Evaluating Neural Network Approaches: CNN vs. RNN-LSTM in Detecting Fake News

Course #: LDSCI7226
Course Title: Natural Language Processing with Deep Learning
Course Leader: Dr. Mahsa Abazari Kia
Student #23220057

Abstract

The ability to detect fake news effectively using machine learning (ML) models holds significant potential for preserving information integrity. This report aims to develop and compare two Neural Network models, Convolutional Neural Network (CNN) and Recurrent Neural Network with Long Short-Term Memory (RNN-LSTM), to perform fake news detection. An extensive review of the literature was conducted to assess current ML models used in fake news detection tasks. Subsequently, CNN and RNN-LSTM models were designed and evaluated using a dataset from Kaggle. The performance of these models was measured based on the metrics accuracy, precision, recall, and F1 score. The results indicated that both models perform similarly, with CNN having greater accuracy in detecting fake news, but RNN-LSTM had more consistent performance across classes. Efforts to perform hyperparameter tuning were limited by processing power and disk space constraints. Nonetheless, these findings suggest that both approaches, but particularly CNN-based approaches, could be further refined and adopted to enhance the detection of fake news, contributing to the broader effort to mitigate the spread of misinformation.

Introduction

Fake news has become a pervasive issue, significantly impacting public perception, political stability, and social harmony. The spread of false information can lead to misinformation, influencing elections, inciting violence, and undermining trust in legitimate news sources. Therefore, effective detection of fake news is crucial for maintaining the integrity of information and safeguarding democratic processes [1].

Machine learning (ML), a subset of artificial intelligence (AI), presents innovative solutions for detecting fake news by leveraging data-driven algorithms to identify patterns and anomalies in textual data. Natural Language Processing (NLP), an important component of ML, enhances these solutions by enabling computers to understand, interpret, and generate human language. This paper aims to develop and compare the effectiveness of two Neural Network models—Convolutional Neural Network (CNN) and Recurrent Neural Network with Long Short-Term Memory (RNN-LSTM)—in detecting fake news. By analyzing a dataset of 44,898 data points, which includes titles, texts, subjects, and dates of both fake and real news articles, this paper seeks to determine which model offers superior performance and can be further refined for enhanced fake news detection.

Description of python files included in submission

NLP AE2 CNN Final - final version of CNN without date feature
NLP AE2 RNN Final - final version of RNN without date feature
NLP AE2 CNN with dates - original CNN with date feature included

Literature review

A. Fake News Overview

Fake news is a significant global issue, exacerbated by digital media and social networking platforms [2]. Its prevalence has surged during events like elections and pandemics, in turn misleading the public, increasing social polarization, and undermining trust in legitimate news sources. The impact extends beyond misinformation by influencing public opinion, swaying electoral outcomes, inciting social unrest, and causing economic repercussions like market destabilization and consumer trust erosion.

Manual fact-checking is impractical due to sheer volume and imitation of credible news by fake news, as well as psychological biases further complicating detection efforts [3]. To address these challenges, innovative solutions leveraging ML and NLP are needed. These technologies can analyze large datasets, detect patterns, and classify news articles accurately. By employing advanced algorithms, such systems can continuously improve and adapt to new forms of fake news, making them important tools in the fight against misinformation.

B. Existing Applications of Machine Learning in Fake News Detection

Machine learning has been increasingly applied to detect and classify fake news. For instance, Mridha et al. (2021) provide a comprehensive review on fake news detection with deep learning, highlighting the advantages of deep learning techniques over traditional machine learning [4]. Their review discusses various architectures, including CNNs, RNNs, and transformers like BERT, emphasizing the importance of automated feature extraction and handling high-dimensional data to improve detection accuracy.

Another study by Saleh et al. (2021) introduced the OPCNN-FAKE model, an optimized CNN for fake news detection [5]. The model enhances feature extraction through optimized CNN layers and hyperparameter tuning, achieving superior performance compared to other models such as RNNs and traditional classifiers like Decision Trees and Random Forests.

