HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and Communication Technology

----- □ & □ -----

Capstone project

Sentiment Analysis of movie reviews

Subject: Introduction to Deep Learning - IT3320E

Supervisor: Professor. Nguyễn Hùng Sơn

Group: 17

Authors:

Vũ Hoàng Nhật Anh	20225471
Vũ Tùng Linh	20210523
Trần Nhật Minh	20225511
Trần Nam Tuấn Vượng	20225540

Hanoi, 12/2024

Abstract

This report presents a detailed analysis of sentiment classification using the SST-5 dataset, a widely used resource for evaluating the performance of sentiment analysis models. The dataset provides multiple sentences along with sentiment labels from very negative to very positive, providing a robust framework for training and testing machine learning models. Our study explores the application of various natural language processing (NLP) techniques and applying machine learning algorithms for prediction, with a focus on deep learning models. We evaluate the performances of these models, compare their results across different sentiment classes. Additionally, we investigate some challenges posed by the dataset. Through this analysis, we demonstrate the efficacy of advanced models in sentiment analysis tasks and provide insights into optimizing sentiment prediction for real-world applications.

Table of Contents

Part I: Introduction

Part II: Dataset

Part III: Methodology

Part IV: Results and Comparison

References

Workshare

Vũ Hoàng Nhật Anh	Report, infer.py, data analysis, demo, CNN, Albert model
Vũ Tùng Linh	Report, deBerta, Xlnet model
Trần Nhật Minh	LSTM model, data preprocessing
Trần Nam Tuấn Vượng	Report, Bert model, demo

Part I: Introduction

1.Background

Sentiment Analysis, also known as opinion mining, or emotion AI, is an approach to natural language processing (NLP) that is used to determine the sentiment that is expressed in a piece of text, a sentence, or a paragraph,etc. In other words, the main goal of sentiment analysis is to understand the emotions provided by a person. The term "sentiment analysis" can be traced back to the middle of the 19th century, in the book "The General Inquirer: A Computer Approach to Content Analysis" published in 1966. The lack of techniques and tools at the time meant that researchers would not be able to gain much knowledge. Many decades later, we experienced through the rise of Machine Learning in the 1990s, the era of Internet and Social Media in the 2000s and great achievements in natural language processing and deep learning in the 2010s, and great achievements have breathed new life into this decades-old practice. Nowadays, sentiment analysis is now widely used worldwide and has various applications. The customers can write their comments, reviews on social media, on their personal blogs or on review pages, and those pieces of information can be collected and analyzed to provide insights to how their emotions were expressed when they wrote the sentences. Companies can use customer feedback analysis to make meaningful changes, to provide a better experience for current and future customers. In conclusion, Sentiment analysis has come a long way from simple text classification methods to lexicon-based methods or simple machine learning classifiers to complex deep learning models capable of understanding nuanced human emotions. Its application continues to grow, making it a vital tool for marketing, brand management, or even social science research. The future of sentiment analysis may improve even further when more sophisticated models can outperform the current ones in the task of capturing the emotions of human languages.

2.Problem formulation

In the context of this project, we provide several models to tackle the challenge of this NLP task, more specifically, a fine-grained sentiment analysis using the SST-5 (Stanford Sentiment Treebank) dataset. This task involves classifying textual data into one of five sentiment categories: very negative, negative, neutral, positive and very positive. Fine-grained sentiment analysis goes beyond

traditional binary or ternary classifications, requiring a deeper understanding of linguistic and contextual nuances.
The problem can also be formulated as a multi-class classification task, where our models need to predict the sentiment of input text with fine-grained detail.

