Comparative Study of Deep Learning Models for Text Classification: LSTM, CNN with GloVe Embedding, and BERT

Hoda Fakharzadehjahromy
(hodfa840)
https://github.com/hodfa840/text-minig-project

Abstract

This project aimed to compare the performance of three deep learning models, namely Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Bidirectional Encoder Representations from Transformers (BERT) for classification of Yahoo Answers dataset. The models were trained and evaluated based on the performance metrics of accuracy, F1-score, and ROC-AUC score. The CNN and LSTM models utilized pre-trained GloVe word embeddings to improve their performance, while the BERT model used its own embedding mechanism. The experimental results demonstrated that the BERT model achieved the best performance with an accuracy of 0.73*, outperforming the CNN and LSTM models. However, it required a significantly longer training time compared to the other models. The LSTM and CNN models achieved comparable performance, with an accuracy of* 0.67 *and* 0.64*, respectively. In conclusion, the BERT model demonstrated superior performance in this task, but at the cost of longer training time, while the LSTM and CNN models could provide a balance between performance and computational efficiency.*

1. Introduction

In recent years, with the rapid growth of digital data, text classification has become a crucial problem in the field of natural language processing (NLP). Deep learning has shown remarkable success in various NLP tasks, including text classification. In this study, we explore the effectiveness of three popular deep learning models, namely Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Bidirectional Encoder Representations from Transformers (BERT), for text classification.

In particular, we use the Yahoo Answers dataset, which consists of over one million questions and their corresponding categories. To obtain vector representations of text data, we use pre-trained word embeddings such as GloVe and BERT embeddings. We then train the models using the vector representations and compare their performance in terms of accuracy, F1 score, and ROC AUC score.

Previous studies have shown that deep learning models have outperformed traditional machine learning algorithms in text classification tasks [1]. Moreover, pre-trained embeddings have been found to be effective in improving the performance of deep learning models. For instance, the use of pre-trained BERT embeddings [2] has shown state-of-the-art performance in various NLP tasks.

Therefore, this study aims to compare the effectiveness of different deep learning models for text classification and to investigate the impact of pre-trained embeddings on the performance of these models. Our findings could provide insights for developing more effective text classification models for practical applications.

The base model used in this study is logistic regression, which achieves an accuracy of 0.61.

2. Theory

Deep learning models are known for their ability to learn and extract complex features from raw data. In this study, various deep learning models have been used, such as the CNN with GloVe embeddings, LSTM with GloVe embeddings, and BERT with BERT embeddings. These models have been investigated in this study to evaluate their performance in text classification.

2.1. Convolutional neural networks

Convolutional neural networks (CNNs) have been used for text classification tasks, particularly for tasks that involve sequence-based inputs such as sentences or documents [1]. In CNNs, a filter or kernel slides over the input sequence to generate a feature map, which captures the presence of certain patterns or n-grams in the input. These feature maps are then fed into a pooling layer, which reduces the dimensionality of the input and further extracts important features. The pooling layer also prevents overfitting. Finally, the resulting features are passed through fully connected layers to produce the final classification output.

The convolution operation in CNN can be expressed as follows:

$$y_{i,j} = \sigma(\sum_{k,l} w_{k,l} x_{i+k-1,j+l-1} + b) \quad (1)$$

where x represents the input sequence, w is the filter, b is the bias term, σ is the activation function, i is the index of the output feature map, j is the index of the filter weights, and k is the filter size.

The pooling operation is typically either max pooling or average pooling, and can be expressed as follows:

$$y_{i,j} = \max_{k,l} x_{i+k-1,j+l-1} \quad (2)$$

or

$$y_{i,j} = \frac{1}{n} \sum_{k,l} x_{i+k-1,j+l-1} \quad (3)$$

where n is the size of the pooling kernel.

The fully connected layers at the end of the CNN can be expressed as follows:

$$y = \sigma(Wx + b) \quad (4)$$

where x is the input features, y is the output class probabilities, W is the weight matrix, b is the bias term, and σ is the activation function.

Overall, CNNs have shown to be effective in text classification tasks, especially when combined with pre-trained word embeddings such as GloVe or Word2Vec. An example of a CNN architecture for text classification is shown in Figure 1.

2.2. Long Short-Term Memory

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) that are widely used in natural language processing tasks like text classification. The main advantage of LSTMs over traditional RNNs is their ability to capture long-term dependencies in sequential data, making them particularly effective in tasks that involve analyzing sequences of text [3]. This is because RNNs are vulnerable to the problem of vanishing and exploding gradients as mentioned in [4].

LSTMs operate on individual elements of a sequence, such as words in the context of text classification. In addition to the usual output vector, LSTMs pass the old cell state to the next cell, which is also referred to as the *hidden state*. The next cell's inputs include both the hidden state and the second element of the sequence. By utilizing this hidden state, information can be preserved and transmitted to subsequent cells.

At the core of an LSTM network is the LSTM cell, which is responsible for processing the input sequence and updating the hidden state at each time step. The LSTM cell has several key components: the input gate, forget gate, output gate, cell state, and hidden state. The cell state is responsible for storing long-term information, while the hidden state stores short-term information. The input and forget gates control the flow of information into and out of the cell state, while the output gate controls the flow of information to the output. The forget gate is responsible for deciding how much of the previous cell state should be retained for the current cell. The input gate is responsible for deciding how much new information should be added to the cell state and the output gate is responsible for deciding how much of the cell state should be output at the current time step.

