Natural Language Processing

TP3 REPORT

ROUBACHE Islam Group 03

2024/2025

PART 01

BERT

Utilizing **BERT's uncased pre-trained model,** we've crafted a text classification algorithm. I'll present the findings and contrast them with the outcomes from my prior model, which employed **Skip-Gram alongside MLP**.

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art natural language processing model developed by **Google.** It excels in understanding the context and nuances of language by training on vast amounts of text data. BERT's architecture allows it to capture bidirectional relationships between words, leading to significant improvements in various NLP tasks such as text classification, question answering, and language understanding.

Data Loading & Preprocessing

westart bydataloading&Cleaning,wetakeonly above 10% from the dataset, 1000 for each Author

```python
df = pd.read_csv("preprocessed_data.csv")

#clean dataset from ununsed features
df.drop(['id'],axis=1,inplace=True)

# Group the DataFrame by author and select the first 1000 rows for each author
df_sampled = df.groupby('author').head(1000)

# Concatenate the sampled DataFrames into a new DataFrame
df_new = pd.concat([df_sampled], ignore_index=True)

df_new.author.value_counts()
```

Python

Encode The label Values

	text	author	author_encoded
0	proces however afforded means ascertaining dim...	EAP	0
1	never occurred fumbling might mere mistake	HPL	1
2	left hand gold snuff box capered hil cutting m...	EAP	0
3	lovely spring looked windsor terrace sixteen f...	MWS	2

Bert Tokenization

Type oftokenizationcalled wordpiece tokenization handle both common and rare words effectively.

```python
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = TFBertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=3)
```

Python

```python
for sent in sentences:
    # dictionary format that the BERT model can understand.
    encoded_dict = tokenizer.encode_plus(
        sent,
        add_special_tokens = True,
        max_length = 64,
        pad_to_max_length = True,
        return_attention_mask = True,
        return_tensors = 'tf',
    )

    input_ids.append(encoded_dict['input_ids'])
    attention_masks.append(encoded_dict['attention_mask'])
```

Python

Bert model can understand the dictionary format so we need to convert our text to that formmat.
here is a explanation of the Block:

1. **for sent in sentences:** This line starts a loop that goes through each sentence in the sentences list.
2. **encoded_dict** = tokenizer.encode_plus(: This line uses the BERT tokenizer to convert the sentence into a format that the BERT model can understand. The encode_plus method does several things:
   - sent,: This is the sentence that you want to encode.
   - add_special_tokens = True,: This adds the special [CLS] and [SEP] tokens at the beginning and end of the sentence, respectively. These tokens are required by BERT.
   - max_length = 64,: This sets the maximum length for the sentence. If the sentence is shorter than this, it will be padded with zeros at the end; if it's longer, it will be truncated.
   - pad_to_max_length = True,: This pads the sentence with zeros at the end until it reaches the maximum length.
   - **return_attention_mask** = True,: This creates an attention mask, which is a sequence of 1s and 0s indicating which tokens the model should pay attention to (1s) and which ones it should ignore (0s).
   - return_tensors = 'tf',: This returns the encoded sentence and attention mask as TensorFlow tensors.
3. **input_ids.append(encoded_dict['input_ids']):** This line appends the encoded sentence (now referred to as input_ids) to the input_ids list.
4. **attention_masks.append(encoded_dict['attention_mask']):** This line appends the attention mask to the attention_masks list.

