**Fake News Detection**

**Yonatan Ben Avraham**

**Abstract**

In today’s era of AI-driven content creation, the spread of fake news poses a substantial threat to society and individuals alike, with ramifications ranging from economic burdens to psychological harm and even life-threatening situations. Consequently, there is a pressing need for tools capable of discerning reliable information to reduce the expansion of false reports. This paper addresses that challenge by presenting two distinct modeling approaches: a classic Machine Learning method based on Support Vector Machines (SVM) and a Deep Learning solution using LSTM networks. First, these models are compared. Subsequently, the performance of the LSTM model is thoroughly examined by exploring various hyperparameters and evaluating distinct architectural configurations of the network. Together, these investigations provide insights into which configurations and design choices hold the most promise for mitigating the fake news phenomenon.

**Keywords:** Fake News Detection, Classification, ML, SVM, DL, LSTM, WELFake.

**Introduction**

The wide expansion of fake news has become a critical issue in the digital age, particularly with the rise of social media platforms that enable the rapid spread of misinformation. Fake news can have severe consequences, ranging from political manipulation and economic disruptions to public health crises. Given the limitations of manual fact-checking and the sheer volume of online content, automated fake news detection has gained significant research attention in recent years.

Machine Learning (ML) and Deep Learning (DL) techniques have emerged as prominent solutions for addressing this challenge. Traditional ML models, such as Support Vector Machines (SVM), Naïve Bayes, and Decision Trees, have been widely used to classify news articles based on linguistic and statistical features. However, these models heavily rely on feature engineering and may struggle with complex textual patterns. In contrast, DL models, particularly Long Short-Term Memory (LSTM) networks, can capture sequential dependencies in text, making them well-suited for analyzing news content.

This paper explores two distinct approaches for fake news detection: an SVM-based classification model and an LSTM-based deep learning model. First, a comparative analysis is conducted to evaluate the performance of these models using key evaluation metrics. Next, the LSTM model is further optimized by exploring various hyperparameter configurations and architectural adjustments. The goal of this study is to identify effective configurations for enhancing fake news detection accuracy while maintaining model interpretability and efficiency.

**Literature Review**

Fake news detection has been a growing research area in recent years, with various approaches developed to improve accuracy and robustness. Earlier methods primarily relied on traditional machine learning (ML) algorithms such as Support Vector Machines (SVM), Naïve Bayes, Random Forest, and Decision Trees to classify fake and real news based on linguistic and statistical features [1]. These models performed well when trained on manually engineered features such as TF-IDF representations, sentiment scores, and n-gram analysis, but their effectiveness depended heavily on feature selection [2], [3].

With advancements in deep learning (DL), researchers have increasingly adopted Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, which excel at capturing sequential dependencies in text [4], [5]. LSTM-based models have shown superior performance compared to traditional ML models, particularly when dealing with long and complex news articles. Some studies have demonstrated that CNN models can also be effective, extracting local textual patterns for classification [6]. A comprehensive review by [7] discusses various deep learning-based techniques for fake news detection, including attention mechanisms, generative adversarial networks, and bidirectional encoder representations for transformers. Their analysis highlights the advantages of deep learning, such as automated feature extraction and improved classification accuracy, while also identifying key challenges like dataset limitations and the need for robust evaluation metrics.

Hybrid approaches that combine traditional ML classifiers with deep learning-based text representations (e.g., BERT embeddings with SVM or TF-IDF with LSTM) have been explored to improve accuracy while maintaining computational efficiency [8], [9]. In addition, some studies have investigated social context features, incorporating user behavior, source credibility, and network propagation patterns into the detection process [1].

Despite these advancements, SVM and LSTM remain among the most widely used methods due to their balance between interpretability and performance [10]. However, recent research highlights the need for robust models resistant to adversarial attacks, as well as better generalization to real-world misinformation [11]. Future research should focus on combining ML techniques with DL architectures, improving model explainability, and integrating external knowledge sources (e.g., fact-checking databases and user credibility signals) to enhance reliability. Given the rapid evolution of misinformation, ongoing refinements in feature engineering, data augmentation, and model robustness remain crucial for improving fake news detection systems [12].