The LSTM cell can be mathematically represented as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad (5)$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (6)$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (7)$$
$$\tilde{c}t = \tanh(W_c x_t + U_c ht - 1 + b_c) \quad (8)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}t \quad (9)$$
$$h_t = o_t \odot \tanh(c_t) \quad (10)$$

where x_t is the input at time t, h_{t-1} is the hidden state at time $t-1$, f_t is the forget gate, i_t is the input gate, o_t is the output gate, $\tilde{c}_t$ is the candidate cell state, c_t is the current cell state, and h_t is the current hidden state. The W and U matrices are the weight matrices, and b is the bias vector. The σ function is the sigmoid function, and $\odot$ denotes element-wise multiplication.

Overall, the LSTM cell is able to selectively forget and retain information from the previous cell state, and add new information to the cell state based on the current input. This allows the LSTM network to learn and remember long-term dependencies in sequential data, making it a powerful tool for text classification tasks.

An example of using LSTM for text classification can be seen in the work of [5]. An unfolded LSTM architecture is illustrated in Figure 2.

Bidirectional LSTM (BiLSTM): is a type of LSTM that processes input sequences in both forward and backward directions, allowing the network to capture contextual information from both past and future time steps. The main advantage of BiLSTMs is their ability to capture long-term dependencies in both directions, making them particularly effective in tasks that involve analyzing sequences of [6].

2.3. GloVe (Global Vectors)

GloVe is a popular word embedding technique used in natural language processing tasks. GloVe embedding is a type of unsupervised learning method that captures the semantic and syntactic information of words based on their co-occurrence in a large corpus of text. GloVe embedding is widely used in tasks such as text classification, sentiment analysis, and machine translation.

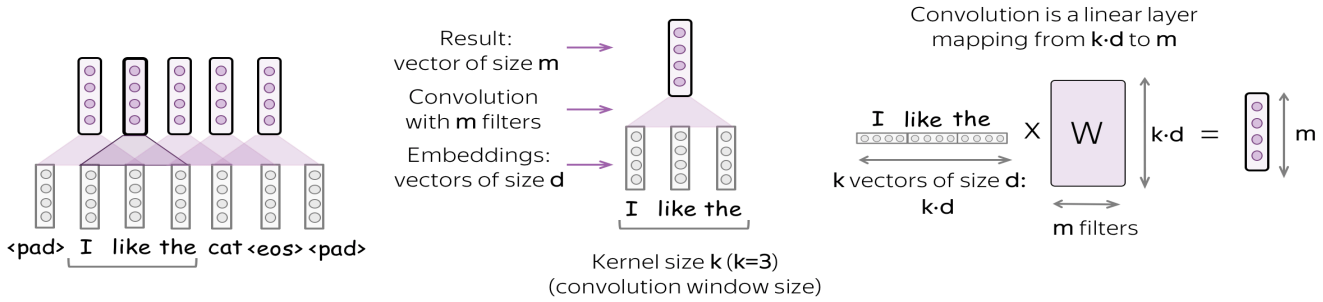


Figure 1. Illustrative Convolutional neural network Source.

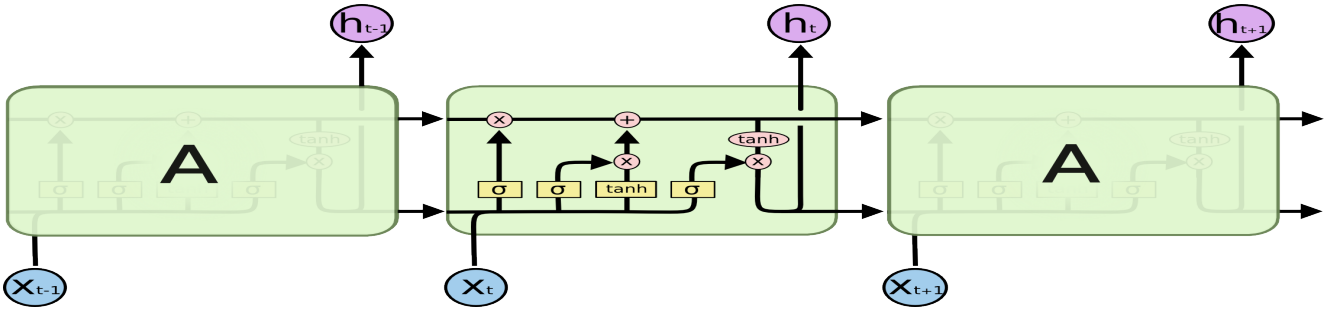


Figure 2. Unfolded LSTM showing the information flow. Source.

At the core of the GloVe algorithm is the co-occurrence matrix, which captures the frequency of word co-occurrence in a corpus. The co-occurrence matrix is then used to calculate the probability distribution of word co-occurrence using the following formula:

$$P_{ij} = \frac{X_{ij}}{\sum_k X_{ik}} \tag{11}$$

where P_{ij} is the probability that word j appears in the context of word i, and X_{ij} is the number of times word j appears in the context of word i. The denominator of the formula is the total number of times any word appears in the context of word i. This probability distribution is then used to calculate a weighted least squares objective function:

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(w_i^T \tilde{w}_j + b_i + \tilde{b_j} - \log(X_{ij}) \right)^2 \tag{12}$$

where V is the vocabulary size, $f(X_{ij})$ is a weighting function that downweights the importance of highly frequent co-occurrences, w_i and $\tilde{w_j}$ are the word vectors for word i and j respectively, and b_i and $\tilde{b_j}$ are the bias terms for words i and j respectively.

