Early Stopping on CNN-LSTM Development to Improve Classification Performance

M. Khairul Anam[1,*], Sarjon Defit[2], Haviluddin[3], Lusiana Efrizoni[4], Muhammad Bambang Firdaus[5]

[1]*Universitas Samudra, Jl, Prof. Dr. Syarief Thayeb, Meurandeh, Langsa 24416, Indonesia*

[2]*Universitas Putra Indonesia YPTK Padang, Jl. Raya Lubuk Begalung, Padang 25145, Indonesia*

[3,5]*Universitas Mulawarman, Jl. Kuaro No.1, Samarinda, 75123, Indonesia*

[4]*Universitas Sains dan Teknologi Indonesia, Jl. Purwodadi Indah KM.10 Panam, Pekanbaru 28294, Indonesia*

(Received: May 29, 2024; Revised: Juni 13, 2024; Accepted: July 1, 2024; Available online: August 5, 2024)

Abstract

Currently, CNN-LSTM has been widely developed through changes in its architecture and other modifications to improve the performance of this hybrid model. However, some studies pay less attention to overfitting, even though overfitting must be prevented as it can provide good accuracy initially but leads to classification errors when new data is added. Therefore, extra prevention measures are necessary to avoid overfitting. This research uses dropout with early stopping to prevent overfitting. The dataset used for testing is sourced from Twitter; this research also develops architectures using activation functions within each architecture. The developed architecture consists of CNN, MaxPooling1D, Dropout, LSTM, Dense, Dropout, Dense, and SoftMax as the output. Architecture A uses default activations such as ReLU for CNN and Tanh for LSTM. In Architecture B, all activations are replaced by Tanh, and in Architecture C, they are entirely replaced by ReLU. This research also performed hyperparameter tuning such as the number of layers, batch size, and learning rate. This study found that dropout and early stopping can increase accuracy to 85% and prevent overfitting. The best architecture entirely uses ReLU activation as it demonstrates advantages in computational efficiency, convergence speed, the ability to capture relevant patterns, and resistance to noise.

Keywords: CNN-LSTM, Early Stopping, Overfitting, ReLU, Tanh

1. Introduction

The hybrid CNN-LSTM algorithm is a combination of the Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) algorithms [1]. CNN-LSTM exploits the advantages of each architecture in spatial and temporal data processing tasks [2]. CNNs effectively create spatial patterns and structures using convolution and pooling layers [3]. Then, LSTM, an algorithm, is used to create and model temporal dependencies or sequences in the data. LSTM is suitable for processing sequential data, such as text, audio, or real-time [4]. The combination of CNN and LSTM allows the model to effectively create and integrate spatial and temporal information [5]. CNN-LSTM works alternately; CNN is used to extract features from input data, and then LSTM processes the features in a time sequence context, allowing the model to make predictions based on long-term dependencies in the data [6].

Several studies have developed CNN-LSTM, and research conducted by [7] carried out stock price correlation analysis using CNN-LSTM with a dataset from Twitter. In the CNN-LSTM architecture, this research uses Hyperparameters and obtains the highest accuracy of 75.75%. Another study [8] used CNN-LSTM to classify text. By comparing epochs, the research achieved an accuracy of 79.5, with the highest epoch of 30. Then, another research [9] carried out classification based on heartbeat sounds with CNN-LSTM. The accuracy obtained by carrying out the proposed CNN-LSTM architecture was 81%. Then, others [10] implemented CNN-LSTM to classify public opinion regarding the COVID-19 vaccine in Indonesia. From the trials, the research only obtained an accuracy of 66%.

Several previous studies showed that using hybrid CNN-LSTM can increase the accuracy of the base model. However, these studies often overlook the issue of overfitting, which is a critical problem as it leads to high accuracy on training

***Corresponding author: M. Khairul Anam (khairula210@gmail.com)**
DOI: https://doi.org/10.47738/jads.v5i3.312

data but poor generalization on new data. Overfitting occurs when a model is too closely fitted to the training data, capturing noise rather than the underlying pattern. This inconsistency across various datasets and processing conditions highlights the need for a more robust approach to prevent overfitting and ensure better model performance. This research aims to address this gap by implementing dropout and early stopping techniques to prevent overfitting in CNN-LSTM architectures. Overfitting is a common issue in machine learning where the model performs well on training data but poorly on new, unseen data. This occurs when the model learns the noise and details of the training data too well, leading to poor generalization. To address overfitting, this research implements dropout and early stopping techniques, which help improve the model's ability to generalize by preventing it from fitting too closely to the training data. The research tries to develop a CNN-LSTM architecture and looks at the effect of early stopping in overcoming overfitting. Early stopping is a technique used to overcome overfitting and find the optimal point to stop model training [11].

Developing the CNN-LSTM architecture, this research uses dropout with early stopping to prevent overfitting. Early stopping is particularly useful as it halts the training process once the model's performance on validation data stops improving, thereby preventing overfitting. The criteria for early stopping were determined by monitoring the validation loss; training was stopped if the validation loss did not improve for five consecutive epochs. This method ensures that the model maintains its ability to generalize well to new data. The layers start from the input data, and then CNN is used to extract spatial features from the input data. Furthermore, Maxpooling1D, Dropout, and LSTM capture temporal dependencies in sequential data. The next layers are dense, dropout, and output, finally adding early stopping.

Apart from that, this research also compares the activation functions of Tanh and ReLU. The Tanh function is an activation function that produces an output in the range of -1 to 1. This function is a non-linear transformation that helps capture complex data patterns [12]. Then, the ReLU function is an activation function that produces output in the range 0 to infinity. This function is very popular because of its simplicity and efficiency in solving the frequently occurring vanishing gradient problem [13]. This research uses the same three architectures but differs in their activation functions. The first architecture uses the default activation function, namely CNN, which uses ReLU, and LSTM, which uses Tanh. Then, all activation functions are changed to Tanh in the second architecture, and the third architecture is completely changed to ReLU.

2. The Proposed Algorithm CNN-LSTM

Several previous studies have carried out developments on the CNN-LSTM algorithm, table 1 is a development carried out by previous researchers.

