CS 579 Project Final Report
Fake News Classification

Neil Dhote
Victor Samsonov

1 Problem Description

We are working on the Fake News Classification Task, and the task is defined as follows: Given the title of a fake news article A and the title of a coming news article B, we are asked to classify B into one of the three categories:

- Agreed: B talks about the same fake news as A.
- Disagreed: refutes the fake news in A.
- Unrelated: B is unrelated to A.

To tackle this problem, our team will undertake a comprehensive approach that encompasses various stages of the machine learning process. Initially, we will conduct Exploratory Data Analysis (EDA) to gain a deeper understanding of the data and identify potential trends and patterns. This will be followed by data preprocessing to ensure our data is in the right format for our model to ingest and potentially improve the quality of our data by removing stop words, using tokenization, lemmatization, and assigning a default padding. Since this is a hands-on course, we decided to make the most out of this opportunity and gain experience with Deep Learning architectures, the most relevant one being BERT.

2 EDA and Data Augmentation

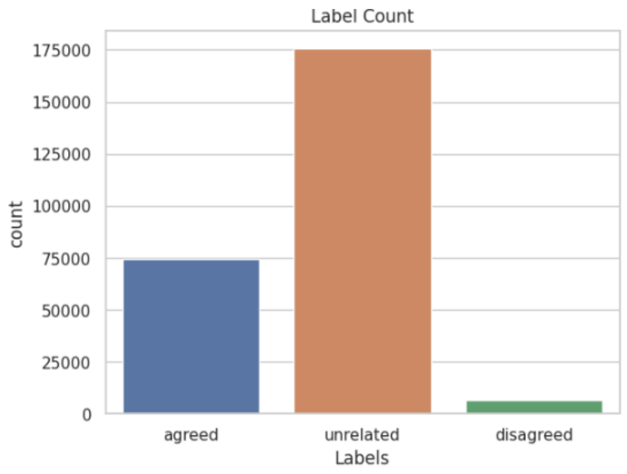


Figure 1: Unbalanced Dataset before Data Augmentation

Upon performing EDA, it was fairly obvious that the dataset was heavily unbalanced. The unrelated labels were the most common, while disagreed barely had any instances within our dataset (agreed had approximately 74,000 samples making it the second most likely label). In our analysis, we determined that it would be very important to predict both disagreed and agreed labels with higher precision and recall since these values are the ones that actually provide useful information to the problem. Unrelated is considered mostly irrelevant for us, and we concluded that predicting an unrelated instance as agreed or disagreed would be an acceptable mistake.

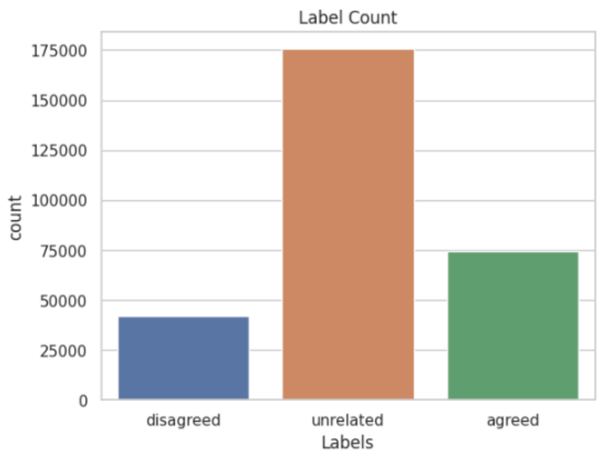
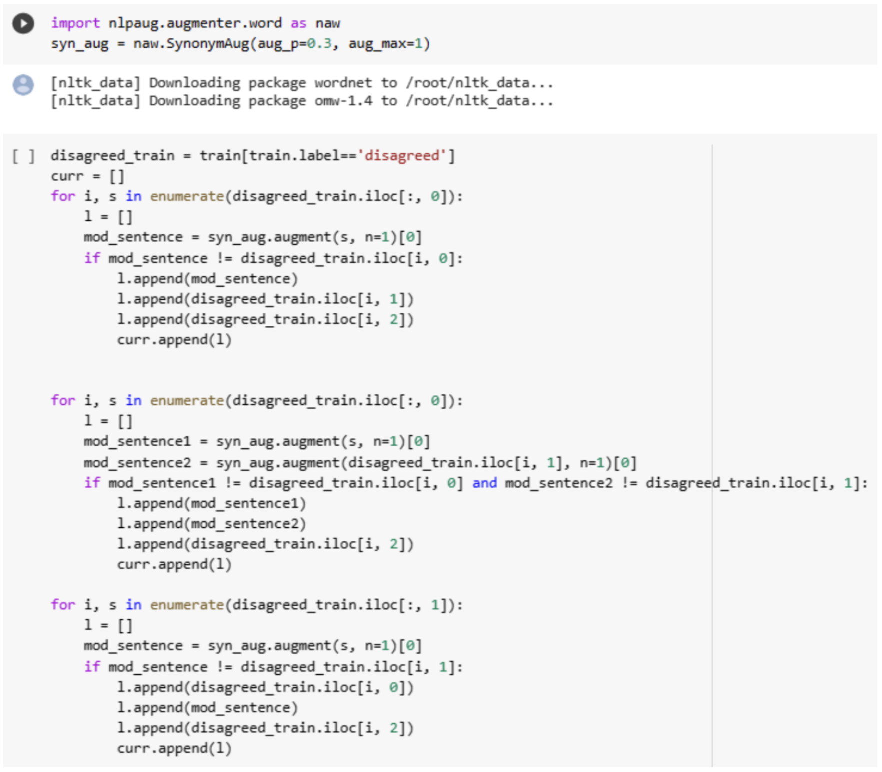


