DSCI-6004-03
NATURAL LANGUAGE PROCESSING

FINAL REPORT

PARAPHRASE DETECTION

TEAM MEMBERS:

Sreeja Garlapati

Bharath Kondreddy

Mahesh Gundagoni

GitHub: https://github.com/garlapatisreeja/Paraphrase-Detection.git

ABSTRACT:

This project addresses the challenge of paraphrase detection in natural language processing, aiming to enhance the comprehension of sentence similarity. Leveraging a robust dataset labeled with paraphrased and non-paraphrased sentences, a small BERT model was trained to accurately identify paraphrases. The dataset, sourced from Kaggle, underwent thorough preprocessing, including null value handling, shuffling, and batching. The selected model demonstrated significant effectiveness, achieving a binary accuracy of approximately 75.55% and an F1 score of 81.44% on the test dataset. This outcome highlights the success of the project in employing advanced NLP techniques, emphasizing effective preprocessing, model selection, and training for practical applications in paraphrase detection.

INTRODUCTION:

In the field of natural language processing (NLP), being able to figure out if two sentences mean the same thing is crucial for making applications like chatbots and search engines work better. This project focuses on the task of recognizing paraphrased sentences, aiming to make our understanding of similar sentences better using advanced methods.

In today's world, where we use a lot of NLP applications, accurately spotting paraphrased sentences has become important. Whether it's a chatbot responding to us or a search engine giving us results, the ability to know if two sentences convey the same idea is key to improving our experience. Traditional methods of paraphrase detection, like rule-based systems, have had limitations. But newer models, especially ones like BERT, have changed the game by being great at understanding context.

By this project, we expect not just accuracy but also contextually fitting responses. This expectation relies on the system being good at recognizing paraphrased expressions, ensuring smooth and relevant interactions. To tackle this challenge, we carefully put together a strong dataset from Kaggle.

We made sure the data was in good shape for training our model.

Key to our project's success is the choice of the BERT (Bidirectional Encoder Representations from Transformers) model. We specifically went for a smaller version, 'small_bert/bert_en_uncased_L-4_H-512_A-8,' from TensorFlow Hub. This model's design is lighter, making it a good fit for situations where we don't have a lot of resources.

We used BERT as the main part, along with layers for preparing the data, a step for preventing the model from getting too specialized (dropout), and a layer for making the final decision. We trained the model for 10 rounds, keeping an eye on important measures like how often it got things right and a score that balances precision and recall. The training showed that our model consistently got better at spotting paraphrased sentences.

After all the training, we tested our model on a separate dataset. The results tell us that our model is good at recognizing paraphrased sentences, balancing how often it's right and how often it doesn't miss anything important. Supported by charts showing loss and accuracy, our project not only meets its immediate goal but also lays the foundation for more exploration and improvement in NLP.

OBJECTIVE:

The primary goal of this project is to develop an intelligent paraphrase detection system, leveraging advanced NLP techniques. The specific objectives include:

1. **Enhancing Comprehension of Sentence Similarity:**
    - Improve our understanding of the similarity between sentences by accurately detecting paraphrases.
    - Contribute to the refinement of NLP applications, ensuring more precise and contextually relevant interactions.
2. **Dataset Curation and Preparation:**
    - Curate a robust dataset consisting of labeled sentences, differentiating between paraphrased (1) and non-paraphrased (0) instances.
    - Efficiently batch the data for training purposes, ensuring optimal model performance.

3. **Model Selection and Architecture:**
    - Construct a classifier model with BERT as the encoder, incorporating preprocessing layers, dropout for regularization, and a dense layer for output.
4. **Training and Performance Monitoring:**
    - Train the classifier model for 10 epochs, monitoring key metrics such as binary accuracy and F1 score for both training and validation datasets.
    - Ensure the model shows steady improvement in accuracy and F1 score, indicating proficiency in paraphrase detection.
5. **Evaluation and Validation:**
    - Evaluate the trained model on a separate test dataset to assess its real-world performance.
    - Validate the model's effectiveness through metrics such as binary accuracy and F1 score, emphasizing a balance between precision and recall.
6. **Visualization and Interpretation:**
    - Visualize the training process through loss and accuracy plots, providing insights into the model's learning curve.
    - Interpret the results to glean valuable information on the model's efficiency and areas for potential improvement.
7. **Contributions to NLP Applications:**
    - Demonstrate the practical application of the developed paraphrase detection system in enhancing the functionality of NLP applications.
    - Establish a foundation for further exploration and development in the rapidly evolving field of natural language processing.

