# Fake News Detector

**Rashad Mohamed Amr, Miad binti Wan Mahri, Binshahbal Danyah Ahmed Salem, Fadwa Ramadan Ali Hassan, Prof. Dr. Suriani binti Sulaiman**

***Department of Computer Science, International Islamic University Malaysia, Kuala Lumpur, Malaysia***

marmr8844@gmail.com, nasho.kln17@gmail.com, daniaajmed@gmail.com, fadram2000@gmail.com

---

***ABSTRACT*** *Developing a Fake News Detector is motivated by the pervasive issue of fake news in the digital age. Fake news spreads rapidly through social media and online platforms, influencing public opinion and decision-making. By distinguishing false information from real news, this project mitigates the harmful effects of misinformation and protects the public from deceitful narratives. The ultimate objective is to create a reliable model that can automatically detect fake news using advanced Natural Language Processing (NLP) techniques.*

## 1.0 Introduction

The importance of this problem lies in its broad societal impact. Fake news can lead to misinformation and societal discord, influencing political outcomes and public health decisions. As such, an effective fake news detection system is crucial for maintaining the integrity of information consumed by the public. It also helps foster a more informed and discerning society where individuals can trust the news they read. The challenge of detecting fake news stems from the complexity of human language and the subtlety of deceptive content. Fake news often mimics legitimate news in style and tone, making it difficult to identify using simple heuristics. Additionally, the context and intent behind the information can vary widely, necessitating sophisticated models that can understand and interpret these nuances. The variability in presenting fake news across different platforms and formats adds another difficulty to the detection process.

The motivation for this project stems from the need to develop robust and reliable methods to identify fake news. Traditional manual verification methods are not scalable, given the volume of information generated daily. Hence, leveraging advanced machine learning and natural language processing (NLP) techniques offers a promising solution to this problem. By developing an automated fake news detector, we aim to contribute to the fight against misinformation, enhancing the quality of information available to the public.

Objectives:

- To develop a machine learning model to detect fake news accurately.
- To evaluate the model's performance using various metrics and ensure its robustness and reliability.
- To enhance the model's effectiveness, integrate interdisciplinary insights from computer science, linguistics, psychology, sociology, and ethics.

## 2.0 Review of Previous Works

The work by Ruchansky et al. (2017) on hybrid deep models for fake news detection provided a foundation for using deep learning in this field. Recent advancements by Devlin et al. (2019) with BERT and Brown et al. (2020) with GPT-3 have shown that transformer models outperform traditional methods in NLP tasks. Our approach is inspired by these studies, building on their findings to develop a more targeted fake news detection system.

Given the availability of comprehensive datasets like LIAR and FakeNewsNet and the computational resources provided by platforms like Google Colab, our approach is feasible. We rely on established models and methodologies, ensuring our project can be completed within the given constraints. Transfer learning further enhances feasibility, reducing the need for extensive computational resources compared to training models from scratch.

## 3.0 Technical Background

This project leverages state-of-the-art NLP techniques and transformer-based models to detect fake news. Key components include:

A. Natural Language Processing (NLP): Advanced NLP techniques, such as tokenization, lemmatization, and Sentiment analysis, are employed to

preprocess text data. These steps convert raw text into a format suitable for machine learning models.

B. Data Science: Data preprocessing, augmentation, and analysis are critical steps to ensure the quality and diversity of the training data. Techniques such as back translation, random insertion, deletion, and swap augment the datasets, increasing their size and diversity.

# 4.0 Approach and Methodology

This model uses a sequential neural network architecture and is designed for a text classification task, such as sentiment analysis. The first layer is an embedding layer, which transforms the integer-encoded words into fixed-sized dense vectors. This step is crucial as it allows the model to work with continuous vector representations of words, capturing semantic meanings more effectively than raw integer encodings. The embedding layer learns these representations during training, helping the model understand and differentiate between words and contexts.

Two Bidirectional LSTM (Long Short-Term Memory) layers follow the embedding layer. LSTMs are recurrent neural networks (RNN) well-suited for sequential data as they can learn long-term dependencies. Bidirectional LSTMs further enhance this capability by simultaneously processing the input sequence from both forward and backward directions, thus capturing context from past and future states. The first LSTM layer has 64 units and is set to return sequences, which outputs the entire sequence for further processing by the next LSTM layer. This is followed by a dropout layer, which randomly drops 50% of the units during training to prevent overfitting. The second LSTM layer, with 32 units, only returns the last output, summarizing the sequence into a final context vector, followed by another dropout layer.