3. Aims

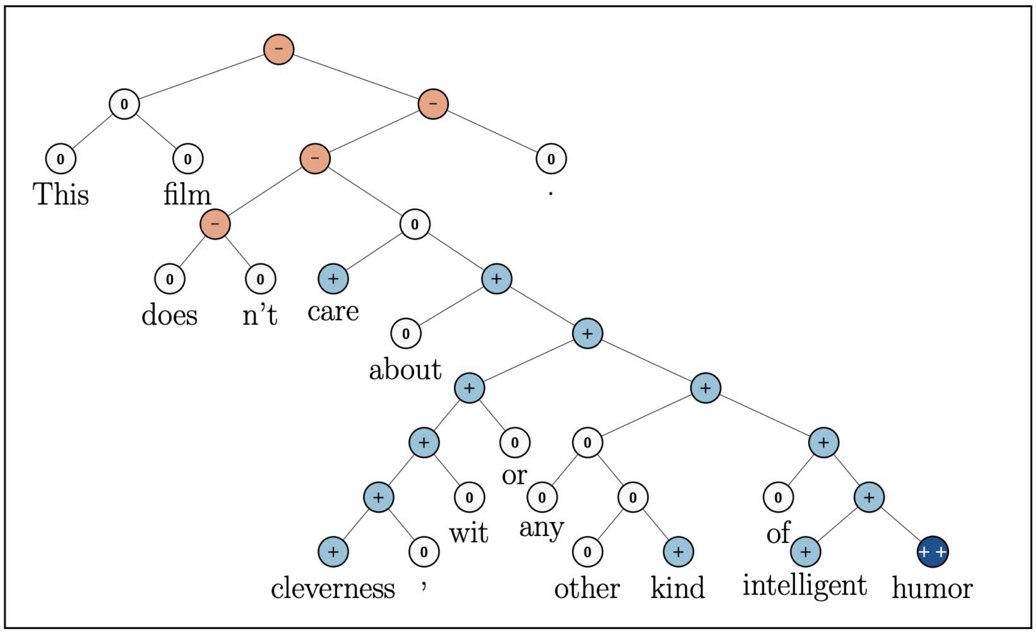
The primary aim of this project is to develop and evaluate effective models for fine-grained sentiment analysis using the SST-5 dataset. The objectives include:
+) Achieving Accurate Sentiment classification: Building models capable of classifying text into one of five sentiment categories,
+) Compare the performances of different machine learning and deep learning models to identify the most effective approach.
+) Benchmark performances: Evaluate the model to benchmark their performance on the SST-5 dataset.
By achieving all the above aims and goals, this project seeks to advance the understanding and application of fine-grained sentiment analysis techniques in natural language processing.

Part II: Dataset

1.Introduction

The SST-5, also known as the Stanford Sentiment Treebank, is one of the most prominent datasets in the field of natural language processing, specifically for sentiment analysis tasks. The dataset is an improvement of the previously released SST-2 dataset, introduced in 2013 in the seminar paper "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank," by Socher et al, creates a milestone in sentiment analysis research.The SST dataset was originally designed to address key limitations of previous sentiment analysis benchmarks, which at the time, often relied on coarse-grained sentiment labels (positive vs negative), and lacked the granularity required to capture nuanced sentiment expressions. SST introduced the concept of fine-grained sentiment classification, splitting sentiment into five categories: very negative, negative, neutral, positive, and very positive.
As the name suggests, the dataset is uniquely known for its hierarchical treebank structure, not limited to sentence-level annotations. Sentiment labels are assigned to individual phrases and sub phrases within sentences. This granularity allows models to analyze how sentiment is distributed and composed within a sentence, offering insights into the role of syntactic and semantic structure in sentiment expression.

This film doesn't care about wit, cleverness, or any other kind of intelligent humor.



```
(2 (3 (3 Effective) (2 but)) (1 (1 too-tepid) (2 biopic)))
```

Picture: an example of hierarchical tree structure

The SST-5 dataset is relatively small. contains 11,855 rows or sentences, each with a corresponding label tag, ranging from the length of 4 characters to 283 characters:

Sentences	label	label_text
like a can of 2-day old coke .	0	very negative
apparently reassembled from the cutting-room floor of any given daytime soap .	1	negative
um , no. .	2	neutral
smaller numbered kidlets will enjoy .	3	positive
a stirring , funny and finally transporting re-imagining of beauty and the beast and 1930s horror films	4	very positive

An example of some sentences in the dataset

In the dataset, the labels are encode from 0 to 4 as follows:

+) Very negative (label 0): Sentences that express strong negative sentiment or intense dissatisfaction. These sentences often include harsh criticism, severe disapproval, or strongly negative emotional tones.