Once the GloVe model is trained on a large corpus, the resulting word embeddings can be used in downstream NLP tasks. In this study GloVe embedding has been used withs CNN and LSTM model.

2.4. Bidirectional Encoder Representations from Transformers (BERT)

BERT is a bidirectional transformer encoder used for natural language processing tasks that has achieved state-of-the-art results on several benchmark datasets [2]. It is trained on a large corpus of text using masked language modeling (MLM) and next sentence prediction (NSP). The MLM objective involves predicting masked words based on the surrounding context to learn contextual representations of words for downstream tasks. The NSP objective helps the model learn relationships between sentences and can be useful for tasks like question answering. BERT can handle both sentence-level and token-level tasks with a single model using special tokens [CLS] and [SEP]. It has been successful in a wide range of NLP tasks, including sentiment analysis, question answering, and natural language inference [7].

Figure Figure 3 shows an overview of the BERT architecture.

3. Data

The data set used in this project is The Yahoo! Answers dataset. The Yahoo Answers dataset is a popular dataset for natural language processing and machine learning research. It contains a large collection of questions and answers posted on the Yahoo Answers website between 2006 and 2010. The dataset was released by Yahoo! Research

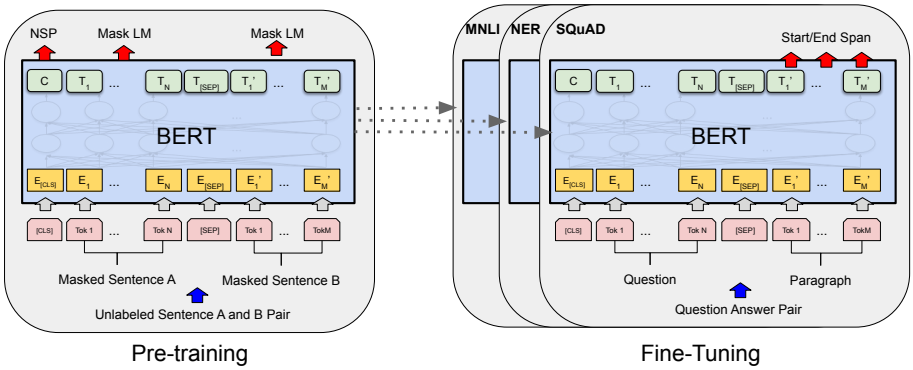


Figure 3. An overview of the BERT architecture: The pre-training and fine-tuning processes of BERT share the same architecture except for the output layers. The pre-trained model parameters are utilized to initialize the models for different downstream tasks. During the fine-tuning process, all the parameters are fine-tuned. Additionally, each input example is pre-pended with a special [CLS] symbol, while a special [SEP] token is used as a separator between questions and answers [2].

and has since been widely used in research projects.

The dataset consists of over 1.4 million questions and answers. It consists of 10 categories; Society & Culture, Science & Mathematics, Health, Education & Reference, Computers & Internet, Sports, Business & Finance, Entertainment & Music, Family & Relationships, and Politics & Government.

The dataset used for our research can be accessed at the following link. We downloaded the training set from this dataset and, due to resource limitations, selected a balanced subset consisting of 30% of the data. This subset was then divided into training, validation, and testing sets at a ratio of 70:15:15, respectively.

Each sample in the chosen dataset contains four columns: 'title', 'content', 'best answer', and 'class'. For the scope of our analysis, we chose to exclude the 'content' column. Instead, our models were constructed using the 'title' and 'best answer' columns as features, with the 'class' column serving as the target variable, indicating the category to which each sample belongs.

Figure 4 shows the class distribution for each category.

The preprocessing is performed by using the spaCy library [8].

In order to utilize a CNN and LSTM model for text analysis, the text data is preprocessed by constructing a dictionary consisting of class labels and their respective question-answer tuples. The subsequent step involves encoding the text data, which entails converting it into a numerical representation suitable for machine learning algorithms. This process is accomplished by tokenizing the text, assigning a unique integer value to each token, and padding each sequence of integers to a fixed length (with a maximum length increased by 10). Additionally, the label encoding step transforms each class label into a one-hot encoded vector.

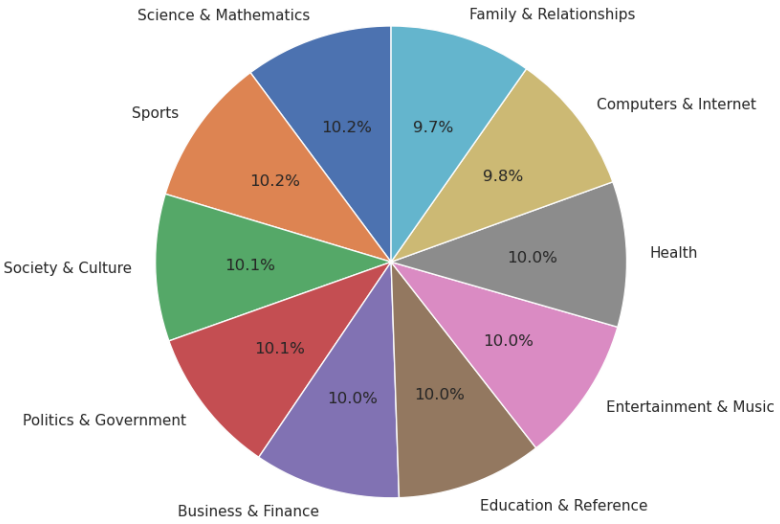


Figure 4. Class distribution.

