality padded the sequences to the maximum length by adding 0 values in the end.

The first layer in this CNN sequence was an embedding layer, which we were built in the previous embedding layer section. The embedding layer passed the output to a 1D convolution layer, which created a convolution kernel that was convolved with the layer input over a single spatial dimension in order to produce a tensor of outputs (Keras, n.d.b.). This conv1D layer produced the output matrix through the actions of a few parameters, such as the number of output filters in the convolution (filters), length of the convolution windows (kernel_size), activation function (activation), and the penalty for the loss function via regularization method (kernel_regularizer). This study chose the hyperparameters' values from Table 5. We used the filters that have the range of 140-200, kernel_size of 5, activation as '*relu*' or '*tanh*', and kernel_regularizer as l_2 or l_1 with 0.001 weight for this convolution layer.

The output of the convolution layer was received by a maximum pooling layer that occurred via the Keras MaxPooling1D function (Keras, n.d.e.). This study set the maximum pooling size as two, i.e., the pooling operation that calculated the two largest values in each patch of each feature map. If the dimension size was set as 140 in the previous layer, the pooling operation returned the same output dimension. This matrix

passed to a dropout layer, and that randomly assigned 0 weights to the neurons in the network. If the dropout rate was 0.5, 50% of the neurons received a zero weight. This dropout operation made the network become less sensitive when reacting to smaller variations in the tweet. That further increased the model's accuracy on unseen data. The output dimension of the dropout layer still had the same dimensions of the conv1d layer.

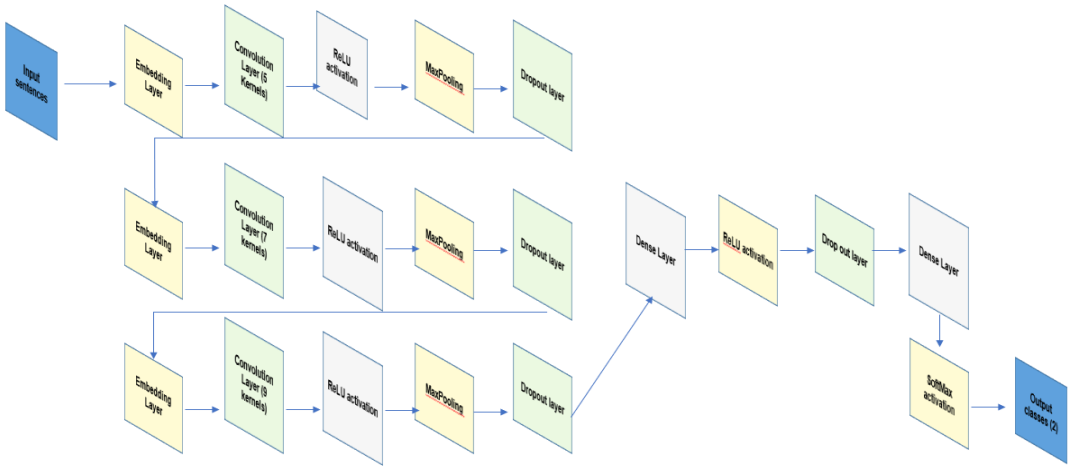

Figure 12. Simple CNN model

The results of the dropout layer were fed to another convolution layer (convolution 2) with a kernel_size of 7. That was passed to a maximum pooling layer, followed by a dropout layer, as successors of convolution 2. From there, the convolution 2 was passed to a third convolution layer with a kernel_size of 8. This convolution 3 layer was fed to a global maximum pooling layer, which was defined by the Keras function GlobalMaxPooling1D (Keras, n.d.e.). The global maximum pooling layer was only taking the max vector of each input word. This pooling layer was passed to a dropout layer with a dropout rate of 0.5.

The third dropout layer was fed into a dense layer with n number of hidden layers, as well as '*relu*' or '*tanh*' as an activation function and a regularization function to avoid overfitting issues. This layer was defined by the Keras dense function (Keras, n.d.f.). The

output dimensionality of the dense layer was based on the number of the hidden layer's hyperparameters. For instance, if the hidden layer was set as 32 and the input dimension was set as 150, the output dimension became 32. The same regularization method used previously was a parameter for this dense layer, which then was passed to a dropout layer with the same dropout rate. This dropout layer dimension was the same as its parent layer.

Consequently, the dropout layer passed the output through a softmax layer for producing the expected sentiment analysis class as an encoded format of [0, 4]. This softmax layer was a dense layer that used the number of output classes (2) as hidden layer units, and '*softmax*' as activation function parameters. The final dense layer reduced the number of dimensions to two. The training time for the simple CNN model with custom word2vec and pre-trained word2vec models was around 20 to 40 minutes in Google's colab environment with a GPU runtime.

3.6.2 Multichannel CNN

Figure 13 depicts the multichannel CNN model that was used in this dissertation. This model had three channels; each channel had an input layer in which the dimension was the same as the maximum length parameter of the model. This input layer defined the shape of the input vector. The input layer was passed to an embedding layer, a convolution 1-dimension layer, a global maximum pooling layer, and a dropout layer. The channels' convolution layers' kernel_size values were 5, 7, and 8, respectively. All three channels' dropout layer output vectors were concatenated and fed to a dense layer with n number of hidden layers, '*relu*' or '*tanh*' as an activation function, and a regularization function. This layer output was passed to a dropout and softmax layer as established in the previous CNN model.

These two CNN models were compiled using '*categorical_crossentropy*' as a loss

function and '*adagrad*' or '*adam*' as an optimizer through the Keras compile function (Keras, n.d.g.). After that, these models were trained with the number of iterations (epochs) and the number of samples per epoch (batch_size), which happened through the Keras fit function (Keras, n.d.h.). Table 5 describes a few hyperparameters that allowed for the CNN to tune the accuracy of the models.

Table 5. CNN Model hyperparameters

Parameter name	Value(s)
Maximum number of words (restricted for the same length)	30-42
Number of classes for output layer	2
Size of the vocabulary	25,000-30,000
Output dimensionality	140-200
Size of the kernel	3-8
Activation functions	relu, tanh
Batch size	16, 32,64
Pooling types	2, maximum
Number of epochs (number of cycles)	20-50
Dropout rate probability	0.2-0.5
Optimizer	adagrad, adam
Loss function	Categorical crossentropy
Number hidden nodes	16-64

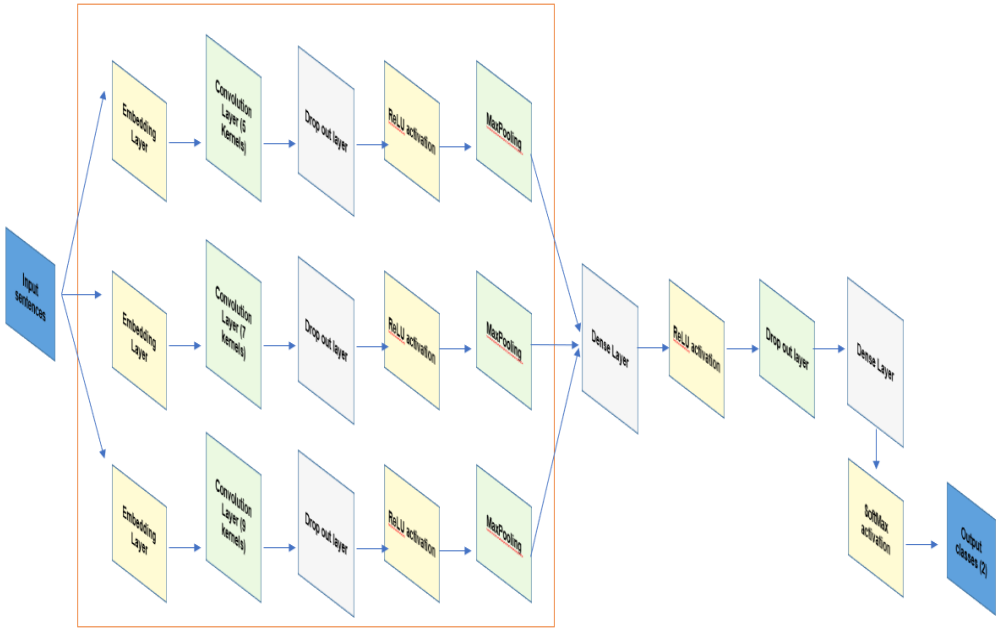

Figure 13. Multichannel CNN model

The variety of simple and multichannel CNN models are illustrated in Table 6. The overall training time for the multichannel CNN model with custom word2vec and pre-trained word2vec models is around 30 to 60 minutes in Google's colab environment with a GPU runtime.

Table 6. Variety of CNN models

CNN Model Name	Custom word2vec	Pre-trained word2vec	Pre-trained GloVe	Pre-trained fastText
Simple CNN	√	√	√	√
Multichannel CNN	√	√	√	√

3.7 LSTM Model

As discussed earlier, Long Short-Term Memory (LSTM) units were built inside of RNNs that encapsulate information about long-term dependencies in the text. The study followed Cliché's (2017) BiLSTM model architecture, depicted in Figure 14. This model builds two LSTM units to train two LSTMs on the input tweets. The reversed copy of the first LSTM layer should be a second LSTM. These neural networks are constructed with sequential data by sharing their weights across the sequence. This mechanism was maintained through the past and future features for a certain time. The length of input training and test dataset tweet messages for this LSTM model were normalized across all the tweets. The normalizing of the input text and encoding of the labels' strategy was the same as the CNN model which was explained in the previous section.

The LSTM network is a linear stack of layers that was built using the Keras API. The first layer was an embedding layer, which encoded the input sequence into a sequence of dense vectors of the embedding dimension. The embedding layer was chosen from one of the embedding models that have been explained in section 3.5. The embedding layer output

was connected to a LSTM layer that was derived from the Keras LSTM function (Keras, n.d.i.). The LSTM transformed the vector sequence into a single vector, based on the output dimension size parameter that contained information about the entire sequence. In addition to the output dimension size, this method was used by the dropout hyperparameter as it contained a fraction of the units that were dropped for the linear transformation of the inputs, as well as the recurrent_dropout hyperparameter for maintaining the dropout rate in the recurrent state. The Bidirectional LSTM layer was implemented via the Keras (Keras, n.d.a.) Bidirectional layer wrapper function.

Normally, all words in a tweet do not contribute equally to people's understanding of the message context of a given tweet. Hence, an attention mechanism, which ignored emoticons in tweets, captured the words (Baziotis et al., 2017). After the attention layer, a dense vector collected the weights of various word vectors. This LSTM layer had an attention layer that received input from the BiLSTM. The attention layer was defined from the work of Yang et al., (2016) which had the sentiment analysis (Keras, n.d.j.) with a BiLSTM model with an attention mechanism.

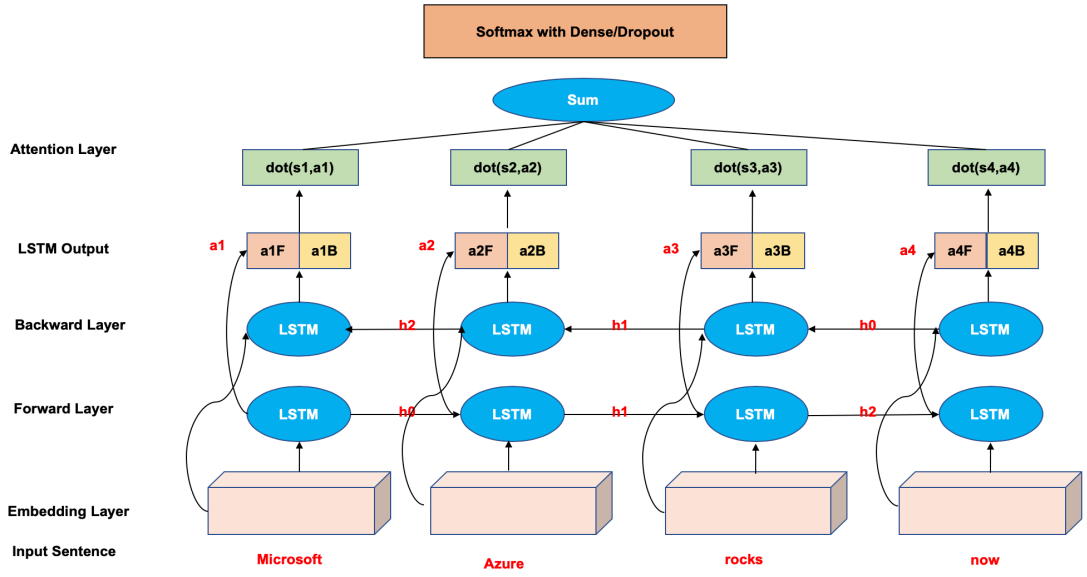

Figure 14. BiLSTM model with attention layer

The output of the attention layer was passed to a dense layer with n number of hidden neurons that were configured in the hyperparameter, and the activation function for this layer was '*tanh'* or '*relu*'. The dense layer output was fed to a dropout layer with the dropout rate used across the model to avoid overfitting problems. The output of this dropout followed a dense layer of 2 hidden layers, using '*softmax*' as an activation function. That produced the sentiment label that belonged to one of the two classes. This study adjusted a few parameters for the BiLSTM model in order to tune the accuracy of the model, as shown in Table 7.

Table 7. LSTM model hyperparameters

Parameter name	Value(s)
Maximum number of words	30-42
Number of classes for output layer	2
Size of the vocabulary	25,000-30,000
Output dimensionality for LSTM	140-200
Activation functions	relu, tanh
Batch size	16, 32,64
Number of epochs	20-50
Dropout keep probability	0.2-0.5
Optimizer	Adagard, adam
Loss function	Categorical crossentropy
Number of hidden nodes	32-256
Number of nodes in Dense Layer	32-256

This model needed to tune the parameters or add additional channels to achieve a better performance. The training time for the BiLSTM model with the pre-trained word2vec model is around 2 to 2.5 hours in Google's colab environment with a GPU runtime. We created another BiLSTM model without the attention layer for sentiment classification. We tested both experiments with the variety of BiLSTM models, such as with an attention layer and without an attention layer, along with four different embedding models, which are illustrated in Table 8.

Table 8. Variety of LSTM models

LSTM Model Name	Custom word2vec	Pre-trained word2vec	Pre-trained GloVe	Pre-trained fastText
BiLSTM	√	√	√	√
BiLSTM with attention layer	√	√	√	√

3.8 Comparison of extended approaches to baseline models

In this research, we trained the models using a training dataset for both experiments separately. First, we tested the BASE_MODEL with the deep learning models and evaluated its performance through the list of evaluation metrics. Next, we compared the same process for the WITH_EMOTICON experiment and evaluated it against the BASE_MODEL's performance metrics to address the emoticons' impact on the sentiment classification. We then repeated the same process by adjusting the LSTM and CNN models' parameters and evaluating their performances. In addition, we measured the training time for each of those models.

4 Results

The goal of this dissertation was to develop deep learning models for sentiment analysis and to investigate whether inclusion of emoticons as features improves classification accuracy. This chapter presents the results of the models, the validation of the final model, and the comparison between the base model and with emoticon experiments. The rest of this chapter is structured as follows: Section 4.1 explains the various embedding vectors used inside the deep neural networks. Section 4.2 reviews the linear model's results for the emoticon and base model experiments. Section 4.3 summarizes the CNN model's results for the emoticon and base model experiments. Section 4.4 describes the LSTM model's results for both experiments. Section 4.5 validates all the model results using the validation metrics. Section 4.6 compares all the model results for the BASE_MODEL and WITH_EMOTICON datasets. Section 4.7 selects the best model for both datasets. Section 4.8 summarizes the model results.

4.1 Embedding vectors

The following five different embedding vectors have been used in all the CNN and LSTM models.