Figure 2: Balanced Dataset after Data Augmentation

We tried multiple approaches when it came to handling our label imbalance. We initially decided to downsample unrelated labels and generate synthetic data for the disagreed category by replacing words at random with synonyms, resulting in approximately 45,000 total disagreed labels. Later on, we discovered that not modifying the number of unrelated labels was ideal since it resulted in improved performance.

We generated the synthetic data with a library called nlpaug, by replacing words with synonyms. We had 2 synonym augmentors, one that would simply replace one word with a synonym, and the other would replace 3 words. This was done for title1_en and title2_en individually and then title1_en and title2_en at the same time.



```
import nlpaug.augmenter.word as naw
syn_aug = naw.SynonymAug(aug_p=0.3, aug_max=1)
```

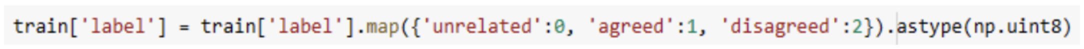
```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
```

```
disagreed_train = train[train.label=='disagreed']
curr = []
for i, s in enumerate(disagreed_train.iloc[:, 0]):
    l = []
    mod_sentence = syn_aug.augment(s, n=1)[0]
    if mod_sentence != disagreed_train.iloc[i, 0]:
        l.append(mod_sentence)
        l.append(disagreed_train.iloc[i, 1])
        l.append(disagreed_train.iloc[i, 2])
        curr.append(l)


for i, s in enumerate(disagreed_train.iloc[:, 0]):
    l = []
    mod_sentence1 = syn_aug.augment(s, n=1)[0]
    mod_sentence2 = syn_aug.augment(disagreed_train.iloc[i, 1], n=1)[0]
    if mod_sentence1 != disagreed_train.iloc[i, 0] and mod_sentence2 != disagreed_train.iloc[i, 1]:
        l.append(mod_sentence1)
        l.append(mod_sentence2)
        l.append(disagreed_train.iloc[i, 2])
        curr.append(l)

for i, s in enumerate(disagreed_train.iloc[:, 1]):
    l = []
    mod_sentence = syn_aug.augment(s, n=1)[0]
    if mod_sentence != disagreed_train.iloc[i, 1]:
        l.append(disagreed_train.iloc[i, 0])
        l.append(mod_sentence)
        l.append(disagreed_train.iloc[i, 2])
        curr.append(l)
```