After encoding the cleaned text, the next step is to create an embedding matrix for the words in the text using GloVe [9]. Specifically, we use the pre-trained GloVe word embeddings (glove.6B.100d) to create an embedding matrix, which will be used to initialize the embedding layer of the neural network model.

Tokenization for BERT is different from traditional tokenization methods because it uses a technique called WordPiece tokenization. In this method, words are broken down into subwords based on their frequency of occurrence in a large corpus of text. The most common words are kept as single tokens, while less common words are split into multiple subwords.

In this project we will use Ktrain library [10]. Ktrain

uses the FullTokenizer from the bert-for-tf2 library to tokenize text for BERT models.

4. Method

The models constructed for this research were specifically designed to operate on the Yahoo Answers dataset, performing multi-class text classification across ten distinct categories. Our chosen baseline model, a logistic regression, achieved a noteworthy accuracy of 0.61 on the test set.

For replicating the results the implementation can be found in the corresponding GitHub repository[1].

4.1. CNN

The Convolutional Neural Network (CNN) model was constructed using the Keras functional API, with input sequences of length 10. These sequences were transformed into dense, fixed-length vectors via an embedding layer. The weights of the embedding layer were initialized using pre-trained word embeddings from GloVe, for the following layers the initialization recommendations from Andrej Karpathy's "Neural Network Recipes" [11] has been used.

The CNN model consists of several layers. The convolutional layers and dense layers are initialized using the HeNormal initializer. The first convolutional layer consists of 112 filters with a kernel size of 3, followed by a dropout layer with a rate of 0.15. The second convolutional layer consists of 80 filters with a kernel size of 3, followed by another dropout layer with a rate of 0.45. The output is then flattened and passed through two dense layers, the first with 256 units and ReLU activation, and the second with 10 units and softmax activation, representing the final prediction. The model is compiled with the categorical cross-entropy loss function, Adam optimizer, and accuracy as the evaluation metric. The total number of parameters in the model is $10,319,258$.

For a detailed model depiction, refer to Figure 9a and Appendix A.

4.1.1 Hyperparameter Tuning

Hyperparameter optimization is a crucial aspect of machine learning, significantly affecting the model's performance. It facilitates achieving the highest possible model accuracy and computational efficiency. For the CNN model, we utilized the Keras Tuner library [12] to conduct hyperparameter tuning. This process entailed adjusting parameters such as the number of filters, kernel size, dropout rate, and learning rate. More details are depicted in Table 1.

To prevent overfitting and enhance the model's ability to generalize, we implemented regularization strategies such as early stopping and learning rate reduction. Early stopping, with a patience of 12 epochs, halts training when no further improvements in the model's validation performance are evident. Learning rate reduction decreases the learning rate when the model's validation performance plateaus, maintaining a balance between exploration and exploitation during the optimization process. The hyperparameter tuning process for CNN, including these regularization strategies, took approximately 3 hours to complete, reflecting a significant investment in model optimization and performance enhancement.

The hyperparameter values for the CNN model can be found in Appendix A.

4.2. LSTM

The Long Short-Term Memory (LSTM) model was implemented as the second model in our study. It is a type of recurrent neural network (RNN) known for its ability to capture long-term dependencies in sequential data.

The architecture of the LSTM model consists of multiple LSTM layers, bidirectional connections, dropout layers, and dense layers. The model is designed to process input sequences of length 10 and learn the temporal dependencies within the text data.

Similar to the CNN model, the LSTM model includes hyperparameters that need to be tuned to optimize its performance. These hyperparameters include the number of LSTM units, dropout rates, and the learning rate.

During the hyperparameter tuning process, the Keras Tuner library was utilized. The Hyperband algorithm, which combines random search with early stopping, was employed. Each trial in the tuning process was trained for a maximum of 50 epochs. The best hyperparameters were selected based on the validation accuracy of the model.

The number of LSTM units, which determines the capacity and complexity of the LSTM layers, was tuned using the 'Int' hyperparameter. The range explored was from 32 to 512, with a step size of 32. This allowed us to find the optimal number of units for capturing the temporal dependencies in the input sequences.

The dropout rate, a L2 regularization technique that helps prevent overfitting, was tuned using the 'Float' hyperparameter. The range explored was from 0.0 to 0.5, with a step size of 0.05. By tuning the dropout rate, we aimed to strike a balance between model complexity and regularization more detail are shown in Table 1.

To prevent overfitting during training, early stopping was implemented with a patience of 12 epochs. If the model's validation performance did not improve for 12 consecutive epochs, the training was stopped early. This helped us find the optimal hyperparameters while avoiding unnecessary computational costs.

The hyperparameter tuning process for the LSTM model

[1] Projet link

Model	Hyperparameter	Range
CNN	Filter 1	[32, 128], step=16
	Kernel 1	[3,5]
	Dropout 1	[0.0, 0.5], step=0.05
	Filter 2	[32, 128], step=16
	Kernel 2	[3,5]
	Dropout 2	[0.0, 0.5], step=0.05
	LR	[1e-4, 1e-2] (log scale)
LSTM	LSTM 1	[32, 512], step=32
	Dropout 1	[0.0, 0.5], step=0.05
	LSTM 2	[32, 512], step=32
	Dropout 2	[0.0, 0.5], step=0.05
	LSTM 3	[32, 512], step=32
	Dropout 3	[0.0, 0.5], step=0.05
	Dense	[32, 512], step=32
	Dropout 4	[0.0, 0.5], step=0.05
	LR	[1e-2, 1e-3, 1e-4]
BERT	LR	Find optimal value

Table 1. Hyperparameters tuning for CNN, LSTM, and BERT

took approximately 23 hours due to the extensive search space and the need to train multiple models. Total parameters for this model is $28,253,870$. The model diagram is shown in Figure 9b and information about the hyperparameter values can be found in Appendix A.