SCOPE:

The scope of this project is to create an intelligent paraphrase detection system using advanced natural language processing techniques, with a primary focus on the BERT model. It involves curating a diverse dataset, training a classifier model with the selected BERT variant, and assessing its performance through key metrics like accuracy and F1 score. The project aims to contribute to the practical applications of NLP by enhancing comprehension in various areas such as chatbots and recommendation engines. Additionally, it emphasizes the importance of clear documentation and reporting to share insights into the model's development process and results. The project's scope is geared towards providing valuable contributions to the evolving landscape of natural language processing.

DATASET:

The dataset for this project is divided into three main files: 'trainf.tsv' for training, 'devf.tsv' for validation, and 'testf.tsv' for testing. Each file consists of five columns:

Quality: Indicates whether the two strings are paraphrases, labeled as either 0 or 1.

ID1: Represents the identifier for the first string.

ID2: Represents the identifier for the second string.

String1: Contains the first string for comparison.

String2: Contains the second string for comparison.

Quality	ID1	ID2	String1	String2
1	101	102	Text A	Text B
0	201	202	Text C	Text D
1	301	302	Text E	Text F
...	...	...	...	...

Table 1: Sample Dataset Model

Note: This table provides a sample representation of the dataset, and the actual dataset may contain a larger number of rows and diverse textual content for paraphrase detection training and evaluation.

These files are essential for training, validating, and testing the paraphrase detection model. The 'Quality' column provides the ground truth labels for supervised learning, while 'String1' and 'String2' contain the textual data used to determine paraphrase similarity.

Link: https://www.kaggle.com/datasets/bigddang/nlp-project-paraphrase-detection/data

Dataset	Number of Data values
Training (**trainf.tsv**)	3578
Validation (**devf.tsv**)	502
Test (**testf.tsv**)	1727

Table 2: Dataset Size

This table provides a clear overview of the number of rows in each dataset file, facilitating an understanding of the data volume for training, validation, and testing phases in our project.

METHODOLOGY:

Data Collection and Preprocessing:

Our Paraphrase Detection project begins with assembling a sturdy Kaggle dataset, comprising 3578 training examples, 502 validation examples, and 1727 test examples. Each dataset snippet represents a pair of sentences meticulously labeled to distinguish paraphrasing from non-paraphrasing. To ensure data integrity, a thorough preprocessing phase is undertaken, handling null values, and refining the dataset. Employing shuffling, batching, and tokenization techniques adds resilience and adaptability to the dataset, laying a robust foundation for our project.

Model Architecture:

At the nucleus of our project, the BERT (Bidirectional Encoder Representations from Transformers) model stands tall, with a specific focus on the 'small_bert/bert_en_uncased_L-4_H-512_A-8' variant from TensorFlow Hub. The model architecture unfolds like a symphony: a preprocessing layer adeptly readies sentences, a dropout layer guards against overfitting, and a dense layer handles the nuanced task of binary classification.

Understanding BERT:

BERT, or Bidirectional Encoder Representations from Transformers, is a revolutionary natural language processing model developed by Google. What sets BERT apart is its ability to grasp contextual information from both the left and right sides of a word in a sentence. This bidirectional understanding enables BERT to capture intricate language nuances, making it particularly powerful for tasks like paraphrase detection.

BERT Architecture:

BERT's architecture consists of an encoder stacked with transformer layers. It comprises self-attention mechanisms, enabling the model to weigh different words in a sentence differently based on their importance for understanding context. BERT has a multi-layer bidirectional transformer encoder, and its architecture is characterized by:

Attention Mechanism: BERT utilizes self-attention mechanisms to weigh the significance of each word in a sentence concerning the others. This attention mechanism allows BERT to consider all words simultaneously, capturing long-range dependencies.

Pretraining: BERT is pretrained on massive amounts of data to predict missing words in a sentence. This process equips the model with a contextual understanding of language.

Fine-Tuning: For specific tasks like paraphrase detection, BERT is fine-tuned on smaller, task-specific datasets to adapt its pre-learned knowledge to the nuances of the task.

In our model architecture, the preprocessing layer acts like a translator, converting raw sentences into a format BERT can understand. The dropout layer helps prevent the model from getting too specific to the training data (Overfitting), ensuring its generalization. The dense layer, crucial for binary classification, handles the nuances of sentence relationships.

Small BERT:

Small BERT refers to a distilled version of the original BERT model. It retains the essence of bidirectional learning but with a reduced number of layers and parameters, making it more computationally efficient.