Figure 3: Code generating augmented_train dataset in Data_Augmentation.ipynb

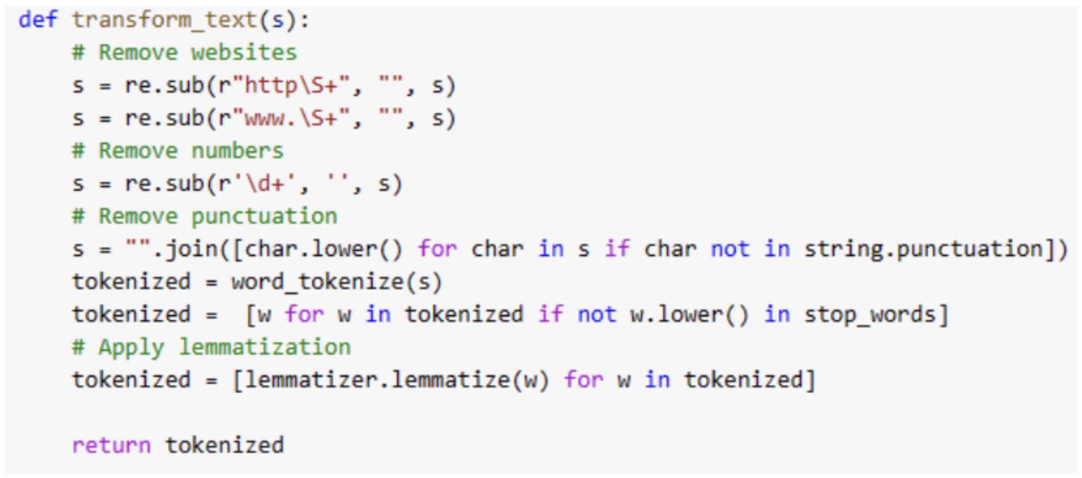
3 Data Preprocessing

We preprocessed the collected data by removing irrelevant information such as links, hashtags, and emojis to improve the quality of the data for further analysis. We also applied tokenization, lemmatization, removed stop words, assigned each word an index from 1 to N, and lastly padded our sequences to a default length. Below, we include screenshots for relevant code snippets in our data preprocessing.



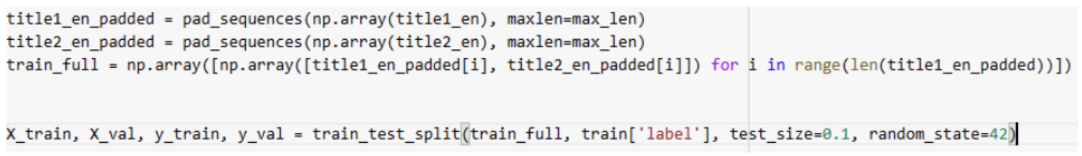
```python
train['label'] = train['label'].map({'unrelated':0, 'agreed':1, 'disagreed':2}).astype(np.uint8)
```

Figure 4: Converting the label from Categorical to Int



```python
def transform_text(s):
    # Remove websites
    s = re.sub(r"http\S+", "", s)
    s = re.sub(r"www.\S+", "", s)
    # Remove numbers
    s = re.sub(r'\d+', '', s)
    # Remove punctuation
    s = "".join([char.lower() for char in s if char not in string.punctuation])
    tokenized = word_tokenize(s)
    tokenized =  [w for w in tokenized if not w.lower() in stop_words]
    # Apply lemmatization
    tokenized = [lemmatizer.lemmatize(w) for w in tokenized]

    return tokenized
```

Figure 5: Transformation function used for title1_en and title2_en



```python
title1_en_padded = pad_sequences(np.array(title1_en), maxlen=max_len)
title2_en_padded = pad_sequences(np.array(title2_en), maxlen=max_len)
train_full = np.array([np.array([title1_en_padded[i], title2_en_padded[i]]) for i in range(len(title1_en_padded))])


X_train, X_val, y_train, y_val = train_test_split(train_full, train['label'], test_size=0.1, random_state=42)
```

Figure 6: Padding the sequences and splitting into training and validation (max_len=50)

4 Models

We implemented two distinct models:

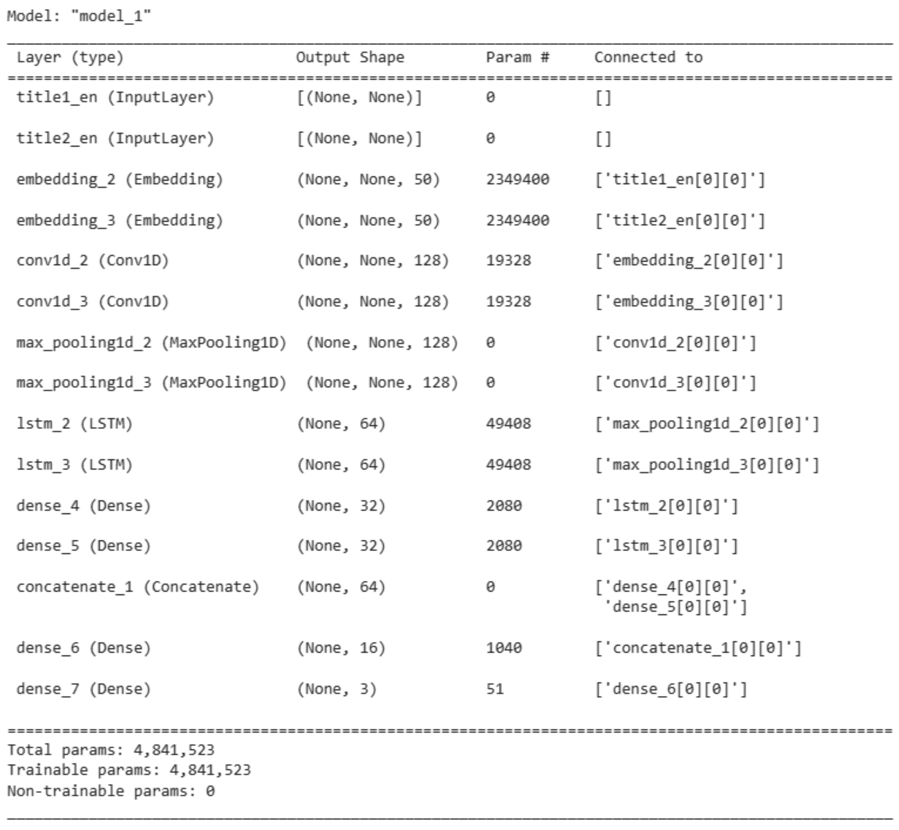
- Custom Embedding LSTM
- BERT LSTM

4.1 Custom Embedding LSTM

For this model, we decided to devise an architecture that takes two inputs, which are fed to distinct Embedding layers. The outputs of the embedding layers are forwarded to Conv1D layers and Maxpooling1D layers, which extract relevant information. Next, an LSTM layer processes the remaining sequences, which is ideal in an NLP problem. This output is subsequently given to a dense layer, and the result is concatenated to additional fully connected layers.

Below, we include relevant characteristics of our model:

- Layers Used:
  - Embedding
  - Conv1D
  - Maxpooling1D
  - Dense
- Optimizer:
  - Adam
- Loss Function:
  - Sparse Categorical Cross-Entropy
- Activation Functions Used:
  - Relu
  - Softmax



```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 title1_en (InputLayer)         [(None, None)]       0           []

 title2_en (InputLayer)         [(None, None)]       0           []

 embedding_2 (Embedding)        (None, None, 50)     2349400     ['title1_en[0][0]']

 embedding_3 (Embedding)        (None, None, 50)     2349400     ['title2_en[0][0]']

 conv1d_2 (Conv1D)              (None, None, 128)    19328       ['embedding_2[0][0]']

 conv1d_3 (Conv1D)              (None, None, 128)    19328       ['embedding_3[0][0]']

 max_pooling1d_2 (MaxPooling1D)  (None, None, 128)   0           ['conv1d_2[0][0]']

 max_pooling1d_3 (MaxPooling1D)  (None, None, 128)   0           ['conv1d_3[0][0]']

 lstm_2 (LSTM)                  (None, 64)           49408       ['max_pooling1d_2[0][0]']

 lstm_3 (LSTM)                  (None, 64)           49408       ['max_pooling1d_3[0][0]']

 dense_4 (Dense)                (None, 32)           2080        ['lstm_2[0][0]']

 dense_5 (Dense)                (None, 32)           2080        ['lstm_3[0][0]']

 concatenate_1 (Concatenate)    (None, 64)           0           ['dense_4[0][0]',
                                                                  'dense_5[0][0]']

 dense_6 (Dense)                (None, 16)           1040        ['concatenate_1[0][0]']

 dense_7 (Dense)                (None, 3)            51          ['dense_6[0][0]']

==================================================================================================
Total params: 4,841,523
Trainable params: 4,841,523
Non-trainable params: 0
__________________________________________________________________________________________________
```