4.1.1 Custom word2vec

In this model, we initially trained the word2vec embedding for both datasets. The word2vec model was built using a genism python package word2vec function. We set the following hyperparameters in the word2vec training function: a dimensionality size as 200, the maximum distance between the current and predicted words as 8, ignoring all the words with a minimum word frequency count as 10, the number of iterations for training as 32, and the number of threads for training as 8. This model produced a vocabulary around the

size of 9,116 for both datasets. These word2vec models were used for creating the embedding layers for deep neural network models.

4.1.2 Pre-trained word2vec – Google news

We downloaded the pre-trained word2vec embedding model using Google news from https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz for building the embedding layers. It has 300 dimensions and a 3,000,000 vocabulary size of embedding vectors.

4.1.3 Pre-trained word2vec – Twitter data

We also used another pre-trained word2vec embedding model using twitter data from https://drive.google.com/uc?id=1lw5Hr6Xw0G0bMT1ZllrtMqEgCTrM7dzc&export=download for building the embedding layers. It has 400 dimensions and a 3,039,345 vocabulary size of embedding vectors.

4.1.4 Pre-trained GloVe

Next, we used Stanford's GloVe pre-trained embedding model from http://nlp.stanford.edu/data/Glove.twitter.27B.zip for building the embedding layers. This model was built by the Stanford NLP lab using Twitter data and has 200 dimensions, as well as a 1,193,514 vocabulary size of embedding vectors.

4.1.5 Pre-trained fastText

Subsequently, we leveraged Facebook's fastText pre-trained embedding model from https://dl.fbaipublicfiles.com/fasttext/vectors-english/crawl-300d-2M.vec.zip for creating the embedding layers. This model was built by Facebook AI lab using a wiki and it has 300 dimensions as well as a 2,000,000 vocabulary size of embedding vectors.

4.2 Linear model results

We tokenized the BASE_MODEL and WITH_EMOTICON datasets using the BERT

tokenizer, which then produced around 118,370 tokens in each training dataset, and 13,153 tokens in each test dataset. The input text was padded with the maximum length of the input text and varied in all the datasets. The BASE_MODEL experiment maximum length of training and test datasets were 64 and 51 respectively, and the WITH_EMOTICON experiment maximum length of training and test datasets were 65 and 49 respectively. We then trained a few linear models, such as logististic regression, stochastic gradient descent classifiers (SGD), ridge classifiers, and perceptron classifiers, using training datasets for both experiments. The logistic regression model performed better than other linear models for both training datasets. We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 80.3% and 80.7% respectively. Table 9 and 10 describe the logistic regression confusion matrix, precision, and recall for the test dataset. We considered this our base model for evaluating the deep learning models.

Table 9. Logistic Regression - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N=**13153	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6798	995	6812	981	**7793**
	Negative	1602	3758	1555	3805	**5360**
		8400	**4753**	**8367**	**4786**	

Table 10. Logistic Regression - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.81	0.87	0.84	7793	0.81	0.87	0.84	7793
negative	0.79	0.70	0.74	5360	0.80	0.71	0.75	5360
macro average	0.80	0.79	0.79	13153	0.80	0.79	0.80	13153
weighted average	0.80	0.80	0.80	13153	0.81	0.81	0.81	13153

4.3 CNN model results

A variety of simple and multichannel CNN models were tried for Sentiment classification. The CNN models were trained and made predictions using four different embedding vectors for the BASE_MODEL and WITH_EMOTICON datasets with the hyperparameters mentioned in Table 5. We adjusted a few hyperparameters in the CNN models based on the execution time and model performance of each dataset. The rest of the sections will go through each of the model's outputs.

4.3.1 CNN with custom word2vec

We tokenized both datasets using the Keras tokenizer, which then produced around 112,069 words in each dataset. The input text was padded with the maximum length of the input text and varied in both datasets. The BASE_MODEL experiment maximum length was 40 and the WITH_EMOTICON maximum length was 42. The rest of the CNN and LSTM models used these steps initially before creating the embedding layer. Next, we made an embedding layer using the custom word2vec vector. The simple CNN model was fitted using both training datasets. We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 15 and Figure 16 illustrate these metrics for this model. While fitting the models, if the validation dataset accuracy did not improve, we stopped the iterations through early stopping parameters we had devised. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 77.22% and 78.42% respectively. Table 9 and 10 describe the confusion matrix, precision, and recall for the test dataset.

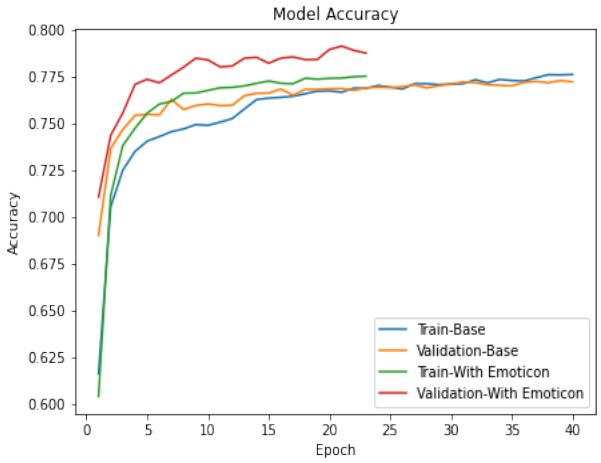

Figure 15. CNN with custom word2vec – accuracy

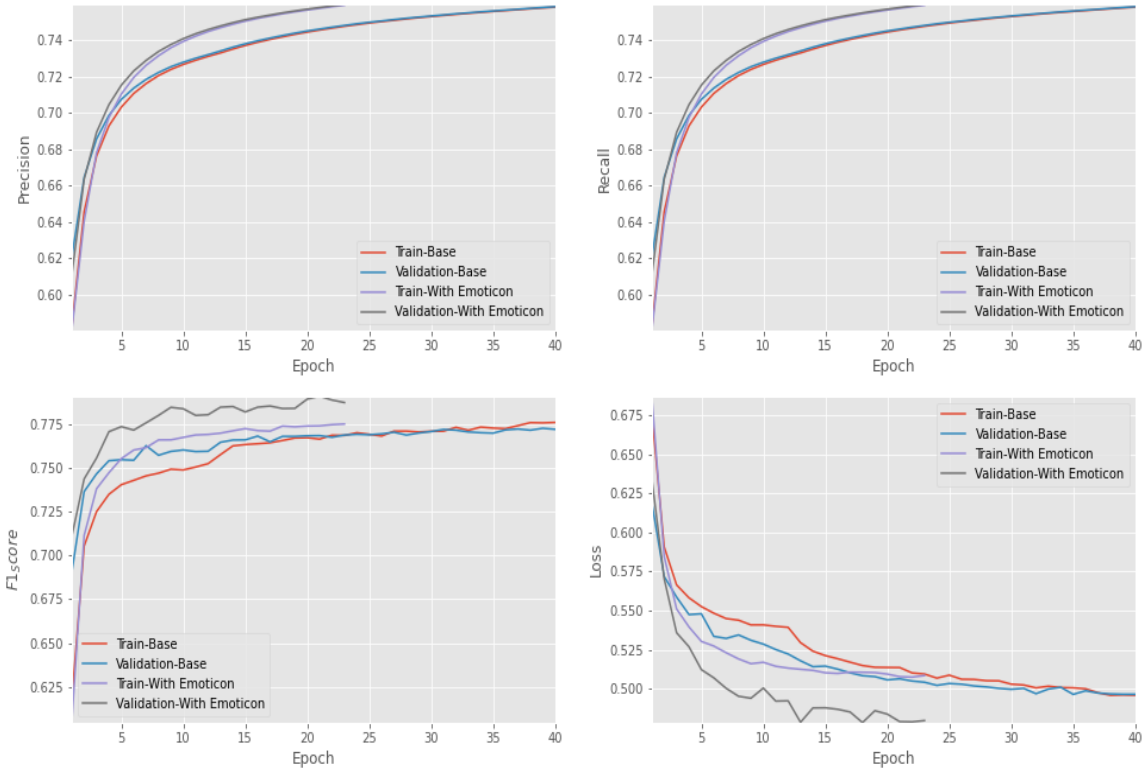

Figure 16. CNN with custom word2vec - precision, recall, F_1 score and loss

Table 11. CNN with custom word2vec - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N**=13153	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6605	1188	6565	1228	**7793**
	Negative	1808	3552	1610	3750	**5360**
		8413	**4740**	**8175**	**4978**	

Table 12. CNN with custom word2vec - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.79	0.85	0.82	7793	0.80	0.84	0.82	7793
negative	0.75	0.66	0.70	5360	0.75	0.70	0.73	5360
macro average	0.77	0.76	0.76	13153	0.78	0.77	0.77	13153
weighted average	0.77	0.77	0.77	13153	0.78	0.78	0.78	13153

4.3.2 CNN with pre-trained Google word2vec

In this model, we made an embedding layer using the pre-trained Google word2vec vector for the CNN model. We fitted this model using both training datasets. We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 17 and Figure 18 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 76.21% and 77.56% respectively. Table 13 and 14 describe the confusion matrix, precision, and recall for the test dataset.

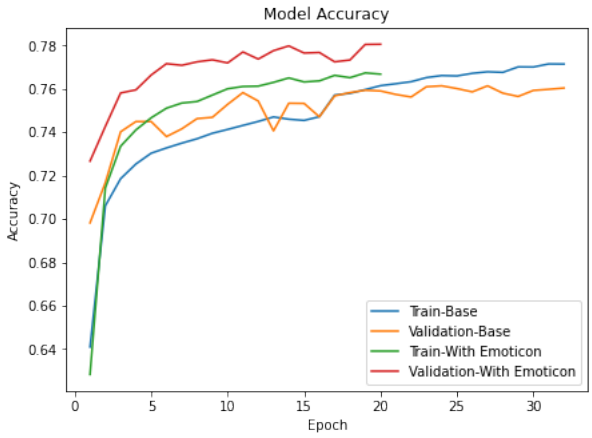

Figure 17. CNN with pre-trained Google word2vec - accuracy

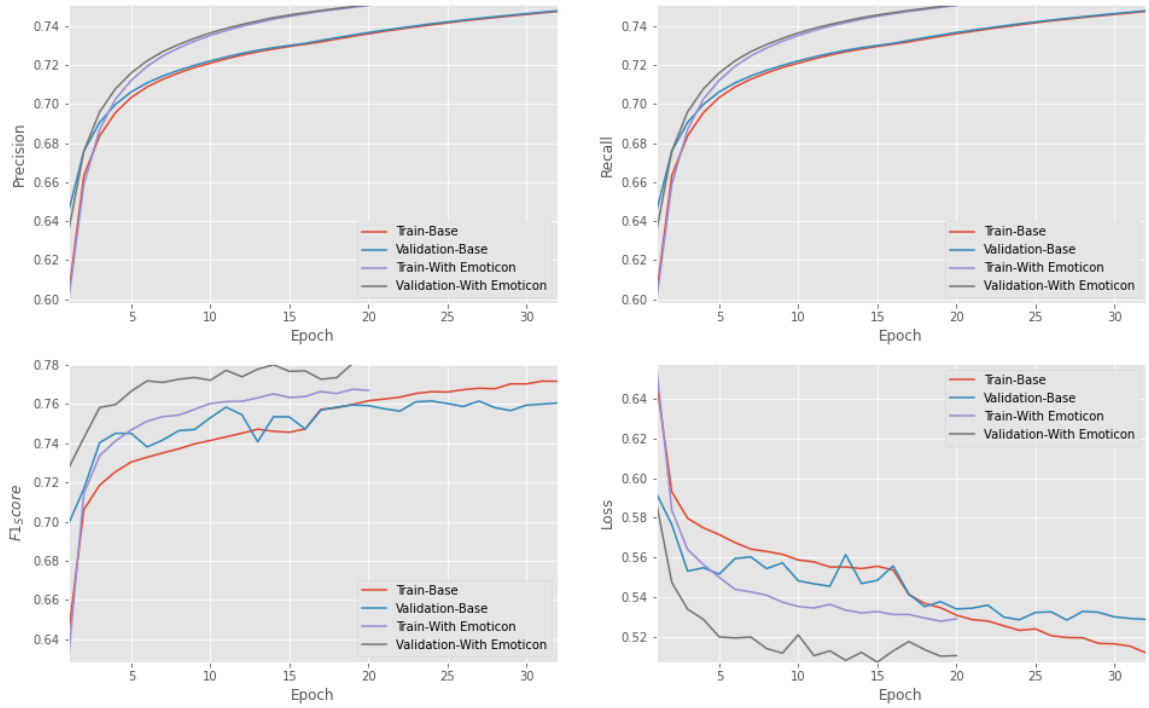

Figure 18. CNN with pre-trained Google word2vec - precision, recall, F_1 score and loss

Table 13. CNN with pre-trained Google word2vec - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N**=13153	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6752	1041	6531	1262	**7793**
	Negative	2088	3272	1689	3671	**5360**
		8840	**4313**	**8220**	**4933**	

Table 14. CNN with pre-trained Google word2vec - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.76	0.87	0.81	7793	0.79	0.84	0.82	7793
negative	0.76	0.61	0.68	5360	0.74	0.68	0.71	5360
macro average	0.76	0.74	0.74	13153	0.77	0.76	0.76	13153
weighted average	0.76	0.76	0.76	13153	0.77	0.78	0.77	13153

4.3.3 CNN with pre-trained Twitter word2vec

In this model, we made an embedding layer using the pre-trained twitter-based word2vec vector for the CNN model. We fitted this model using both training datasets. We

evaluated the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 19 and Figure 20 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 77.23% and 79.67% respectively. Table 15 and 16 describe the confusion matrix, precision, and recall for the test dataset.

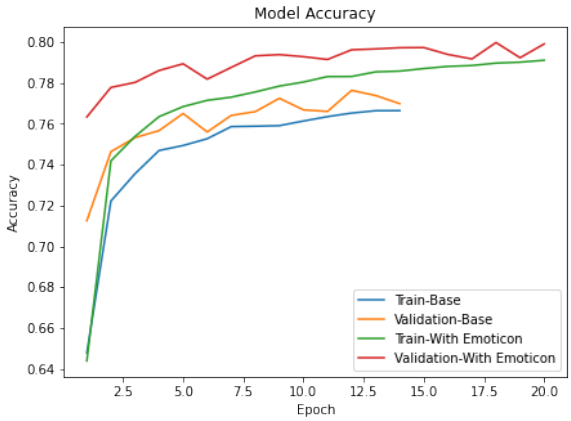

Figure 19. CNN with pre-trained twitter-based word2vec – accuracy

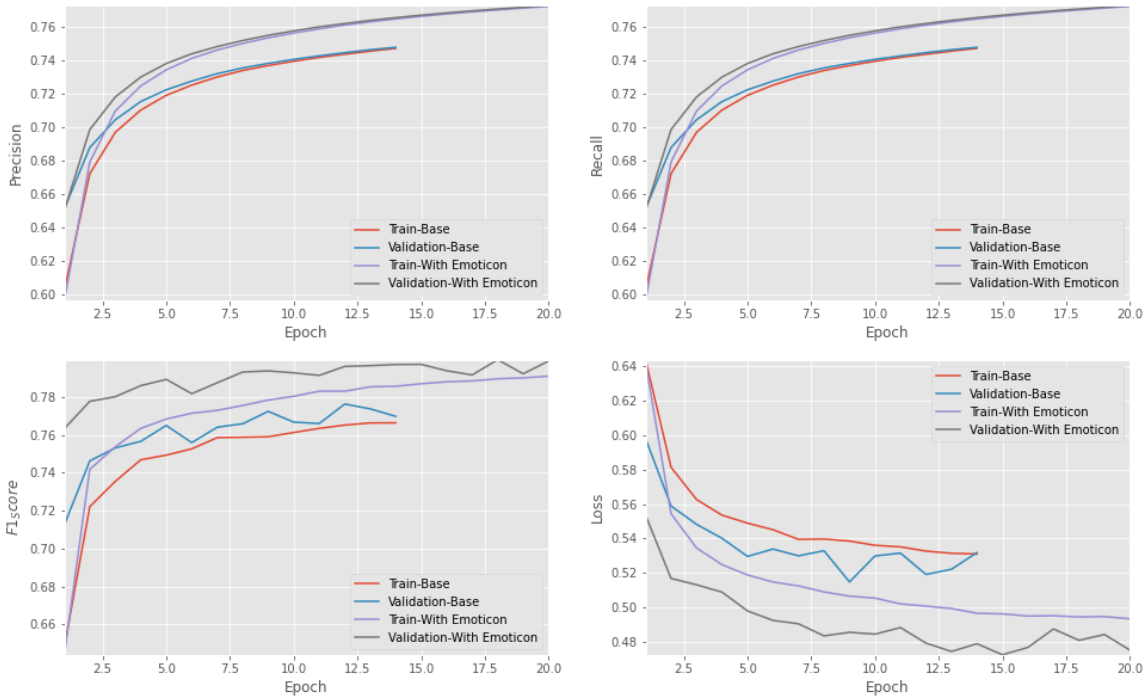

Figure 20. CNN with pre-trained twitter-based word2vec - precision, recall, F_1 score and loss

Table 15. CNN with pre-trained twitter-based word2vec - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N=13153**	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6564	1229	6548	1245	**7793**
	Negative	1766	3594	1429	3931	**5360**
		8330	**4823**	**7977**	**5176**	

Table 16. CNN with pre-trained twitter-based word2vec - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.79	0.84	0.81	7793	0.82	0.84	0.83	7793
negative	0.75	0.67	0.71	5360	0.76	0.73	0.75	5360
macro average	0.77	0.76	0.76	13153	0.79	0.79	0.79	13153
weighted average	0.77	0.77	0.77	13153	0.80	0.80	0.80	13153

4.3.4 CNN with pre-trained GloVe

In this model, we made an embedding layer using the pre-trained GloVe vector for the CNN model. We fitted this model using both training datasets. We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 21 and Figure 22 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 77.78% and 79.37% respectively. Table 17 and 18 describe the confusion matrix, precision, and recall for the test dataset.