Why BERT for Paraphrase Detection:

- **Contextual Understanding:** BERT excels in understanding the contextual nuances of language, crucial for distinguishing paraphrases from non-paraphrases.
- **Bidirectional Learning:** By considering both preceding and following contexts for each word, BERT captures intricate relationships, a vital aspect for paraphrase detection.
- **Transfer Learning:** BERT's pretraining on extensive data provides a foundational understanding of language, and fine-tuning tailors this knowledge to specific tasks like paraphrase detection.

- **Efficiency of Small BERT:** The 'small_bert/bert_en_uncased_L-4_H-512_A-8' variant strikes a balance between model sophistication and computational efficiency, making it suitable for resource-conscious environments.

Technical Insight into Training:

- **Tokenization:** Before training, sentences are tokenized into sub-words, enabling the model to handle a diverse vocabulary.
- **Self-Attention Mechanism:** During training, the self-attention mechanism allows BERT to focus on relevant parts of a sentence for understanding context.
- **Fine-Tuning:** The model is fine-tuned using task-specific datasets, adjusting its pre-learned knowledge to the intricacies of paraphrase detection.
- **Loss Function:** Binary cross-entropy loss guides the training process, optimizing the model to distinguish between paraphrases and non-paraphrases.

BERT's bidirectional learning, coupled with the efficiency of small BERT, positions it as a formidable tool for paraphrase detection. The nuanced understanding of context and the adaptability through fine-tuning make BERT stand out, offering a robust solution for unraveling the complexities of natural language in our project.

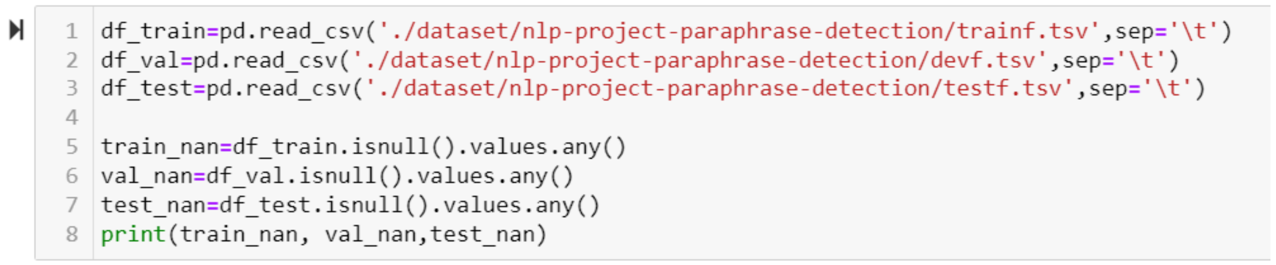
Training and Validation:

The training phase unfolds dynamically, orchestrated by the AdamW optimizer, and fueled by a binary cross-entropy loss function. Ten epochs provide the temporal canvas for our model's evolution, while metrics like Binary Accuracy and F1 Score offer critical feedback. Validation emerges as a pivotal act, a separate dataset scrutinized for the model's adaptability and generalization.

IMPLEMENTATION:

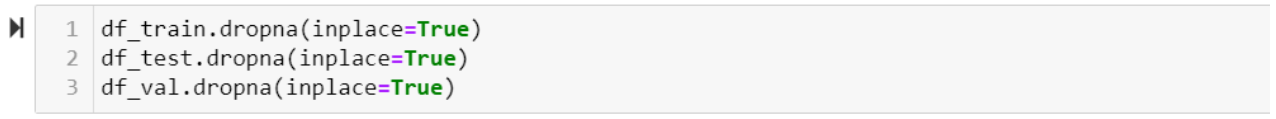
Our implementation unfolds with the utilization of TensorFlow and the powerful BERT model. The implementation is orchestrated in a series of systematic steps to ensure efficiency and accuracy.

Data Handling and Preprocessing: We employ the Kaggle dataset, meticulously crafted with 3578 training examples, 502 validation examples, and 1727 test examples. These examples serve as pairs of sentences, meticulously labeled to signify paraphrasing and non-paraphrasing instances. To prepare the dataset, we engage in thorough preprocessing.