Additionally, Awan et al. (2020) investigated the use of a hybrid Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) model for fake news classification [6]. This bimodal approach leveraged CNNs for feature extraction and LSTMs for capturing sequential dependencies, enhancing the model's ability to understand textual nuances. The hybrid model showed improved performance over traditional methods in classifying fake news across multiple datasets, demonstrating its effectiveness in analyzing complex linguistic patterns in news articles.

These studies collectively illustrate how ML, enhanced by deep learning and NLP techniques, can improve fake news detection accuracy and provide scalable solutions to combat misinformation.

Methodology

A. Fake News Classification Dataset

The dataset comprises 44,898 samples, 4 features, and 2 classes. The features are article title, text, subject, and the date the article was published on. The classes are real (0) and fake (1) news. This dataset was collected from real world sources. The truthful articles were obtained by crawling articles from the news website Reuters.com. The fake news articles were collected from unreliable websites that were flagged by the US fact-checking organization Politifact and Wikipedia. The dataset contains different types of articles on different topics, however, the majority of articles focus on political and world news topics. The dataset was sourced from the public domain and is available on Kaggle [7].

A second dataset from the public domain available on Kaggle was utilized for testing. This dataset contained 5,200 samples with 2 features and 2 classes, with the features being article title and text and the classes also being real (0) and fake (1). This data was used in a fake news detection competition led by the UTK Machine Learning Club. Information regarding where the data was originally sourced from was not provided [8].

B. Software and Libraries

This paper utilizes Python 3 for all aspects, including data preprocessing, model development, and evaluation. Key libraries utilized are Pandas for data manipulation, NumPy for numerical operations, and TensorFlow with Keras for model development and training. Specific functionalities from Keras include model compilation and fitting. Keras Tuner is utilized for hyperparameter optimization in CNN models. Scikit-learn is used for data splitting, model evaluation, and metrics such as train_test_split and classification_report. Matplotlib is employed for data visualization. Jupyter Notebooks serve as the integrated development environment.

C. Model selection

Convolutional Neural Networks (CNN) and Recurrent Neural Networks with Long Short-Term Memory (RNN-LSTM) were selected for their established efficacy in similar fake news detection tasks [3], [4], [5], [6], [9]. CNNs are good at handling data with spatial hierarchy. This makes them adept at recognizing patterns in text when transformed into embedding matrices, which is vital for detecting nuances in fake news content. RNN-LSTMs are chosen for their ability to understand sequence and context in text data, which is essential for capturing long-term dependencies and subtle cues indicative of misinformation.

D. Data pre-processing

The initial processing routine involved data loading and cleaning using the Pandas library. The data from both the Real and Fake files were loaded into a Pandas DataFrame. The files did not have labels, so a 'label' column was added to each to denote whether the news was real or fake. Then the dataframes were combined into one to store both real and fake news in one dataframe. 209 duplicates were found in which all features contained the same information so these were deleted leaving 44,679 samples. There were no null values.

As mentioned, the data contained the features of article title, text, subject, and date. Initially all features were kept. The date column was normalized and then checked for invalid dates. There were 10 instances of invalid dates so these rows were removed. The temporal features of 'year' and 'day_of_year' were extracted. These were chosen as both used together will always correlate to a specific date. New columns were created containing each of these values.

Next, the 'title', 'text' and 'subject' columns were combined into one string. This was done as it helps provide a richer context and more comprehensive representation of the data, which can improve model performance. The combined text column was then run through a text normalization function. First, all text was converted to lowercase to ensure uniformity and reduce case sensitivity issues. URLs were removed as they are usually irrelevant for text analysis. Contractions were expanded to their full forms to standardize the text. Punctuation marks and numerical digits were removed to focus solely on the words themselves. Extra whitespace was reduced to single spaces for cleaner text. Stopwords from the NLTK Stopwords library were eliminated as these words carry less meaningful information. Lastly, the text was lemmatized to reduce words to their base or root form, improving consistency in text representation.

The data was then split into training and test sets using a 80/20 split. A tokenizer was run on the train data to split the text into individual pieces or tokens. The tokenizer was only fit on the training content to prevent information leakage from the test set into the training process. By fitting the tokenizer on the training content, we create a vocabulary based solely on the training data. This mirrors real-world scenarios where the model encounters new, unseen data during deployment. Then both the training and testing text data were converted into sequences of integers based on the vocabulary learned from the training data.