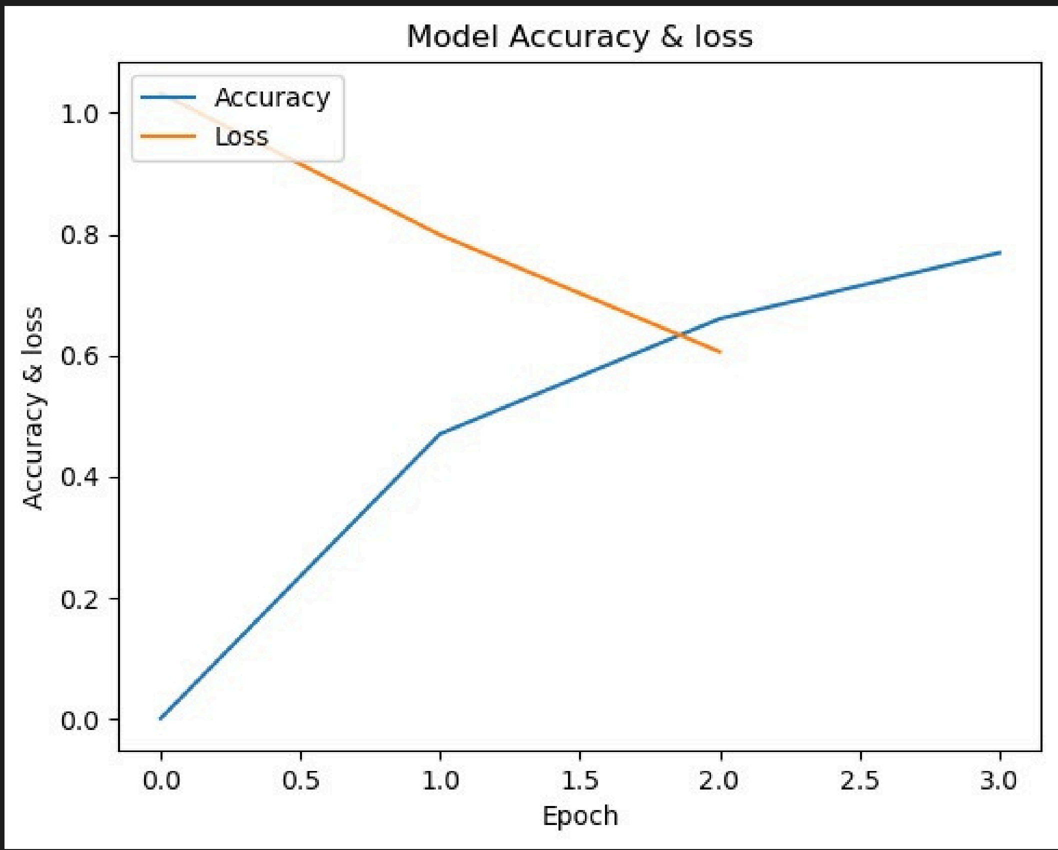
Model Training

We will try only 3 iterations with Batch_size of 32

```
model.compile(
            optimizer=tf.keras.optimizers.Adam(1e-5),
            loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=['accuracy']
            )
model.fit([train_inputs, train_masks], train_labels, batch_size=32, epochs=3)
import matplotlib.pyplot as plt
```

Evaluation

Wegofrom0.47 as **Accuracy** to 0.76 and from 1.03 as **Loss** into 0.60.



I'm satisfied with how well my text classification model based on **BERT** has performed. In contrast to the MLP approach, where we achieved an **accuracy** of **0.60**, our BERT model reached **0.768 accuracy**. However, to enhance its performance further, I'm planning to explore strategies such as **fine-tuning hyperparameters** to fine-tune the model and incorporating **Early Stopping**. This involves monitoring the validation set's loss during training and halting the process whenthe loss startsto rise, preventing overfitting on the training data.

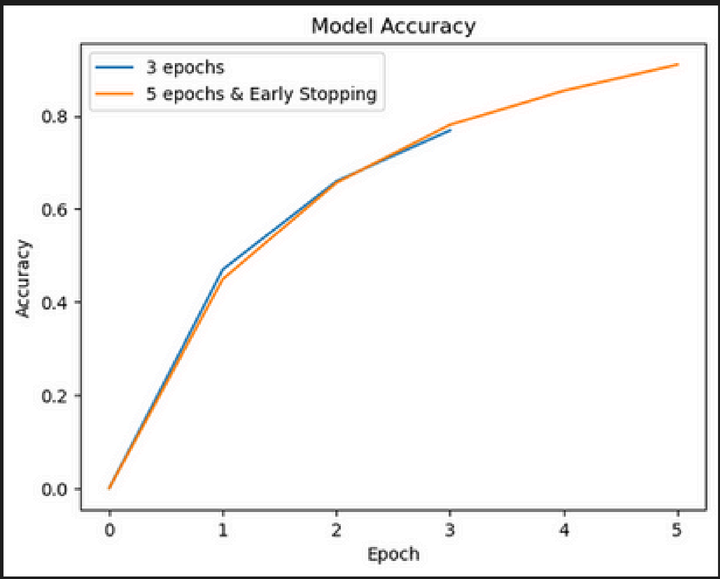
Performance Enhancing:

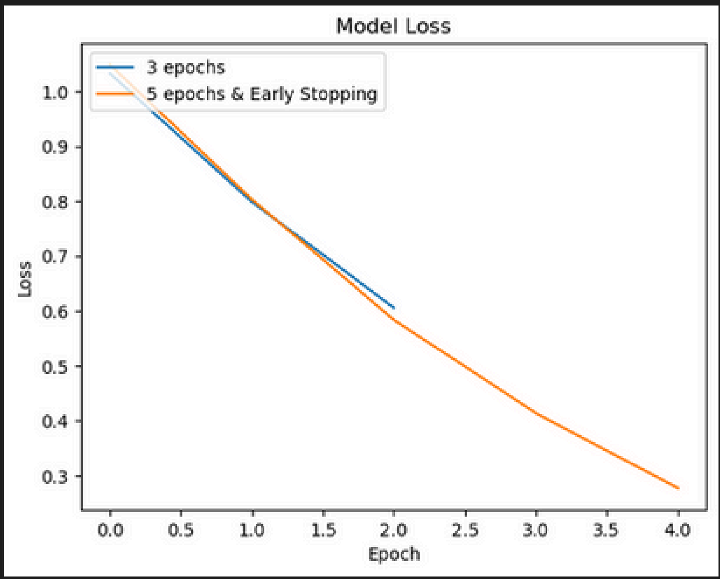
I’veaddedEarlyStoppingandincrease The number of iterations into 5

```python
from tensorflow.keras.callbacks import EarlyStopping
model2 = TFBertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=3)

# Define the early stopping callback
early_stopping = EarlyStopping(monitor='loss', patience=3)

model2.compile(
            optimizer=tf.keras.optimizers.Adam(1e-5),
            loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=['accuracy']
            )
model2.fit([train_inputs, train_masks], train_labels,callbacks=[early_stopping], batch_size=32, epochs=5)
```