**Methodology**

This section outlines the methodology employed in this project, which was organized into several main phases. In the first phase, the selecteddataset, the working environment, and the Exploratory Data Analysis (EDA) performed on the data are presented. The second phase describes the development of a Support Vector Machine (SVM) model. The third phase details the construction of an LSTM model configured with defaulthyperparameters, followed by an assessment of its test set performance and a comparative analysis of the results from the SVM and LSTM modelsusing various evaluation metrics. The fourth phase comprises an investigation of several different LSTM architectures with varying hyperparameters, including variations in the

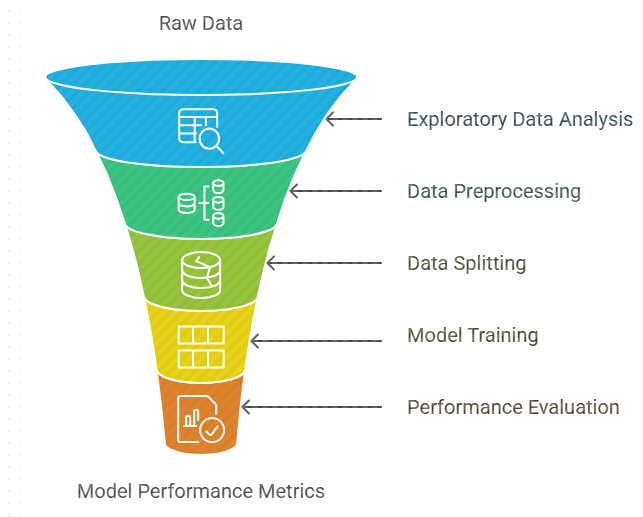
 number of data records, embedding dimensions, class balance levels, and architectural modifications—with the most "successful" model (according to the considered performance metrics) among those examined being subsequently presented. The following sections provide a detailed discussion of each phase.

Figure 1

1. Initially, the WELfake dataset was selected for this study. This dataset comprises a collection of news articles, with each record containing columns for the article headline, article text, and a label indicating whether the news is genuine or fabricated. Subsequently, the working environment was established using Python, and an extensive exploratory data analysis (EDA) was conducted. The EDA involved examining the number and percentage of records with missing values, identifying duplicate articles, detecting articles containing invalid characters, filtering out non-English articles, determining the longest articles within the dataset, calculating the average article length, and analyzing the distribution of the labels.
2. The process of building SVM model comprises an initial preprocessing phase to refine the dataset, followed by partitioning the cleansed data into training, validation, and test sets. Subsequently, the model is trained using the training set while its performance is monitored on the validation set, and finally, the efficacy of the trained model is evaluated on the test set using a variety of performance metrics.

The Preprocessing steps are described as follows:

1. Removal of Null Records: Eliminate any rows with missing values.
2. Removal of Duplicate Texts: Discard duplicate entries to prevent redundant information from biasing the model.
3. Removal of Articles Containing HTML Tags.
4. Removal of Non-English Articles: Exclude articles not written in English to maintain language consistency throughout the dataset.
5. Expansion of Contractions: Convert contractions (e.g., "can't" to "cannot") to their full forms to standardize the text.
6. Conversion of texts to Lowercase.
7. Removal of punctuation and whitespaces: Eliminate punctuation marks and redundant whitespace to focus on the meaningful content.
8. Removal of Stop Words: Filter out common words that do not contribute significant meaning to improve the quality of features.
9. Lemmatization: Reduce words to their base or dictionary form to normalize different inflected variants and enhance consistency.
10. Concatenation of Title and Text: For each record, merge the article title and its main text into a single string to create a unified input.
11. Generation of Word Embeddings**:** For each token (word) in the concatenated string, generate an embedding using the "glove-wiki-gigaword-50" model.
12. Averaging of Embeddings: Compute the average of all word embeddings for each record to obtain a single representative vector.
13. Saving Preprocessed Data: Store all the preprocessed data (features and labels) in NPZ file.

Following these steps, the NPZ files are loaded, and the data is split into training, validation, and test sets. Thereafter, cross-validation is performed, and the model is trained using an RBF kernel. Finally, the model’s performance is evaluated on the test set.