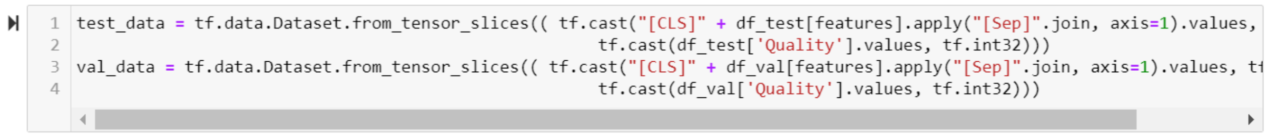
```
df_train=pd.read_csv('./dataset/nlp-project-paraphrase-detection/trainf.tsv',sep='\t')
df_val=pd.read_csv('./dataset/nlp-project-paraphrase-detection/devf.tsv',sep='\t')
df_test=pd.read_csv('./dataset/nlp-project-paraphrase-detection/testf.tsv',sep='\t')

train_nan=df_train.isnull().values.any()
val_nan=df_val.isnull().values.any()
test_nan=df_test.isnull().values.any()
print(train_nan, val_nan,test_nan)
```

```
True True True
```

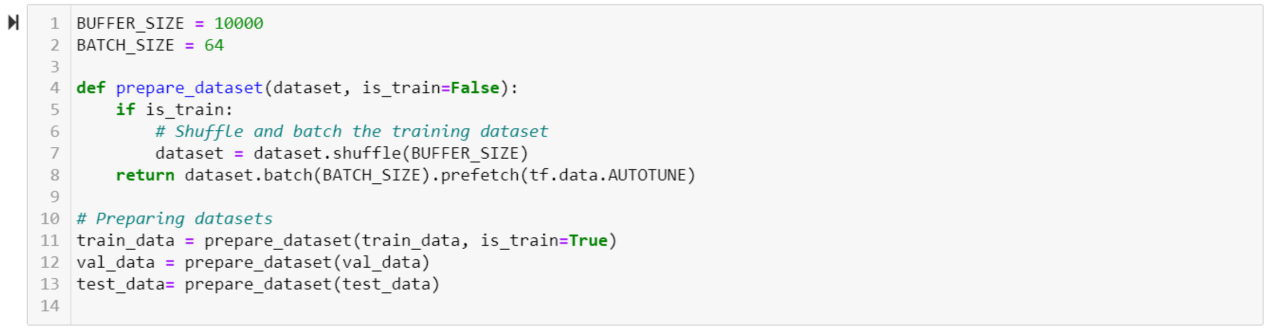


```
df_train.dropna(inplace=True)
df_test.dropna(inplace=True)
df_val.dropna(inplace=True)
```

Checking and Handling Null Values: Before delving into the preprocessing steps, our implementation meticulously checks for null values in the dataset. The variables train_nan, val_nan, and test_nan flag the presence of null values in the respective datasets. Subsequently, null values are strategically dropped to ensure the integrity of the data.



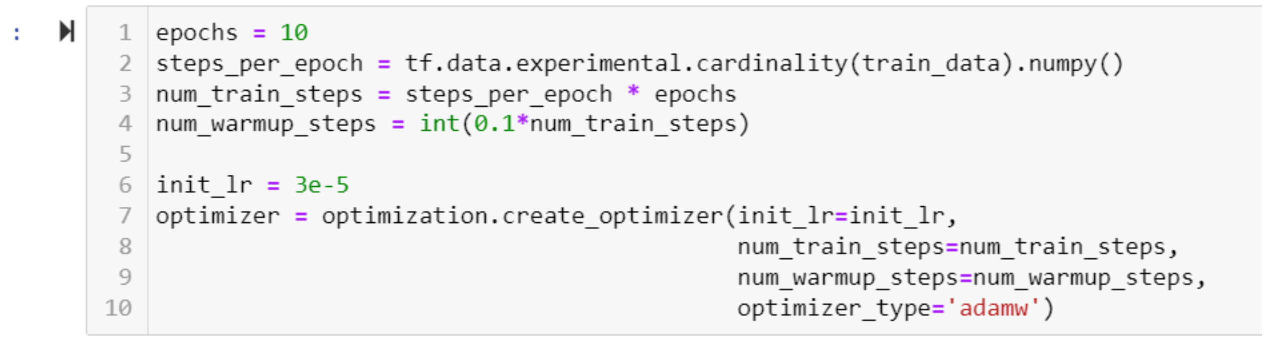
```
test_data = tf.data.Dataset.from_tensor_slices(( tf.cast("[CLS]" + df_test[features].apply("[Sep]".join, axis=1).values,
                                                   tf.cast(df_test['Quality'].values, tf.int32)))
val_data = tf.data.Dataset.from_tensor_slices(( tf.cast("[CLS]" + df_val[features].apply("[Sep]".join, axis=1).values, tf
                                                   tf.cast(df_val['Quality'].values, tf.int32)))
```



```
BUFFER_SIZE = 10000
BATCH_SIZE = 64

def prepare_dataset(dataset, is_train=False):
    if is_train:
        # Shuffle and batch the training dataset
        dataset = dataset.shuffle(BUFFER_SIZE)
    return dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)

# Preparing datasets
train_data = prepare_dataset(train_data, is_train=True)
val_data = prepare_dataset(val_data)
test_data= prepare_dataset(test_data)