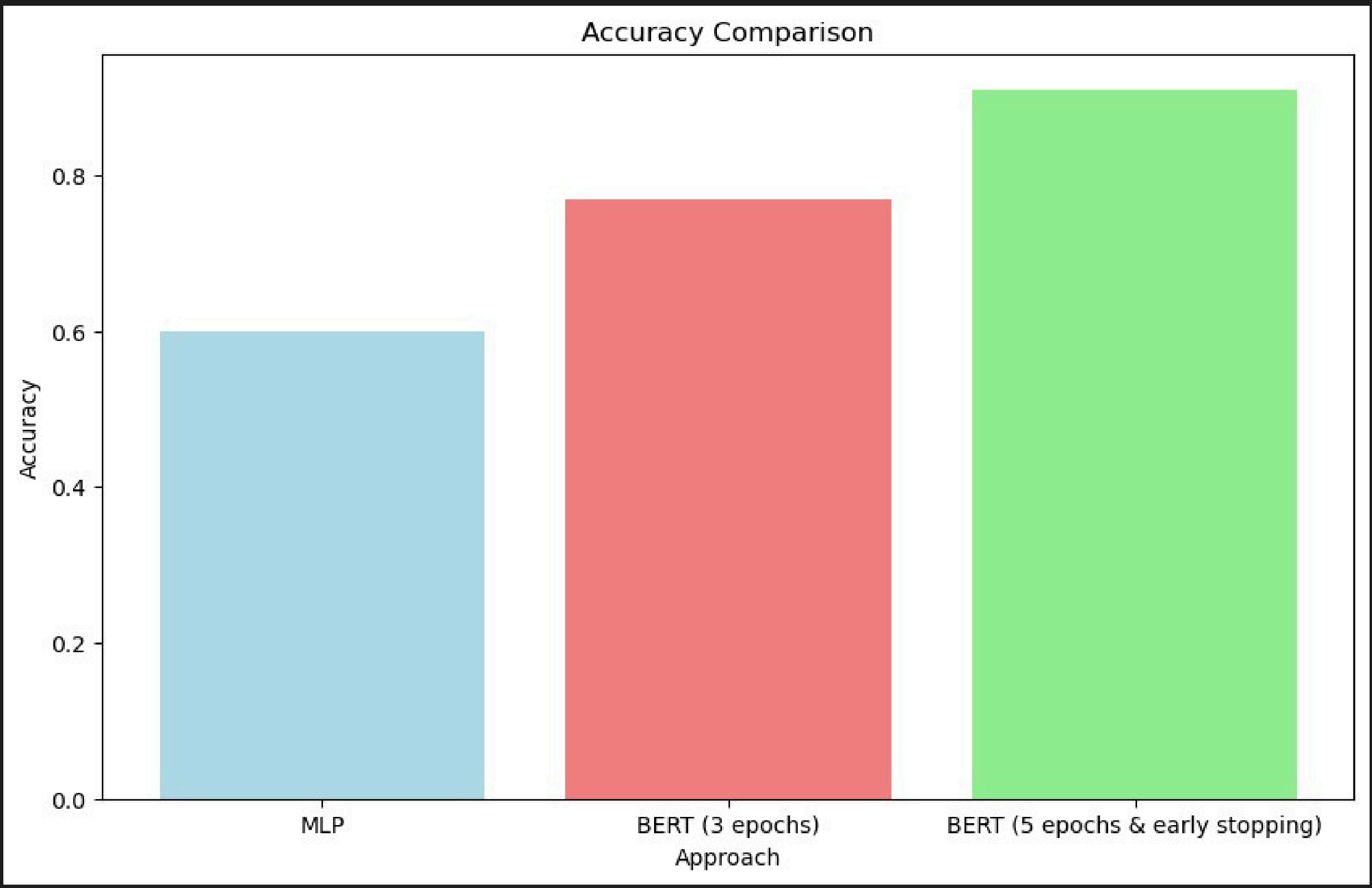
Python

Comparing the new Bert model To the previous one:





As evident from the results, our **accuracy** surged from **0.7689** to an impressive **0.91**, while the loss plummeted from **0.60** to a mere **0.27**. Notably, these enhancements were achieved without encountering any overfitting, all thanks to the implementation of early stopping. This outcome undoubtedly surpasses even the previous best performance.



Comparison between MLP, Bert , Bert with 5 epochs and Early stopping

PART 02

CNN

CNN (convolutional neural networks) model automatically learn and extract relevant features from text input. By applying one-dimensional convolutions over the input text, the model captures patterns and structures at different levels of granularity. These extracted features are then processed and combined through pooling layers before being fed into fully connected layers for classification into predefined categories or labels.

Data Loading & Preprocessing

westart bydataloading&Cleaning,wetakeonly above 10% from the dataset, 1000 for each Author

```
df = pd.read_csv("preprocessed_data.csv")

#clean dataset from ununsed features
df.drop(['id'],axis=1,inplace=True)

# Group the DataFrame by author and select the first 1000 rows for each author
df_sampled = df.groupby('author').head(1000)

# Concatenate the sampled DataFrames into a new DataFrame
df_new = pd.concat([df_sampled], ignore_index=True)

df_new.author.value_counts()
```
Python

Encode The label Values

	text	author	author_encoded
0	proces however afforded means ascertaining dim...	EAP	0
1	never occurred fumbling might mere mistake	HPL	1
2	left hand gold snuff box capered hil cutting m...	EAP	0
3	lovely spring looked windsor terrace sixteen f...	MWS	2

Tokenization And pad Sequences

weuse pad_sequencestoensurethatallsequences in a list have the same length.

```
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(df_new['text'])
sequences = tokenizer.texts_to_sequences(df_new['text'])
data = pad_sequences(sequences, maxlen=200)
```
✓ 0.2s Python

Model Defining

we useModel with4 hidden layers :

- Embedding layer used for word embeddings where words or phrases from the vocabulary are mapped to vectors of real numbers.
- Conv1D: This layer creates a convolution kernel
- GlobalMaxPooling1D: This layer applies max pooling operation for temporal data.
- Dense: This layer is a regular fully-connected neural network layer.

and output layer using Softmax activation function.

```
model = Sequential()
model.add(layers.Embedding(input_dim=5000, output_dim=50, input_length=200))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(3, activation='softmax'))
```
✓ 0.1s Python