Next, the text sequences were padded to ensure uniform length across all sequences. This step handles sequences of varying lengths by padding shorter sequences with zeros and truncating longer ones. Next, the 'year' and 'day_of_year' columns were extracted from the training and testing dataframes and converted into numpy arrays. These date elements were then standardized using the StandardScaler. The scaler was fit on the training data and applied to both the training and testing data for consistent scaling. The padded text sequences were then horizontally stacked with the standardized date elements to create the final feature sets for the training and testing. Lastly, the labels for the training and testing data were extracted to be used in training and evaluating the model.

E. CNN Model Architecture Definition and Compilation

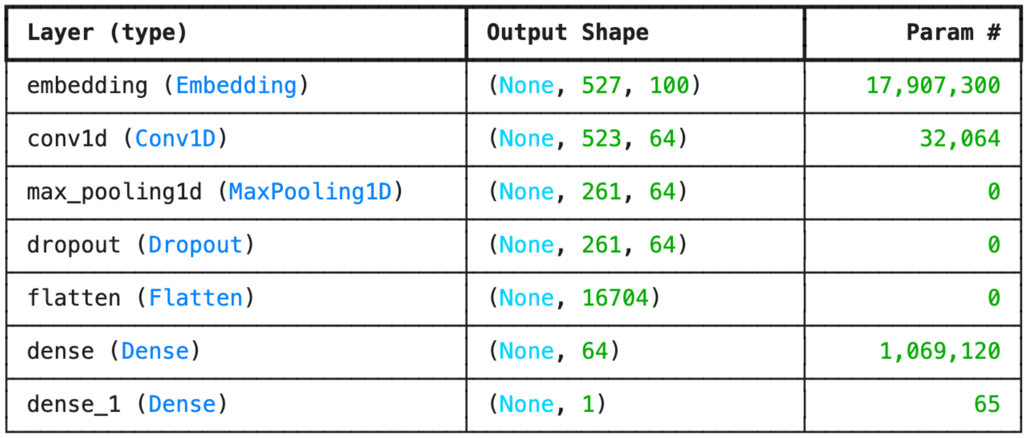
To create the CNN model architecture, first three important aspects were defined: the maximum sequence length, vocabulary size, and embedding dimension size. The sequence length was determined by calculating the lengths of all sequences from both the training and testing sets and determining the 95th percentile of these sequence lengths. This was done as the maximum sequence length was significantly long due to a few outliers. Therefore it was chosen to only utilize the length below which 95% of the sequences fell. The vocabulary size was set by determining the number of unique words in the tokenizer's word index and then adding 1 to account for a special token, such as padding or an unknown word, to ensure the vocabulary size covered all possible tokens the model might encounter. Lastly, the embedding dimension was set to 100, which specifies that each word in the vocabulary will be represented by a 100-dimensional vector in the embedding layer of the model. This was chosen because it provides a balance between capturing sufficient semantic information and maintaining computational efficiency

Next, the model was designed and compiled. The model was designed to process and integrate both textual and temporal data for binary classification. First was the text input, which was fed into an embedding layer. This layer maps each word in the input sequence to a dense vector space, capturing semantic relationships and reducing dimensionality from the raw text data. The output of the embedding layer was then passed through a 1D convolutional layer with 64 filters and a kernel size of 5. This convolutional layer detects local patterns and features in the text by sliding the filters over the embedded word vectors. It applied the ReLU activation function to introduce non-linearity and enable the model to learn complex features.

After the convolutional layer, max pooling was performed with a pool size of 2, which reduced the dimensionality of the feature maps by selecting the maximum value in each pooling window. This operation helps in retaining the most significant features and mitigating the risk of overfitting. A dropout layer with a rate of 0.3 was then applied to randomly set a fraction of input units to zero during training, further reducing overfitting and promoting generalization.