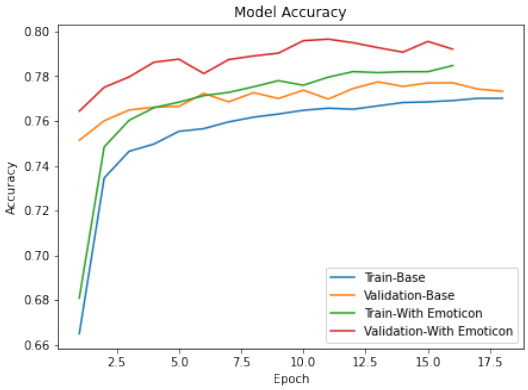

Figure 21. CNN with pre-trained GloVe - accuracy

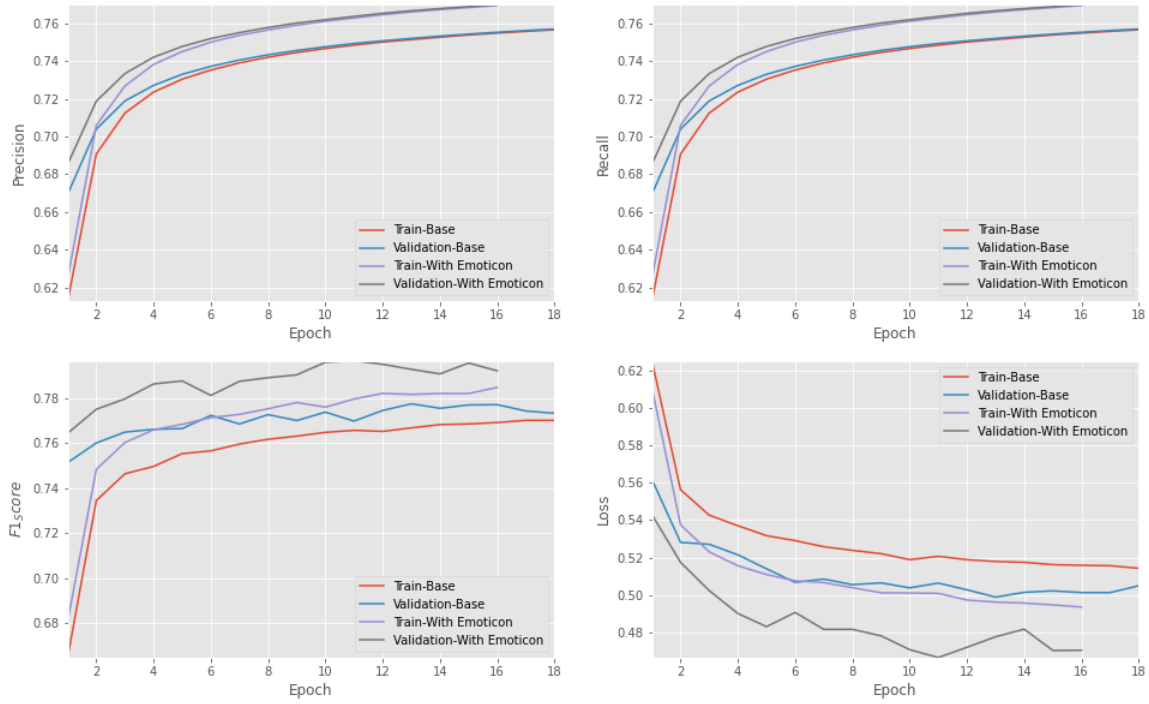

Figure 22. CNN with pre-trained GloVe - precision, recall, F_1 score and loss

Table 17. CNN with pre-trained GloVe - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N**=13153	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6465	1328	6636	1157	**7793**
	Negative	1594	3766	1557	3803	**5360**
		8059	**5094**	**8193**	**4960**	

Table 18. CNN with pre-trained GloVe - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.80	0.83	0.82	7793	0.81	0.85	0.83	7793
negative	0.74	0.70	0.72	5360	0.77	0.71	0.74	5360
macro average	0.77	0.77	0.77	13153	0.79	0.78	0.78	13153
weighted average	0.78	0.78	0.78	13153	0.79	0.79	0.79	13153

4.3.5 CNN with pre-trained fastText

In this model, we crafted an embedding layer using the pre-trained fastText vector for the CNN model. We fitted this model using both training datasets. We measured the

model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets that are shown in Figure 23 and Figure 24. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 77.21% and 79.76% respectively. Table 19 and 20 describe the confusion matrix, precision, and recall for the test dataset.

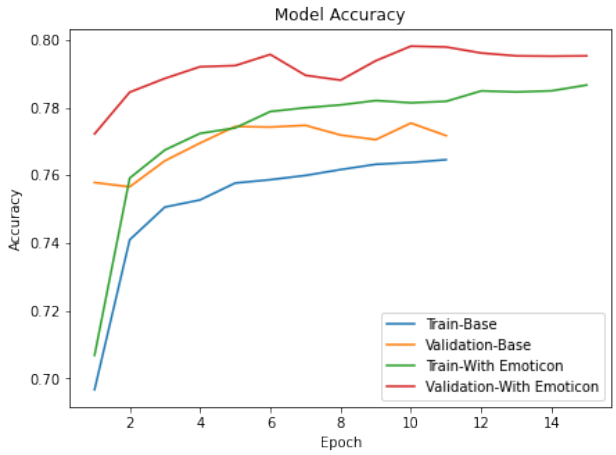

Figure 23. CNN with pre-trained fastText - accuracy

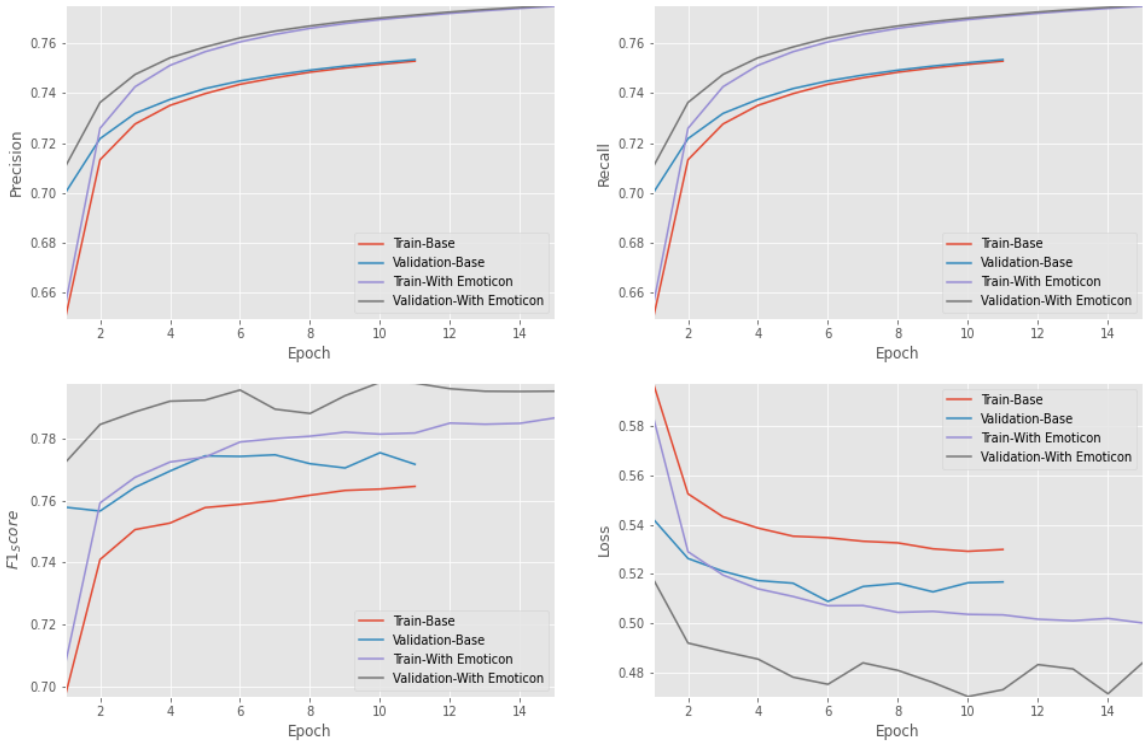

Figure 24. CNN with pre-trained fastText - precision, recall, F_1 score and loss

Table 19. CNN with pre-trained fastText - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	N=13153	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6536	1257	6407	1386	**7793**
	Negative	1740	3620	1276	4084	**5360**
		8276	**4877**	**7683**	**5470**	

Table 20. CNN with pre-trained fastText - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.79	0.84	0.81	7793	0.83	0.82	0.83	7793
negative	0.74	0.68	0.71	5360	0.75	0.76	0.75	5360
macro average	0.77	0.76	0.76	13153	0.79	0.79	0.79	13153
weighted average	0.77	0.77	0.77	13153	0.80	0.80	0.80	13153

4.3.6 Multichannel CNN with custom word2vec

In this model, we made an embedding layer using the custom word2vec vector for the Multichannel CNN model. We fitted this model using both training datasets. We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 25 and Figure 26 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 78.93% and 79.38% respectively. Table 21 and 22 describe the confusion matrix, precision, and recall for the test dataset.

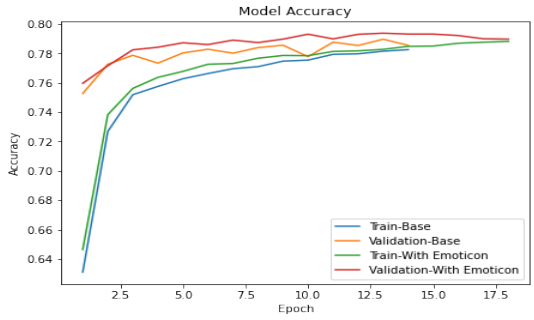

Figure 25. Multichannel CNN with custom word2vec - accuracy

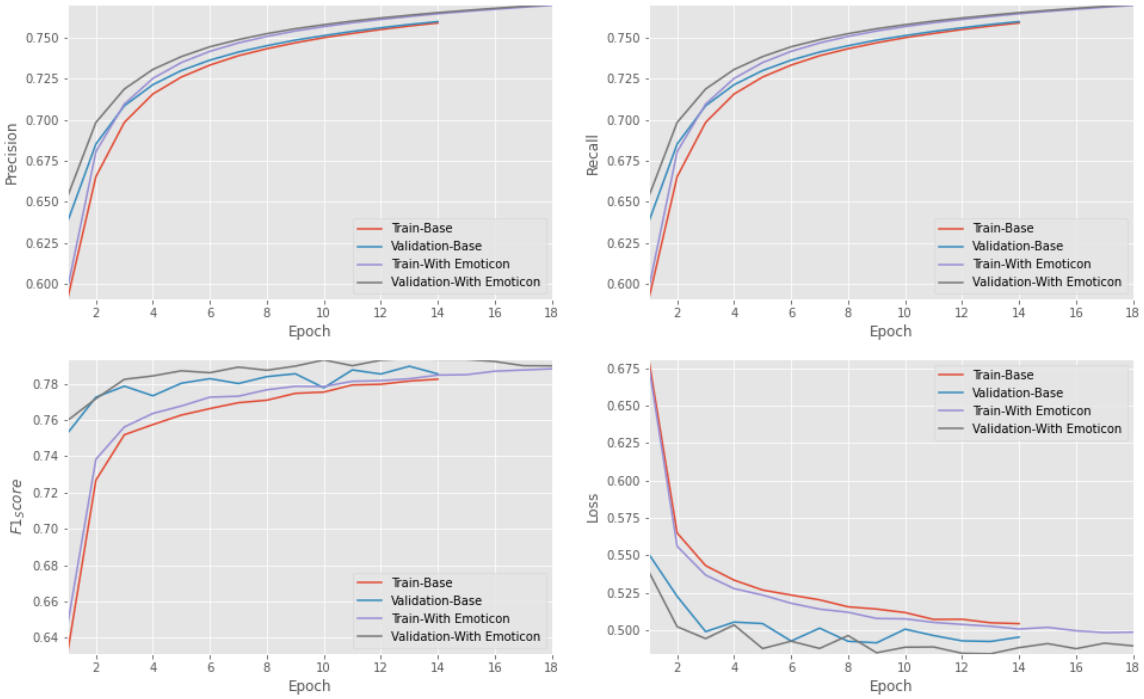

Figure 26. Multichannel CNN with custom word2vec - precision, recall, F_1 score and loss

Table 21. Multichannel CNN with custom word2vec - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N**=13153	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6478	1315	6606	1187	**7793**
	Negative	1456	3904	1525	3835	**5360**
		7934	**5219**	**8131**	**5022**	

Table 22. Multichannel CNN with custom word2vec - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.82	0.83	0.82	7793	0.81	0.85	0.83	7793
negative	0.75	0.73	0.74	5360	0.76	0.72	0.74	5360
macro average	0.78	0.78	0.78	13153	0.79	0.78	0.78	13153
weighted average	0.79	0.79	0.79	13153	0.79	0.79	0.79	13153

4.3.7 Multichannel CNN with pre-trained Google word2vec

In this model, we made an embedding layer using the pre-trained Google news word2vec vector for the multichannel CNN models. We fitted this model using both

training datasets. We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 27 and Figure 28 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 77.15% and 77.38% respectively. Table 23 and 24 describe the confusion matrix, precision, and recall for the test dataset.