Model Summary

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 200, 50)           250000

 conv1d_1 (Conv1D)           (None, 196, 128)          32128

 global_max_pooling1d_1 (Gl  (None, 128)               0
 obalMaxPooling1D)

 dense_2 (Dense)             (None, 10)                1290

 dense_3 (Dense)             (None, 3)                 33

=================================================================
Total params: 283451 (1.08 MB)
Trainable params: 283451 (1.08 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Model Train

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

✓ 0.0s Python

Early Stoping Defining:

```python
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

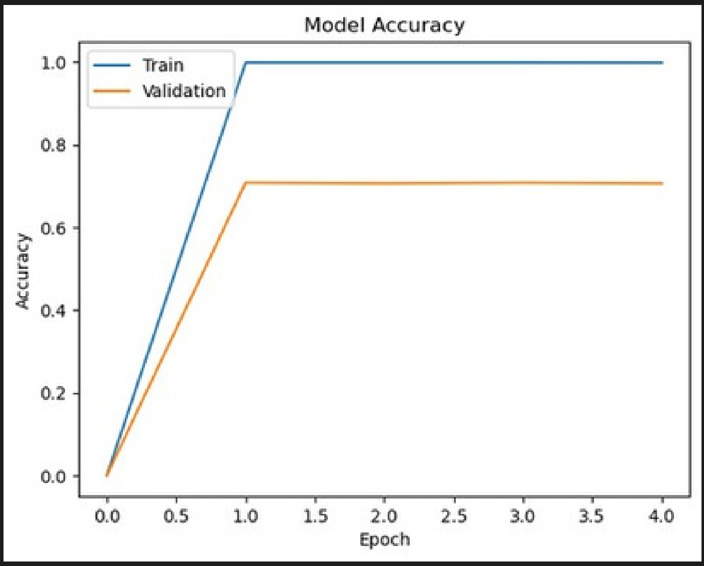
✓ 0.0s Python

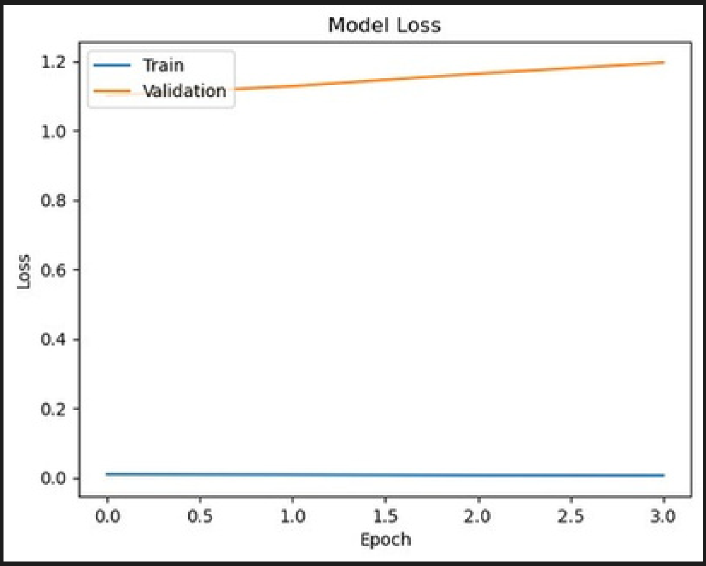
The Model Stop train after only 4 iterations while i declare 10

```
Epoch 1/10
38/38 [==============================] - 2s 48ms/step - loss: 0.0092 - accuracy: 0.9987 - val_loss: 1.1003 - val_accuracy: 0.7083
Epoch 2/10
38/38 [==============================] - 2s 45ms/step - loss: 0.0080 - accuracy: 0.9987 - val_loss: 1.1276 - val_accuracy: 0.7067
Epoch 3/10
38/38 [==============================] - 2s 44ms/step - loss: 0.0065 - accuracy: 0.9987 - val_loss: 1.1637 - val_accuracy: 0.7083
Epoch 4/10
38/38 [==============================] - 2s 43ms/step - loss: 0.0060 - accuracy: 0.9983 - val_loss: 1.1953 - val_accuracy: 0.7067