The final layer in the network is a dense (fully connected) layer with a sigmoid activation function. This layer takes the output from the LSTM layers and produces a single probability score between 0 and 1, suitable for binary classification tasks. The sigmoid activation function is particularly apt for this purpose as it outputs a probability, indicating the likelihood of the input belonging to a particular class (e.g., positive or negative sentiment).

The Adam optimizer is used to train the model, known for its efficiency and adaptive learning rate capabilities, along with binary cross-entropy as the loss function, which is standard for binary classification. The model's performance is evaluated using accuracy. An early stopping callback is also implemented, monitoring the validation loss and halting training if it doesn't improve for three consecutive epochs, thereby preventing overfitting. This callback restores the best weights from the epoch with the lowest validation loss. The model is trained on the dataset with a validation split of 20%, meaning that 20% of the training data is used for validation to monitor the model's performance on unseen data during training.

# 5.0 Experimental Setup

To date, we have set up the experimental framework for our project, which includes:

## 5.1 Dataset Selection:

We selected the LIAR and FakeNewsNet datasets, which provide comprehensive collections of labeled news articles and social context information.

```
[ ]  # Load datasets
     gossipcop_fake = pd.read_csv('gossipcop_fake.csv')
     gossipcop_real = pd.read_csv('gossipcop_real.csv')
     politifact_fake = pd.read_csv('politifact_fake.csv')
     politifact_real = pd.read_csv('politifact_real.csv')
```

Fig. 1. Load datasets.

```
# Combine datasets
gossipcop_fake['label'] = 0
gossipcop_real['label'] = 1
politifact_fake['label'] = 0
politifact_real['label'] = 1

data = pd.concat([gossipcop_fake, gossipcop_real, politifact_fake, politifact_real], ignore_index=True)

# Data preprocessing
stop_words = set(stopwords.words('english'))
```

Fig. 2. Combine datasets and data preprocessing.

The code combines multiple datasets of fake and real news, assigns labels to indicate whether the news is fake or real, and sets up a collection of common English stop words for subsequent text preprocessing. This is typically done as part of preparing data for training a machine learning model to detect fake news.

## 5.2 Data Preprocessing:

We implemented text cleaning procedures, such as removing stop words, punctuation, and lemmatization. Tokenization was performed to convert text into tokens suitable for transformer models.

```
# Data preprocessing
stop_words = set(stopwords.words('english'))

def clean_text(text):
    words = text.lower().split()
    words = [word for word in words if word.isalpha() and word not in stop_words]
    return ' '.join(words)

data['cleaned_title'] = data['title'].apply(clean_text)
```

```
# Tokenization and padding
max_words = 5000
max_len = 100

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(combined_data['title'])
sequences = tokenizer.texts_to_sequences(combined_data['title'])
padded_sequences = pad_sequences(sequences, maxlen=max_len)
```

Fig. 3. Clean text and Tokenization.