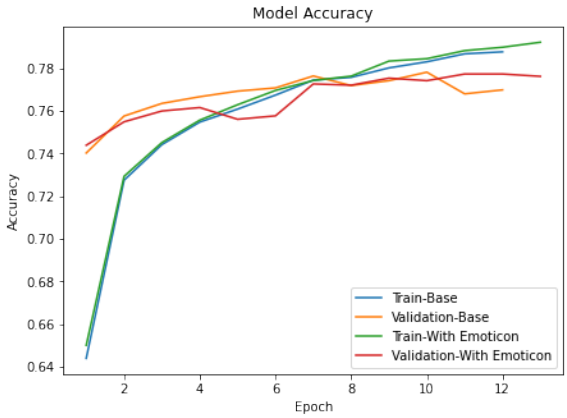

Figure 27. Multichannel CNN with pre-trained Google news word2vec - accuracy

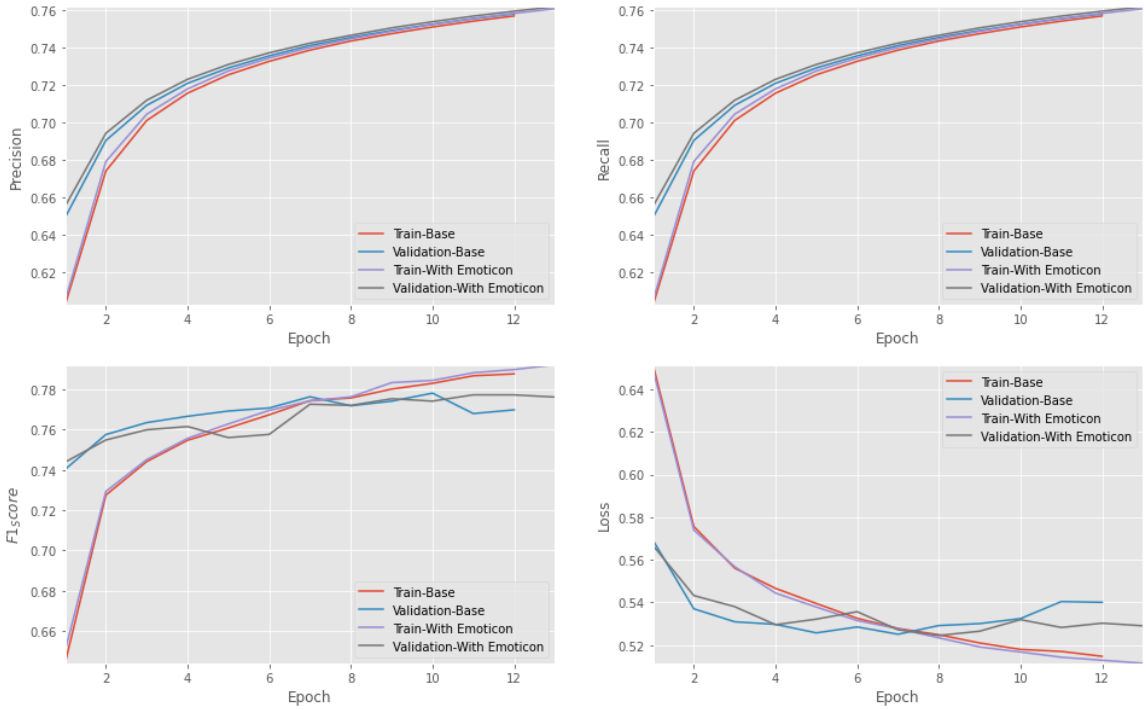

Figure 28. Multichannel CNN with pre-trained Google news word2vec - precision, recall, F_1 score and loss

Table 23. Multichannel CNN with pre-trained Google news word2vec - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N=13153**	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6652	1141	6678	1115	**7793**
	Negative	1864	3496	1860	3500	**5360**
		8516	**4637**	**8538**	**4615**	

Table 24. Multichannel CNN with pre-trained Google news word2vec - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.78	0.85	0.82	7793	0.78	0.86	0.82	7793
negative	0.75	0.65	0.70	5360	0.76	0.65	0.70	5360
macro average	0.77	0.75	0.76	13153	0.77	0.75	0.76	13153
weighted average	0.77	0.77	0.77	13153	0.77	0.77	0.77	13153

4.3.8 Multichannel CNN with pre-trained Twitter word2vec

In this model, we crafted an embedding layer using the pre-trained twitter-based word2vec vector for the Multichannel CNN model. We fitted this model using both training datasets. We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 29 and Figure 30 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 79.59% and 80% respectively. Table 25 and 26 describe the confusion matrix, precision, and recall for the test dataset.

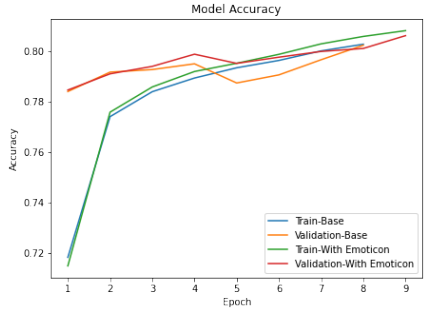

Figure 29. Multichannel CNN with pre-trained Twitter data word2vec - accuracy

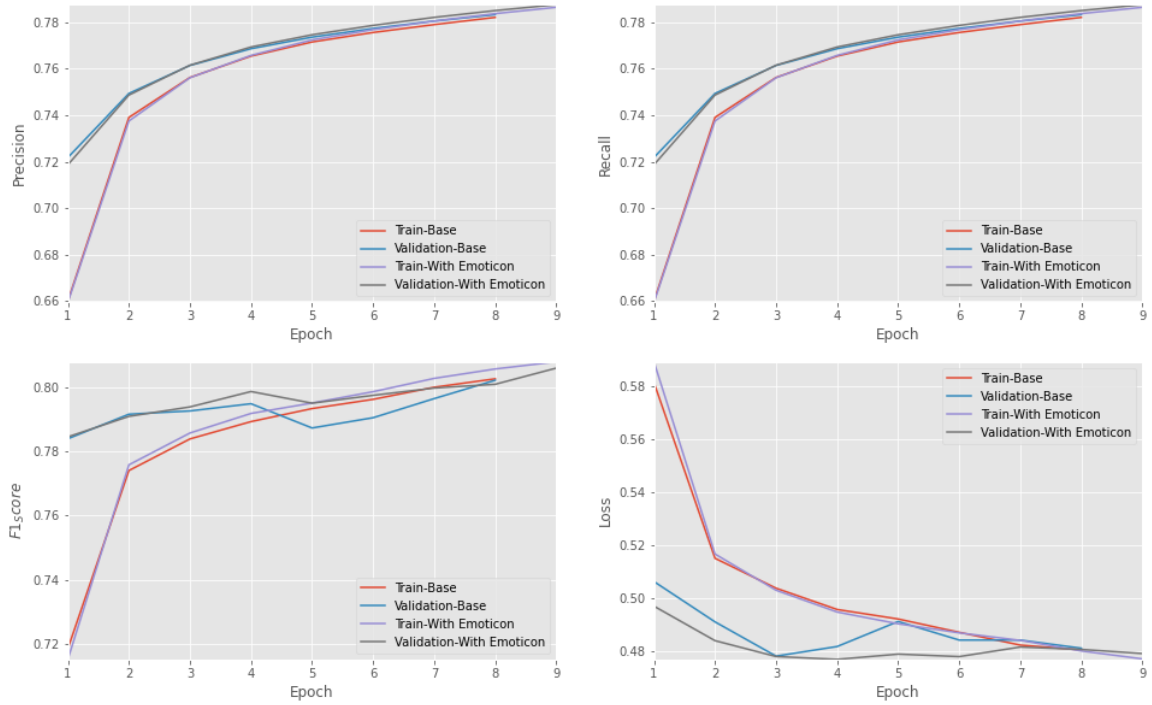

Figure 30. Multichannel CNN with pre-trained Twitter data word2vec - precision, recall, F_1 score and loss

Table 25. Multichannel CNN with pre-trained Twitter data word2vec - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N**=13153	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6519	1274	6463	1330	**7793**
	Negative	1411	3949	1300	4060	**5360**
		7930	**5223**	**7763**	**5390**	

Table 26. Multichannel CNN with pre-trained Twitter data word2vec - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.82	0.84	0.83	7793	0.83	0.83	0.83	7793
negative	0.76	0.74	0.75	5360	0.75	0.76	0.76	5360
macro average	0.79	0.79	0.79	13153	0.79	0.79	0.79	13153
weighted average	0.80	0.80	0.80	13153	0.80	0.80	0.80	13153

4.3.9 Multichannel CNN with pre-trained GloVe

In this model, we created an embedding layer using the pre-trained GloVe vector for the Multichannel CNN model. We fitted this model using both training datasets. We

evaluated the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 31 and Figure 32 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 79.97% and 80.1% respectively. Table 27 and 28 described the confusion matrix, precision, and recall for the test dataset.

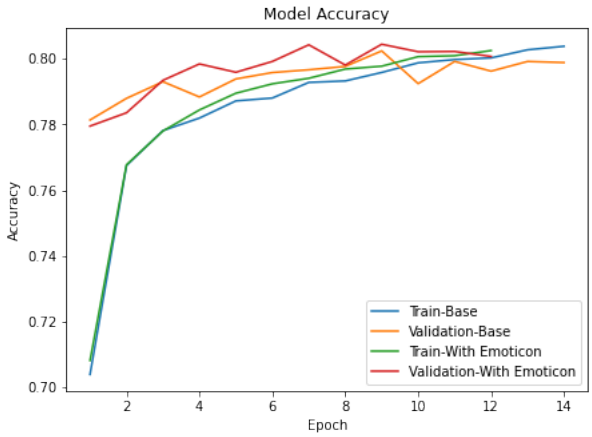

Figure 31. Multichannel CNN with pre-trained GloVe - accuracy

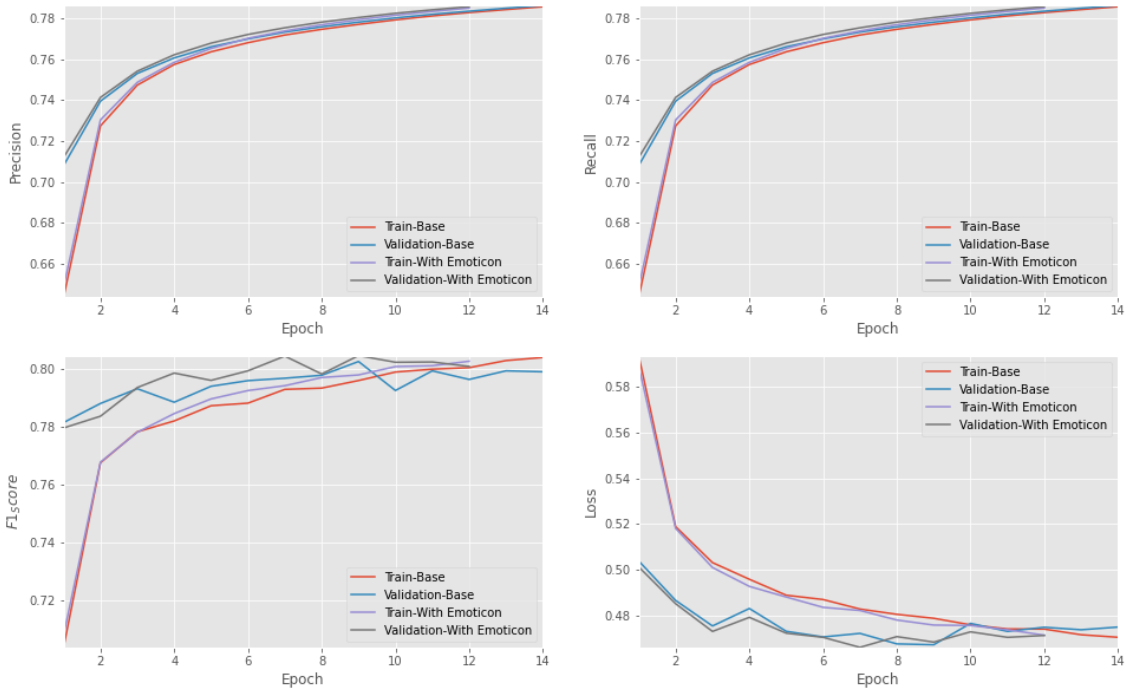

Figure 32. Multichannel CNN with pre-trained GloVe - precision, recall, F_1 score and loss

Table 27. Multichannel CNN with pre-trained GloVe - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N=13153**	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6456	1337	6600	1193	**7793**
	Negative	1298	4062	1424	3936	**5360**
		7754	**5399**	**8024**	**5129**	

Table 28. Multichannel CNN with pre-trained GloVe - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.83	0.83	0.83	7793	0.82	0.85	0.83	7793
negative	0.75	0.76	0.76	5360	0.77	0.73	0.75	5360
macro average	0.79	0.79	0.79	13153	0.79	0.79	0.79	13153
weighted average	0.80	0.80	0.80	13153	0.80	0.80	0.80	13153

4.3.10 Multichannel CNN with pre-trained fastText

In this model, we built an embedding layer using the pre-trained fastText for the Multichannel CNN model. We fitted this model using both training datasets. We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 33 and Figure 34 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 80.15% and 80.57% respectively. Table 29 and 30 describe the confusion matrix, precision, and recall for the test dataset.

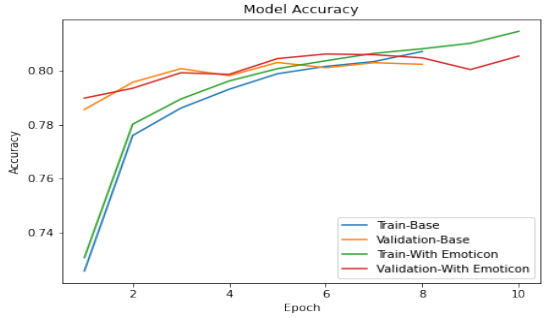

Figure 33. Multichannel CNN with pre-trained fastText - accuracy

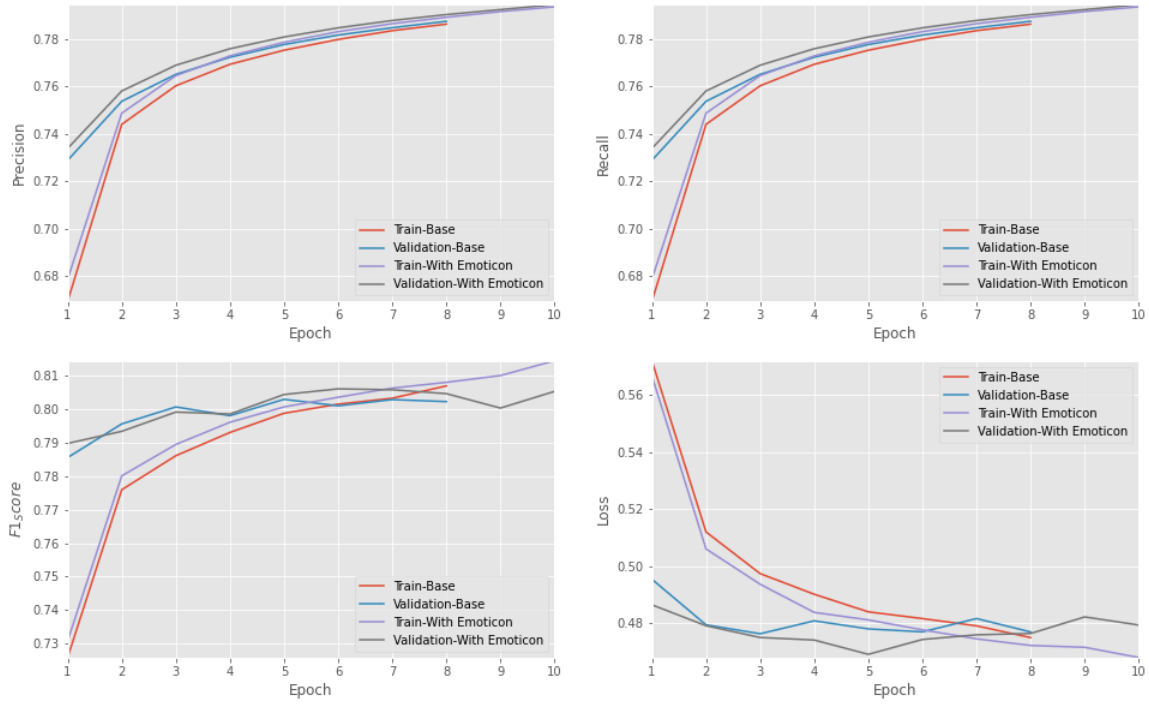

Figure 34. Multichannel CNN with pre-trained fastText - precision, recall, F_1 score and loss

Table 29. Multichannel CNN with pre-trained fastText - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N**=13153	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6618	1175	6441	1352	**7793**
	Negative	1436	3924	1203	4157	**5360**
		8054	**5099**	**7644**	**5509**	

Table 30. Multichannel CNN with pre-trained fastText - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.82	0.85	0.84	7793	0.84	0.83	0.83	7793
negative	0.77	0.73	0.75	5360	0.75	0.78	0.76	5360
macro average	0.80	0.79	0.79	13153	0.80	0.80	0.80	13153
weighted average	0.80	0.80	0.80	13153	0.81	0.81	0.81	13153

While comparing all the CNN models' performance metrics, the WITH_EMOTICON dataset performed a bit better than the BASE_MODEL dataset. Also, we observed that models with a fastText embedding vector predicted the results more correctly than the other

embedding vectors, and that the multichannel CNN model worked better for these datasets than the sequential CNN model with all the embedding vectors factored in.