<keras.src.callbacks.History at 0x1dddf7ced50>
```

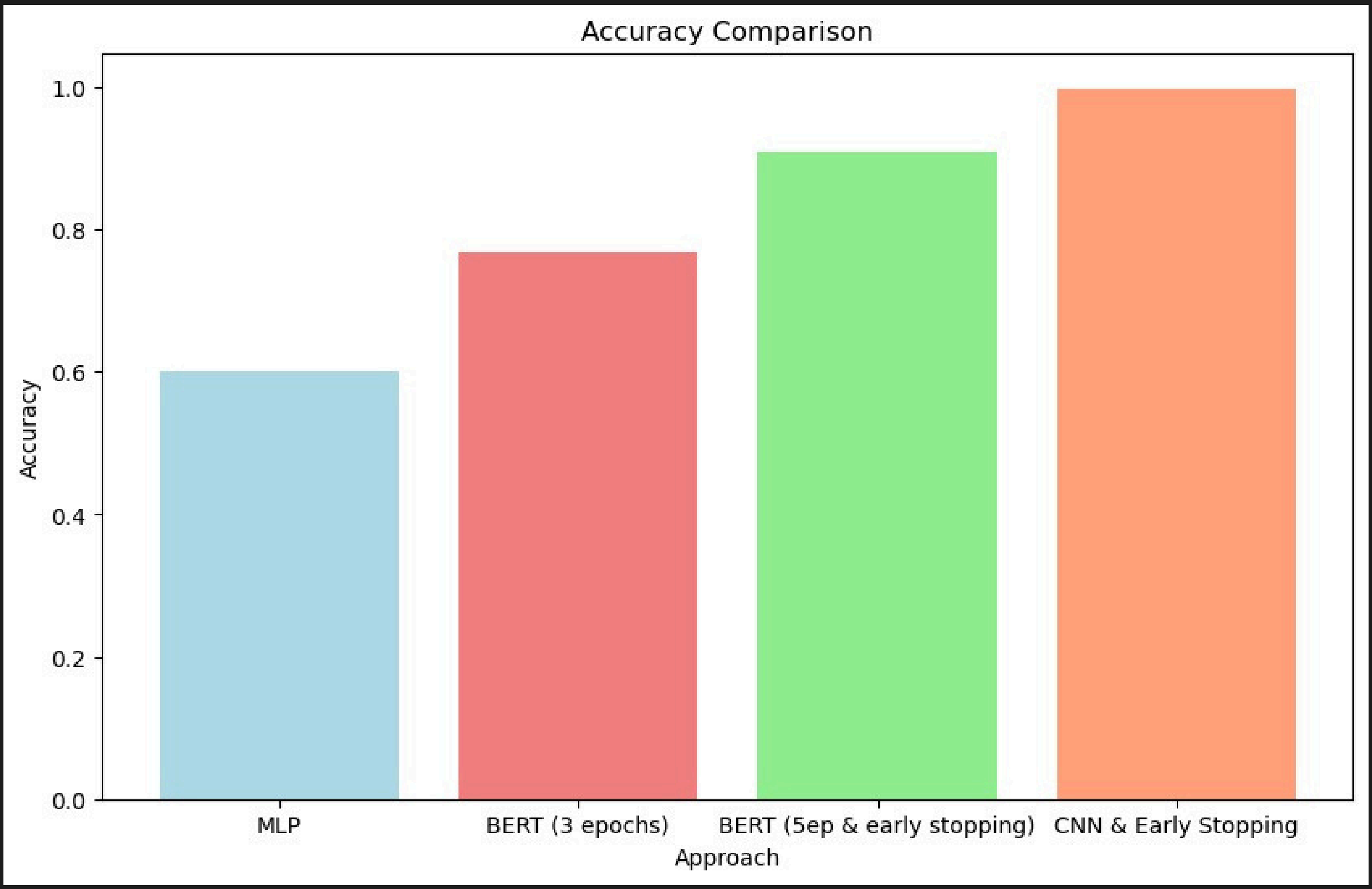
Evaluation

Wegoet**0.99**as **Accuracy** and from **0.008** as **Loss .**





I'm really satisfied with how well my text classification model based on CNN has performed.



Comparison between MLP, Bert , Bert with 5 epochs and Early stopping, CNN