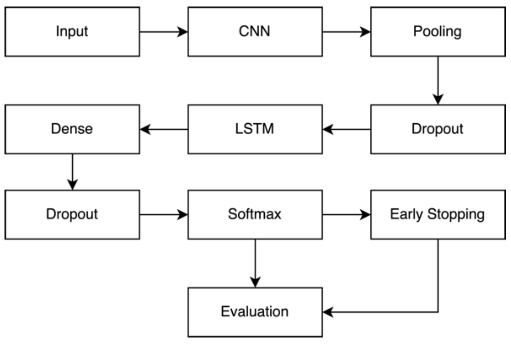
Table 1. Previous Research

Researcher	Year	Architecture
[14]	2024	Embedding → Convolutional 1D → Max Pooling 1D → LSTM → Dense
[15]	2020	Input → Convolutional 1D → Max Pooling → LSTM → Fully Connected Layer → Output Layer
[16]	2022	Input → LSTM → Concatenation → CNN → Fully Connected → Output
[17]	2023	Embedding → Dropout → Convolutional 1D → Max Pooling → Convolutional 1D → Max Pooling → Convolutional 1D → Max Pooling → LSTM → Dropout → Output
[18]	2022	Embeddings → Convolution → Activation Function layer → Features Maps for Regions Size → LSTM → Fully Connected Layer

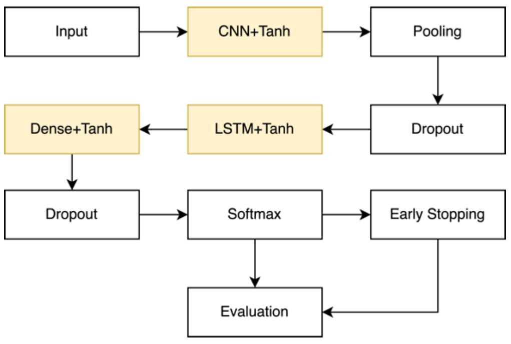
From several of these studies, figure 1 is the architectural development carried out in this research. Architecture A does not mention a specific activation function for the CNN, LSTM, and dense layers. Instead, this architecture uses the default activation function, ReLU for CNN and Tanh for LSTM. MaxPooling1D is still used to reduce data dimensions. Dropout is applied to reduce overfitting, and SoftMax is used on the output for classification. Without early stopping, the model can overfit, but with early stopping, training is stopped early to prevent overfitting and improve performance on validation data.

Architecture B uses the tanh activation function on all CNN, LSTM, and dense layers. The Tanh activation function helps capture more complex non-linearities in the data. MaxPooling1D reduces the dimensionality of the data, while dropout is applied after the CNN and dense layers to prevent overfitting. Without early stopping, the model may overfit if trained too long, while with early stopping, training is stopped early to avoid overfitting and improve generalization.

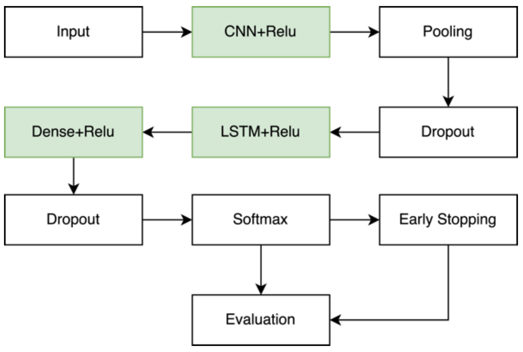
The C architecture uses the ReLU activation function in all CNN, LSTM, and Dense layers. ReLU is often used because it is effective in overcoming the vanishing gradient problem and accelerating convergence. MaxPooling1D reduces the dimensionality of the data, and dropout is applied to reduce overfitting. The output layer uses SoftMax for classification. Without early stopping, the model can experience overfitting, whereas, with early stopping, training is stopped early to avoid overfitting and increase the generalization ability of the model.



Normal Activation



Activation With Tanh



Activation with ReLU

Figure 1. CNN-LSTM Architecture Development

Using different activation functions in the CNN-LSTM architecture is necessary to understand how each function affects the model's performance. Activation functions play a crucial role in determining how neural networks learn and process information. Tanh and ReLU are two commonly used activation functions, but they have different characteristics that can significantly impact the model's outcomes. Testing these functions helps identify the most effective activation function for a specific architecture and dataset.

The Tanh (Hyperbolic Tangent) activation function produces outputs in the range of -1 to 1. Tanh is a non-linear function that can help the model capture more complex patterns. This function is also symmetric around the origin (0,0), meaning negative and positive values are treated equally, making it useful in contexts where the data distribution is centered around zero. However, Tanh can suffer from the vanishing gradient problem, where gradients become very small, slowing down the learning process in the early layers of the neural network.

On the other hand, ReLU (Rectified Linear Unit) produces outputs in the range of 0 to infinity. ReLU is known for its simplicity and efficiency in mitigating the vanishing gradient problem, making it very effective for deeper networks. ReLU activates neurons only if the input is positive, which means it can create sparsity in the representation, helping to make the model more efficient and faster in convergence. However, ReLU also has a drawback, known as the "dying ReLU" problem, where neurons can become inactive permanently if the input is continuously negative. By testing both

ISSN 2723-6471

functions, this research aims to determine which activation function provides the best performance in terms of accuracy, convergence speed, and the model's ability to capture relevant patterns and resist noise in the data.

With early stopping, all architectures can avoid overfitting if trained too long. The use of dropout helps reduce this risk but is not always enough. With early stopping, training is stopped early based on performance on validation data, helping prevent overfitting and improving model generalization on new data. Each architecture combines CNN capabilities for spatial feature extraction with LSTM capabilities for temporal dependency modelling, and the chosen activation function (Tanh or ReLU) influences how the model captures patterns in the data.

3. Method

To streamline this research, the methodological flowchart in figure 2 has been meticulously updated for enhanced clarity. This flowchart visually encapsulates each critical phase of the research process ensuring an intuitive understanding of the entire workflow. The detailed steps include data collection, preprocessing, word embedding using Word2Vec, CNN-LSTM modeling with dropout and early stopping and model evaluation using various metrics such as accuracy, precision, recall, and F1-score. Each step is visually represented making the research process easy to follow and replicate thereby facilitating a smoother and more efficient execution of the study.

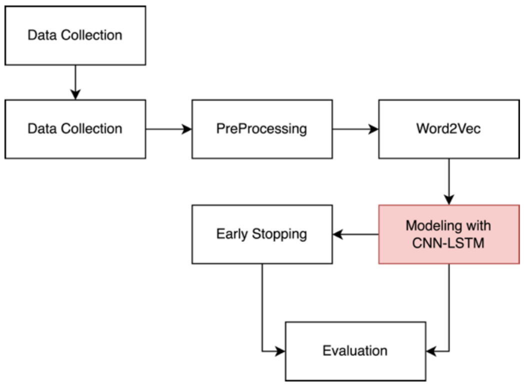


Figure 2. Methodology Flow

3.1. Data Collection

The dataset used for testing was meticulously collected from Twitter, covering the period from January 2020 to December 2020. This data predominantly comprises tweets in Indonesian, reflecting a wide range of sentiments and topics discussed by users during that time. To ensure the dataset's quality and relevance, a series of preprocessing steps were undertaken. These steps included the removal of retweets to eliminate redundant information and the normalization of text to standardize the format and reduce variability. Tokenization was then applied to break down the text into individual words or tokens, making it easier to analyze. Additionally, stopwords, which are common words that do not contribute significant meaning, were removed to focus on the more substantive parts of the text.