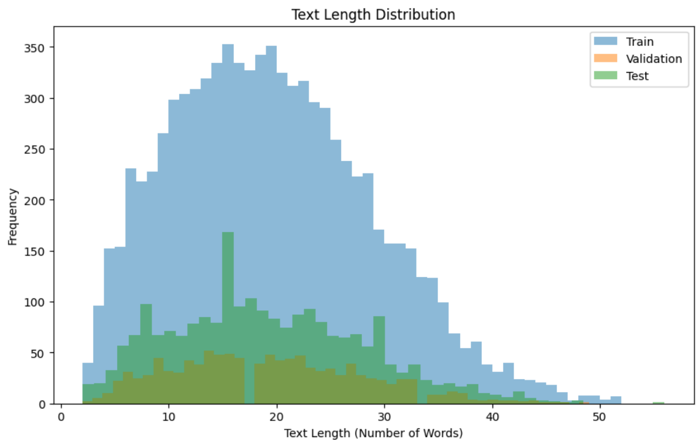
+) Negative (label 1): Sentences that convey a general sense of negativity or disapproval, but with less intensity than "very negative." These sentences may critique or highlight shortcomings without being overly harsh.

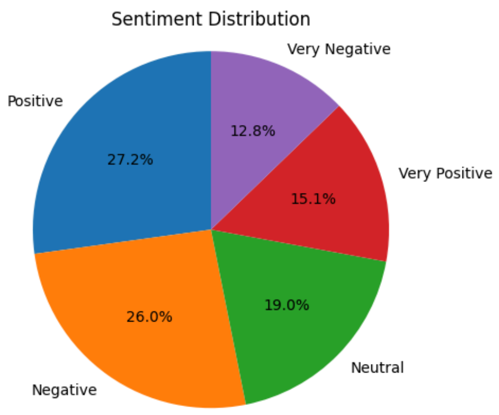
+) Neutral (label 2): Sentences that are objective, balanced, or express mixed sentiment without leaning clearly positive or negative. These sentences might present facts, provide descriptions, or offer lukewarm opinions.

+) Positive (Label 3): Sentences that exhibit positive sentiment or approval, expressing general satisfaction or appreciation. These sentences highlight good qualities without extreme enthusiasm.

+) Very Positive (Label 4): Sentences that reflect strong positive sentiment or high praise, showing intense enthusiasm, admiration, or delight.

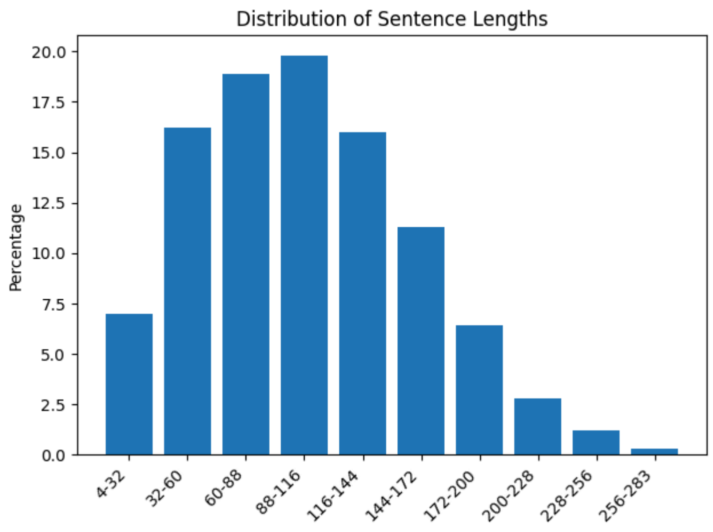
The corresponding distribution of the 5 labels are as follows:





Graph: The text length distribution in train, validation and test set

Graph: Sentiment distribution of the dataset



Graph: The distribution of the lengths of sentences in the train set

2.Data preprocessing

In order to prepare our data for use in model training and testing processes, we apply a processing procedure which contains multiple natural language processing techniques with the primary aim to remove noise from the original data. Some preprocessing techniques were applied in models and here are the following:

+) *Punctuation removal and lowercase*: All text is converted to lowercase to maintain uniformity. Punctuation is removed to focus on the meaningful words.