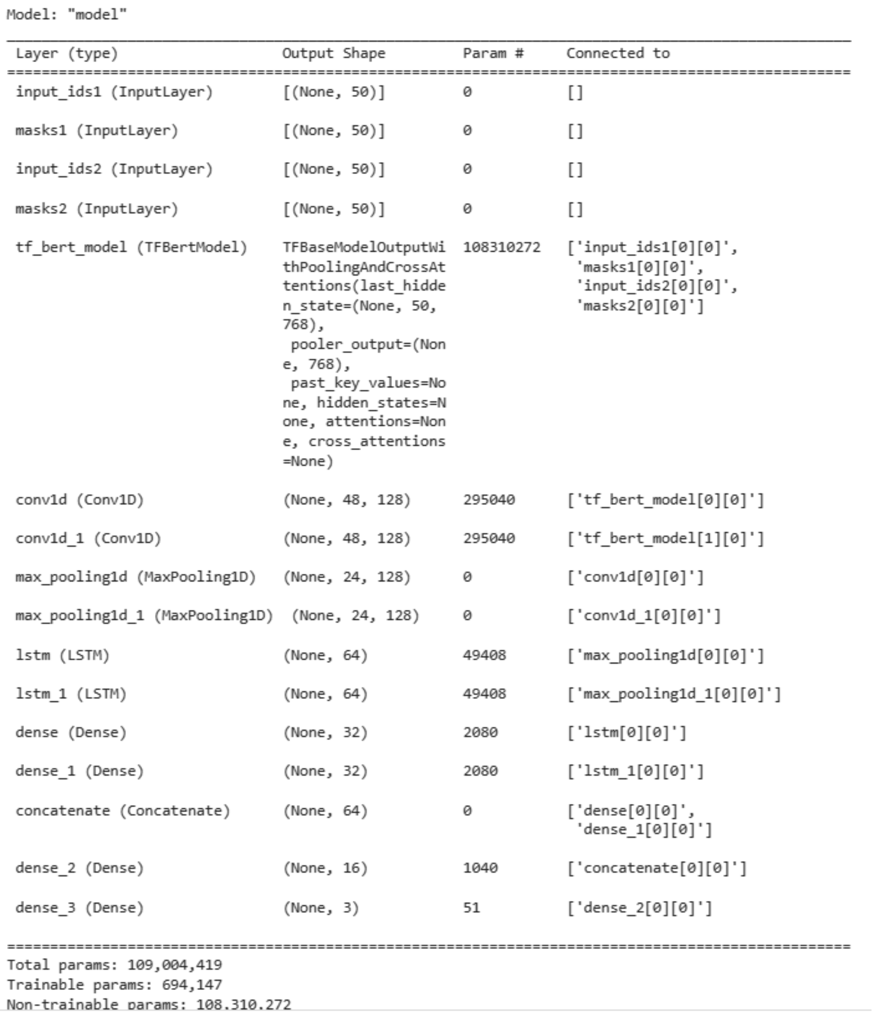
Figure 7: Custom Embedding LSTM Model Architecture

4.2 BERT LSTM

Upon performing further research into transformers, we realized that we can use pretrained models to extract the underlying representations and contexts of words within our texts. Therefore, we decided to use the transformers library to load a pretrained instance of BERT, which acts as an "embedding" layer in this case. It is fair to note that we didn't fine-tune BERT since it has many parameters; therefore, we decided to freeze it during training.

Below, we include relevant characteristics of our model, which is pretty much the same as the previous architecture:

- Layers/Models Used:
  - BERT
  - Conv1D
  - Maxpooling1D
  - Dense
- Optimizer:
  - NAdam
- Loss Function:
  - Sparse Categorical Cross-Entropy
- Activation Functions Used:
  - Relu
  - Softmax