Following these preprocessing steps, the dataset was refined to consist of 10,000 tweets, each carefully labeled to facilitate sentiment analysis. Out of these, 7,854 tweets were categorized as positive, reflecting favorable or optimistic sentiments, while 3,028 tweets were labeled as negative, indicating unfavorable or critical sentiments. This comprehensive labeling process was crucial for training and evaluating the CNN-LSTM model. Figure 3 illustrates the distribution of positive and negative labels within the dataset, highlighting the balance and representation of sentiments captured in this research.

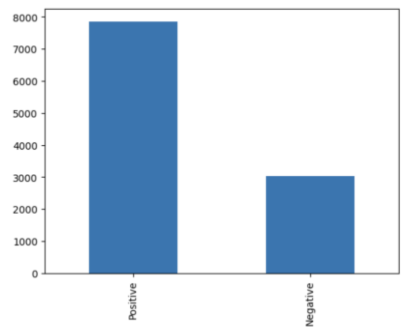


Figure 3. Number of Positive and Negative Labels

3.2. Preprocessing

The text preprocessing stage is an important step in processing raw text data to be ready for further analysis [19]. This stage aims to improve the quality of the data used in the analysis and ensure more accurate and reliable analysis results [20]. Several text preprocessing steps in this research include Data Cleaning, Case Folding, Text Normalization, Tokenization, Filtering, Stemming, and Data Transformation.

Data Cleaning aims to remove or correct incomplete, inaccurate, or irrelevant data, such as unnecessary punctuation, numbers, or special characters [21]. Then, Case Folding changes all letters to lowercase to avoid differences caused by capitalization [22]. Next, Text Normalization changes various word forms into the same standard form, such as changing the word "no" to "tak" [23]. Tokenization breaks down text into smaller units, such as words or sentences, to facilitate further analysis [24]. Filtering removes common words that do not provide much information, such as stop words (conjunctions, prepositions) [25]. Certain stop words are removed to reduce noise and focus on the more meaningful words, improving the performance of the model.

Stemming reduces words to their base form by removing suffixes or prefixes, for example, “play” becomes “main” [26]. This step is crucial to ensure that words with the same root are treated as the same term, which improves the model's ability to learn from the data. The stemming process helps in reducing the dimensionality of the feature space by grouping different forms of a word into a single term.

Finally, Data Transformation involves converting text data into a format more suitable for analysis, such as vector representation, using techniques such as TF-IDF, Word2Vec, and other word embeddings [27]. By applying these preprocessing steps, raw text data is transformed into a more structured and meaningful form, improving the performance of machine learning models and producing more accurate and effective analysis results [28].

3.3. Word2Vec

Before the Word2Vec process is carried out, figure 4 is a token density distribution plot that shows the distribution of the number of tokens in the dataset used. The token density distribution shows that most of the documents in the dataset are between 10 and 30 tokens long, with some documents being longer by up to 50 tokens. It guides the embedding process using Word2Vec and subsequent adjustment of model parameters, such as padding length and model architecture, to effectively handle variations in document length [29].

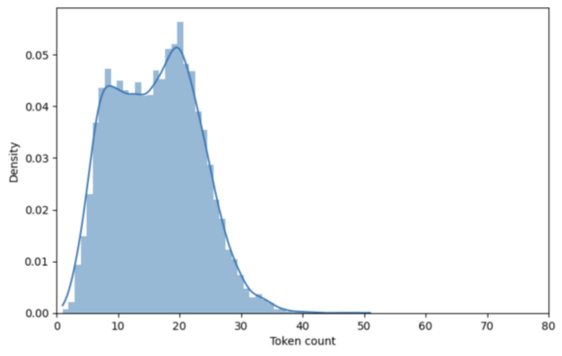


Figure 4. Token Density Distribution Plot

Word2Vec was chosen over other embedding methods because it efficiently captures the semantic relationships between words by learning distributed representations [30]. Unlike traditional methods such as one-hot encoding, which creates high-dimensional and sparse vectors, Word2Vec creates dense vectors that embed similar words close to each other in the vector space. This helps the model better understand the context and meaning of words in the dataset.

The saved Word2Vec model is loaded using Gensim, and vector embeddings are retrieved for each word in the dictionary. The embedding matrix is initialized to an appropriate size and filled with embedding vectors for each word found in the Word2Vec dictionary [31]. This step prepares the text data for machine learning by leveraging Word2Vec embedding, ensuring that all text sequences have the same length and appropriate embedding vectors for use in the model [32]. By using Word2Vec, the model benefits from a richer representation of the text data, which enhances its ability to capture complex patterns and improves overall performance.

3.4. CNN-LSTM

This stage involves data modeling using the CNN-LSTM architecture. A combination of CNN and LSTM captures spatial and temporal patterns in the data. CNN extracts feature from Word2Vec embeddings, while LSTM captures temporal relationships in text data. Word2Vec is implemented by first training a Word2Vec model on the preprocessed text data to create dense vector representations for each word. These vectors capture the semantic relationships between words, allowing similar words to have similar representations. The trained Word2Vec model is then used to convert the text data into sequences of word vectors, which serve as input to the CNN-LSTM model.

The CNN component processes these word vectors to extract local patterns and features through convolutional layers. These layers apply filters to the input data, generating feature maps that highlight important patterns in the text. The formula for a convolution operation in one dimension is:

$$y_i = f\left(\sum_{j=0}^{k-1} x_{i+j} w_j + b\right) \tag{1}$$

where y_i is the output feature map, x is the input sequence, w is the filter, b is the bias term, and f is the activation function, typically ReLU.

After the convolutional layers, MaxPooling is applied to reduce the dimensionality of the feature maps and focus on the most important features. The pooled feature maps are then passed to the LSTM component, which processes the sequences to capture long-term dependencies and temporal relationships in the data. The LSTM layers use the following equations to update their hidden states:

$$\begin{gathered}
f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \\
i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \\
\tilde{C}_t = \tanh(W_C.[h_{t-1}, x_t] + b_C) \\
C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \\
h_t = o_t * \tanh(C_t)
\end{gathered} \tag{2}$$

where f_t is the forget gate, i_t is the input gate, $\tilde{C}_t$ is the candidate cell state, $\tilde{C}_t$ is the cell state, ot is the output gate, and h_t is the hidden state. These equations enable the LSTM to selectively remember and forget information, capturing temporal patterns in the data.