1. The process of building the LSTM model consists of several key stages. First, the data goes thorough preprocessing to ensure it is suitable for modeling. Next, the preprocessed data is split into training, validation, and test sets. During model training, performance is continuously monitored on the validation set. Finally, the performance of the trained model is evaluated on the test set using the same set of performance metrics employed for the SVM model, thereby assessing its generalizability and enabling a direct comparison with the SVM’s results.

The Preprocessing steps are described as follows:

1. Removal of Null Records: Eliminate any rows with missing values.
2. Removal of Duplicate Texts: Discard duplicate entries to prevent redundant information from biasing the model.
3. Removal of Articles Containing HTML Tags.
4. Removal of Non-English Articles: Exclude articles not written in English to maintain language consistency throughout the dataset.
5. Expansion of Contractions: Convert contractions (e.g., "can't" to "cannot") to their full forms to standardize the text.
6. Conversion of texts to lowercase.
7. Removal of punctuation and whitespaces: Eliminate punctuation marks and redundant whitespace to focus on the meaningful content.
8. Removal of Stop Words: Filter out common words that do not contribute significant meaning to improve the quality of features.
9. Lemmatization: Reduce words to their base or dictionary form to normalize different inflected variants and enhance consistency.
10. Conversion to sequences: Using a tokenizer that maps each word to a unique numerical value, the texts are transformed into sequences of numbers. For sequences that are shorter than the predetermined length, padding is applied to ensure consistent input dimensions.
11. Saving Preprocessed Data: Store all the preprocessed data (features and labels) in NPZ file.

After these steps, the NPZ files are loaded, 90% of the preprocessed data is split into training, validation, and test sets. Next, the model is trained using default hyperparameters: training is conducted for 1 epoch with a batch size of 32 and the RMSprop optimizer. The model architecture comprises an Embedding layer with a vocabulary size of 20,000 and an embedding dimension of 128. This is followed by an input sequence length of 512 timesteps, an LSTM layer with 32 units, which connects to a Dense layer containing a single neuron with a Sigmoid activation function. The loss function used is Binary Cross-Entropy. Finally, the performance of the trained model is evaluated on the test set.

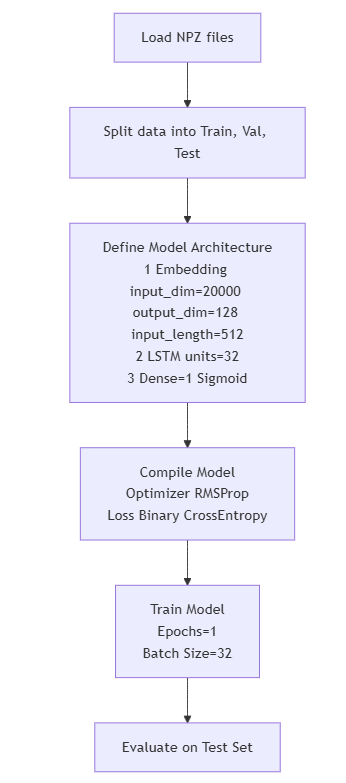


Figure 2

1. This section focuses on the exploration and evaluation of multiple LSTM architectures by systematically varying three hyperparameters, each tested with three different values, to assess their impact on model performance. Additionally, modifications to the dataset, including the addition and removal of records, are performed to analyze their influence on the model's predictive capabilities. Further enhancements to the network architecture itself are examined to identify potential improvements. Moreover, the dataset's balance is adjusted to three different levels to study its effect on classification outcomes. Finally, dimensionality reduction is applied to the embedding layer, and the model's performance is evaluated accordingly.

**Results**

Exploratory Data Analysis (EDA)

The following part shows the results of Exploratory Data Analysis (EDA) process over the data.