4.4 LSTM model results

We experimented with a few different types of Bidirectional LSTM (BiLSTM) as well as a BiLSTM with an attention layer models for sentiment classification. Both of the BiLSTM models were trained and made predictions using the four embedding vectors for the BASE_MODEL and WITH_EMOTICON datasets with the hyperparameters mentioned in Table 6. We tuned a few parameters based on the execution time and model performance of each dataset. The sections that follow analyze the output of each model.

4.4.1 BiLSTM with custom word2vec

We tokenized both datasets using a Keras tokenizer that generated around 112,069 tokens of words in each dataset. The input text in the dataset was padded with its maximum length of the text from the input data. Maximum length was different for both datasets, the BASE_MODEL experiment maximum length was 40 and the WITH_EMOTICON maximum length was 42. All the BiLSTM models initially used these steps before creating the embedding layer. After that, we made an embedding layer using the custom word2vec vector. The BiLSTM model was fitted using both training datasets. We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 35 and Figure 36 illustrate these metrics for this model. While fitting the models, if the validation dataset accuracy had not improved, we stopped the iterations through early stopping parameters. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 80.48% and 81.07% respectively. Table 29 and 30 describe the confusion

matrix, precision, and recall for the test dataset.

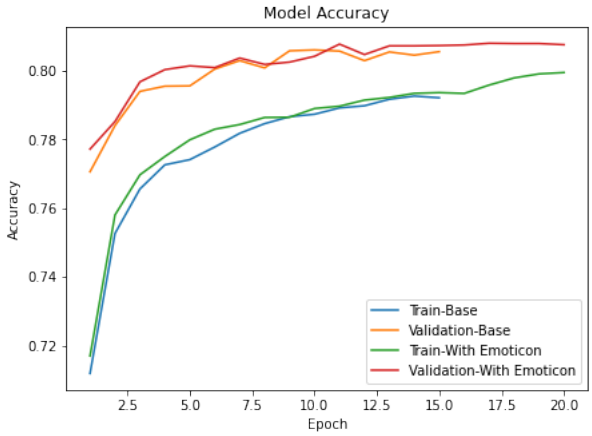

Figure 35. BiLSTM with custom word2vec - accuracy

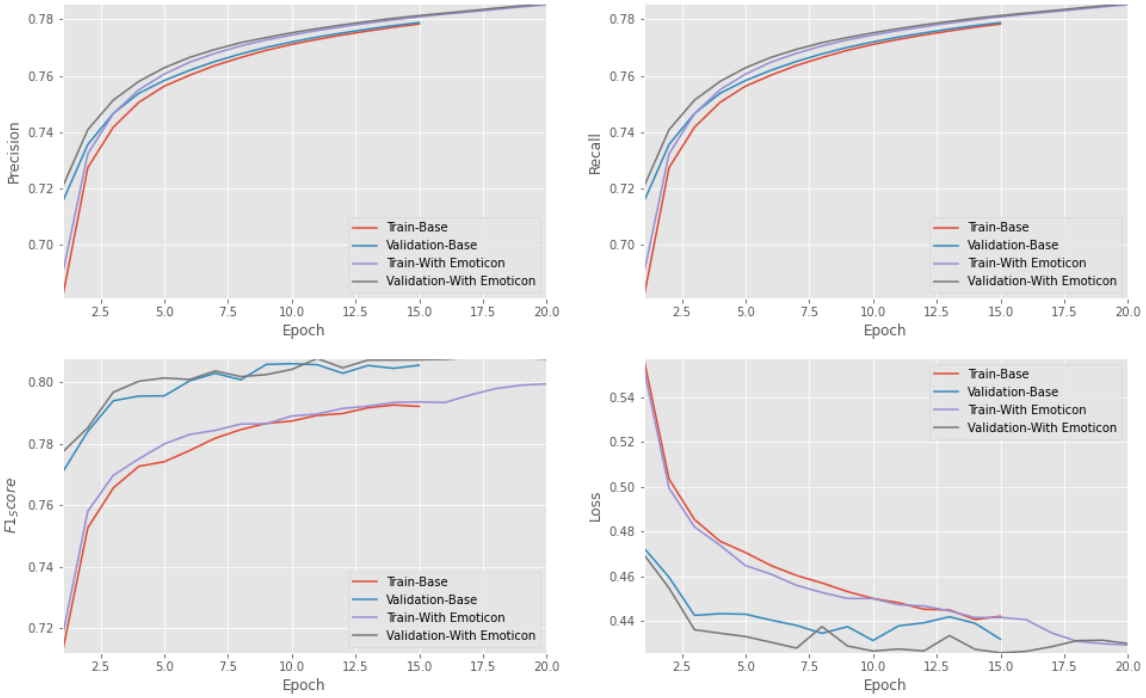

Figure 36. BiLSTM with custom word2vec - precision, recall, F_1 score and loss

Table 31. BiLSTM with custom word2vec - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N**=13153	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6625	1168	6634	1159	**7793**
	Negative	1400	3960	1331	4029	**5360**
		8025	**5128**	**7965**	**5188**	

Table 32. BiLSTM with custom word2vec - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.83	0.85	0.84	7793	0.83	0.85	0.84	7793
negative	0.77	0.74	0.76	5360	0.78	0.75	0.76	5360
macro average	0.80	0.79	0.80	13153	0.80	0.80	0.80	13153
weighted average	0.80	0.80	0.80	13153	0.81	0.81	0.81	13153

4.4.2 BiLSTM with pre-trained Google word2vec

In this model, we created an embedding layer using the pre-trained Google word2vec vector for the BiLSTM model. We fitted this model using both training datasets. We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 37 and Figure 38 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 80.83% and 81.03% respectively. Table 33 and 34 describe the confusion matrix, precision, and recall for the test dataset.

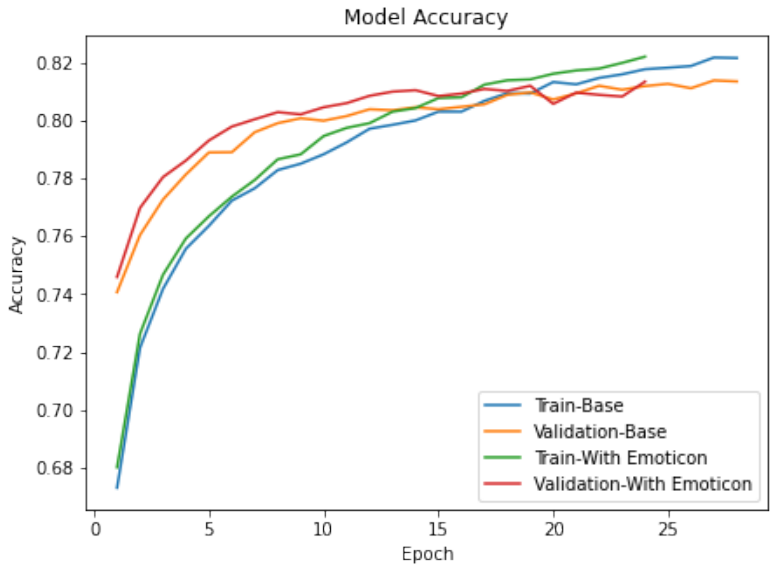

Figure 37. BiLSTM with pre-trained Google word2vec - accuracy

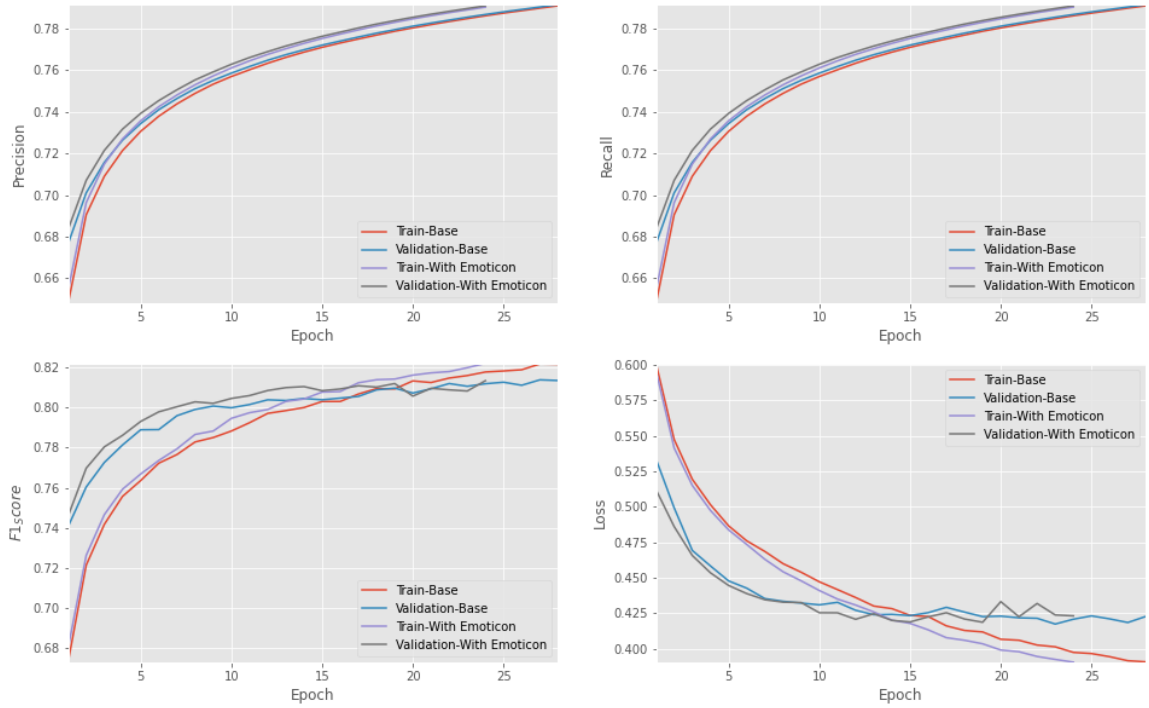

Figure 38. BiLSTM with pre-trained Google word2vec - precision, recall, F_1 score and loss

Table 33. BiLSTM with pre-trained Google word2vec - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N**=13153	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6734	1059	6481	1312	**7793**
	Negative	1459	3901	1183	4177	**5360**
		8193	**4960**	**7664**	**5489**	

Table 34. BiLSTM with pre-trained Google word2vec - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.82	0.86	0.84	7793	0.85	0.83	0.84	7793
negative	0.79	0.73	0.76	5360	0.76	0.78	0.77	5360
macro average	0.80	0.80	0.80	13153	0.80	0.81	0.80	13153
weighted average	0.81	0.81	0.81	13153	0.81	0.81	0.81	13153

4.4.3 BiLSTM with pre-trained Twitter word2vec

In this model, we made an embedding layer using the pre-trained twitter-based word2vec vector for the BiLSTM model. We fitted this model using both training datasets.

We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 39 and Figure 40 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 82.26% and 82.75% respectively. Table 35 and 36 describe the confusion matrix, precision, and recall for the test dataset.

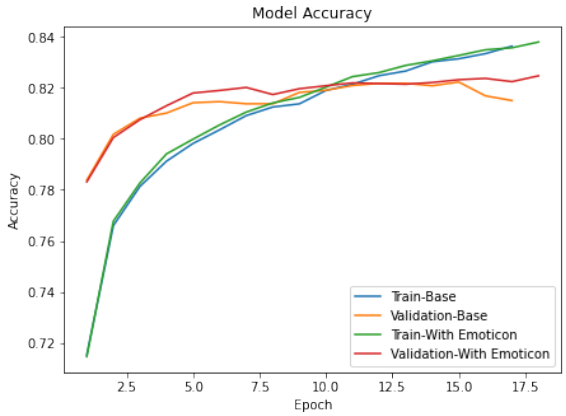

Figure 39. BiLSTM with pre-trained twitter data based word2vec - accuracy

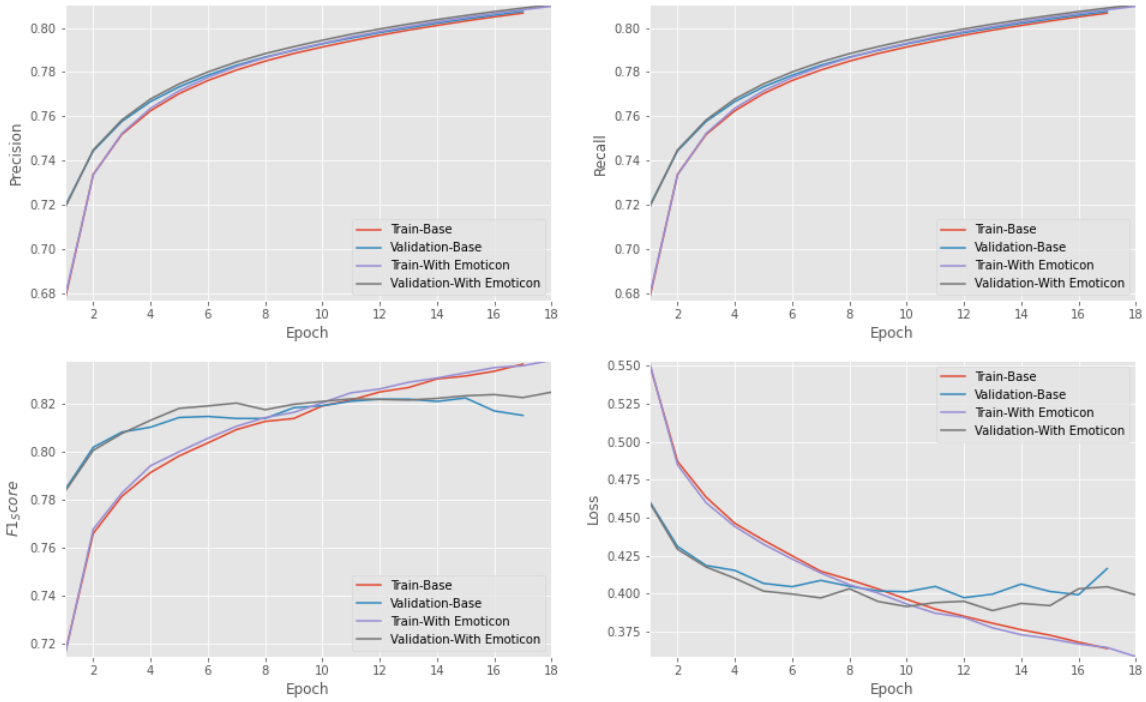

Figure 40. BiLSTM with pre-trained twitter data based word2vec - precision, recall, F_1 score and loss

Table 35. BiLSTM with pre-trained twitter data based word2vec - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N=13153**	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6639	1154	6571	1222	**7793**
	Negative	1179	4181	1047	4313	**5360**
		7818	**5335**	**7618**	**5535**	

Table 36. BiLSTM with pre-trained twitter data based word2vec - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.85	0.85	0.85	7793	0.86	0.84	0.85	7793
negative	0.78	0.78	0.78	5360	0.78	0.80	0.79	5360
macro average	0.82	0.82	0.82	13153	0.82	0.82	0.82	13153
weighted average	0.82	0.82	0.82	13153	0.83	0.83	0.83	13153

4.4.4 BiLSTM with pre-trained GloVe

In this model, we made an embedding layer using the pre-trained GloVe vector for the BiLSTM model. We fitted this model using both training datasets. We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 41 and Figure 42 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 81.91% and 82.7% respectively. Table 37 and 38 describe the confusion matrix, precision, and recall for the test dataset.