This process also includes implementing early stopping to prevent overfitting by monitoring the performance on validation data and halting training if the performance does not improve for a specified number of epochs. Early stopping is particularly useful as it helps find the optimal point to stop training, ensuring the model does not overfit to the training data and maintains good generalization to new data.

ISSN 2723-6471

3.5. Evaluation

To evaluate the performance of the CNN-LSTM model, we used several metrics including accuracy, precision, recall, and F1-score. These metrics provide a comprehensive assessment of the model's performance across different aspects of classification.

Accuracy measures the proportion of correct predictions out of the total predictions made [33]. It provides an overall performance metric and is calculated as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3}$$

where TP represents true positives, TN represents true negatives, FP represents false positives, and FN represents false negatives. Accuracy is a useful metric for understanding the general effectiveness of the model in making correct predictions.

Precision quantifies the number of true positive predictions made out of all positive predictions [34]. It is crucial for understanding the model's performance in identifying relevant results, especially in scenarios where the cost of false positives is high. Precision is calculated as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4}$$

High precision indicates that the model has a low false positive rate, meaning it is effective in making positive predictions with high confidence.

Recall, or sensitivity, measures the proportion of actual positives that were correctly identified by the model [35]. It is an indicator of the model's effectiveness in capturing all relevant cases, and is calculated as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5}$$

High recall indicates that the model has a low false negative rate, meaning it is effective in identifying most of the positive cases.

F1-score is the harmonic mean of precision and recall, providing a balanced measure of accuracy, especially useful when dealing with imbalanced datasets [36]. It is calculated as:

$$\text{F1} - \text{Score} = 2 \text{ x } \frac{\text{Precision x Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$

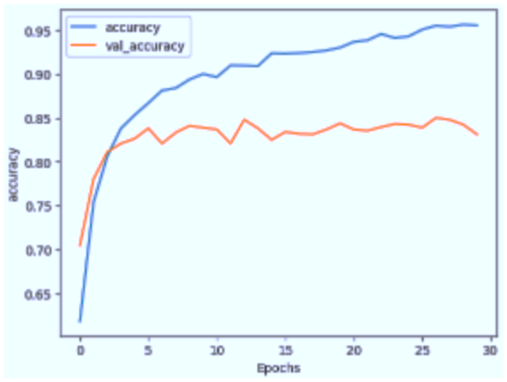
The F1-score is particularly valuable when the dataset has a significant imbalance between the classes, as it considers both false positives and false negatives. By using these metrics, we can obtain a detailed understanding of the model's performance, identifying strengths and weaknesses in different aspects of its classification capability.

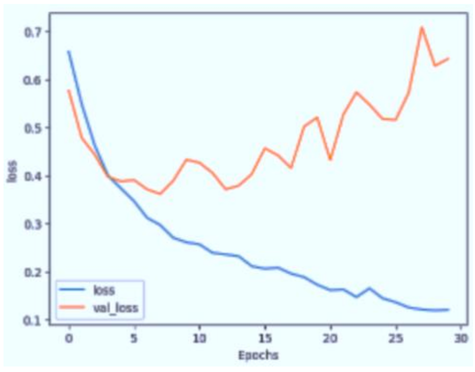
Additionally, the model's performance was evaluated using a confusion matrix, which provides a detailed breakdown of true positives, true negatives, false positives, and false negatives. This matrix helps visualize the types of errors the model makes, enabling a more nuanced analysis of its predictive performance.

By combining these metrics and the confusion matrix, we gain a thorough evaluation of the CNN-LSTM model's performance, ensuring that it not only achieves high accuracy but also maintains balanced precision and recall, thereby providing robust and reliable classification results.

4. Results and Discussion

This research starts from data collection and pre-processing which has been carried out previously, then this research is carried out with the three architectures mentioned in figure 1. Figure 5 results in testing architecture b or activation using tanh on CNN and LSTM.

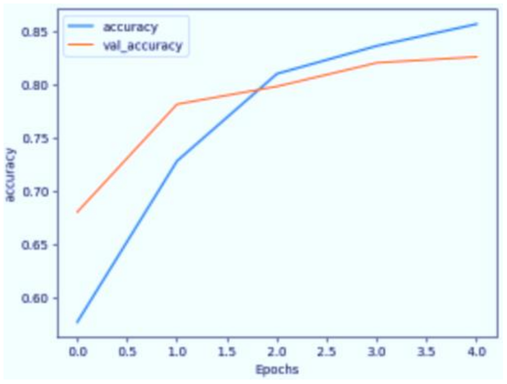


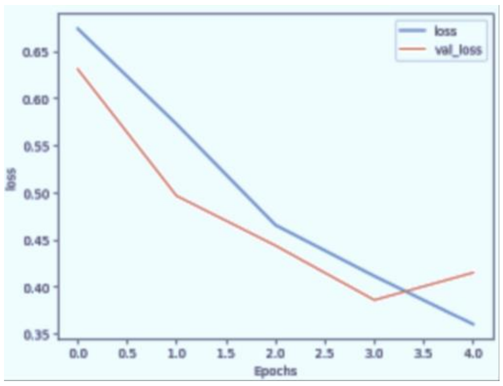


Accuracy Loss

Figure 5. Plotting Graphs Without Using Early Stopping

From figure 5, it can be seen that the model performance worked for 30 training epochs without early stopping. From the results of the plot, overfitting occurred during the training process. Early in the training process, the model worked well and improved in both training and validation. However, after the fifth epoch, validation accuracy began to level off and decreased slightly, while training accuracy continued to increase. This difference indicates that the model needs to be more adapted to the training data, thus failing to generalize well to new data. Then, in plotting loss, training loss decreases steadily, while validation loss increases after the fifth epoch. This indicates overfitting because the model performance in validation loss continues to worsen while training performance continues to improve. This research uses early stopping to prevent overfitting, as seen in figure 6.