* Basic information about the dataset:

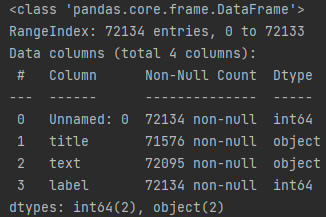


Figure 3

* Number of rows having missing values:



Figure 4

* Number of duplicates:

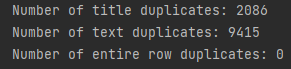


Figure 5

* Number of articles containing HTML content:



Figure 6

* Number of non-English articles:



Figure 7

* Statistical information about articles lengths:

תמונה שמכילה טקסט, צילום מסך, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

Figure 8

* The 10 longest lengths of articles (including duplicates):



Figure 8

* Labels distribution:

תמונה שמכילה טקסט, צילום מסך, גופן, עיגול

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

Figure 10

Performance Evaluation - SVM:

The following table presents the accuracy, recall, precision, specificity, and F1-score values of the SVM model (with kernel = "rbf") evaluated on the test set:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Accuracy | Precision | Recall | Specificity | F1-Score |
| SVM(Kernel = 'rbf') | 0.747 | 0.751 | 0.64 | 0.831 | 0.691 |

Table 1

Performance Evaluation - LSTM Model (Using Default Hyperparameters):

This table presents the same metrics of the LSTM network on the same test set:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Accuracy | Precision | Recall | Specificity | F1-Score |
| LSTM | 0.59 | 0.758 | 0.108 | 0.972 | 0.189 |

Table 2

We can derive from these tables that the SVM model significantly outperforms the LSTM model in accuracy, recall, and F1-score, making it a more reliable choice in its current form.

The LSTM model is highly precise and has strong specificity but struggles with recall, meaning it fails to capture a significant portion of positive cases.

These results suggest that further improvements to the LSTM model's architecture, hyperparameters, or training strategy may be necessary to achieve better performance.

Hyperparameters Tuning Results:

The following table shows the metrics values of the LSTM model on the test set. The same LSTM model was trained using various configurations of three key hyperparameters: epochs, batch size, and optimizer. Each hyperparameter was tested with three different values:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Hyperparameter | Accuracy | Precision | Recall | Specificity | F1-Score |
| Epochs = 50 | 0.954 | 0.943 | 0.955 | 0.954 | 0.949 |
| Epochs = 100 | 0.962 | 0.965 | 0.949 | 0.973 | 0.957 |
| Epoch = 200 | 0.953 | 0.953 | 0.94 | 0.963 | 0.947 |
| Optimizer="Adam", Epochs = 50 | 0.949 | 0.954 | 0.931 | 0.964 | 0.942 |
| Optimizer="SGD",  Epochs = 50 | 0.557 | 0.5 | 0.001 | **0.999** | 0.002 |
| Optimizer="AdaGrad"  Epochs = 50 | 0.748 | 0.699 | 0.755 | 0.742 | 0.726 |
| Batch size = 16  Epochs = 50 | **0.966** | 0.962 | **0.961** | 0.97 | **0.961** |
| Batch size = 64  Epochs = 50 | 0.951 | **0.975** | 0.913 | 0.981 | 0.943 |
| Batch size = 128  Epochs = 50 | 0.964 | 0.966 | 0.951 | 0.973 | 0.959 |

Table 3

It is evident that increasing the number of epochs has had a significant impact on the values of these metrics. Increasing the number of epochs from 50 to 100 improved accuracy, precision, and F1-score, indicating better overall performance.

However, increasing to 200 epochs led to a slight decline in performance (accuracy: 0.953, precision: 0.953, recall: 0.94). This could suggest overfitting, where the model learns too much from the training data but generalizes poorly on unseen data.

The RMSprop optimizer (default in the first three rows) performed well across different epoch settings, achieving consistently high accuracy and balanced performance across metrics.

Adam optimizer performed comparably to RMSprop (accuracy: 0.949, precision: 0.954, recall: 0.931), making it a viable alternative.

SGD optimizer performed extremely poorly (accuracy: 0.557, recall: 0.001, F1-score: 0.002), indicating that it is not suitable for this dataset and LSTM architecture.

AdaGrad optimizer showed moderate performance (accuracy: 0.748, recall: 0.755, F1-score: 0.726) but was significantly weaker than RMSprop or Adam.

A batch size of 16 achieved the highest overall performance (accuracy: 0.966, recall: 0.961, F1-score: 0.961), suggesting that smaller batch sizes help in capturing patterns better in this dataset.

If accuracy is considered the most important metric, then the model trained with 50 epochs, the RMSprop optimizer, and a batch size of 16 achieved the highest performance (0.966).