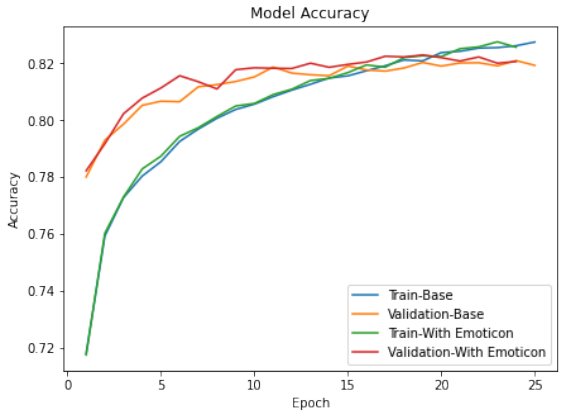

Figure 41. BiLSTM with pre-trained GloVe - accuracy

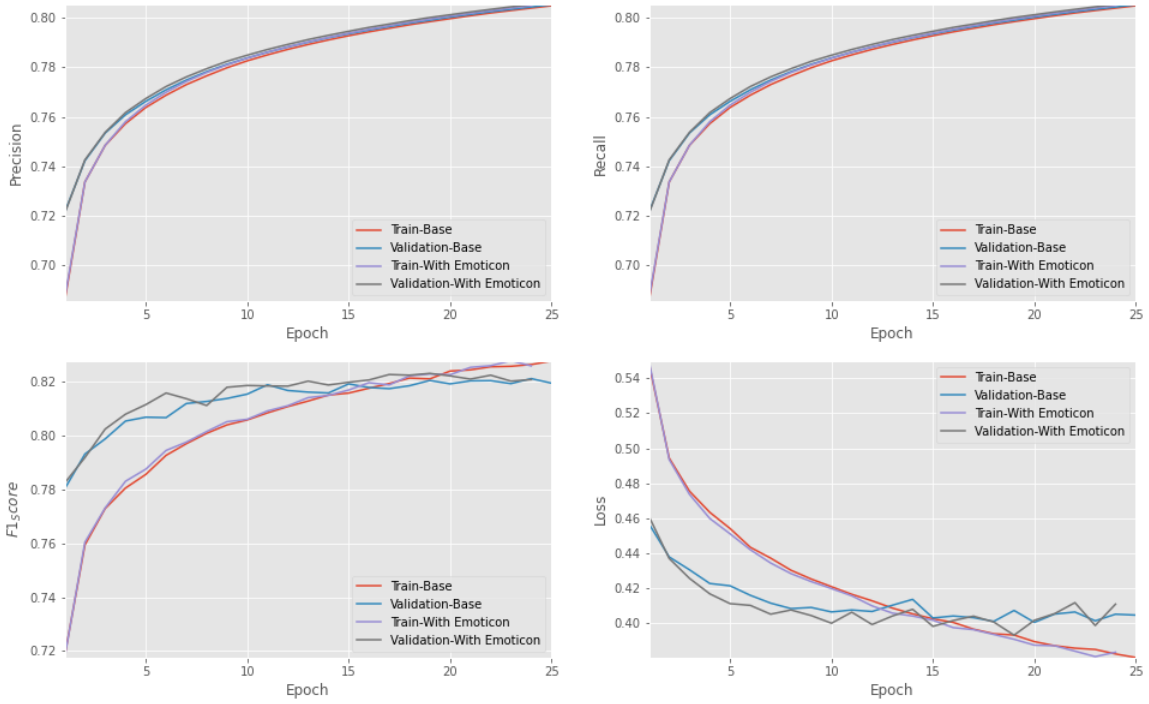

Figure 42. BiLSTM with pre-trained GloVe - precision, recall, F_1 score and loss

Table 37. BiLSTM with pre-trained GloVe - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N**=13153	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6565	1228	6642	1151	**7793**
	Negative	1152	4208	1125	4235	**5360**
		7717	**5436**	**7767**	**5386**	

Table 38. BiLSTM with pre-trained GloVe - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.85	0.84	0.85	7793	0.86	0.85	0.85	7793
negative	0.77	0.79	0.78	5360	0.79	0.79	0.79	5360
macro average	0.81	0.81	0.81	13153	0.82	0.82	0.82	13153
weighted average	0.82	0.82	0.82	13153	0.83	0.83	0.83	13153

4.4.5 BiLSTM with pre-trained fastText

In this model, we made an embedding layer using the pre-trained fastText vector for the BiLSTM model. We fitted this model using both training datasets. We measured the

model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 43 and Figure 44 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 82.31% and 82.73% respectively. Table 39 and 40 describe the confusion matrix, precision, and recall for the test dataset.

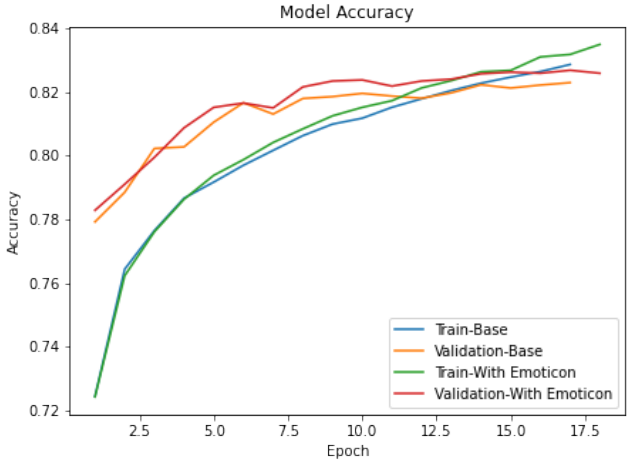

Figure 43. BiLSTM with pre-trained fastText - accuracy

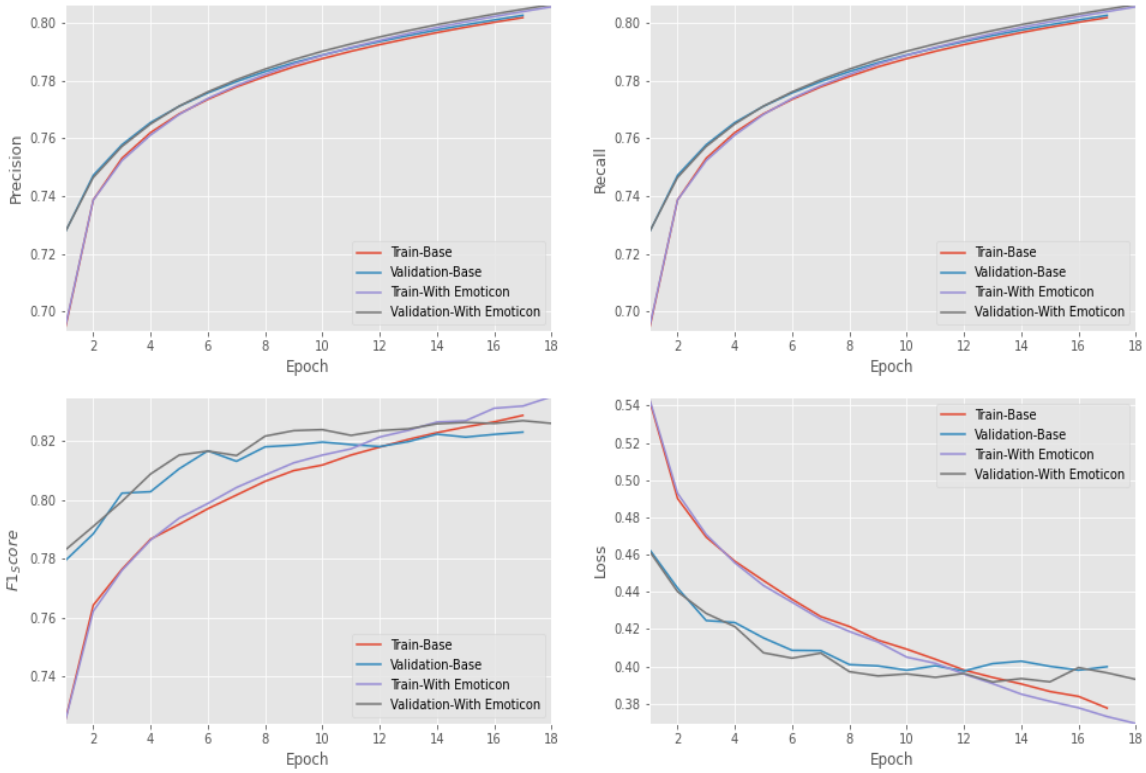

Figure 44. BiLSTM with pre-trained fastText - precision, recall, F_1 score and loss

Table 39. BiLSTM with pre-trained fastText - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N=13153**	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6861	932	6512	1281	**7793**
	Negative	1395	3965	991	4369	**5360**
		8256	**4897**	**7503**	**5650**	

Table 40. BiLSTM with pre-trained fastText - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.83	0.88	0.86	7793	0.87	0.84	0.84	7793
negative	0.81	0.74	0.77	5360	0.77	0.82	0.79	5360
macro average	0.82	0.81	0.81	13153	0.82	0.83	0.82	13153
weighted average	0.82	0.82	0.82	13153	0.83	0.83	0.83	13153

4.4.6 BiLSTM with attention layer - custom word2vec

In this model, we created an embedding layer using the custom word2vec vector and fitted the BiLSTM with an attention layer model using both training datasets. We evaluated the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 45 and Figure 46 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 81.21% and 81.52% respectively. Table 41 and 42 describe the confusion matrix, precision, and recall for the test dataset.

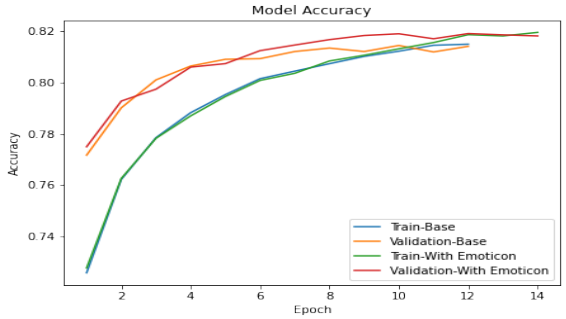

Figure 45. BiLSTM with attention layer – custom word2vec accuracy

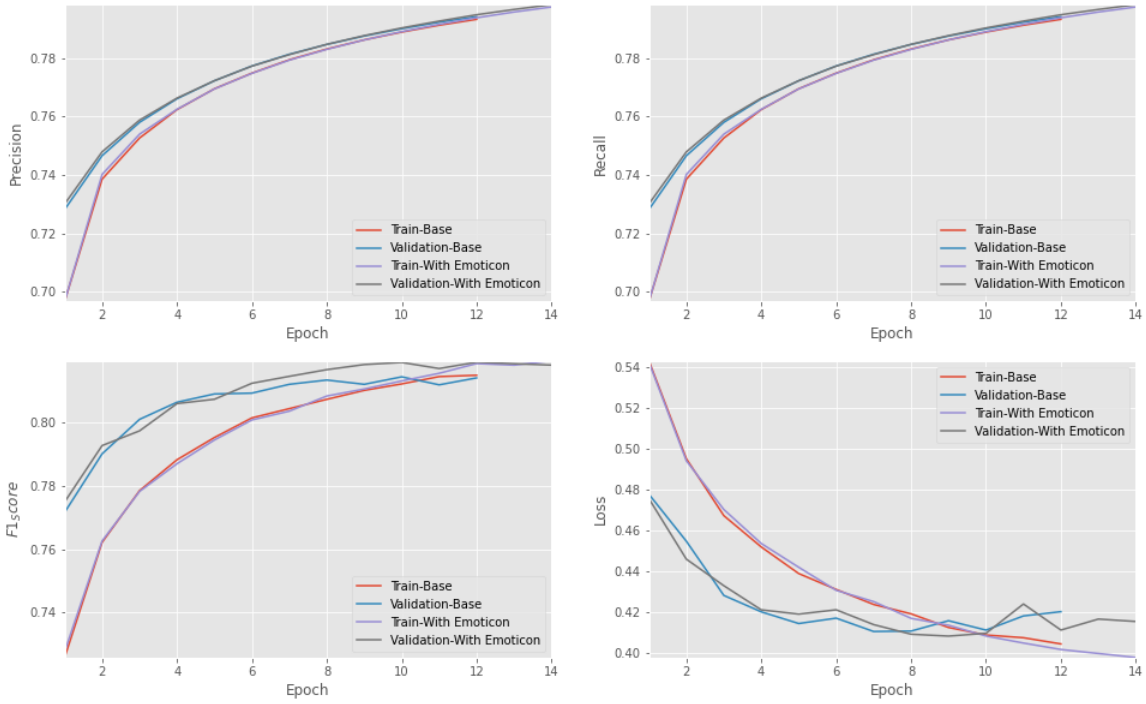

Figure 46. BiLSTM with attention layer – custom word2vec precision, recall, F_1 score and loss

Table 41. BiLSTM with attention layer – custom word2vec confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N**=13153	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6638	1155	6712	1081	**7793**
	Negative	1316	4044	1350	4010	**5360**
		7954	**5199**	**8062**	**5091**	

Table 42. BiLSTM with attention layer – custom word2vec precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.83	0.85	0.84	7793	0.83	0.86	0.85	7793
negative	0.78	0.75	0.77	5360	0.79	0.75	0.77	5360
macro average	0.80	0.80	0.80	13153	0.81	0.80	0.81	13153
weighted average	0.81	0.81	0.81	13153	0.81	0.82	0.81	13153

4.4.7 BiLSTM with attention layer - pre-trained Google word2vec

In this model, we made an embedding layer using the pre-trained Google news word2vec vector for the BiLSTM model with an attention layer. We fitted this model fitted

using both training datasets. We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 47 and Figure 48 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 81.65% and 82.67% respectively. Table 43 and 44 describe the confusion matrix, precision, and recall for the test dataset.