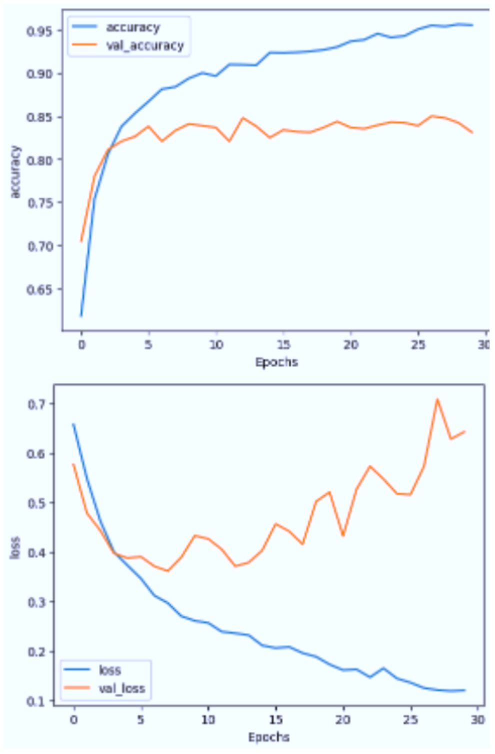




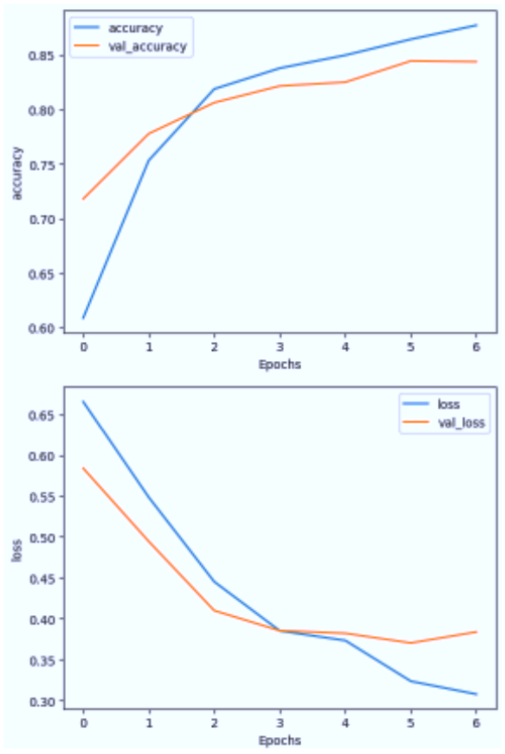
Accuracy Loss

Figure 6. Plotting Graphs Using Early Stopping

From figure 6, it can be seen that the model stops at epoch 5 using early stopping. In plotting a, it can be seen that both the blue and orange lines increase as the epoch increases. However, in the validation data or the orange line starts to flatten at the fifth epoch, this shows that the model stops learning more than necessary which has been set for 30 epochs. Then, the loss value in the training and validation data decreases as the epoch increases. With early stopping, the loss also stops at the fifth epoch. By stopping this model, overfitting does not occur. Then, figure 7 shows the result of graph plotting using architecture 1 or the normal activation function.



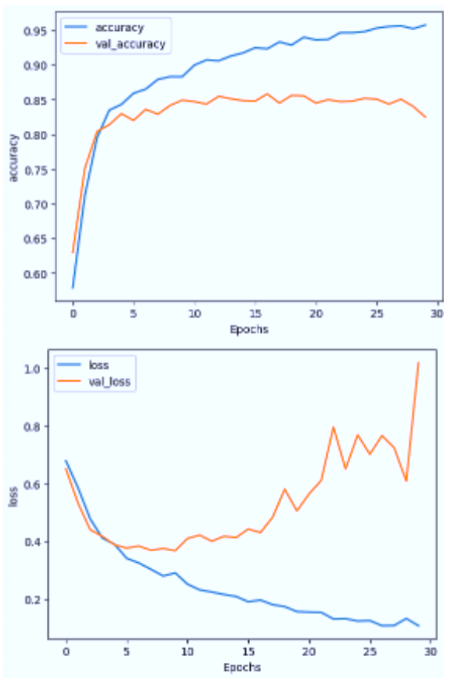
Without Early Stopping



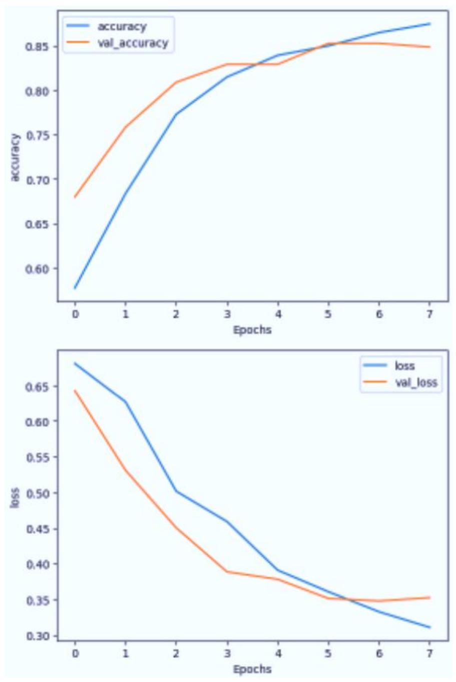
With early stopping

Figure 7. Plotting Graphs with Architecture A

Not using Early Stopping, training is carried out on the model architecture for 30 epochs. The plotting results show that there is overfitting in model training. Judging from the plotting results, excessive overfitting occurs after epoch 7. Early Stopping can prevent excessive overfitting from occurring. It can be seen from the val_loss value, which is not much greater than the loss value. With Early Stopping, model training stops at epoch 7. Next is figure 8, which is the result of plotting using the C architecture.



Without Early Stopping



With Early Stopping

Figure 8. Plotting Graphs with C Architecture

By not using Early Stopping, training is carried out on the model architecture for 30 epochs. The results of plotting show that there is overfitting in model training. Judging from the plotting results, excessive overfitting occurs after epoch 8. After that, using Early Stopping can prevent excessive overfitting from occurring. It can be seen from the val_loss value, which is similar to the loss value. With Early Stopping, model training stops at epoch 8.

The paper includes various plotting graphs, but the interpretations are not detailed enough. There should be a more thorough analysis of what the graphs reveal about the model's performance. For instance, the accuracy and loss plots in figure 5 indicate that the model starts to overfit after the fifth epoch, as evidenced by the widening gap between training and validation accuracy and the increasing validation loss. In figure 6, the use of early stopping successfully mitigates overfitting by halting training at the optimal point where validation performance stabilizes. Figure 7 and figure 8 further demonstrate the impact of early stopping across different architectures, showing how it can effectively balance training duration and model generalization. A detailed analysis of these graphs is essential to understand how different training strategies and architectural choices influence model performance and overfitting. After looking at the plotting results in table 2, we will see the accuracy results produced by architectures a, b, and c.