Table 1 shows the range of values used for hyperparameter optimization in the CNN, LSTM, and BERT models.

4.3. BERT

BERT, a pre-trained language model, can be fine-tuned for various natural language processing tasks by adjusting hyperparameters and updating the model on task-specific datasets [2]. In this study, due to resource limitations, the focus was primarily on optimizing the learning rate. The learning rate finder plot, generated using the ktrain library [13], revealed three potential learning rate values (Figure 5). The longest valley in the plot, corresponding to a learning rate of 9.53E-06, indicated a stable region with a low and flat loss [14]. This learning rate was chosen as it strikes a balance between convergence and stability during training. While other hyperparameters and layer configurations can be explored, given the constraints, the emphasis was placed on optimizing the learning rate. Thus, the learning rate of 9.53E-06 was selected to fine-tune the BERT model for the text classification task, maximizing performance within the available resources [14].

It is important to note that the choice of learning rate is highly dependent on the specific dataset, task complexity, and model behavior. Additional experiments and evaluations should be conducted to validate the performance of the selected learning rate and fine-tuned BERT model on the target text classification task.

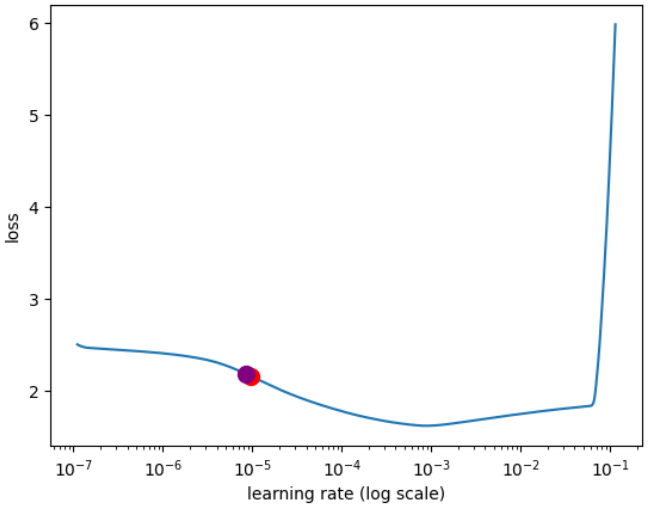


Figure 5. Learning Rate Finder Plot for Bert model. The red dot is chosen as the optimal learning rate

5. Results

The results of our experiments are presented in Table 2. The CNN model achieved an accuracy of 0.64, an F1 score of 0.64, and a ROC-AUC score of 0.91. The LSTM model achieved an accuracy of 0.68, an F1 score of 0.67, and a ROC-AUC score of 0.93. The BERT model outperformed both CNN and LSTM models, achieving an accuracy of 0.73, an F1 score of 0.73, and a ROC-AUC score of 0.95.

In terms of running time, the CNN model was the fastest, since it has less parameters. The LSTM model took approximately 1 hour to run for 13 epochs due to the early stopping layer. The BERT model took the longest, running for 7 hours to complete 12 epochs.

Table 2. Performance Metrics for CNN, LSTM, and BERT

Model	Performance Metrics		
	Accuracy	F1 Score	ROC-AUC Score
CNN	0.64	0.64	0.91
LSTM	0.68	0.67	0.93
BERT	**0.73**	**0.73**	**0.95**

The visualization report (Figure 6) presents a comprehensive analysis of the training and validation performance of CNN, LSTM, and BERT models. The accuracy and loss curves (Figure 6a, Figure 6b, Figure 6c, Figure 6d, Figure 6e, Figure 6f) demonstrate the models' learning progress during training, reflecting improvements in accuracy and reduction in loss on training set.

The accuracy curves demonstrate the models' ability to correctly classify the data as training progresses. By observing the changes in accuracy, we can assess the models'

learning capabilities and their performance on unseen data.

In addition to accuracy and loss, the visualization report also includes ROC curves (Figure 7a, Figure 7b, Figure 7c) and confusion matrices for the unseen test data (Figure 8a, Figure 8b, Figure 8c). The ROC curves provide insights into the models' classification performance across different thresholds, helping to evaluate their ability to discriminate between classes. The confusion matrices visualize the distribution of correctly and incorrectly classified instances across different classes, providing a deeper understanding of the models' performance on individual classes.

6. Discussion

The results of our experiments indicate that the BERT-based model outperformed both the CNN and LSTM models in terms of accuracy, F1 score, and ROC-AUC score. However, it is important to note that the BERT model also had a significantly longer running time compared to the CNN and LSTM models.

Current state-of-the-art model for Yahoo Answer dataset classification is achieved by BERT-ITPT-FiT model [15] and is about 77% accuracy. We achieved 73% accuracy by using a single GPU NVIDIA 3070 RTX. Due to resource constraint we dropped the 'content' column. We expect our final model improve by including this feature in the model.