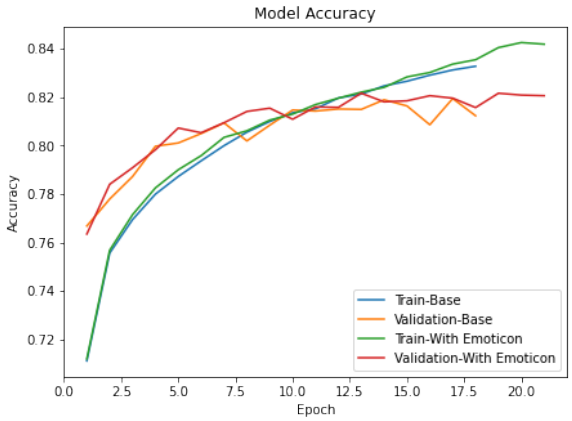

Figure 47. BiLSTM attention with pre-trained Google news word2vec - accuracy

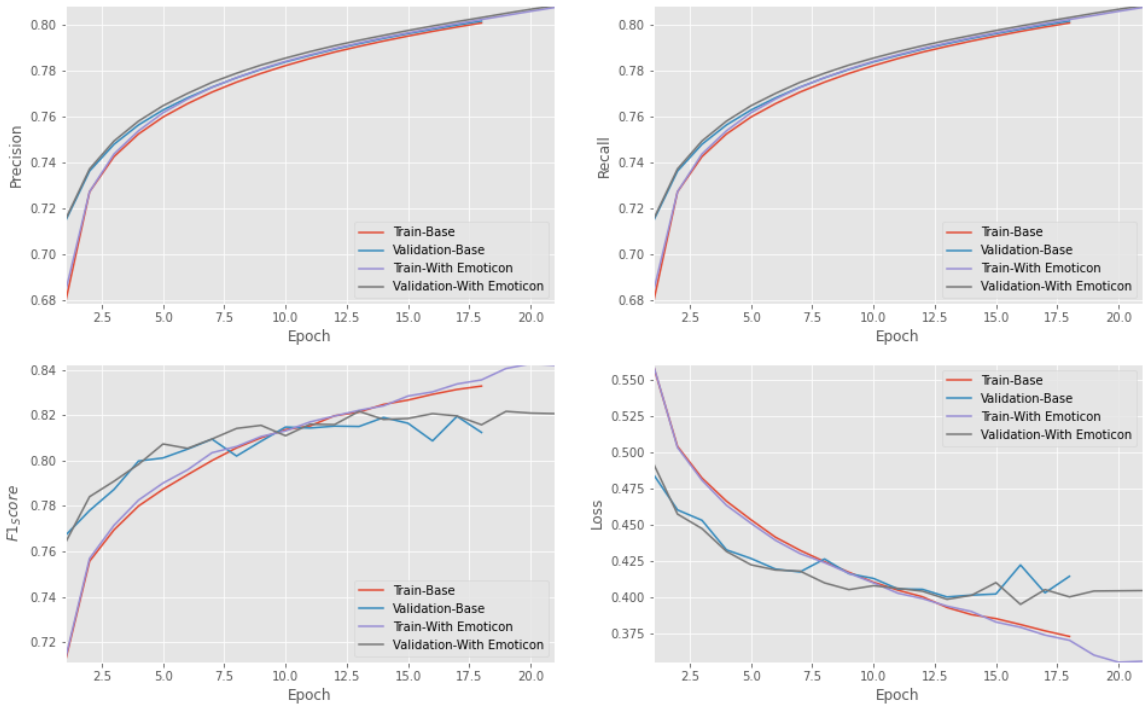

Figure 48. BiLSTM attention with pre-trained Google news word2vec - precision, recall, F_1 score and loss

Table 43. BiLSTM attention with pre-trained Google news word2vec - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N=13153**	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6515	1278	6629	1164	**7793**
	Negative	1135	4225	1116	4244	**5360**
		7650	**5503**	**7745**	**5408**	

Table 44. BiLSTM attention with pre-trained Google news word2vec - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.85	0.84	0.84	7793	0.86	0.85	0.85	7793
negative	0.77	0.79	0.78	5360	0.78	0.79	0.79	5360
macro average	0.81	0.81	0.81	13153	0.82	0.82	0.82	13153
weighted average	0.82	0.82	0.82	13153	0.83	0.83	0.83	13153

4.4.8 BiLSTM with attention layer - pre-trained Twitter word2vec

In this model, we made an embedding layer using the pre-trained twitter-based word2vec vector for the BiLSTM model with an attention layer. We fitted this model using both training datasets. We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 49 and Figure 50 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 83.24% and 83.4% respectively. Table 45 and 46 describe the confusion matrix, precision, and recall for the test dataset.

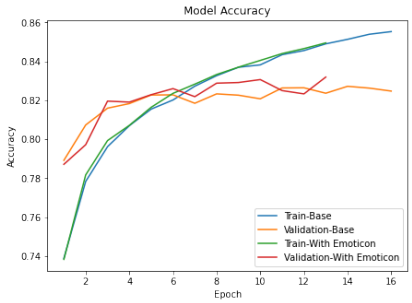

Figure 49. BiLSTM attention with pre-trained Twitter data word2vec - accuracy

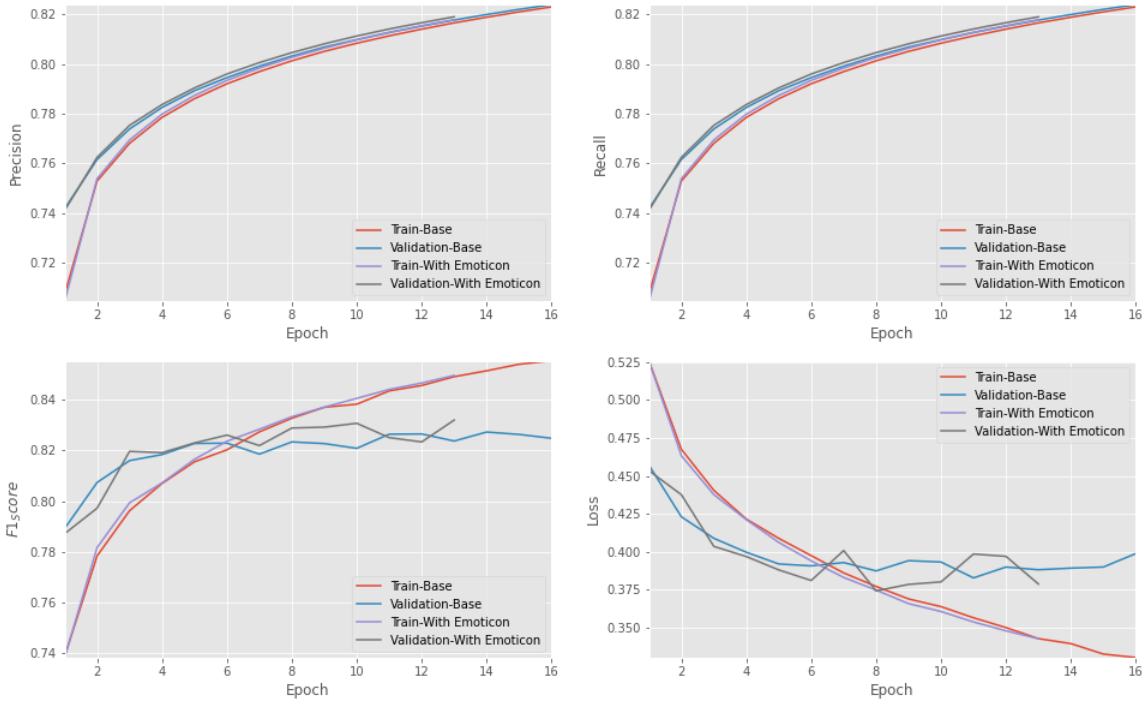

Figure 50. Figure 51. BiLSTM attention with pre-trained Twitter data word2vec - precision, recall, F_1 score and loss

Table 45. BiLSTM attention with pre-trained Twitter data word2vec - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N=13153**	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6631	1162	6714	1079	**7793**
	Negative	1042	4318	1105	4255	**5360**
		7673	**5480**	**7819**	**5334**	

Table 46. BiLSTM attention with pre-trained Twitter data word2vec - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.86	0.85	0.86	7793	0.86	0.86	0.86	7793
negative	0.79	0.81	0.80	5360	0.80	0.79	0.80	5360
macro average	0.83	0.83	0.83	13153	0.83	0.83	0.83	13153
weighted average	0.83	0.83	0.83	13153	0.83	0.83	0.83	13153

4.4.9 BiLSTM with attention layer - pre-trained GloVe

In this model, we made an embedding layer using the pre-trained GloVe vector and fitted the BiLSTM with an attention layer model using both training datasets. We measured

the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 51 and Figure 52 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 82.29% and 83.38% respectively. Table 47 and 48 describe the confusion matrix, precision, and recall for the test dataset.

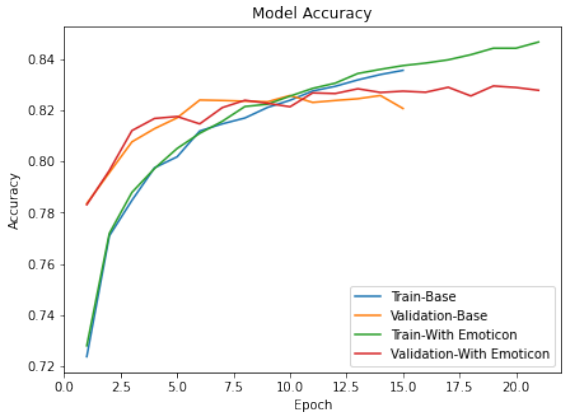

Figure 52. BiLSTM attention with pre-trained GloVe - accuracy

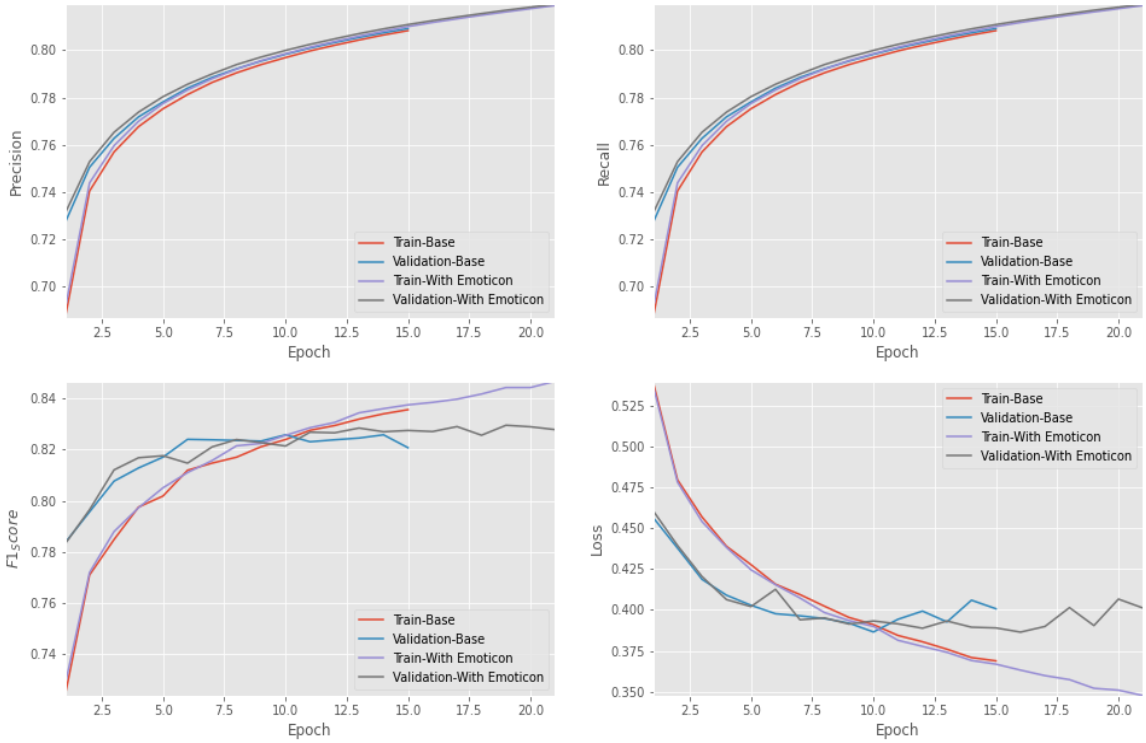

Figure 53. BiLSTM attention with pre-trained GloVe - precision, recall, F_1 score and loss

Table 47. BiLSTM attention with pre-trained GloVe - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N=13153**	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6612	1181	6631	1162	**7793**
	Negative	1149	4211	1024	4336	**5360**
		7761	**5392**	**7655**	**5498**	

Table 48. BiLSTM attention with pre-trained GloVe - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.85	0.85	0.85	7793	0.87	0.85	0.86	7793
negative	0.78	0.79	0.78	5360	0.79	0.81	0.80	5360
macro average	0.82	0.82	0.82	13153	0.83	0.83	0.83	13153
weighted average	0.82	0.82	0.82	13153	0.83	0.83	0.83	13153

4.4.10 BiLSTM with attention layer with pre-trained fastText

In this model, we crafted an embedding layer using the pre-trained fastText vector for the BiLSTM model with an attention layer. We fitted this model using both training datasets. We measured the model's accuracy, precision, recall, F_1 score and loss metrics for both of the experiments using validation datasets. Figure 53 and Figure 54 illustrate these metrics for this model. We then measured the test dataset prediction accuracy for both datasets. Accuracies for the base model and the model with emoticons were 83.02% and 83.56% respectively. Table 49 and 50 describe the confusion matrix, precision, and recall for the test dataset.

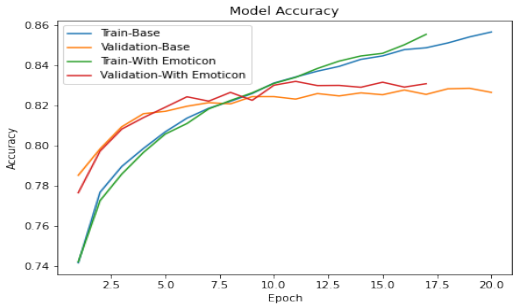

Figure 54. BiLSTM with attention pre-trained fastText - accuracy

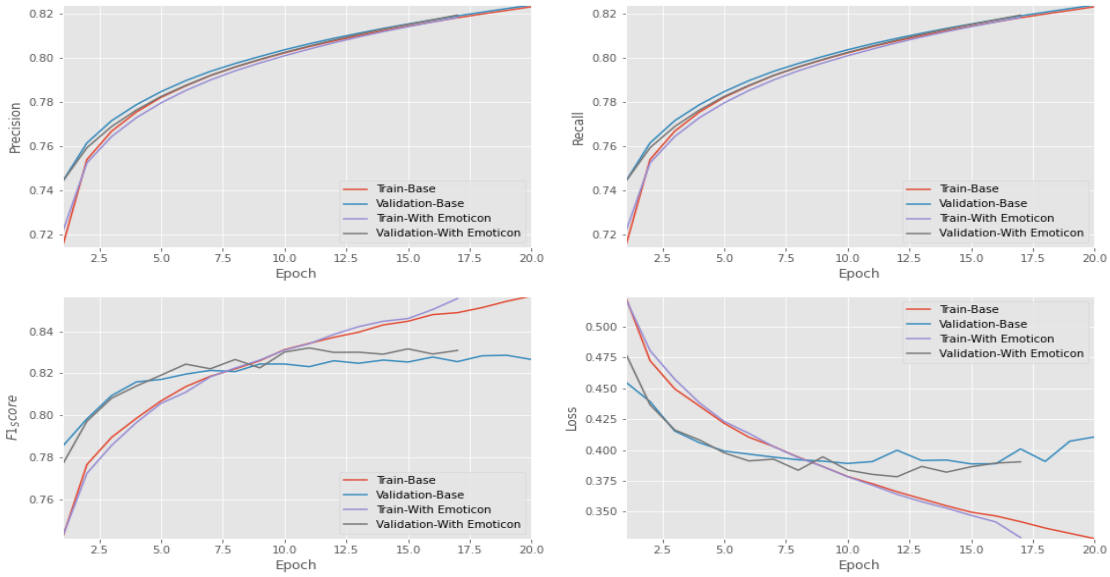

Figure 55. BiLSTM with attention pre-trained fastText - precision, recall, F_1 score and loss

Table 49. BiLSTM with attention pre-trained fastText - confusion matrix

		Predicted label				
		Base		With Emoticon		
True label	**N**=13153	**Positive**	**Negative**	**Positive**	**Negative**	
	Positive	6564	1229	6576	1217	**7793**
	Negative	1004	4356	946	4414	**5360**
		7568	**5585**	**7522**	**5631**	

Table 50. BiLSTM with attention pre-trained fastText - precision, recall, F_1 score, support, accuracy, macro average, and micro average

	Base				With Emoticon			
	precision	**recall**	**F_1 score**	**support**	**precision**	**recall**	**F_1 score**	**support**
positive	0.87	0.84	0.85	7793	0.87	0.84	0.86	7793
negative	0.78	0.81	0.80	5360	0.78	0.82	0.80	5360
macro average	0.82	0.83	0.83	13153	0.83	0.83	0.83	13153
weighted average	0.83	0.83	0.83	13153	0.84	0.84	0.84	13153

While comparing all the BiLSTM models' accuracy, precision, recall, and F_1 score, the WITH_EMOTICON dataset performed better than the BASE_MODEL dataset. All the LSTM models performed better in the WITH_EMOTICON experiment for sentiment classification than the other dataset. Also, we observed that the BiLSTM models performed

better than CNN models, as well as that the BiLSTM model with an attention layer worked better for these datasets than the BiLSTM model with all the embedding vectors.