Table 2. Accuracy Comparison of Each Architecture

Test	Epoch	Accuracy
Architecture a without Early Stopping	30	83%
Architecture a Using Early Stopping	7	84%
Architecture b Without Early Stopping	30	85%
Architecture b Using Early Stopping	5	83%
Architecture c Without Early Stopping	30	83%
Architecture c Using Early Stopping	8	85%

Table 2 presents the results of experiments on three different model architectures, each tested with and without the use of Early Stopping to detect overfitting. The first architecture consists of Input, CNN, Pooling (MaxPooling1D), Dropout, LSTM, Dense, Dropout, and Dense layers with the SoftMax activation function as output. In experiments without Early Stopping, the model was trained for 30 epochs and achieved an accuracy of 83%. However, when Early Stopping was applied, training stopped earlier at the seventh epoch, slightly increasing accuracy to 84%. Early Stopping can help avoid overfitting by stopping training early when performance on validation data is no longer improving.

The second architecture uses Tanh for all function activations. Without Early Stopping, this model was trained for 30 epochs and achieved the highest accuracy of 85%. However, when Early Stopping is applied, training stops at the fifth epoch, and accuracy decreases slightly to 83%. Even though the use of Early Stopping reduces the number of epochs required, there is no overfitting with early.

The third architecture, ReLU, is an activation function in all layers. In experiments without Early Stopping, the model was trained for 30 epochs and achieved an accuracy of 83%. Training stops at the eighth epoch when early stopping is applied, and accuracy increases by 85%. This shows that Early Stopping reduces the number of epochs required and improves model performance.

Overall, Early Stopping proved beneficial in reducing the number of epochs required for training without significantly sacrificing accuracy. The second architecture shows the highest accuracy without Early Stopping, while the third architecture shows the best increase in accuracy when using Early Stopping. This emphasizes the importance of proper hyperparameter settings and configuration in achieving optimal model performance. In this study, hyperparameters such as the number of layers, batch size, learning rate, dropout rate, and early stopping criteria were meticulously tuned. The learning rate was initially set to 0.001 and adjusted using a validation set to avoid overfitting. Batch sizes of 32, 64, and 128 were tested to find the optimal balance between training speed and model performance. The dropout rate was set at 0.5 to prevent overfitting by randomly dropping neurons during training. Early stopping was implemented with a patience of 5 epochs, meaning training was halted if the validation loss did not improve for 5 consecutive epochs. These careful adjustments and configurations of hyperparameters were crucial in enhancing the model's accuracy and generalization ability, as evidenced by the increased performance metrics achieved in this study. Early Stopping can help find the balance between underfitting and overfitting, ensuring the model can generalize well on new data.

Then, Tanh and ReLU functions have their respective advantages, and their implementation in Architectures 2 and 3 shows how they can be used to capture complex patterns in data. In this study, ReLU shows superior computational efficiency, convergence speed, better ability to capture relevant patterns, and noise robustness. In this research, using ReLU was proven to be more effective in increasing model accuracy and avoiding overfitting than Tanh. Makes ReLU a better choice for the model architecture used in this research. All the experiments carried out were still above 80%, meaning that the model performance worked well and the results of this research were better compared to previous research related to CNN-LSTM development, as seen in table 3.

Table 3. Comparison with Previous Research

Researcher	Architecture	Dataset	Overfitting Prevention	Accuracy
[37]	Input → Conv ID → Max Pooling ID → LSTM → Dropout → Output	real-world EHR	Dropout	84%
[17]	Conv ID → Max Pooling ID → Conv ID → Max Pooling ID → Conv ID → Max Pooling ID → LSTM → Dropout → Output	Twitter	Dropout	79%
[9]	Conv ID → Max Pooling ID → Batch Normalization → Conv ID → Max Pooling ID → Batch Normalization → Conv ID → Max Pooling ID → Batch Normalization → LSTM → LSTM → Dense → Dropout → Dense → Dropout → Dense → Dropout	heartbeats	Dropout	81%
[38]	Input Data → 1D CNN with ReLU → Pooling → LSTM → Flattened → Output	HARD Dataset	Dropout	84%
[14]	Embedding → Convolutional 1D → Max Pooling 1D → LSTM → Dense	Cyberbullying Dataset	-	81%
This Study	Input → CNN+ReLU → Pooling → Dropout → LSTM+ReLU → Dense+ReLU → Dropout → Softmax → Early Stopping	Twitter	Dropout and Early Stopping	85%

Table 3 compares the architecture, dataset, overfitting prevention techniques, and accuracy achieved in this study with several previous studies. The study by [37] used an architecture consisting of 1D convolutional layers, Max Pooling 1D, LSTM, and Dropout, achieving 84% accuracy on real-world electronic health records (EHR) data. The study by [17] used multiple 1D convolutional layers and LSTM with Dropout, achieving 79% accuracy on Twitter data. The study by [9] employed a more complex architecture with multiple 1D convolutional layers, Batch Normalization, LSTM, and several Dropout layers, achieving 81% accuracy on heartbeat data. The study by [38] used data from the HARD Dataset with an architecture consisting of 1D convolutional layers with ReLU, Pooling, LSTM, Flattened, and Dropout, achieving 84% accuracy. The study by [14] utilized an architecture that included Embedding, multiple 1D convolutional layers and LSTM, achieving 78% accuracy on the Cyberbullying Dataset. This study used an architecture consisting of CNN layers with ReLU, Pooling, Dropout, LSTM with ReLU, Dense layers with ReLU, Dropout, and Softmax, and employed Early Stopping as an overfitting prevention technique. This study achieved the highest accuracy of 85% on Twitter data, demonstrating that the combination of Dropout and Early Stopping is effective in improving model performance.

5. Conclusion

Based on the results of the tests that have been carried out adding early stopping can improve and prevent overfitting on certain architectures. Then, ReLU activation is better than Tanh in data processing using the Twitter dataset. However, this research still has several shortcomings that need to be addressed for further research. Future research is recommended to explore other overfitting prevention techniques, such as L2 regularization and data augmentation as well as testing combinations of these techniques on various datasets to test the generalization of the methods used. For example, L2 regularization can be implemented by adding a penalty term to the loss function and its effect can be evaluated by comparing the model's performance with and without regularization across different datasets. Data augmentation can be applied by artificially increasing the size of the training data through techniques such as cropping, padding, and flipping, and its effectiveness can be measured by observing changes in the model's accuracy and robustness. In addition, hyperparameter optimization for dropout and early stopping can be carried out using grid search or random search methods to find the best combination to improve model accuracy further. More complicated models, such as transformers or hybrid models that integrate many architectures should be tested to gain a better understanding of how to overcome overfitting. For example, transformer models can be evaluated by implementing them with varying layer depths and comparing their performance against traditional models. Finally, an in-depth analysis of when and how overfitting occurs in models would be extremely beneficial in designing more effective prevention strategies. This can be achieved by regularly varying training durations and monitoring validation performance to identify specific points where overfitting begins.