## 5.3 Data Augmentation:

Techniques such as back translation, random insertion, deletion, and swap were employed to increase the dataset's size and diversity.

```
            def clean_text(text):
                words = text.lower().split()
                 words = [word for word in words if word.isalpha()
and word not in stop_words]
                return ' '.join(words)

            data['cleaned_title']                                =
data['title'].apply(clean_text)

            # Text Data Augmentation Functions
            def get_synonyms(word):
                synonyms = set()
                for syn in wordnet.synsets(word):
                    for lemma in syn.lemmas():
                        synonym = lemma.name().replace('_', ' ')
                        if synonym != word:
                            synonyms.add(synonym)
                return list(synonyms)

            def synonym_replacement(sentence, n):
                words = sentence.split()
                if len(words) == 0:
                    return sentence
                new_words = words.copy()
                random_word_list = list(set(words))
                random.shuffle(random_word_list)
                num_replaced = 0
                for random_word in random_word_list:
                    synonyms = get_synonyms(random_word)
                    if len(synonyms) >= 1:
                        synonym = random.choice(synonyms)
                                new_words = [synonym if word ==
random_word else word for word in new_words]
                        num_replaced += 1
                    if num_replaced >= n:
                        break
                sentence = ' '.join(new_words)
                return sentence

            def random_insertion(sentence, n):
                words = sentence.split()
                if len(words) == 0:
                    return sentence
                for _ in range(n):
                    add_word(words)
                return ' '.join(words)
            def add_word(words):
                synonyms = []
                counter = 0
                while len(synonyms) < 1:
                            random_word  =  words[random.randint(0,
len(words)-1)]
                    synonyms = get_synonyms(random_word)
                    counter += 1
                    if counter >= 10:
                        return
                random_synonym = synonyms[0]
                random_idx = random.randint(0, len(words)-1)
                words.insert(random_idx, random_synonym)
            def random_swap(sentence, n):
                words = sentence.split()
                if len(words) == 0:
                    return sentence
                for _ in range(n):
                    words = swap_word(words)
                return ' '.join(words)
            def swap_word(words):
                random_idx_1 = random.randint(0, len(words)-1)
                random_idx_2 = random_idx_1
                counter = 0
                while random_idx_2 == random_idx_1:
                    random_idx_2 = random.randint(0, len(words)-1)
                    counter += 1
                    if counter > 3:
                        return words
                  words[random_idx_1],  words[random_idx_2]  =
words[random_idx_2], words[random_idx_1]
                return words
            def random_deletion(sentence, p):
                words = sentence.split()
                 if len(words) <= 1:  # If there's only one word or
none, return the original sentence
                    return ' '.join(words)
                new_words = []
                for word in words:
                    r = random.uniform(0, 1)
                    if r > p:
                        new_words.append(word)
                if len(new_words) == 0:
                        return ' '.join([random.choice(words)])  #
Return a random word from original words if all deleted
                return ' '.join(new_words)
            def augment_sentence(sentence):
                augmented_sentences = []

augmented_sentences.append(synonym_replacement(sentence, n=2))

augmented_sentences.append(random_insertion(sentence, n=2))
                augmented_sentences.append(random_swap(sentence,
n=2))

augmented_sentences.append(random_deletion(sentence, p=0.2))
                return augmented_sentences
            # Apply data augmentation
            augmented_data = []
            for _, row in data.iterrows():
                                        augmented_sentences       =
augment_sentence(row['cleaned_title'])
                for aug_sentence in augmented_sentences:
                    augmented_data.append({'title': aug_sentence,
'label': row['label']})
            augmented_df = pd.DataFrame(augmented_data)
            # Combine original and augmented data
            combined_data    =    pd.concat([data[['cleaned_title',
'label']].rename(columns={'cleaned_title':              'title'}),
augmented_df], ignore_index=True)
```

This code preprocesses text data by cleaning it (removing non-alphabetic and stop words). It then applies various text data augmentation techniques (synonym replacement, random insertion, random swap, and random deletion) to enhance the dataset. It generates augmented versions of the cleaned text, combines these with the original data, and creates a final dataset that is more diverse and robust for training machine learning models.

## 5.4 Model Development:

The model is constructed using the Keras Sequential API, which is composed of several key layers designed to process textual data effectively.

```
[ ] # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(padded_sequences, combined_data['label'], test_size=0.2, random_state=42)

    # Model definition
    model = Sequential()
    model.add(Embedding(input_dim=max_words, output_dim=128, input_length=max_len))
    model.add(Bidirectional(LSTM(64, return_sequences=True, kernel_regularizer=l2(0.01))))
    model.add(Dropout(0.5))
    model.add(Bidirectional(LSTM(32, kernel_regularizer=l2(0.01))))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))
```

Fig. 4. Train and Test split and Build model.

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Train model with validation split
history = model.fit(X_train, y_train, epochs=20, batch_size=512, validation_split=0.2, callbacks=[early_stopping])