4.5 Model evaluation

All the CNN and BiLSTM models were trained using training datasets of the BASE_MODEL and WITH_EMOTICON experiments with the hyperparameters mentioned previously. While fitting each model, we validated the model's performance using a validation dataset with 40 epochs. To avoid overfitting, we added the early stopping option in Keras, and this option stopped the training process based on the parameters. These are the parameters we used in the early stopping callback methods: the '*monitor*' parameter which allowed us to specify the performance measurement to monitor and end training. We used '*loss*' on the validation dataset; the mode parameter in the early stopping process was set as '*auto*,' and that allowed us to minimize loss or maximize accuracy; we set a '*patience*' parameter as 5, which delayed the trigger in terms of the number of epochs on which we would like to see no improvement; the '*min_delta*' parameter was set as '*1e-4*' for minimum improvement in each epoch; the '*restore_best_weights*' was set as '*True*' to retain the best weight for the models. This stopped the fitting earlier when there was an overfitting issue. Also, we used a regularizer at l_2 regularization and added a dropout rate for avoiding the overfitting issues.

We used the same approach to overcome overfitting issues across all the CNN and LSTM models to get the best models. The trained model validated the number of epochs for the test dataset and measured the model's performance using model accuracy, precision, recall, F_1 score, and macro average metrics. Based on the performance validation and twitter message length, we used the following hyperparameters for the CNN and

Multichannel CNN models shown in Table 51.

Table 51. CNN model recommended hyperparameters

Parameter name	Value(s)
Maximum number of words	40,42
Size of the vocabulary	25,000
Output dimensionality	140-400
Size of the kernel	3, 5, and 6
Activation functions	relu
Batch size	32
Pooling types	2, maximum
Number of epochs	20-50
Dropout rate probability	0.5
Optimizer	adam
Loss function	categorical_crossentropy
Filters	32
Filter Length	4
Number of nodes in Dense layer	32

Table 52 describes the BiLSTM models' recommended hyperparameters for the given datasets. We noted down the average execution time for these four different deep neural network models. The average training time for the CNN or sequential CNN, Multichannel CNN, BiLSTM, and BiLSTM with an attention layer models was 2-8 minutes, 2-10 minutes, 1.5 hours, and 2 hours respectively, on the Google colab pro with GPU platform.

Table 52. BiLSTM models' recommended hyperparameters

Parameter name	Value(s)
Maximum number of words	40,42
Number of classes for output layer	2
Size of the vocabulary	25,000
Output dimensionality for LSTM	140-400
Activation functions	relu
Batch size	64
Number of epochs	20-40
Dropout keep probability	0.5
Optimizer	adam
Loss function	categorical_crossentropy
Number of LSTM nodes	256
Number of Dense nodes	128

4.6 Model comparison

As we detailed in the previous section, all the models were compared with various performance metrics, such as accuracy, precision, recall, F_1 score, and etc. Table 51 describes the models' performances using accuracy metrics utilizing a test dataset from the BASE_MODEL and WITH_EMOTICON experiments. While comparing all the 20 models for both experiments, the BiLSTM with an attention layer and pre-trained fastText embedding vector model achieved the highest classification accuracy for the given datasets. The BASE_MODEL dataset produced around 83.02% accuracy and the WITH_EMOTICON dataset returned 83.56% accuracy for the BiLSTM with an attention layer and pre-trained fastText embedding vector model.

Table 53. Models with accuracy for BASE_MODEL and WITH_EMOTICON datasets

	Accuracy (in %)		**Difference (in %)**
	Base Model	With Emoticon	
BiLSTM - attention + pre-trained fastText	83.02	83.56	0.65%
BiLSTM - attention + pre-trained GloVe	82.29	83.38	1.31%
BiLSTM - attention + pre-trained word2vec (Twitter data)	83.18	83.27	0.11%
BiLSTM - pre-trained word2vec (Twitter data)	82.26	82.75	0.59%
BiLSTM - pre-trained fastText	82.31	82.73	0.51%
BiLSTM - pre-trained GloVe	81.91	82.7	0.96%
BiLSTM - attention + pre-trained word2vec (Google news)	81.65	82.67	1.23%
BiLSTM - attention + custom word2vec	81.21	81.52	0.38%
BiLSTM - custom word2vec	80.48	81.07	0.73%
BiLSTM - pre-trained word2vec (Google news)	80.86	81.03	0.21%
MultiCNN - pre-trained fastText	80.15	80.57	0.52%
MultiCNN - pre-trained GloVe	79.97	80.1	0.16%
MultiCNN - pre-trained word2vec (Twitter data)	79.59	80	0.51%
CNN - pre-trained fastText	77.21	79.76	3.20%
CNN - pre-trained word2vec (Twitter data)	77.23	79.67	3.06%
MultiCNN - custom word2vec	78.93	79.38	0.57%
CNN - pre-trained GloVe	77.78	79.37	2.00%
CNN - custom word2vec	77.22	78.42	1.53%
CNN - pre-trained word2vec (Google news)	76.21	77.56	1.74%
MultiCNN - pre-trained word2vec (Google news)	77.15	77.38	0.30%

We noticed that all of the models' performances were better for the WITH_EMOTICON dataset than the BASE_MODEL dataset. While comparing the

models with other metrics, such as the confusion matrix, class level precision, recall, and F_1 score, all the WITH_EMOTICON dataset trained model performances and accuracy were better. The BiLSTM with an attention layer models were in the top 10 lists for the given datasets. We also identified that the BiLSTM models performed significantly better for this sentiment classification than the CNN and multichannel CNN models. The word2vec (twitter), GloVe, and fastText pre-trained embedding vectors contributed better results than the pre-trained word2vec (Google news).

4.7 Selected model

To get a better insight into the best quantification approaches, we considered the confusion matrix, precision, recall, F_1 score, support, accuracy, macro average, and micro average performance metrics across all the models. The BiLSTM model with an attention layer and pre-trained fastText embedding layer achieved the best results among the tested models. This model accuracy metric is better than the baseline linear model accuracy metrics for both datasets. The precision, recall, F_1 score, and roc auc score for the BASE_MODEL experiments were 0.83, 0.83, 0.83, and 0.9076 respectively. The precision, recall, F_1 score, and roc auc score for the WITH_EMOTICON experiments were 0.82, 0.82, 0.84, and 0.9035 respectively. Those results were better than all of the other deep learning models for the current datasets.

In the selected model, the positive and negative classes' F_1 score was higher for this model than the other models in this research. Hence, we recommend the BiLSTM attention layer with pre-trained fastText embedding vector model as the best model for the BASE_MODEL as well as WITH_EMOTICON datasets.

4.8 Summary

This chapter presented twenty sentiment classification models for the BASE_MODEL and WITH_EMOTICON datasets. The first step was to train a model using the training datasets, and the second step was to use the test datasets to determine the model's performance. We evaluated the models using several performance metrics, such as the confusion matrix, accuracy, precision, recall, F_1 score, and macro-average for these two datasets separately. We identified that BiLSTM models performed better than CNN models for this sentiment classification.

We also discovered that the dataset with emoticons resulted in a slightly better performance than the dataset without emoticons. Primary results show that a BiLSTM model with an attention layer and pre-trained fastText embedding vector has the potential to classify sentiment classification with acceptably high accuracy. Therefore, we decided on this BiLSTM model as the best model when classifying the sentiment labels as positive and negative.

5 Conclusions, and Future work

The growth of social media has allowed customers to post their feedback on products and services. Posted opinions contain vital information for businesses and governmental organizations because they can steer marketing campaigns and help decision makers sense the public's wishes on events such as elections or product promotions. However, with the massive volume of data and different types of signals coming from customers, including things like slang and emoticons, extracting and classifying the sentiments of the comments is too complex a task to be done manually. NLP applications and tools can help in this regard, and many different approaches have been offered to address this problem. This research considered posts written in English that contained emoticons.

For addressing this issue, we used Twitter data and considered emoticons in order to determine the sentiment polarity within two classes: POSITIVE and NEGATIVE. We experimented with the Twitter dataset using two experiments, building one with emoticons as English language text, and another with the dataset's emoticons replaced with unknowns. These two datasets were trained and tested with a variety of CNN and LSTM neural network models. Those models made use of five different embedding vectors such as custom word2vec, pre-trained word2vec, GloVe, and fastText. Finally, we compared the model performance using a few different performance metrics, such as the confusion matrix, accuracy, precision, recall, F_1 score, and macro-average for each model.

After reviewing the performance metrics for all the models, we concluded that the BiLSTM model with the attention layer that used the pre-trained fastText embedding vector produced better classification for these two datasets, with an accuracy above 83%. We also found that emoticons add more value when determining sentiment classifications.

Since there are more users who are providing their comments and feedback using emoticons, we should consider those signals when identifying a customer's emotions.

There are specific areas that are of interest and relevance to further sentiment analysis, such as additional signals with a similar impact to emoticons in the text. Our future work includes the following:

a. Considering multi-language user feedback or comments on products or services.
b. Converting all the other signals, such as images, audio, and video clips, in addition to text for sentiment classifications.
c. Developing models that can detect sarcasm. Sarcasm is of special interest to us because of its complex nature and because it is common on social media.

6 References

Baroni, M., Dinu, G., & Kruszewski, G. (2014). Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. *In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Vol.1, pp. 238-247).

Baziotis, C., Pelekis, N., & Doulkeridis, C. (2017, August). Datastories at semeval-2017 task 4: Deep lstm with attention for message-level and topic-based sentiment analysis. *In Proceedings of the 11th International Workshop on Semantic Evaluation* (SemEval-2017) (pp. 747-754).

Cambria, E., Poria, S., Gelbukh, A., & Thelwall, M. (2017). Sentiment analysis is a big suitcase. *IEEE Intelligent Systems*, 32(6), 74-80.

Chen, X., Qiu, X., Zhu, C., Liu, P., & Huang, X. (2015). Long short-term memory neural networks for chinese word segmentation. *In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1197-1206).

Cliche, M. (2017). Bb_twtr at semeval-2017 task 4: Twitter sentiment analysis with cnns and lstms. *arXiv preprint arXiv:*1704.06125.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research,* 12(Aug), 2493-2537.

dos Santos, C., & Gatti, M. (2014). Deep convolutional neural networks for sentiment analysis of short texts. *In Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers* (pp. 69-78).

FastText. (n.d). *Pretrained model.* Retrieved from https://dl.fbaipublicfiles.com/fasttext/vectors-english/crawl-300d-2M.vec.zip

Giatsoglou, M., Vozalis, M. G., Diamantaras, K., Vakali, A., Sarigiannidis, G., & Chatzisavvas, K. C. (2017). Sentiment analysis leveraging emotions and word embeddings. *Expert Systems with Applications*, 69, 214-224.

GloVe. (n.d). *Pretrained model.* Retrieved from http://nlp.stanford.edu/data/GloVe.twitter.27B.zip

Go, A., Bhayani, R., & Huang, L. (2009). Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1(12), 2009.

Hochreiter, Sepp and J¨urgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9(8):1735–1780.

Hogenboom, A., Bal, D., Frasincar, F., Bal, M., de Jong, F., & Kaymak, U. (2013, March). Exploiting emoticons in sentiment analysis. *In Proceedings of the 28th Annual ACM Symposium on Applied Computing* (pp. 703-710). ACM.

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. *In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers) (Vol. 1, pp. 655-665).

Karpathy, A., Johnson, J., and Fei-Fei, Li (2016). *Visualizing and Understanding Recurrent Networks*. ICLR.

Keras. (n.d.a). *Biredirection LSTM*. Retrieved from https://keras.io/layers/wrappers/#bidirectional

Keras. (n.d.b). *Convolutional*. Retrieved from https://keras.io/layers/convolutional/

Keras. (n.d.c). *Tokenizer*. Retrieved from https://keras.io/preprocessing/text/

Keras. (n.d.d). *Embedding layer*. Retrieved from https://keras.io/layers/embeddings/

Keras. (n.d.e). *Pooling layer*. Retrieved from https://keras.io/layers/pooling/

Keras. (n.d.f). *Dense layer*. Retrieved from https://keras.io/layers/core/#dense

Keras. (n.d.g). *Compile*. Retrieved from https://keras.io/models/model/#compile

Keras. (n.d.h). *Fit*. Retrieved from https://keras.io/preprocessing/image/#fit

Keras. (n.d.i). *LSTM*. Retrieved from https://keras.io/layers/recurrent/#lstm

Keras. (n.d.j). *LSTM Attention Layer.* Retrieved from https://gist.github.com/cbaziotis/7ef97ccf71cbc14366835198c09809d2

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. *In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1746-1751).

Kiritchenko, S., Zhu, X., & Mohammad, S. M. (2014). Sentiment analysis of short informal texts. *Journal of Artificial Intelligence Research*, *50*, 723-762.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Nakov, P., Ritter, A., Rosenthal, S., Sebastiani, F., & Stoyanov, V. (2016). SemEval-2016 task 4: Sentiment analysis in Twitter. *Proceedings of SemEval*, 1-18.

Poria, S., Cambria, E., Howard, N., Huang, G. B., & Hussain, A. (2016). Fusing audio, visual and textual clues for sentiment analysis from multimodal content. *Neurocomputing*, 174, 50-59.

Rosenthal, S., Farra, N., & Nakov, P. (2017). SemEval-2017 task 4: Sentiment analysis in Twitter. *In Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)* (pp. 502-518).

Rosenthal, S., Nakov, P., Kiritchenko, S., Mohammad, S. M., Ritter, A., & Stoyanov, V. (2015, June). Semeval-2015 task 10: Sentiment analysis in twitter. In *Proceedings of the 9th international workshop on semantic evaluation (SemEval 2015)* (pp. 451-463).

Sentiment140. (2009). *For Academics*. Retrieved from http://help.sentiment140.com/for-students

Wang, H., & Castanon, J. A. (2015). Sentiment expression via emoticons on social media. arXiv preprint arXiv:1511.02556.

Wang, Y., Huang, M., & Zhao, L. (2016, November). Attention-based LSTM for aspect-level sentiment classification. *In Proceedings of the 2016 conference on empirical methods in natural language processing* (pp. 606-615).

Wehrmann, J., Becker, W., Cagnini, H. E., & Barros, R. C. (2017, May). A character-based convolutional neural network for language-agnostic Twitter sentiment analysis. *In Neural Networks (IJCNN), 2017 International Joint Conference on* (pp. 2384-2391). IEEE.

Wikipedia. (n.d). *List of emoticons*. Retrieved from https://en.wikipedia.org/wiki/List_of_emoticons

Word2vec. (n.d). *Pretrained model.* Retrieved from https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016, June). Hierarchical attention networks for document classification. *In Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies* (pp. 1480-1489).

Zhang, Y., & Wallace, B. (2017). A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *IJCNLP*.