6. Declarations

6.1. Author Contributions

Conceptualization: M.K.A., S.D., H., L.E., and M.B.F.; Methodology: L.E.; Software: M.K.A.; Validation: M.K.A., S.D., H., L.E., and M.B.F.; Formal Analysis: M.K.A., S.D., H., L.E., and M.B.F.; Investigation: M.K.A.; Resources: L.E.; Data Curation: L.E.; Writing Original Draft Preparation: M.K.A., S.D., H., L.E., and M.B.F.; Writing Review and Editing: L.E., M.K.A., S.D., H., and M.B.F.; Visualization: M.K.A.; All authors have read and agreed to the published version of the manuscript.

6.2. Data Availability Statement

The data presented in this study are available on request from the corresponding author.

6.3. Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

6.4. Institutional Review Board Statement

Not applicable.

6.5. Informed Consent Statement

Not applicable.

6.6. Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] J. M. Ahn, J. Kim, and K. Kim, “Ensemble Machine Learning of Gradient Boosting (XGBoost, LightGBM, CatBoost) and Attention-Based CNN-LSTM for Harmful Algal Blooms Forecasting,” *Toxins (Basel)*, vol. 15, no. 10, pp. 1–15, Oct. 2023, doi: 10.3390/toxins15100608.

[2] A. Bousmina, M. Selmi, M. A. Ben Rhaiem, and I. R. Farah, “A Hybrid Approach Based on GAN and CNN-LSTM for Aerial Activity Recognition,” *Remote Sens (Basel)*, vol. 15, no. 14, pp. 1–30, Jul. 2023, doi: 10.3390/rs15143626.

[3] A. Anton, N. F. Nissa, A. Janiati, N. Cahya, and P. Astuti, “Application of Deep Learning Using Convolutional Neural Network (CNN) Method For Women’s Skin Classification,” *Scientific Journal of Informatics*, vol. 8, no. 1, pp. 144–153, May 2021, doi: 10.15294/sji.v8i1.26888.

[4] M. Ghislieri, G. L. Cerone, M. Knaflitz, and V. Agostini, “Long short-term memory (LSTM) recurrent neural network for muscle activity detection,” J Neuroeng Rehabil, vol. 18, no. 1, pp. 1–15, Dec. 2021, doi: 10.1186/s12984-021-00945-w.

[5] B. Pan et al., “A CNN–LSTM Machine-Learning Method for Estimating Particulate Organic Carbon from Remote Sensing

in Lakes," *Sustainability (Switzerland)*, vol. 15, no. 17, pp. 1–15, Sep. 2023, doi: 10.3390/su151713043.

[6] L. Liu, J. Feng, J. Li, W. Chen, Z. Mao, and X. Tan, "Multi-layer CNN-LSTM network with self-attention mechanism for robust estimation of nonlinear uncertain systems," *Front Neurosci*, vol. 18, no. 1379495, pp. 1–14, 2024, doi: 10.3389/fnins.2024.1379495.

[7] M. N. I. Sina and E. B. Setiawan, "Stock Price Correlation Analysis with Twitter Sentiment Analysis Using The CNN-LSTM Method," *sinkron*, vol. 8, no. 4, pp. 2190–2202, Oct. 2023, doi: 10.33395/sinkron.v8i4.12855.

[8] N. Mohd, H. Singhdev, and D. Upadhyay, "Text Classfication Using CNN and CNN-LSTM," *Webology*, vol. 18, no. 4, pp. 2440–2446, 2021, doi: 10.29121/web/v18i4/149.

[9] N. B. Aji, Kurnianingsih, N. Masuyama, and Y. Nojima, "CNN-LSTM for Heartbeat Sound Classification," *International Journal on Informatics Visualization*, vol. 8, no. 2, pp. 735–741, 2024, doi: 10.62527/joiv.8.2.2115.

[10] S. Saadah, K. M. Auditama, A. A. Fattahila, F. I. Amorokhman, A. Aditsania, and A. A. Rohmawati, "Implementation of BERT, IndoBERT, and CNN-LSTM in Classifying Public Opinion about COVID-19 Vaccine in Indonesia," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 6, no. 4, pp. 648–655, Aug. 2022, doi: 10.29207/resti.v6i4.4215.

[11] C. W. Bartlett et al., "Invasive or More Direct Measurements Can Provide an Objective Early-Stopping Ceiling for Training Deep Neural Networks on Non-invasive or Less-Direct Biomedical Data," *SN Comput Sci*, vol. 4, no. 2, pp. 1–12, Mar. 2023, doi: 10.1007/s42979-022-01553-8.

[12] Y. Gong, "STL: A Signed and Truncated Logarithm Activation Function for Neural Networks," *arxiv.org*, vol. 14, no. 8, pp. 1–5, Jul. 2021, doi: 10.48550/arXiv.2307.16389.

[13] G. Madhu, S. Kautish, K. A. Alnowibet, H. M. Zawbaa, and A. W. Mohamed, "NIPUNA: A Novel Optimizer Activation Function for Deep Neural Networks," *Axioms*, vol. 12, no. 3, pp. 1–13, Mar. 2023, doi: 10.3390/axioms12030246.

[14] T. P. Handayani and M. I. Abas, "Comparative Analysis of CNN-LSTM and LSTM Models for Cyberbullying Detection with Increasing Dataset Sizes," *JURNAL ILMU KOMPUTER (JUIK)*, vol. 4, no. 2, pp. 75–85, 2024, doi: 10.31314/juik.v4i2.3185.

[15] H. Kaur, "Sentiment Analysis Of User Review Text Through Cnn And Lstm Methods," *PalArch's Journal of Archaeology of Egypt / Egyptology*, vol. 17, no. 12, pp. 290–306, 2020.

[16] P. N. Anggreyani and W. Maharani, "Hoax Detection Tweets of the COVID-19 on Twitter Using LSTM-CNN with Word2Vec," *Jurnal Media Informatika Budidarma*, vol. 6, no. 4, pp. 2432–2437, Oct. 2022, doi: 10.30865/mib.v6i4.4564.

[17] M. Srisankar and K. P. Lochanambal, "Hybrid CNN-LSTM Based Model for Sentiment Analysis in Tweets," *Journal of Data Acquisition and Processing*, vol. 38, no. 3, pp. 548–556, 2023, doi: 10.5281/zenodo.7922943.