The output from the dropout layer was flattened into a 1D vector, which prepared the text features for combination with the additional date features. The date input was processed separately and directly fed into the model, providing temporal context. The flattened text features and date features were concatenated to form a single feature vector that integrates both sources of information.

This combined feature vector was then passed through a dense layer with 64 units and ReLU activation, allowing the model to learn complex interactions between the text and date features. Finally is the output layer; a dense layer with a single unit and a sigmoid activation function, which produces a probability score for binary classification. The model was compiled with the Adam optimizer, known for its efficiency in training deep learning models, and binary crossentropy loss, suitable for evaluating the performance of binary classification tasks. The CNN model summary can be seen in Table 1.



Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 527, 100)	17,907,300
conv1d (Conv1D)	(None, 523, 64)	32,064
max_pooling1d (MaxPooling1D)	(None, 261, 64)	0
dropout (Dropout)	(None, 261, 64)	0
flatten (Flatten)	(None, 16704)	0
dense (Dense)	(None, 64)	1,069,120
dense_1 (Dense)	(None, 1)	65

Table 1: CNN Model Summary

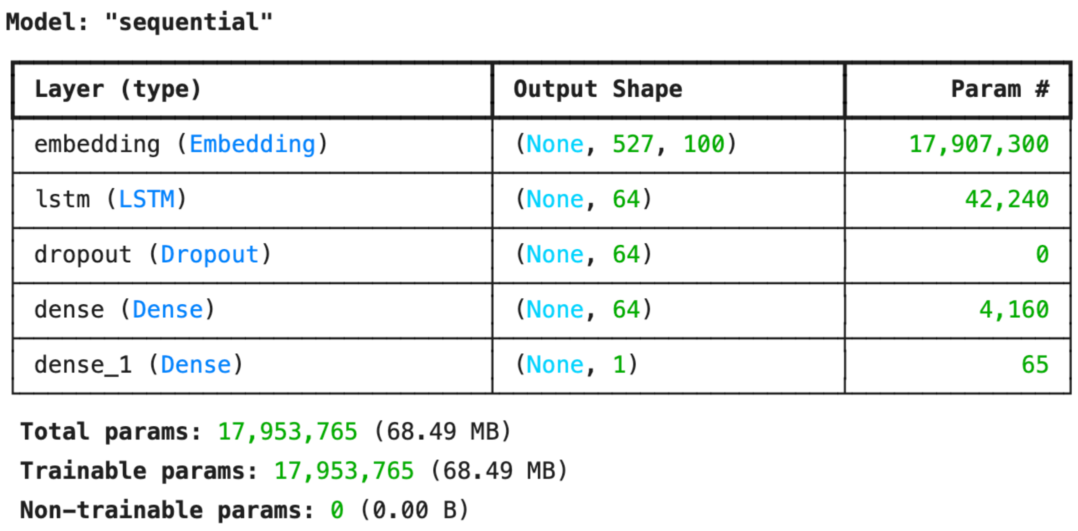
F. RNN-LSTM Model Architecture Definition and Compilation

The RNN-LSTM started with an embedding layer that transformed each word in the text input into dense vectors, capturing semantic relationships and reducing the dimensionality of the raw text data. These embeddings were then fed into an LSTM layer, which excels at learning long-range dependencies and capturing contextual information within the sequence. The LSTM processed the sequential data, capturing intricate patterns and temporal relationships.

Following the LSTM layer, a dropout layer was incorporated to mitigate overfitting. This layer randomly deactivated a portion of the units during training, which helped in regularizing the model and preventing it from relying too heavily on any particular feature.

The output from the dropout layer was then passed to a dense layer with 64 units. This dense layer applied a ReLU activation function, enabling the model to learn complex, non-linear interactions from the features extracted by the LSTM. The resulting representations were then directed to a final dense layer with a single unit, which use a sigmoid activation function to produce a probability score for binary classification.