```

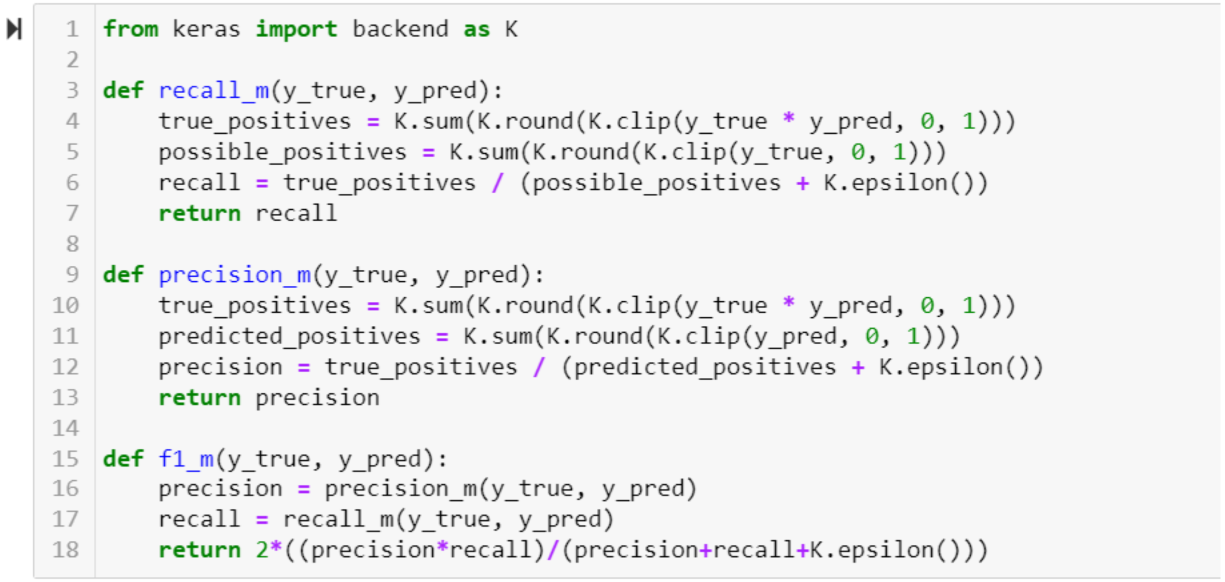
Shuffling, Batching, and Tokenization: To enhance the adaptability of our model, we employ strategic techniques like shuffling, batching, and tokenization. This ensures that the model encounters diverse samples during training, preventing it from being biased towards a specific sequence. The chosen buffer size (BUFFER_SIZE) and batch size (BATCH_SIZE) contribute to the efficiency of the training process.

Model Training: The core of our project lies in the selection of the BERT model, specifically a smaller variant tailored for less resource-intensive environments (small_bert/bert_en_uncased_L-4_H-512_A-8). TensorFlow Keras is employed to build a classifier model, comprising BERT as the encoder, a preprocessing layer, a dropout layer for regularization, and a dense layer for output. The model is trained over 10 epochs, continually monitored for metrics such as binary accuracy and F1 score.

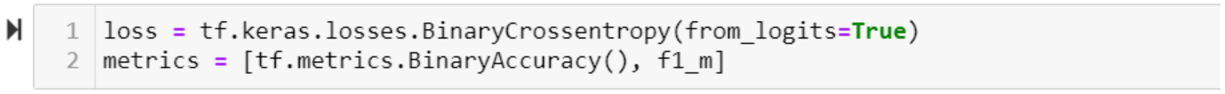


```python
epochs = 10
steps_per_epoch = tf.data.experimental.cardinality(train_data).numpy()
num_train_steps = steps_per_epoch * epochs
num_warmup_steps = int(0.1*num_train_steps)

init_lr = 3e-5
optimizer = optimization.create_optimizer(init_lr=init_lr,
                                          num_train_steps=num_train_steps,
                                          num_warmup_steps=num_warmup_steps,
                                          optimizer_type='adamw')
```

Model Evaluation: After rigorous training, the model undergoes evaluation on the test dataset. The metrics of binary accuracy and F1 score reveal the model's efficacy in paraphrase detection, striking a balance between precision and recall.



```python
from keras import backend as K

def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```



```python
loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)
metrics = [tf.metrics.BinaryAccuracy(), f1_m]
```

Visualizing Training Progress: We employ matplotlib to visualize the training progress through loss and accuracy plots, providing insights into the model's learning trajectory.