+) *Stop words removal*: Typically, stop words like "the", "is", "in", etc., are removed as they don't add significant meaning to the text.

+) *Tokenization & Lemmatization*: The TweetTokenizer was employed for tokenizing the text. It is suitable for handling the complexities of social media text, such as hashtags and mentions. Lemmatization was applied to convert words to their base form, ensuring that words like "running" and "runner" are treated as "run".

+) *Fixing Apostrophes*: A custom function fix_apostrophes was applied to handle apostrophes in contractions. This helped correct common issues like "wo n't" to "won't".

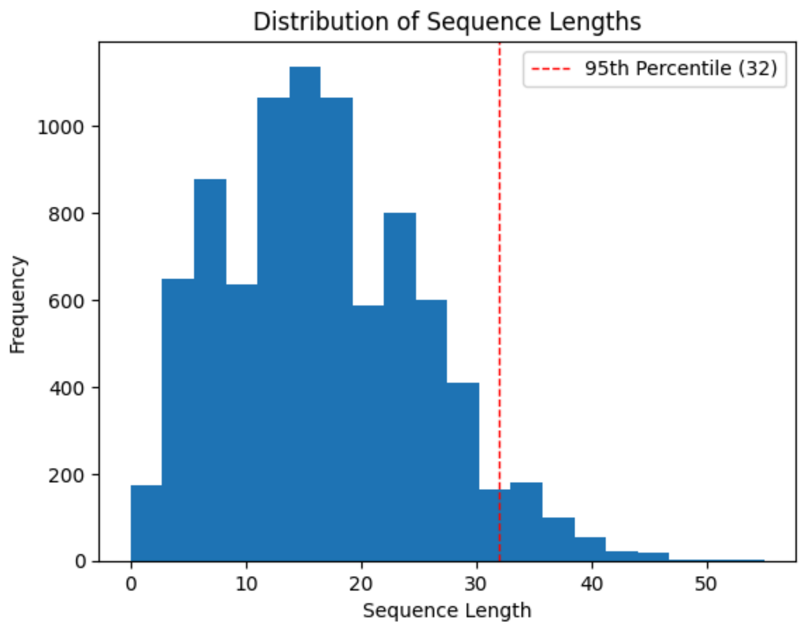
+) *Contraction Expansion*: The function fix_contractions was used to expand common contractions, ensuring that words like "won't" and "can't" are treated consistently as "will not" and "cannot".

Input Preparation

+) *Padding*: Padding is a technique used to ensure that all sequences of text (or tokenized words) have the same length before being fed into a model. The tokenized sequences were padded to a uniform length of 32 tokens using the Keras `pad_sequences`. This uniform length, represented by `max_length`, is necessary for processing with the convolutional neural network.

Before padding	After padding
["movie", "great"]	["movie", "great", 0, 0, 0…,0]
["plot", "was", "really", "good"]	["plot", "was", "really", "good", 0..,0]

* `max_length`: The padded sequences are of length 32. This parameter is used for padding. The number 32 was chosen because it was the 95th percentile of sequence lengths in the training set



Graph: Distribution of Sequence Lengths

+) *Embedding Matrix*: The embedding matrix holds word vectors in a 2D array, where each row corresponds to a word and its corresponding embedding vector. When we train a model, instead of representing words by one-hot encoding, we use these vectors to provide more meaningful representations of words. Each word in our vocabulary is represented as a 300-dimensional vector. These vectors are designed to capture the meaning and context of words, such that words with similar meaning have a similar vector representation. We utilized a Word2Vec model, trained using the Gensim library, to generate word embeddings for the tokens. The model was trained as follows:

* The `custom_word2vec` model was trained on the training text data.