The model was compiled using the Adam optimizer and employed binary crossentropy as the loss function, which is appropriate for binary classification problems. The RNN-LSTM model summary can be seen in Table 2.



```
Model: "sequential"
┌─────────────────────────────────┬────────────────────────┬───────────────┐
│ Layer (type)                    │ Output Shape           │       Param # │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ embedding (Embedding)           │ (None, 527, 100)       │    17,907,300 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm (LSTM)                     │ (None, 64)             │        42,240 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 64)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 64)             │         4,160 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 1)              │            65 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 17,953,765 (68.49 MB)
 Trainable params: 17,953,765 (68.49 MB)
 Non-trainable params: 0 (0.00 B)
```

Table 2: RNN-LSTM Model Summary

G. Note on Final Data Pre-processing and Model Architecture

The final versions of these models did not include date or subject data. All steps followed are the same except that the date and subject columns were removed before text normalization and were not included in the training or test datasets. This was to allow the model to take a third set of unseen data to be tested on. This data only contained the article title and text, so the final models were trained using only those aspects of the original dataset. Details as to why this was done can be found in the Results section.

Results

A. Results overview

The performance of measures of classifiers measures the decision-making capability of the classifier. The measures used to determine the performance are accuracy, precision, recall, F-score. Accuracy gives an overall effectiveness of a classifier. Precision is a measure of the class agreement of the data labels with the positive labels given by the classifier. Recall represents the classifier's effectiveness to identify class labels. Lastly, F- Score gives the relationship between positive labels and those given by the classifier. Both models were assessed on these metrics to determine which had higher performance.

B. Initial results

The initial results for both models showed 100% across all metrics, meaning that both models correctly identified every instance of fake and real news. While this may seem like a success, it is highly unlikely that such perfect performance reflects the true capabilities of the models. Achieving 100% accuracy often indicates that the models may be overfitting to the training data or that the dataset is not sufficiently challenging or diverse. Overfitting can occur when models become too tailored to the specific examples in the training set, failing to perform well on new, real-world data. Therefore, it is crucial to validate models with rigorous testing and cross-validation to assess their robustness and reliability in practical applications.

Steps to reduce overfitting such as a dropout layer, early stopping, and model checkpoint had already been employed. Therefore, it was decided to test the model on a completely new dataset. While the test data created from the initial dataframe was unseen to the model, it still followed a similar format to the training data. This could make it so that the patterns learned by the model during training apply effectively. However, if the new dataset includes variations or anomalies not present in the training data, the model's performance could reveal whether it generalizes well beyond the familiar patterns it was exposed to. This testing will provide a clearer picture of the model's robustness and its ability to perform reliably in real-world scenarios, where data can often be more diverse and unpredictable.

C. Subsequent results

The new data did not contain features for 'subject' and 'date', therefore these fields were removed from the original data before training the model again. The new results still showed 99-100% for all metrics for both models, showing that the lack of the 'subject' and 'date' columns did not have a significant impact on model performance for the training and testing data. When the model was deployed on the new data, however there was a significant decrease in performance.

As seen in tables 1-4 below, results showed that the CNN generally exhibited higher accuracy, with a score of 0.60 compared to 0.57 for the RNN-LSTM. In terms of precision, the CNN performed better in identifying real data with a score of 0.67 versus 0.53 for the RNN-LSTM, while both models had equal precision for fake data at 0.59. The recall scores show that the RNN-LSTM is more effective in recognizing real data with a recall of 0.42 compared to 0.21 for the CNN, but the CNN is better at identifying fake data with a recall of 0.91 compared to 0.69 for the RNN-LSTM. Finally, the F1 score, which balances precision and recall, is higher for the CNN when identifying fake data at 0.72 versus 0.64 for the RNN-LSTM, and higher for the RNN-LSTM when identifying real data at 0.47 versus 0.33 for the CNN. Overall, the CNN shows strength in identifying fake data with higher accuracy and recall, whereas the RNN-LSTM demonstrates a better capability in recognizing real data through higher recall and F1 scores.