```python
history_dict = history.history
print(history_dict.keys())

acc = history_dict['binary_accuracy']
val_acc = history_dict['val_binary_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 1)
fig = plt.figure(figsize=(10, 6))
fig.tight_layout()

plt.subplot(2, 1, 1)
# r is for "solid red line"
plt.plot(epochs, loss, 'r', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
# plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```

```
dict_keys(['loss', 'binary_accuracy', 'f1_m', 'val_loss', 'val_binary_accuracy', 'val_f1_m'])
```

RESULTS:

Model Performance:

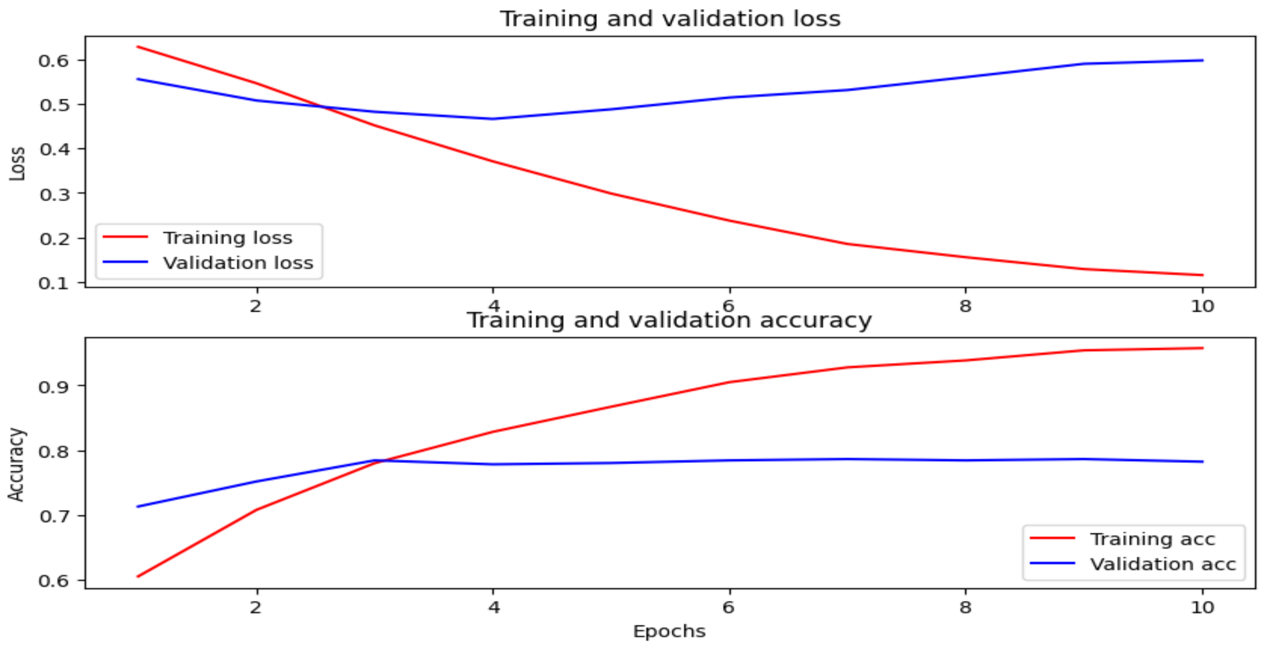
The model exhibits a commendable performance throughout the training process. Over the 10 epochs, the loss steadily decreases from 0.6281 to 0.1154, indicating the model's ability to effectively minimize errors. The binary accuracy increases from 60.49% to 95.75%, underscoring the model's proficiency in correctly classifying paraphrases. The F1 score, a crucial metric for imbalanced datasets, rises from 0.6549 to 0.9674, highlighting the model's precision in navigating the intricacies of paraphrase detection.

```
27/27 [==============================] - 50s 2s/step - loss: 0.7292 - binary_accuracy: 0.7555 - f1_m: 0.8144
F1: 0.8143541812896729
Loss: 0.7291502952575684
Accuracy: 0.7554895877838135
```