* `vector_size`: The embedding vector size used was 300, corresponding to `embedding_dim`.

* `window`: A context window size of 5 was used to define the maximum distance between the current and predicted word within a sentence.

* `min_count`: The model ignores words with a frequency lower than 2.

* `workers`: 4 worker threads were used for faster training.

The Word2Vec vectors were stored in a numpy array that can be used as a weight for the Embedding layer.

Padding and Embedding matrix is specifically used for convolutional Neural network and Long-short-term-mermoy, as Bert-related models don't require these techniques. The stop-words removal technique is used carefully in certain models so that it does not affect the quality of the original dataset and accidentally make it harder for our model to work with.

3. Results

The following is an example of the dataset after the data preprocessing step:

Original Text	I can't believe it, this isn't working! I have been running all day, and I won't stop 🙂.
Lowercasing and Punctuation Removal	"i cant believe it this isnt working i have been running all day and i wont stop:)"
Fixing Apostrophes	i cannot believe it this isnt working i have been running all day and i will not stop
Contraction Expansion	i cannot believe it this isnt working i have been running all day and i will not stop
Tokenization	["i", "cannot", "believe", "it", "this", "isnt", "working", "i", "have", "been", "running", "all", "day", "and", "i", "will", "not", "stop"]

Lemmatization	["i", "cannot", "believe", "it", "this", "isnt", "work", "i", "have", "be", "run", "all", "day", "and", "i", "will", "not", "stop"]

The preprocessing steps improved the dataset by removing irrelevant noise and standardizing the text. The dataset is now ready for model training, with clean, meaningful features that will allow the machine learning model to better learn the sentiment patterns. These preprocessing techniques have enhanced the quality and consistency of the data, setting the foundation for effective model performance.

Part III: Methodology

In the context of this project, we have tried a few models (trained and pretrained-included) to do the task of sentiment analysis. Convolutional Neural Network is the first model we tried but the result was poor. We then examined different types of model including the Long-short-term-memory (LSTM), BERT and some BERT variations, and Xlnet. In the following pages, we dive deeper in the architecture, the applied method of each model.

1. Convolutional Neural Network (CNN)
2. Long-Short-Term-Memory (LSTM)
3. BERT
4. DeBERTa
5. Albert
6. XLNet

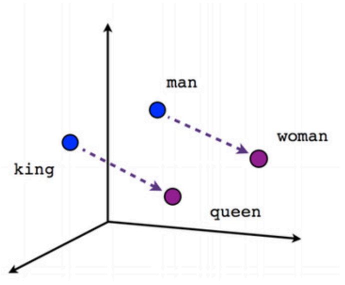
1. Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) have revolutionized machine learning in recent years, enabling breakthroughs in tasks that traditional machine learning models struggled with. In this project, a CNN architecture was utilized for analyzing and classifying text data. The approach was designed to process raw text efficiently while leveraging advanced techniques to improve model accuracy.

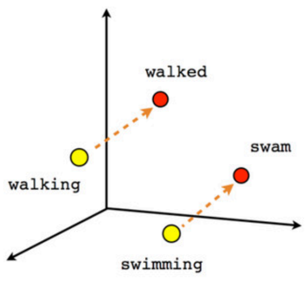
Model Overview

Embedding Layer

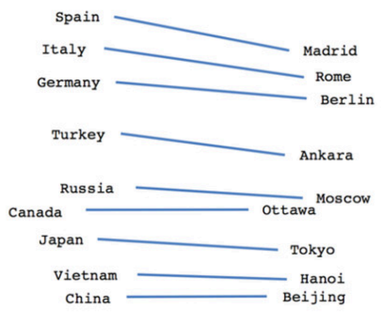
The CNN model begins with a pre-trained Word2Vec embedding layer, leveraging pre-trained word vectors. These embeddings capture semantic relationships between words, enabling the model to understand complex linguistic patterns. The embedding layer was initialized with these pre-trained vectors, which allowed the model to benefit from prior linguistic knowledge while mitigating overfitting by keeping the embeddings trainable.