As often observed in machine learning, the larger the model in terms of the number of parameters, the more prone it is to overfitting. This trend is evident in the learning curves for these models, depicted in Figures 6a to 6f. For the CNN model, as shown in Figure 6d, overfitting started to occur after about 6 epochs, demonstrated by a divergence in training and validation loss. Similarly, in the LSTM model, overfitting was observed after around 2 epochs as evidenced in Figure 6e. However, the BERT model, which has the largest number of parameters, showed the quickest onset of overfitting, with the model starting to overfit after just a single epoch as can be seen in Figures 6c and 6f.

Further evidence of overfitting can be seen by examining the accuracy curves. During training, the accuracy for the training set typically increases, while it decreases or plateaus for the validation set. This discrepancy is an indication of overfitting and can be observed in the accuracy curves for the CNN, LSTM, and BERT models (Figures 6a to 6c).

To manage the overfitting problem, an early stopping callback was used, which was monitoring the validation accuracy with a patience of 12 epochs. This means that the training would stop if the validation accuracy did not improve for 12 consecutive epochs, effectively limiting the overfitting.

Upon evaluation of the ROC curves and confusion matrices for the test set, as depicted in Figures 7a to 7c and 8a to 8c, a consistent trend becomes apparent across all models. Each model demonstrated the highest prediction accuracy for the 'Politics & Government' category and the lowest accuracy for the 'Health' category. Various factors could potentially account for this observation. It might be the result of certain distinct features inherent to the dataset, the effectiveness of the model in deciphering complex patterns related to each category, or perhaps the presence of specific keywords mentioned by users that the model has learned to associate with a particular category. Additional exploration is needed to fully comprehend the underlying causes of these trends and to enhance the model's performance across diverse categories. The overall results and performance metrics can also be visualized as shown in Figure 6.

When balancing speed and precision, both CNN and LSTM models offer a sensible compromise. While they don't reach the same accuracy levels as BERT, they still deliver commendable results with much less training time. This is particularly useful in applications that need real-time or near-real-time processing.

However, BERT's higher accuracy signifies a much greater capacity to learn complex patterns within data. Therefore, for tasks that prioritize depth of understanding over speed, BERT may be more suitable. Ultimately, the choice between these models should depend on the specific needs of the task.

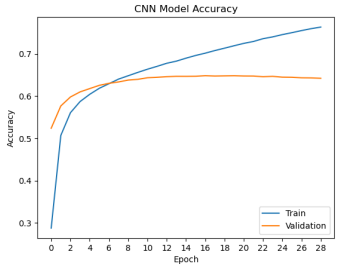
6.1. Limitations

The current study, which offers a comparative analysis of LSTM, CNN with GloVe Embedding, and BERT models for text classification, does come with its set of limitations that need to be taken into account when interpreting the results.

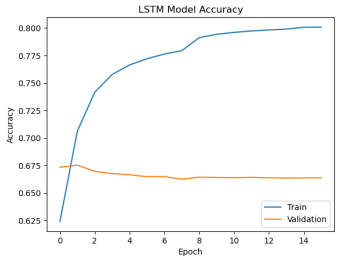
One significant limitation is tied to the dataset size. Due to computational constraints, the study only managed to utilize a mere 30% of the entire dataset. The limited dataset might not represent the complete data distribution accurately, which could limit the broader applicability of the results. A larger, more comprehensive dataset might improve the models' performance and influence the choice of optimal hyperparameters.

Importantly, the study did not employ cross-validation during hyperparameter tuning. This absence could inadvertently bias the model performance assessment towards a more favorable result, especially if the validation set was inherently easier for the models to predict. Despite the robustness that cross-validation methods, such as k-fold cross-validation, offer, they were not employed due to their requirement for significantly more computational resources and time, given the need for k times more trials.

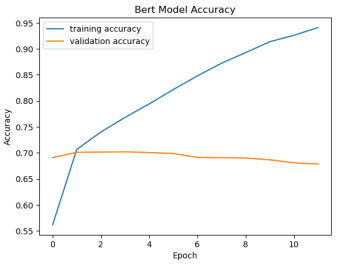
The research also faced resource limitations, which affected the extent of hyperparameter tuning trials, the maximum number of training epochs, and the application of ad-



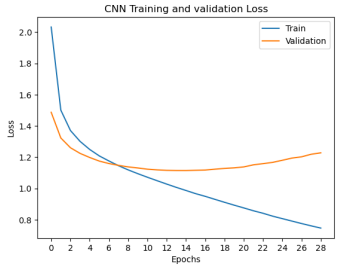
(a) Training & Validation Accuracy for CNN



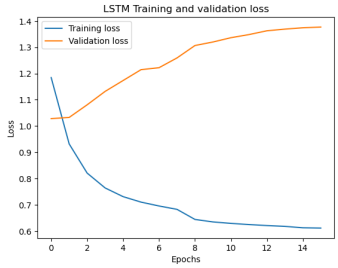
(b) Training & Validation Accuracy for LSTM



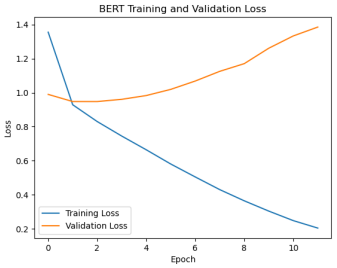
(c) Training & Validation Accuracy for BERT