	CNN	**RNN-LSTM**
Accuracy	0.60	0.57

Table 3: Accuracy

	CNN	**RNN-LSTM**
Real	0.67	0.53
Fake	0.59	0.59

Table 4: Precision

	CNN	**RNN-LSTM**
Real	0.21	0.42
Fake	0.91	0.69

Table 5: Recall

	CNN	**RNN-LSTM**
Real	0.33	0.47
Fake	0.72	0.64

Table 6: F1

The values in the CNN confusion matrix seen in Figure 1 represent the proportion of correct and incorrect predictions normalized by the actual class counts. For real news, 21.5% of cases were correctly classified, while 78.5% were misclassified as fake. For fake news, 91.4% were correctly identified, with 8.6% misclassified as real Overall, the CNN model shows high performance in identifying fake news but struggles with accurately classifying real news.

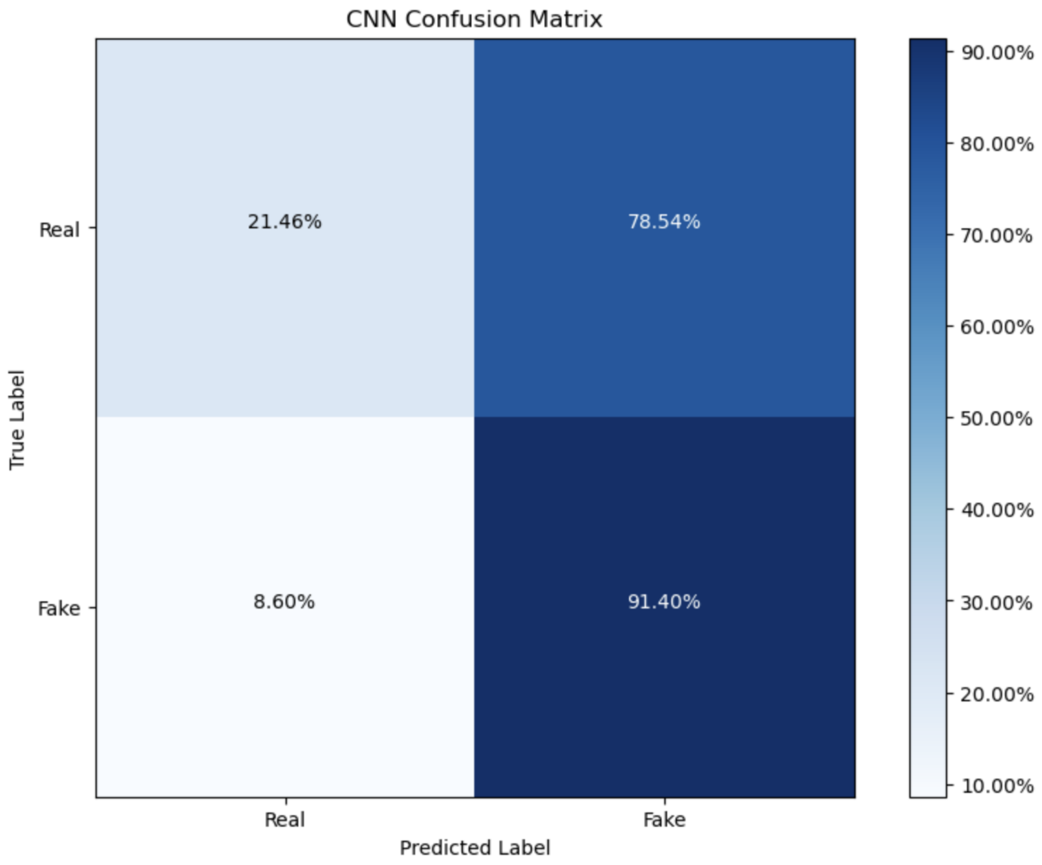


Fig. 1

In the RNN-LSTM confusion matrix in Figure 2, 42.3% of real news cases were correctly classified, with 57.7% misclassified as fake news. For the fake news category, 68.6% were correctly identified, with 31.2% misclassified as real.