[18] L. Khan, A. Amjad, K. M. Afaq, and H. T. Chang, "Deep Sentiment Analysis Using CNN-LSTM Architecture of English and Roman Urdu Text Shared in Social Media," *Applied Sciences (Switzerland)*, vol. 12, no. 5, pp. 1–18, Mar. 2022, doi: 10.3390/app12052694.

[19] M. K. Anam, T. A. Fitri, Agustin, Lusiana, M. B. Firdaus, and A. T. Nurhuda, "Sentiment Analysis for Online Learning using The Lexicon-Based Method and The Support Vector Machine Algorithm," *ILKOM Jurnal Ilmiah*, vol. 15, no. 2, pp. 290–302, 2023, doi: 10.33096/ilkom.v15i2.1590.290-302.

[20] A. N. Ulfah, M. K. Anam, N. Y. S. Munti, S. Yaakub, and M. B. Firdaus, "Sentiment Analysis of the Convict Assimilation Program on Handling Covid-19," *JUITA: Jurnal Informatika*, vol. 10, no. 2, pp. 209–216, 2022, doi: 10.30595/juita.v10i2.12308.

[21] C. S. C. Woolley, I. G. Handel, B. M. Bronsvoort, J. J. Schoenebeck, and D. N. Clements, "Is it time to stop sweeping data cleaning under the carpet? A novel algorithm for outlier management in growth data," *PLoS One*, vol. 15, no. 1, pp. 1–21, Jan. 2020, doi: 10.1371/journal.pone.0228154.

[22] R. S. Putra, W. Agustin, M. K. Anam, L. Lusiana, and S. Yaakub, "The Application of Naïve Bayes Classifier Based Feature Selection on Analysis of Online Learning Sentiment in Online Media," *Jurnal Transformatika*, vol. 20, no. 1, pp. 44–56, Jul. 2022, doi: 10.26623/transformatika.v20i1.5144.

[23] A. R. Lubis and M. K. M. Nasution, "Twitter Data Analysis and Text Normalization in Collecting Standard Word," *Journal of Applied Engineering and Technological Science*, vol. 4, no. 2, pp. 855–863, 2023, doi: 10.37385/jaets.v4i2.1991.

[24] K. I. Roumeliotis and N. D. Tselikas, “ChatGPT and Open-AI Models: A Preliminary Review,” *Future Internet*, vol. 15, no. 6, pp. 1–24, Jun. 2023, doi: 10.3390/fi15060192.

[25] S. Sarica and J. Luo, “Stopwords in technical language processing,” *PLoS One*, vol. 16, no. 8, pp. 1–13, Aug. 2021, doi: 10.1371/journal.pone.0254937.

[26] H. Alshalabi, S. Tiun, N. Omar, F. N. AL-Aswadi, and K. Ali Alezabi, “Arabic light-based stemmer using new rules,” *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 9, pp. 6635–6642, Oct. 2022, doi: 10.1016/j.jksuci.2021.08.017.

[27] Y. Li and Z. Wang, “biSAMNet: A Novel Approach in Maritime Data Completion Using Deep Learning and NLP Techniques,” *J Mar Sci Eng*, vol. 12, no. 6, pp. 1–21, May 2024, doi: 10.3390/jmse12060868.

[28] K. Maharana, S. Mondal, and B. Nemade, “A review: Data pre-processing and data augmentation techniques,” *Global Transitions Proceedings*, vol. 3, no. 1, pp. 91–99, Jun. 2022, doi: 10.1016/j.gltp.2022.04.020.

[29] K. Rahman, A. Ghani, S. Misra, and A. U. Rahman, “A deep learning framework for non-functional requirement classification,” *Sci Rep*, vol. 14, no. 3216, pp. 1–18, Dec. 2024, doi: 10.1038/s41598-024-52802-0.

[30] D. S. Asudani, N. K. Nagwani, and P. Singh, “Impact of word embedding models on text analytics in deep learning environment: a review,” *Artif Intell Rev*, vol. 56, no. 9, pp. 10345–10425, Sep. 2023, doi: 10.1007/s10462-023-10419-1.

[31] M. Ghifari Adrian, S. Suryani Prasetyowati, and Y. Sibaroni, “Effectiveness of Word Embedding GloVe and Word2Vec within News Detection of Indonesian uUsing LSTM,” *JURNAL MEDIA INFORMATIKA BUDIDARMA*, vol. 7, no. 3, pp. 1180–1188, 2023, doi: 10.30865/mib.v7i3.6411.

[32] J. T. Hancock and T. M. Khoshgoftaar, “Survey on categorical data for neural networks,” *J Big Data*, vol. 7, no. 1, pp. 1–41, Dec. 2020, doi: 10.1186/s40537-020-00305-w.

[33] L. Jen and Y.-H. Lin, “A Brief Overview of the Accuracy of Classification Algorithms for Data Prediction in Machine Learning Applications,” *Journal of Applied Data Sciences*, vol. 2, no. 3, pp. 84–92, 2021, doi: 10.47738/jads.v2i3.38.

[34] E. J. Michaud, Z. Liu, and M. Tegmark, “Precision Machine Learning,” *Entropy*, vol. 25, no. 1, pp. 1–17, Jan. 2023, doi: 10.3390/e25010175.

[35] S. A. Hicks et al., “On evaluation metrics for medical applications of artificial intelligence,” *Sci Rep*, vol. 12, no. 1, pp. 1–9, Dec. 2022, doi: 10.1038/s41598-022-09954-8.

[36] S. Orozco-Arias, J. S. Piña, R. Tabares-Soto, L. F. Castillo-Ossa, R. Guyot, and G. Isaza, “Measuring performance metrics of machine learning algorithms for detecting and classifying transposable elements,” *Processes*, vol. 8, no. 6, pp. 1–18, Jun. 2020, doi: 10.3390/PR8060638.

[37] J. Tian, A. Xiang, Y. Feng, Q. Yang, and H. Liu, “Enhancing Disease Prediction with a Hybrid CNN-LSTM Framework in EHRs,” *Journal of Theory and Practice of Engineering Science*, vol. 4, no. 02, pp. 8–14, Feb. 2024, doi: 10.53469/jtpes.2024.04(02).02.

[38] N. Hicham, H. Nassera, and S. Karim, “Enhancing Arabic E-Commerce Review Sentiment Analysis Using a hybrid Deep Learning Model and FastText word embedding,” *EAI Endorsed Transactions on Internet of Things*, vol. 10, no. 1, pp. 1–10, 2023, doi: 10.4108/eetiot.4601.