Evaluating the Impact of Dataset Size on Model Performance:

As previously observed, the model with the default batch size and optimizer achieved an accuracy of 95.4%. This result was obtained using only 90% of the preprocessed data, which was split into training, validation, and test sets. Now, the performance of the same model will be evaluated under different dataset modifications. First, the model will be tested with a reduced dataset by removing records so that the model will be trained using 10% of the available data. Then, the model will be trained using 100% of the available data, split into training, validation, and test sets, followed by artificially increasing the number of training records through duplication. Finally, a comparison will be made between all models to analyze the impact of dataset size on performance. The following table summarizes the metrics results of the models on the test set:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset size | Accuracy | Precision | Recall | Specificity | F1-Score |
| 90% (regular) | 0.954 | 0.943 | 0.955 | 0.954 | 0.949 |
| 10% | 0.899 | 0.883 | 0.89 | 0.906 | 0.887 |
| 200% (Duplicated) | 0.967 | 0.968 | 0.957 | 0.974 | 0.962 |

Table 4

More data improves performance – even though data was duplicated to 200%, led to the highest metrics results, suggesting that increasing the dataset size enhances model generalization.

Small dataset reduces effectiveness – Using only 10% of the data resulted in significantly lower accuracy (89.9%) and specificity (0.906), indicating that insufficient data limits model performance.

Regular dataset performs well – The original 90% dataset achieves high accuracy (95.4%), but further increasing the dataset still provides slight improvements.

Improved Architecture:

After experimenting with various architectural configurations, the model that achieved the highest accuracy (**97.5%**) featured several optimizations. These included a reduced vocabulary size (13,000), a lower embedding dimension (84), the addition of SpatialDropout1D (0.2) after the embedding layer, a dropout rate of 0.2 within the LSTM layer, L2 regularization applied to the Dense layer, and the incorporation of Early Stopping and learning rate decay for improved training efficiency. In addition, the batch size is 16 and the optimizer is "RMSprop".

In summary, the optimized architecture consists of the following components:

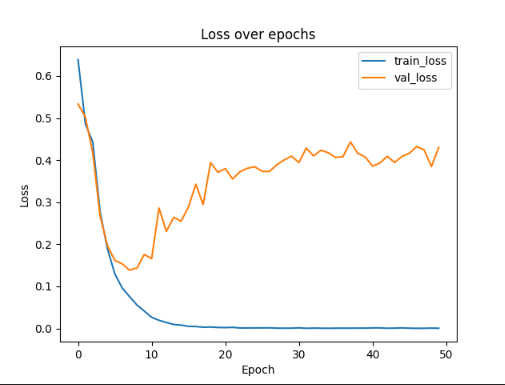
**תמונה שמכילה טקסט, צילום מסך, גופן, עיצוב

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**

Figure 11

Speed of Convergence Comparison

The following graphs illustrate the training and validation loss values across the epochs for two different network architectures. Figure 12a represents the network that achieved 96.6% accuracy. Figure 12b represents the newly introduced architecture:

**תמונה שמכילה טקסט, עלילה, צילום מסך, תרשים

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**

Figure 12b

Figure 12a

Figure 12b demonstrates a significantly faster convergence compared to the first one. The loss decreases rapidly, stabilizing within the first 10 epochs, whereas the first network continues to improve gradually over a longer period. This suggests that the new model learns more efficiently and reaches an optimal state in fewer training epochs.

New Metric

The newly selected metrics for evaluation during the training process include the ROC Curve, AUC (Area Under the Curve), and MCC (Matthews Correlation Coefficient). focusing on the model that achieved 97.5% accuracy. The MCC is a balanced metric that evaluates the quality of binary classifications, even when the classes are imbalanced. It is calculated using the following formula:

MCC =

Where TP = True Positive, FP = False Positive, TN = True Negative, FN = False Negative.

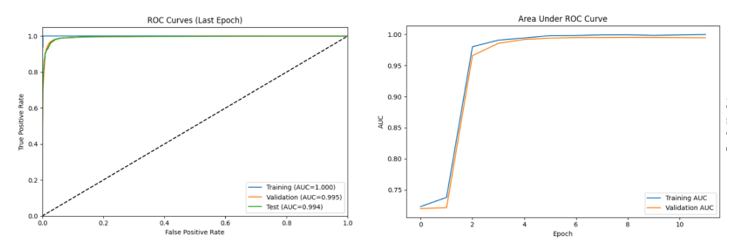


Figure 13

תמונה שמכילה טקסט, תרשים, קו, עלילה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

Figure 14

Impact of Data Imbalance on Model Performance

To analyze the effect of data imbalance on model performance, the dataset was modified to create three different levels of class distribution.