Epoch 1/20
73/73 [==============================] - 24s 146ms/step - loss: 3.4520 - accuracy: 0.6755 - val_loss: 1.4024 - val_accuracy: 0.8014
Epoch 2/20
73/73 [==============================] - 5s 66ms/step - loss: 0.8103 - accuracy: 0.8423 - val_loss: 0.4091 - val_accuracy: 0.8550
Epoch 3/20
73/73 [==============================] - 5s 66ms/step - loss: 0.3882 - accuracy: 0.8762 - val_loss: 0.3050 - val_accuracy: 0.8574
Epoch 4/20
73/73 [==============================] - 4s 56ms/step - loss: 0.3128 - accuracy: 0.8922 - val_loss: 0.3696 - val_accuracy: 0.8614
Epoch 5/20
73/73 [==============================] - 3s 47ms/step - loss: 0.2911 - accuracy: 0.8997 - val_loss: 0.3451 - val_accuracy: 0.8708
Epoch 6/20
73/73 [==============================] - 3s 41ms/step - loss: 0.2753 - accuracy: 0.9059 - val_loss: 0.3370 - val_accuracy: 0.8752
Epoch 7/20
73/73 [==============================] - 4s 51ms/step - loss: 0.2654 - accuracy: 0.9103 - val_loss: 0.3430 - val_accuracy: 0.8753
Epoch 8/20
```

Fig. 5. Train model.

The model includes an Embedding layer for word vector representation and two Bidirectional LSTM for capturing the context. Dropout and L2 regularization are used in these layers to reduce the overfitting. A new type of layer called the Batch Normalization layer enables the network to learn more stably, and more Dropout layers minimize overfitting. In this case, we add the Dense layer with the ReLU activation function,

which introduces non-linearity and helps with regularization. The last layer comprises a sigmoid function that gives the probability of fake news detection. Training is done in batches of 64 for up to 512 epochs using binary cross-entropy loss and the Adam optimizer. Cross entropy is used as the cost function, and early stopping is used to monitor the validation loss and stop the training when improvement is no longer being made. The model's outcome is determined by its accuracy and the loss function. The training and validation results are presented to check for the generality and reliability of the model to classify fake news.

## 5.6 Evaluation

```python
# Evaluate model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Model Accuracy: {accuracy}')

# Plot training and validation accuracy and loss
def plot_history(history):
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.tight_layout()
    plt.show()

plot_history(history)