(d) Training & Validation Loss for CNN

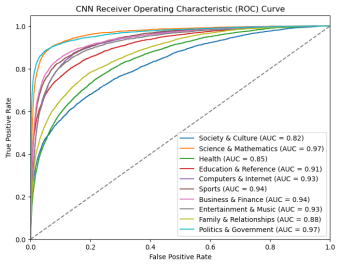


(e) Training & Validation Loss for LSTM

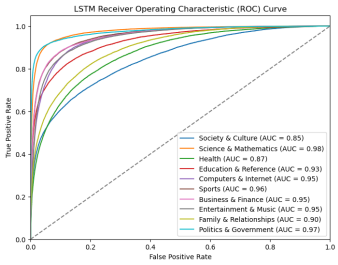


(f) Training & Validation Loss for BERT

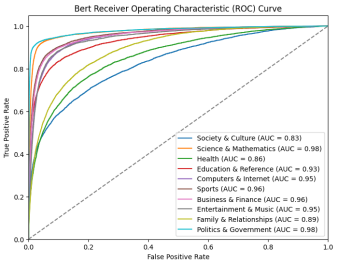
Figure 6. Visualization Report for CNN, LSTM, and BERT on Test set



(a) ROC curve for CNN



(b) ROC curve for LSTM



(c) ROC curve for BERT

Figure 7. ROC curves for CNN, LSTM, and BERT

vanced tuning methodologies like Bayesian Optimization. These constraints precluded the use of multiple data splits for testing.

In addition, the experiments were conducted on a single test set. To enhance the generalizability and credibility of the results, experiments should ideally be repeated with various random seeds, with the outcomes averaged.

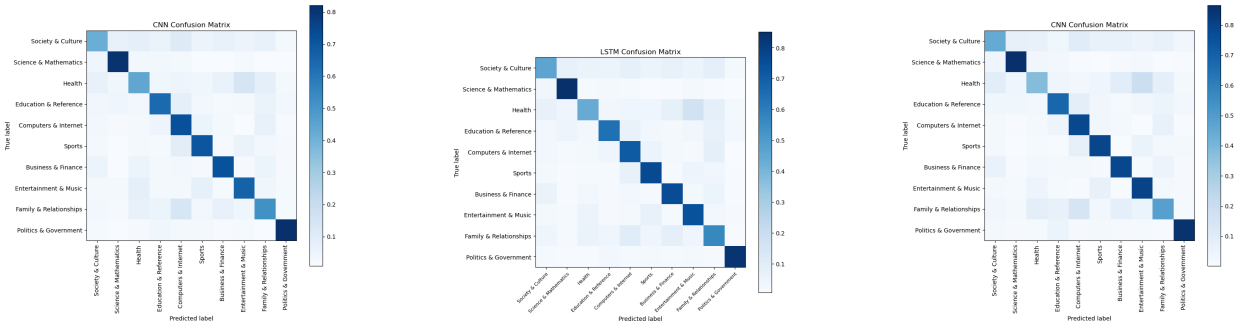
The hyperparameter tuning, particularly for the BERT model, might not have been exhaustive. A broader exploration of hyperparameters, including the number of epochs, batch sizes, and sequence lengths, could potentially improve the models' performance.

Lastly, the study focused on comparing only three types of deep learning models: CNN, LSTM, and BERT. There are numerous other model architectures and variations that might also be effective for text classification, such as Transformer models, GRUs, or Capsule Networks [16]. The limited model diversity in this study could be a potential limitation.

Future research in this area should aim to address these limitations, including increasing the computational resources, utilizing a more extensive and representative dataset, employing advanced and more comprehensive hyperparameter tuning methodologies, adopting multiple data splits for testing, repeating experiments with various random seeds, and exploring a broader range of model architectures.

7. Conclusion and Future Work

This project examined the efficacy of three deep learning models: CNN, LSTM, and BERT, for text classification tasks on the Yahoo Answers dataset. Our experiments demonstrated superior performance from the BERT model in terms of accuracy, F1 score, and ROC-AUC score, albeit with increased training time. Both LSTM and CNN models utilized GloVe word embeddings, enhancing their



(a) Confusion Matrix for CNN (b) Confusion Matrix for LSTM (c) Confusion Matrix for BERT

Figure 8. Confusion Matrices for CNN, LSTM, and BERT

performance, while BERT used its specific pre-trained embeddings.

Notwithstanding, deep learning models can be computationally intensive, necessitating considerable computational resources. However, their effectiveness in text classification tasks is evident, underscoring the importance of appropriate model and embedding selection.

Moving forward, this project can stimulate further exploration in the field of text mining and deep learning. Anticipated future work includes:

- Exploring other pre-trained language models like OpenAI's GPT-4 and RoBERTa, which could offer improved results or efficiency.
- Experimenting with different hyperparameters and architectures, potentially optimizing the models' performance.
- Utilizing larger or more diverse datasets, aiding in model generalization.
- Incorporating the 'content' column as another feature to enhance model performance.

The ever-growing data volume and rapid advancements in deep learning technology amplify the potential for deep learning in text mining, making this research area particularly exciting.