The first imbalance scenario involved a dataset where 0% of the rows belonged to class 1 (fake news), while 100% of the rows belonged to class 0 (real news).

The second imbalance scenario involved a dataset where 35% of the rows belonged to class 1 (fake news), while 65% of the rows belonged to class 0 (real news).

The third imbalance scenario involved a dataset where 80% of the rows belonged to class 1 (fake news), while 20% of the rows belonged to class 0 (real news). The following table shows the metrics results of the same model on each scenario:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Imbalance Level | Accuracy | Precision | Recall | Specificity | F1-Score |
| 0% - 100% | 0.557 | 0 | 0 | 1 | 0 |
| 35% - 65% | 0.973 | 0.965 | 0.975 | 0.972 | 0.97 |
| 80% - 20% | 0.948 | 0.917 | 0.97 | 0.93 | 0.943 |

Table 5

When the imbalance level is 0% - 100% the dataset is entirely composed of one class (real news), the model is unable to classify fake news at all, resulting in a precision, recall, and F1-score of 0. Specificity is 1 since the model perfectly identifies real news.

When the imbalance level is 35% - 65% the data is quite balanced. The model achieves the highest accuracy, precision and F1-score, indicating strong overall performance. This confirms that a well-balanced dataset provides optimal model performance.

When the imbalance level is 80% - 20% the data is moderately imbalanced. The metrics remain high but slightly lower than the balanced case. Precision (0.917) is lower than in the balanced dataset, likely due to an increased number of false positives, suggests that the model starts favoring the dominant class.

Impact of Dimension Reduction on Model Performance

To evaluate the model performance after reducing dimensions the following modification were made:

Reduction of vocabulary size from 13,000 to 5,000: This parameter defines the number of unique words in the model's vocabulary.

Reduction of embedding dimension from 84 to 32: This parameter determines the number of dimensions in which each word is represented in the embedding layer.

Reduction of timesteps from 512 to 256: This parameter represents the maximum sequence length (i.e., the number of words the model considers from each news article).

The following table compares the metric results of the model that achieved 97.5% accuracy (as described in the 'Improved Architecture' section) with the results obtained after reducing the model's dimensions.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | Specificity | F1-Score |
| Baseline Model | 0.975 | 0.97 | 0.973 | 0.976 | 0.972 |
| Reduced-Dimensions Model | 0.973 | 0.969 | 0.965 | 0.976 | 0.967 |

Table 6

The reduced-dimensions model provides almost the same classification performance as the baseline model while being computationally lighter. This suggests that reducing model complexity can be a viable strategy for optimizing efficiency without significant loss in accuracy.

Since the reduced model was trained with lower vocab size, embedding dimensions, and timesteps, it is likely more efficient in terms of memory usage and training speed, while maintaining strong performance.

**Discussion and Conclusions**

This study explored different approaches to fake news detection, leveraging both traditional Machine Learning (SVM) and Deep Learning (LSTM) models. The performance of these models was assessed using various hyperparameter configurations, dataset modifications, and architectural enhancements.

The initial results demonstrated that the SVM model outperformed the LSTM model with default hyperparameters, achieving higher accuracy, recall, and F1-score. However, subsequent experiments with hyperparameter tuning and architectural improvements led to significant enhancements in the LSTM model’s performance, ultimately surpassing SVM in certain configurations.

The study examined the impact of three key hyperparameters—epochs, batch size, and optimizer—on the LSTM model. The findings indicated that increasing the number of epochs generally improved performance up to a certain point, after which overfitting was observed. The RMSprop and Adam optimizers provided the best results, while SGD significantly underperformed. Additionally, a batch size of 16 yielded the highest accuracy, suggesting that smaller batch sizes allow the model to better capture patterns within the dataset.