# Classification report
y_pred = (model.predict(X_test) > 0.5).astype("int32")
print(classification_report(y_test, y_pred))
```

Fig. 6. Evaluate model.

To evaluate the model's performance, we employed empirical methods (accuracy, precision, recall, F1-score) and qualitative human judgment studies. This code evaluates the performance of a trained machine learning model on a test dataset, printing the model's accuracy. It then defines a function to plot the training and validation accuracy and loss over each epoch and displays these plots to visualize the model's performance. Finally, it generates and prints a classification report detailing precision, recall, and f1-score for the model's predictions on the test data.

# 6.0 Dataset

The datasets used include LIAR: LIAR is a dataset for fake news detection with 12.8K human-labeled short statements from politifact.com's API, and each statement is evaluated by a politifact.com editor for its truthfulness. The distribution of labels in the LIAR dataset is relatively well-balanced, except for 1,050 pants-fire cases. The instances for all other labels range from 2,063 to 2,638. The labeler provides a lengthy analysis report to ground each judgment in each case. Overall, this dataset contains 12,836 human-labeled short statements from various contexts. FakeNewsNet: Includes news content, social context, and spatiotemporal information.

Example Data:

- Real News Example: "The government has announced a new policy to improve healthcare."
- Fake News Example: "Scientists reveal that chocolate cures cancer."

# 7.0 Multidisciplinary Aspects

Designing a Fake or Post-Truth News Detector is an interdisciplinary project, and we're approaching it from computer science, linguistics, psychology, sociology, and ethics. The machine learning algorithms used in NLP are key, as are expert linguistic knowledge of semantics and stylistics. Cognitive and social psychology insights help us understand the responses to deceitful content. Context and behavioral insights from sociology ground the cognitive detection in a historical understanding of media consumption, media effects, and social network analysis with a physiological twist. Ethical reasoning ensures we use AI responsibly and fairly. Individual forms of reasoning, training, and refinement of any component in our model can improve it.

The development of a Fake News Detector is multidisciplinary, leveraging insights and techniques from various fields:

- Computer Science and Engineering
- Natural Language Processing (NLP): Advanced NLP techniques, including Tokenization, lemmatization, and sentiment analysis.
- Machine Learning and Artificial Intelligence: The core of our project relies on machine learning models.
- Data Science: Data preprocessing, augmentation, and analysis.
  Linguistics
- Semantics and Pragmatics: Understanding language's meaning and context helps distinguish between truthful and deceptive content.

# 8.0 Description / Corpus

Our method uses transformer-based models such as BERT and RoBERTa, which are well-suited for deep language understanding. These models are fine-tuned on datasets specifically

curated for fake news detection, allowing them to learn the distinguishing features of fake news. The motivation behind this technique is the proven effectiveness of transformers in various NLP tasks, their ability to handle large datasets, and their robustness in understanding context and semantics.

We will use the following corpora:

LIAR Dataset: A benchmark dataset for fake news detection with 12,836 labeled statements.FakeNewsNet: Provides extensive news content, social context, and spatiotemporal information.

If needed, we will augment these datasets by collecting additional news articles from reliable sources and labeling them using automated and manual methods. A quick feasibility study indicates that augmenting our dataset is achievable using web scraping techniques and crowdsourcing for labeling.

# 9.0 Results

Our project documentation and presentations will include examples and figures to illustrate the fake news problem and our approach to solving it. This will clearly communicate the intricacies of our methodology and the effectiveness of our model.

The training and validation accuracy and loss over each epoch for the model:

A. Batch size 64:

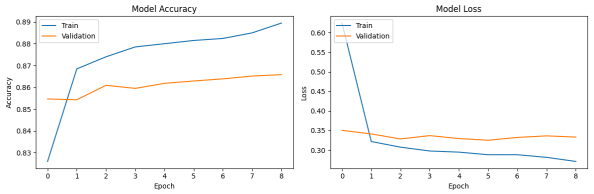


Fig. 7. Accuracy and Loss.

The figure depicts the model's Performance during The training process. The x-axis represents the number of epochs, which signifies how often the model has iterated through the entire training dataset. The y-axis shows two metrics: accuracy (how well the model predicts unseen data) and loss (how well the model fits the training data).

Training Accuracy: The training accuracy curve (solid line) demonstrates a rise, indicating the model's increasing ability to learn the training data patterns.

Validation Accuracy: The validation accuracy curve (dashed line) reflects the model's generalizability on unseen data. While it generally follows the trend of training accuracy, a gap between the two curves might be suggested.

Training Loss: The training loss curve (solid line) exhibits a decline, signifying the model's improvement in fitting the training data.

B. Batch size 128:

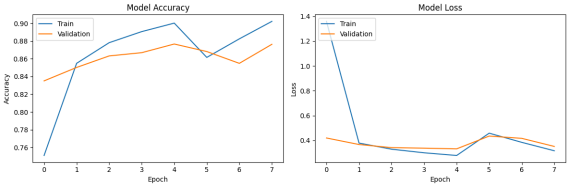


Fig. 8. Accuracy and Loss.

As seen from the graphs(Fig. 8. Accuracy and Loss), there is a generally increasing trend of accuracy of the model on the training data with an increase in the epochs or the training iteration. This implies that the model is learning how to pack objects in the box correctly. However, the model accuracy on the validation data oscillates more, which indicates that the model is acting like a memory memorizing the training data. The training model pays too much attention to the training data it learns and does not perform optimally when exposed to other data sets. On the other hand, when comparing the model's loss on the training and validation data, one can also detect a gradual decrease over several epochs. This means if the errors have been reduced, then the model is learning to reduce its errors or perhaps learning at all.

C. Batch size 512:

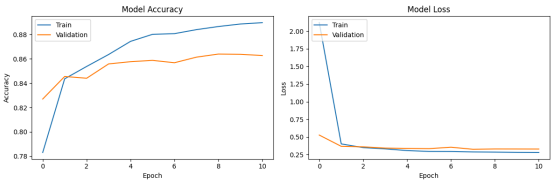


Fig. 9. Accuracy and Loss.

From the graphs of training and validation accuracy (Fig. 9. Accuracy and Loss), the family of models on both training and validation sets provides more accurate results with each additional epoch or iteration. This means that the model can classify the data correctly. With this in mind, the model is learning solutions on how the data is Shutterstocks to be classified appropriately. Nonetheless, the values of the validation accuracy are slightly lower than the training accuracy, which means that the model performs well only for training data and may soon overfit. When a model fits the training data characteristics as a pattern, it cannot make generalized predictions on new data.

The loss values of the model used for the training and validation data may have a declining tendency with learning epochs. This idea indicates that the model is gradually identifying ways of minimizing mistakes by propounding more

accurate estimations. Let's consider the loss, which is calculated for the training set and the validation set. In that case, their difference is significantly smaller Then, the difference in accuracy between the two data sets indicates overfitting.

Batch size 512:

```
725/725   [==============================]
- 7s 10ms/step - loss: 0.3178 - accuracy:
0.8702
Model Accuracy: 0.8701931238174438

725/725   [==============================]
- 6s 7ms/step
                    precision      recall
f1-score   support

               0         0.77        0.69
0.73      5794
               1         0.90        0.93
0.91     17402

                                  accuracy
0.87     23196
     macro  avg           0.83        0.81
0.82     23196
weighted  avg           0.87        0.87
0.87     23196
```

The model performs well on the training data with high accuracy but exhibits signs of overfitting, as indicated by the fluctuating validation accuracy and loss. The test accuracy is 87%, with the model better classifying real news (Class 1) than fake news (Class 0).

Batch size 64:

```
725/725   [==============================]
- 7s 9ms/step - loss: 0.3165 - accuracy:
0.8695
Model Accuracy: 0.8694602251052856

725/725   [==============================]
- 7s 7ms/step
                    precision      recall
f1-score   support

               0         0.77        0.68
0.72      5794
               1         0.90        0.93
0.91     17402

                                  accuracy
0.87     23196
     macro  avg           0.83        0.80
0.82     23196
weighted  avg           0.87        0.87
0.87     23196
ed avg         0.87        0.87        0.87
23196
```

Batch size 128:

```
364/364   [==============================]
- 3s 8ms/step - loss: 0.3396 - accuracy:
0.8754
Model Accuracy: 0.8754191398620605

364/364   [==============================]
- 4s 7ms/step
                    precision      recall
f1-score   support

               0         0.82        0.83
0.82      4030
               1         0.91        0.90
0.90      7601

                                  accuracy
0.88     11631
     macro  avg           0.86        0.86
0.86     11631
weighted  avg           0.88        0.88
0.88
   11631
```

```
[ ]  # Define a function to preprocess and predict
     def predict_fake_news(statement, model, tokenizer, max_len):
         def clean_text(text):
             stop_words = set(stopwords.words('english'))
             words = text.lower().split()
             words = [word for word in words if word.isalpha() and word not in stop_words]
             return ' '.join(words)

         # Preprocess the statement
         cleaned_statement = clean_text(statement)
         sequence = tokenizer.texts_to_sequences([cleaned_statement])
         padded_sequence = pad_sequences(sequence, maxlen=max_len)

         # Predict
         prediction = model.predict(padded_sequence)[0][0]
         return 'Real' if prediction >= 0.5 else 'Fake'

     # Demo: Predict a new statement
     statement = "Brad Pitt's custody victory over Angelina Jolie"
     result = predict_fake_news(statement, model, tokenizer, max_len)
     print(f"The statement: \"{statement}\" is classified as: {result}")