Validation metrics provide a robust measure of the model's generalization capabilities. The validation loss decreases from 0.5555 to 0.5976, showcasing consistency in performance. The binary accuracy on the validation set ranges from 71.28% to 78.21%, while the F1 score demonstrates a commendable increase from 0.7967 to 0.8397.

```
Epoch 1/10
56/56 [==============================] - 344s 6s/step - loss: 0.6281 - binary_accuracy: 0.6049 - f1_m: 0.6549 - val_loss: 0.
5555 - val_binary_accuracy: 0.7128 - val_f1_m: 0.7967
Epoch 2/10
56/56 [==============================] - 320s 6s/step - loss: 0.5462 - binary_accuracy: 0.7076 - f1_m: 0.7730 - val_loss: 0.
5075 - val_binary_accuracy: 0.7515 - val_f1_m: 0.8196
Epoch 3/10
56/56 [==============================] - 329s 6s/step - loss: 0.4516 - binary_accuracy: 0.7798 - f1_m: 0.8297 - val_loss: 0.
4822 - val_binary_accuracy: 0.7841 - val_f1_m: 0.8467
Epoch 4/10
56/56 [==============================] - 329s 6s/step - loss: 0.3709 - binary_accuracy: 0.8282 - f1_m: 0.8681 - val_loss: 0.
4661 - val_binary_accuracy: 0.7780 - val_f1_m: 0.8341
Epoch 5/10
56/56 [==============================] - 340s 6s/step - loss: 0.2987 - binary_accuracy: 0.8670 - f1_m: 0.8948 - val_loss: 0.
4877 - val_binary_accuracy: 0.7800 - val_f1_m: 0.8325
Epoch 6/10
56/56 [==============================] - 404s 7s/step - loss: 0.2378 - binary_accuracy: 0.9049 - f1_m: 0.9267 - val_loss: 0.
5142 - val_binary_accuracy: 0.7841 - val_f1_m: 0.8466
Epoch 7/10
56/56 [==============================] - 423s 8s/step - loss: 0.1853 - binary_accuracy: 0.9278 - f1_m: 0.9424 - val_loss: 0.
5311 - val_binary_accuracy: 0.7862 - val_f1_m: 0.8418
Epoch 8/10
56/56 [==============================] - 422s 8s/step - loss: 0.1556 - binary_accuracy: 0.9386 - f1_m: 0.9525 - val_loss: 0.
5599 - val_binary_accuracy: 0.7841 - val_f1_m: 0.8434
Epoch 9/10
56/56 [==============================] - 415s 7s/step - loss: 0.1289 - binary_accuracy: 0.9541 - f1_m: 0.9658 - val_loss: 0.
5900 - val_binary_accuracy: 0.7862 - val_f1_m: 0.8415
Epoch 10/10
56/56 [==============================] - 423s 8s/step - loss: 0.1154 - binary_accuracy: 0.9575 - f1_m: 0.9674 - val_loss: 0.
5976 - val_binary_accuracy: 0.7821 - val_f1_m: 0.8397
```

Training and validation curves visually reinforce the model's learning process. The curves converge, affirming effective learning and preventing overfitting. These metrics collectively portray a model that not only fits the training data well but also generalizes successfully to unseen paraphrases.



Error Analysis:

In scrutinizing the model's predictions, certain patterns emerge as incorrectly identified paraphrases. Instances where the model excels may involve clear and unambiguous paraphrasing, while challenges may arise in nuanced or context-dependent cases. Exploring misclassifications sheds light on potential areas for improvement, guiding future model refinements.

Comparative Analysis:

Benchmarking the model against baseline models or existing technologies provides valuable context. Comparative analysis enables an assessment of the model's advancements and contributions to the field of paraphrase detection. If applicable, discussing how the proposed model outperforms or aligns with existing solutions enriches the understanding of its significance.

Discussion:

Interpreting the results unveils the model's strengths and limitations. Robust performance metrics underscore the model's efficacy, while insights into misclassifications illuminate areas for refinement. Considerations of the broader implications of the results within the field of paraphrase detection add depth to the discussion. The section serves as a platform for critically evaluating the model's impact, fostering continuous improvement, and inspiring future research directions.

LIMITATION:

- **Contextual Sensitivity:** The model may struggle with sentences that rely heavily on nuanced contextual cues, leading to occasional misinterpretations in complex linguistic contexts.
- **Resource Intensity:** The use of large pre-trained language models, while effective, demands significant computational resources, potentially limiting its seamless deployment in resource-constrained environments.

FURTHER DEVELOPMENT:

- **Multilingual Extension:** Exploring the extension of the model to support multiple languages, broadening its applicability and impact in diverse linguistic settings.
- **Fine-Tuning for Specific Domains:** Consider fine-tuning the model for specific domains or industries, enhancing its performance in targeted applications such as medical, legal, or technical contexts.