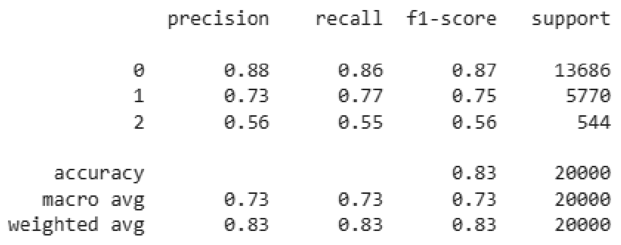


```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_ids1 (InputLayer)        [(None, 50)]         0           []

 masks1 (InputLayer)            [(None, 50)]         0           []

 input_ids2 (InputLayer)        [(None, 50)]         0           []

 masks2 (InputLayer)            [(None, 50)]         0           []

 tf_bert_model (TFBertModel)    TFBaseModelOutputWi  108310272   ['input_ids1[0][0]',
                                thPoolingAndCrossAt               'masks1[0][0]',
                                tentions(last_hidde               'input_ids2[0][0]',
                                n_state=(None, 50,                'masks2[0][0]']
                                768),
                                 pooler_output=(Non
                                e, 768),
                                 past_key_values=No
                                ne, hidden_states=N
                                one, attentions=Non
                                e, cross_attentions
                                =None)

 conv1d (Conv1D)                (None, 48, 128)      295040      ['tf_bert_model[0][0]']

 conv1d_1 (Conv1D)              (None, 48, 128)      295040      ['tf_bert_model[1][0]']

 max_pooling1d (MaxPooling1D)   (None, 24, 128)      0           ['conv1d[0][0]']

 max_pooling1d_1 (MaxPooling1D)  (None, 24, 128)     0           ['conv1d_1[0][0]']

 lstm (LSTM)                    (None, 64)           49408       ['max_pooling1d[0][0]']

 lstm_1 (LSTM)                  (None, 64)           49408       ['max_pooling1d_1[0][0]']

 dense (Dense)                  (None, 32)           2080        ['lstm[0][0]']

 dense_1 (Dense)                (None, 32)           2080        ['lstm_1[0][0]']

 concatenate (Concatenate)      (None, 64)           0           ['dense[0][0]',
                                                                  'dense_1[0][0]']

 dense_2 (Dense)                (None, 16)           1040        ['concatenate[0][0]']

 dense_3 (Dense)                (None, 3)            51          ['dense_2[0][0]']

==================================================================================================
Total params: 109,004,419
Trainable params: 694,147
Non-trainable params: 108,310,272
```

Figure 8: BERT LSTM Model Architecture

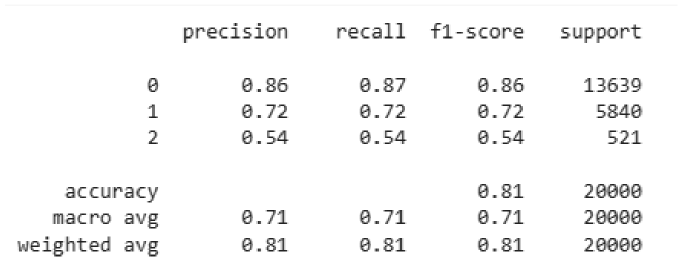
5 Results

We used a **holdout validation set of 20,000 samples** to evaluate the performance of our model. As seen in Figure 9 and Figure 10, the Custom Embedding LSTM model performed better in both overall accuracy and all F1 scores. The Custom Embedding LSTM model also takes significantly less time to make predictions. BERT had 100 million parameters, which is a lot to tune, but if we were to fine-tune BERT, then it should result in much more improved performance.



```
              precision    recall  f1-score   support

           0       0.88      0.86      0.87     13686
           1       0.73      0.77      0.75      5770
           2       0.56      0.55      0.56       544

    accuracy                           0.83     20000
   macro avg       0.73      0.73      0.73     20000
weighted avg       0.83      0.83      0.83     20000
```

Figure 9: Custom Embedding LSTM Model Results



```
              precision    recall  f1-score   support

           0       0.86      0.87      0.86     13639
           1       0.72      0.72      0.72      5840
           2       0.54      0.54      0.54       521

    accuracy                           0.81     20000
   macro avg       0.71      0.71      0.71     20000
weighted avg       0.81      0.81      0.81     20000
```

Figure 10: BERT LSTM Model Results

Overall, we conclude that the steps followed throughout this project are a valid approach, and we were able to improve our F1-score with respect to the disagreed label, which we concluded to be the most important aspect to focus on for our model.

6 References

1. How-to Build a Transformer for Language Classification in TensorFlow - YouTube
   YouTube Video: `https://www.youtube.com/watch?v=GYDFBfx8Ts8&t=1780s`

2. python - How to setup 1D-Convolution and LSTM in Keras - Stack Overflow
   Stack Overflow Thread: `https://stackoverflow.com/questions/51344610/how-to-setup-1d-convolution-and-lstm-`

3. BERT Explained: What it is and how does it work? — Towards Data Science
   Article: `https://towardsdatascience.com/keeping-up-with-the-berts-5b7beb92766`

Tools/Libraries used:

- TensorFlow and Keras
- Pandas
- Numpy
- Transformers
- Nlpaug
- NLTK
- Keras preprocessing