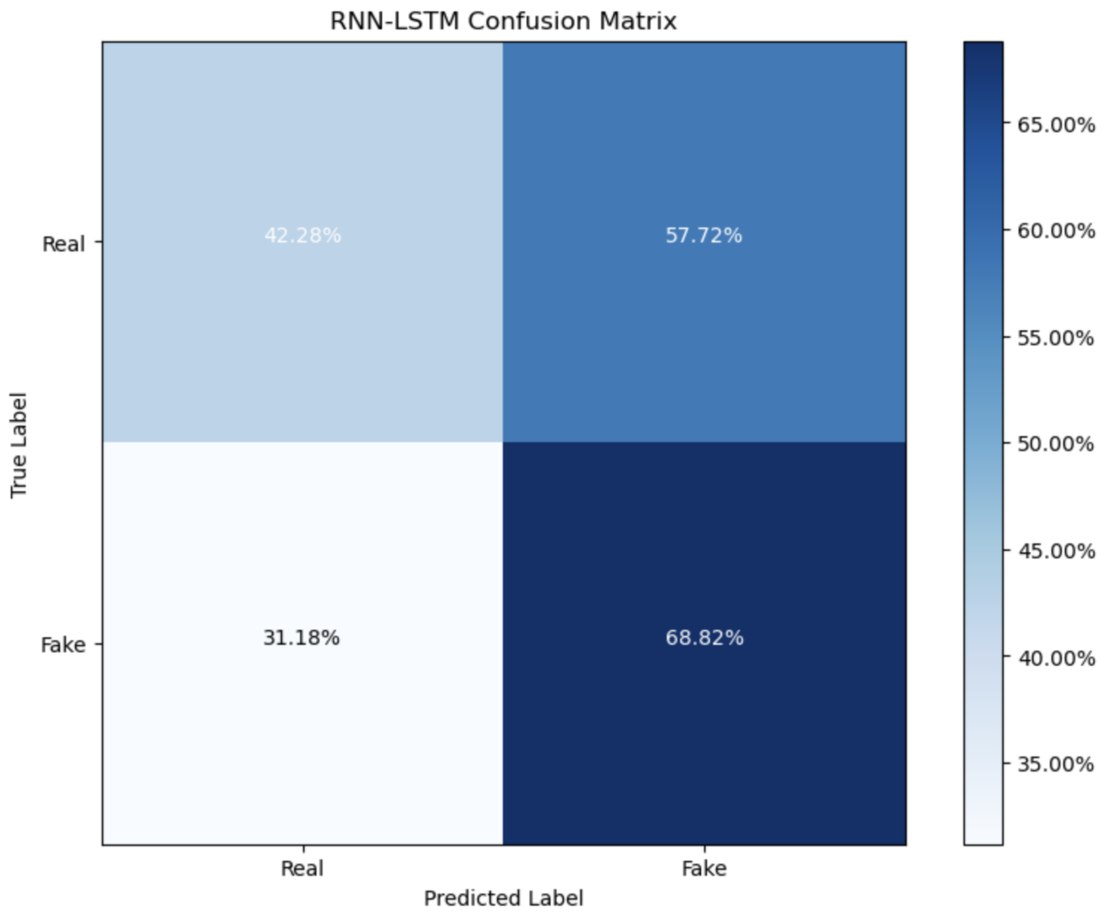


Fig 2.

Comparing the two models, the RNN-LSTM demonstrates better performance in correctly classifying real news (42.3% versus CNN's 21.5%). However, the CNN performs better in identifying fake news, with an accuracy of 91.4% compared to the RNN-LSTM's 68.8%. The CNN shows a higher misclassification rate for real news as fake, whereas the RNN-LSTM has a more balanced misclassification rate between the two categories. Overall, while the RNN-LSTM improves the accuracy for real news, the CNN excels in identifying fake news more accurately.

D. Drawbacks and Further Analysis

While both models had promising results on the training and test sets, its performance on the second unseen data left much to be desired. Steps were taken to perform hyperparameter tuning using the Keras tuner, but local processing power and disk space constraints consistently prevented tuning from running to completion. Performance is particularly critical in the context of fake news detection, where accurate predictions are paramount. Inaccurate detection of fake news can lead to the proliferation of misinformation, eroding public trust, and potentially influencing significant social, political, and economic outcomes. Addressing these constraints is essential not only to enhance model performance but also to ensure the reliability and impact of fake news detection systems in real-world applications.