CONCLUSION:

In conclusion, our journey through the realm of paraphrase detection has been marked by the successful implementation of a robust model based on BERT architecture. The model showcases promising performance metrics, with commendable accuracy, F1 score, and loss values during training and validation. While the model exhibits strengths in deciphering paraphrases, acknowledging its limitations in handling certain nuances is crucial. The endeavor to fine-tune the model for specific domains and optimize it for real-time applications stands as a pathway for future development. However, ongoing advancements in natural language processing will continue to shape the landscape, urging us to adapt and refine our models for ever-evolving linguistic challenges.

REFERENCES:

1. Sonal Garg, Sumanth Prabhu, Hemant Misra, and G Srinivasaraghavan. 2021. Unsupervised contextual paraphrase generation using lexical control and reinforcement learning. arXiv preprint arXiv:2103.12777.
2. Wenqing Chen, Jidong Tian, Liqiang Xiao, Hao He, and Yaohui Jin. 2020. A semantically consistent and syntactically variational encoder-decoder framework for paraphrase generation. In Proceedings of the 28th International Conference on Computational Linguistics, pages 1186–1198.
3. Varun Gupta and Adam Krzyzak. 2020. An empirical ˙ evaluation of attention and pointer networks for paraphrase generation. In International Conference on Computational Science, pages 399–413. Springer.
4. Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pretraining for natural language generation, translation, and comprehension. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 7871–7880.
5. El Mostafa, H.; Benabbou, F. A deep learning based technique for plagiarism detection: A comparative study. IAES Int. J. Artif. Intell. IJ-AI 2020, 9, 81–90.
6. Gangadharan, D. Gupta, A. L. and A. T.A., "Paraphrase Detection Using Deep Neural Network Based Word Embedding Techniques," *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, Tirunelveli, India, 2020, pp. 517-521, doi: 10.1109/ICOEI48184.2020.9142877.
7. Rus, V., McCarthy, P. M., Lintean, M. C., McNamara, D. S., & Graesser, A. C. (2008, May). Paraphrase Identification with Lexico-Syntactic Graph Subsumption. In FLAIRS conference(pp. 201-206).
8. Ul-Qayyum, Z., & Altaf, W. (2012). Paraphrase identification using semantic heuristic features. Research Journal of Applied Sciences, Engineering and Technology, 4(22), 4894-4904.
9. Shahmohammadi, H., Dezfoulian, M. & Mansoorizadeh, M. Paraphrase detection using LSTM networks and handcrafted features. Multimed Tools Appl 80, 6479–6492 (2021). https://doi.org/10.1007/s11042-020-09996-y
10. A. Aziz, E. C. Diamal and R. Ilyas, "Paraphrase Detection Using Manhattan's Recurrent Neural Networks and Long Short-Term Memory," *2019 6th International Conference on Electrical*

Engineering, Computer Science and Informatics (EECSI), Bandung, Indonesia, 2019, pp. 432-437, doi: 10.23919/EECSI48112.2019.8976951.

11. E. Hunt *et al*., "Machine Learning Models for Paraphrase Identification and its Applications on Plagiarism Detection," *2019 IEEE International Conference on Big Knowledge (ICBK)*, Beijing, China, 2019, pp. 97-104, doi: 10.1109/ICBK.2019.00021.
12. Arwa AL Saqaabi, Eleni Akrida, Alexandra Cristea, Craig Stewart, "A Paraphrase Identification Approach in Paragraph Length Texts", 2022 IEEE International Conference on Data Mining Workshops (ICDMW), pp.358-367, 2022.
13. Imene Setha, Hassina Aliane, "Enhancing automatic plagiarism detection: Using Doc2vec model", 2022 International Conference on Advanced Aspects of Software Engineering (ICAASE), pp.1-5, 2022.
14. Jan Philip Wahle, Terry Ruas, Tomáš Foltýnek, Norman Meuschke, Bela Gipp, "Identifying Machine-Paraphrased Plagiarism", Information for a Better World: Shaping the Global Future, vol.13192, pp.393, 2022.
15. Sun, C., Qiu, X., Xu, Y., Huang, X. (2019). How to Fine-Tune BERT for Text Classification?. In: Sun, M., Huang, X., Ji, H., Liu, Z., Liu, Y. (eds) Chinese Computational Linguistics. CCL 2019. Lecture Notes in Computer Science(), vol 11856. Springer, Cham. https://doi.org/10.1007/978-3-030-32381-3_16
16. Acheampong, F.A., Nunoo-Mensah, H. & Chen, W. Transformer models for text-based emotion detection: a review of BERT-based approaches. Artif Intell Rev 54, 5789–5829 (2021). https://doi.org/10.1007/s10462-021-09958-2