Further investigation was conducted by modifying the dataset size, including training with reduced data (10%) and expanded data (duplicated to 200%). The results showed that increasing the dataset size improved generalization and overall accuracy, while a smaller dataset significantly hindered performance.

An optimized LSTM architecture was developed, incorporating dimensionality reduction, dropout layers, L2 regularization, and learning rate scheduling. This architecture achieved the highest accuracy (97.5%) while maintaining stable training dynamics. The model was also tested under different levels of data imbalance, confirming that a balanced dataset yields the best classification performance. In cases of extreme imbalance, the model struggled to correctly classify the minority class.

Finally, dimensionality reduction was applied by lowering the vocabulary size, embedding dimensions, and sequence length (timesteps). The results indicated that reducing model complexity did not significantly degrade performance, while improving computational efficiency and training speed.

**Future Work**

While the improved LSTM model achieved high accuracy, future research can explore more advanced deep learning architectures such as Transformer models (e.g., BERT), attention mechanisms, and ensemble learning techniques to further enhance fake news detection capabilities. Additionally, data augmentation strategies could be implemented to improve model robustness in real-world applications.

**References**

[1] Z. Khanam, B. N. Alwasel, H. Sirafi, and M. Rashid, “Fake News Detection Using Machine Learning Approaches,” *IOP Conf Ser Mater Sci Eng*, vol. 1099, no. 1, p. 012040, Mar. 2021, doi: 10.1088/1757-899X/1099/1/012040.

[2] J. C. S. Reis, A. Correia, F. Murai, A. Veloso, and F. Benevenuto, “Supervised Learning for Fake News Detection,” *IEEE Intell Syst*, vol. 34, no. 2, pp. 76–81, Mar. 2019, doi: 10.1109/MIS.2019.2899143.

[3] S. M. Jaybhaye, V. Badade, A. Dodke, A. Holkar, and P. Lokhande, “Fake News Detection using LSTM based deep learning approach,” *ITM Web of Conferences*, vol. 56, p. 03005, Aug. 2023, doi: 10.1051/itmconf/20235603005.

[4] R. K. Kaliyar, A. Goswami, P. Narang, and S. Sinha, “FNDNet – A deep convolutional neural network for fake news detection,” *Cogn Syst Res*, vol. 61, pp. 32–44, Jun. 2020, doi: 10.1016/j.cogsys.2019.12.005.

[5] Y. Yang, L. Zheng, J. Zhang, Q. Cui, Z. Li, and P. S. Yu, “TI-CNN: Convolutional Neural Networks for Fake News Detection,” Jun. 2018.

[6] M. F. Mridha, A. J. Keya, Md. A. Hamid, M. M. Monowar, and Md. S. Rahman, “A Comprehensive Review on Fake News Detection With Deep Learning,” *IEEE Access*, vol. 9, pp. 156151–156170, 2021, doi: 10.1109/ACCESS.2021.3129329.

[7] B. Upadhayay and V. Behzadan, “Hybrid Deep Learning Model for Fake News Detection in Social Networks (Student Abstract),” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 11, pp. 13067–13068, Jun. 2022, doi: 10.1609/aaai.v36i11.21670.

[8] Q. Abbas, M. U. Zeshan, and M. Asif, “A CNN-RNN Based Fake News Detection Model Using Deep Learning,” in *2022 International Seminar on Computer Science and Engineering Technology (SCSET)*, IEEE, Jan. 2022, pp. 40–45. doi: 10.1109/SCSET55041.2022.00019.

[9] N. K. Conroy, V. L. Rubin, and Y. Chen, “Automatic deception detection: Methods for finding fake news,” *Proceedings of the Association for Information Science and Technology*, vol. 52, no. 1, pp. 1–4, Jan. 2015, doi: 10.1002/pra2.2015.145052010082.

[10] N. Ruchansky, S. Seo, and Y. Liu, “CSI,” in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, New York, NY, USA: ACM, Nov. 2017, pp. 797–806. doi: 10.1145/3132847.3132877.

[11] K. Stahl, “Fake news detection in social media,” 2018.

[12] S. I. Manzoor, J. Singla, and Nikita, “Fake News Detection Using Machine Learning approaches: A systematic Review,” in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, IEEE, Apr. 2019, pp. 230–234. doi: 10.1109/ICOEI.2019.8862770.