To continue developing this model and mitigate issues like overfitting, several next steps are necessary. First, expanding the dataset to include a more diverse set of fake and real news sources can improve the model's generalization capabilities and robustness. Incorporating techniques such as additional regularization methods and advanced cross-validation strategies can help reduce overfitting by ensuring that the model does not excessively memorize the training data. Exploring more powerful computing resources, such as cloud-based GPUs or distributed computing environments, can facilitate more extensive hyperparameter tuning and model optimization. Finally, experimenting with different model architectures and feature engineering approaches, such as leveraging pre-trained word embeddings or incorporating meta-data, may further enhance performance and prevent overfitting.

Discussion/Ethics

The ethical implications of using machine learning for fake news detection are both multifaceted and significant. The primary concern centers around algorithmic bias, where systems trained on datasets that might include historical biases could lead to discriminatory outcomes against certain content or perspectives. For example, these biases might cause a model to unfairly target less mainstream sources or fail to identify misinformation from more prominent outlets. Additionally, the opacity of machine learning models, especially those based on deep learning, can lead to a lack of transparency. This "black box" nature complicates users' understanding of why decisions were made, making it difficult to trust or even verify the system's actions.

There is also a risk of over-reliance on these automated systems. They might not fully grasp context, nuance, or the evolving nature of misinformation, which can suppress legitimate discourse or create a false sense of security about the information's veracity online. Privacy concerns are also important as these systems often access extensive data pools, potentially including personal information. This necessitates robust data protection measures to maintain user trust.

Inaccurate fake news detection can further erode trust and distort public discourse. If these systems fail to identify false information correctly it can spread unchecked, misleading the public and potentially influencing critical outcomes like elections or public policies. Such failures not only undermine trust in detection systems but can lead to increased public skepticism towards all information sources. Therefore, continuous refinement of algorithms, maintaining transparency, establishing clear accountability, and protecting privacy are essential to ensure that machine learning tools support a fair and well-informed public discourse.

References

1. Murphy, K. M. (2023). Fake News and the Web of Plausibility. *Social Media + Society*, *9*(2). https://doi.org/10.1177/20563051231170606
2. Jardine, E. (2019, April 2). *Beware Fake News*. Centre for International Governance Innovation. https://www.cigionline.org/articles/beware-fake-news/?utm_source=google_ads&utm_medium=grant&gad_source=1&gclid=Cj0KCQjwtsy1BhD7ARIsAHOi4xalHST_VzDjV5dNPfuVKqzNbjxZqHa0-nCRg51jNMa5jANsvYEMqCgaAjPXEALw_wcB#gad_source=1
3. Aïmeur, E., Amri, S., & Brassard, G. (2023). Fake news, disinformation and misinformation in social media: a review. *Social network analysis and mining*, *13*(1), 30. https://doi.org/10.1007/s13278-023-01028-5
4. Mridha, M. F., Keya, A. J., Hamid, Md. A., Monowar, M. M., & Rahman, Md. S. (2021). A Comprehensive Review on Fake News Detection With Deep Learning. *IEEE Access*, *9*, 156151–156170. https://doi.org/10.1109/access.2021.3129329
5. Saleh, H., Alharbi, A., & Alsamhi, S. H. (2021). OPCNN-FAKE: Optimized Convolutional Neural Network for Fake News Detection. *IEEE Access*, 1–1. https://doi.org/10.1109/access.2021.3112806
6. Abdullah, Yasin, A., Awan, M., Faisal Shehzad, H., & Ashraf, M. (2020). Fake news classification bimodal using convolutional neural network and long short-term memory. *International Journal of Emerging Technologies in Learning (iJET)*, 11, 209-212.
7. Fake News Detection Datasets. (n.d.). *Kaggle.* www.kaggle.com. https://www.kaggle.com/datasets/emineyetm/fake-news-detection-datasets
8. William Lifferth. (2018). Fake News. *Kaggle*. https://kaggle.com/competitions/fake-news
9. Bahad, P., Saxena, P., & Kamal, R. (2019). Fake News Detection using Bi-directional LSTM-Recurrent Neural Network. *Procedia Computer Science*, *165*, 74–82. https://doi.org/10.1016/j.procs.2020.01.072