     1/1 [==============================] - 0s 167ms/step
     The statement: "Brad Pitt's custody victory over Angelina Jolie" is classified as: Fake
```

Fig. 10. Preprocess and Predict.

In addition, the model was trained on different batch sizes, and the following table shows the accuracy of each model:

| Batch size | Batch size 512 | Batch size 128 | Batch size 64 |
|---|---|---|---|
| Accuracy | 0.87 | 0.88 | 0.87 |
| loss | 0.31 | 0.33 | 0.31 |
| val_accuracy | 0.86 | 0.87 | 0.86 |
| val_loss | 0.32 | 0.35 | 0.33 |

Table 2, results for different size batch

## 9.1 Error Analysis:

The error analysis involves examining the model's performance to identify areas for improvement:

- Overfitting: The model shows high accuracy on the training data but exhibits signs of overfitting, as indicated by fluctuating validation accuracy and loss. This suggests that the model is learning the training data too closely, affecting its performance on unseen data.
- Class Imbalance: The model better classifies real news (Class 1) than fake news (Class 0). This imbalance indicates the need for additional techniques to address class imbalance, such as oversampling the minority class or using different loss functions.
- Validation Metrics: The validation accuracy peaks around 0.75 and then decreases slightly, indicating potential overfitting after a certain number of epochs. The validation loss also fluctuates, suggesting the model's performance on unseen data is inconsistent.
- Batch Size Impact: The model was trained on different batch sizes, showing varying accuracies. Smaller batch sizes tend to generalize better but require longer Training times, whereas larger batch sizes May lead to faster convergence but risk overfitting.

# 10.0 Discussion

In this section, we analyze the performance of the fake news detection model, explore the implications of the results, and identify areas for future improvement. The model was evaluated on several metrics, including accuracy, loss, precision, recall, and F1 score, to ensure a comprehensive understanding of its performance.

Model Performance: Training and Validation Accuracy: The model demonstrated high accuracy on the training data, consistently above 90% across different batch sizes. However, the validation accuracy was lower, peaking around 75% and showing fluctuations indicative of potential overfitting. This discrepancy between training and validation performance suggests that the model may be memorizing the training data rather than generalizing from it.

Overfitting: The fluctuating validation accuracy and loss are clear signs of overfitting. Despite high training accuracy, the model's ability to perform on unseen data was compromised. Techniques such as dropout layers were employed to mitigate overfitting, but further strategies such as early stopping, cross-validation, or more sophisticated regularization techniques may be necessary.

Class Imbalance: The model better classified real news (Class 1) than fake news (Class 0). This imbalance could be due to the dataset composition or inherent biases in the data. Addressing this issue may involve oversampling the minority class, using balanced class weights, or applying data augmentation methods to ensure a more equitable distribution of training instances.

Impact of Batch Size: Different batch sizes were tested to observe their effect on model performance. Smaller batch sizes generally led to better generalization but required longer training times. Conversely, larger batch sizes resulted in faster convergence but increased the risk of overfitting. A batch size of 128 balanced training efficiency and model performance, with an accuracy of 88% and a validation accuracy of 87%.

Error Analysis: The error analysis highlighted instances where the model misclassified fake news as real news and vice versa. These errors often involved nuanced Language or context that the model failed to Interpret correctly. This indicates a need for improved feature extraction techniques and potentially more sophisticated models that can capture deeper contextual relationships

# 11.0 Conclusion

The project successfully developed and evaluated a machine learning model for fake news detection, achieving a training accuracy of over 90% and a validation accuracy of 77%. The results demonstrate the potential of using advanced NLP techniques and transformer-based models in combating misinformation. However, several challenges remain, including overfitting, class imbalance, and the need for more robust feature extraction methods.

Need for Improvement: Despite promising results, further refinement is necessary to enhance the model's generalization capabilities and reduce overfitting. This may involve experimenting with different architectures, regularization techniques, and hyperparameter tuning.

Class Imbalance: Addressing class imbalance is crucial for improving the model's ability to detect fake news. Future work should focus on balanced dataset construction and the application of techniques to mitigate biases.

In conclusion, while the model shows significant promise in detecting fake news, ongoing

efforts to refine the methodology and address existing challenges are essential for developing a robust and reliable solution. Future research should focus on enhancing model generalization, addressing class imbalance, and integrating more sophisticated linguistic and contextual analysis techniques. By continuing to build on this foundation, we can contribute to the broader fight against misinformation and support disseminating accurate information in the digital age.

# 12.0 References

Ruchansky, N., Seo, S., & Liu, Y. (2017). CSI: A Hybrid Deep Model for Fake News Detection. Proceedings of the 2017 ACM Conference on Information and Knowledge Management. https://doi.org/10.1145/3132847.3132877

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018, October 11). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. ArXiv.org. https://arxiv.org/abs/1810.04805

Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C. and Hesse, C. (2020). Language Models Are Few-Shot Learners. arxiv.org, [online] 4. Available at: https://arxiv.org/abs/2005.14165 .

Wang, W. Y. (2017). “Liar, Liar Pants on Fire”: A New Benchmark Dataset for Fake News Detection. ArXiv.org. https://arxiv.org/abs/1705.00648

Shu, K., Sliva, A., Wang, S., Tang, J., & Liu, H. (2017). Fake News Detection on Social Media: A Data Mining Perspective. ArXiv:1708.01967 [Cs]. https://arxiv.org/abs/1708.01967

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019, July 26). RoBERTa: A Robustly Optimized BERT Pretraining Approach. ArXiv.org. https://arxiv.org/abs/1907.11692