References

[1] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015. 1

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. 1, 3, 4, 6

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. 2

[4] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2012. 2

[5] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*, pages 207–212, 2016. 2

[6] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018. 2

[7] Songwen Pei, Lulu Wang, Tianma Shen, and Zhong Ning. Da-bert: Enhancing part-of-speech tagging of aspect sentiment analysis using bert. In *Advanced Parallel Processing Technologies: 13th International Symposium, APPT 2019, Tianjin, China, August 15–16, 2019, Proceedings 13*, pages 86–95. Springer, 2019. 3

[8] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017. 4

[9] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014. 4

[10] Arun Ammar. ktrain: A python library for deep learning with keras. https://github.com/

amaiya/ktrain, 2021. Accessed: March 13, 2023. 4

[11] Andrej Karpathy. Neural network recipes. Blog post, 2019. 5

[12] Tom O'Malley et al. Keras tuner. GitHub repository, 2021. 5

[13] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018. 6

[14] Leslie N. Smith. Learning rate finder. *Fastai*, 2019. 6

[15] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune BERT for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019. 7

[16] Wei Zhao, Jianbo Ye, Min Yang, Zeyang Lei, Zhangming Chan, and Qiang Ding. Investigating capsule networks with dynamic routing for text classification. *arXiv preprint arXiv:1804.00538*, 2018. 8

A. Hyperparameter Values

The following hyperparameter values were used for the CNN model:

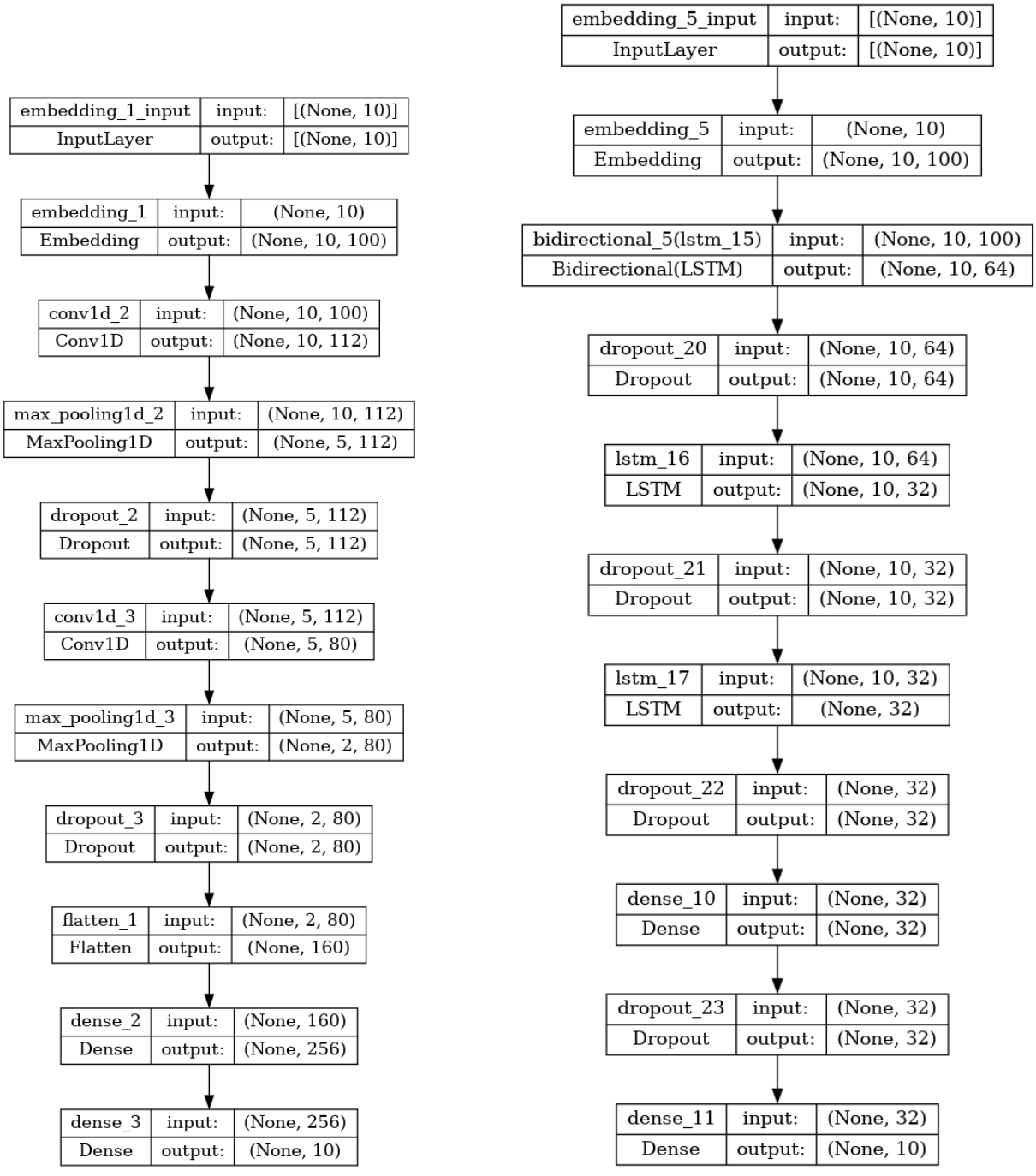
```
{
    "number_of_filters_1": 112.0,
    "kernel_size_1": 3.0,
    "dropout_rate_1": 0.15,
    "number_of_filters_2": 80.0,
    "kernel_size_2": 3.0,
    "dropout_rate_2": 0.45,
    "learning_rate": 0.00043
    "batch_size": 2048
  }
```

The following hyperparameter values were used for the LSTM model:

```
{
    "units_lstm_1": 448.0,
    "dropout_1": 0.45,
    "units_lstm_2": 416.0,
    "dropout_2": 0.25,
    "units_lstm_3": 32.0,
    "dropout_3": 0.25,
    "units_dense": 384.0,
    "dropout_4": 0.2,
    "learning_rate": 0.01
    "batch_size": 2048

    }
```

B. Model Architecture



(a) CNN Diagram

(b) LSTM Diagram

Figure 9. Model architecture