Male-Female



Verb tense

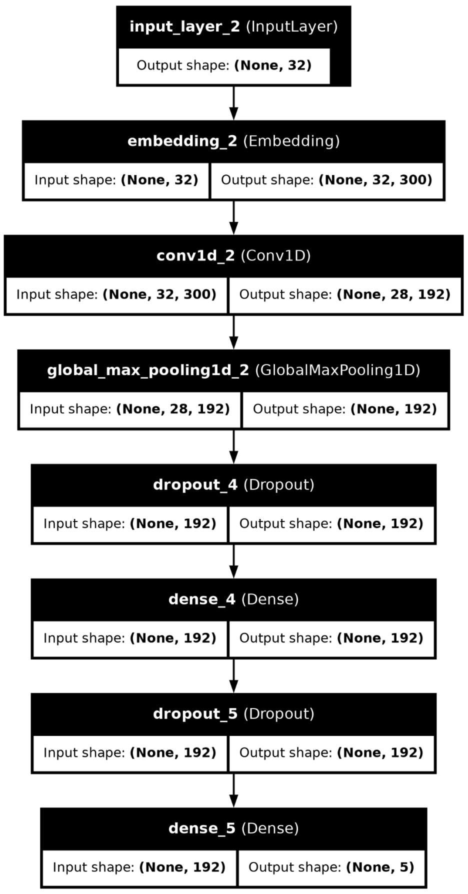


Country-Capital

The CNN architecture is designed to extract features from text data. The input sequences are passed through an embedding layer, **embedding_2**, which maps sequences of tokens into dense vectors of size 300. This layer allows the model to capture semantic relationships between words in the input sequences. Following the embedding layer is a **1D convolutional layer, conv1d_2**, with 192 filters. This layer applies a convolution operation with a kernel size that reduces the output sequence length to 28. A ReLU activation function is applied to this layer to introduce non-linearity, enabling the model to capture local patterns and n-gram relationships within the sequences. Next, a **global max-pooling layer, global_max_pooling1d_2**, is added. This layer reduces the dimensionality of the output from the convolutional layer by selecting the most salient features across the sequence. The pooled features are passed through a **dropout layer, dropout_4**, which applies a dropout rate to prevent overfitting. After this, a **dense layer, dense_4**, with 192 units learns the complex non-linear relationships between the extracted features. Another **dropout layer, dropout_5**, is applied for further regularization. Finally, the output layer, **dense_5**, consists of 5 neurons with softmax activation. This layer produces the final probability distribution corresponding to the five sentiment categories of the SST-5 dataset.

Layer (type)	***Output Shape***	***Param #***
input_layer_2 (InputLayer)	*(None, 32)*	*0*
embedding_2 (Embedding)	*(None, 32, 300)*	*4,100,700*
conv1d_2 (Conv1D)	*(None, 28, 192)*	*288,192*
global_max_pooling1d_2 (GlobalMaxPooling1D)	*(None, 192)*	*0*
dropout_4 (Dropout)	*(None, 192)*	*0*
dense_4 (Dense)	*(None, 192)*	*37,056*
dropout_5 (Dropout)	*(None, 192)*	*0*
dense_5 (Dense)	*(None, 5)*	*965*

```
Total params: 13,280,741 (50.66 MB)

Trainable params: 4,426,913 (16.89 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 8,853,828 (33.77 MB)
```


input_layer_2 (InputLayer)
Output shape: (None, 32)
embedding_2 (Embedding)
Input shape: (None, 32)
Output shape: (None, 32, 300)
conv1d_2 (Conv1D)
Input shape: (None, 32, 300)
Output shape: (None, 28, 192)
global_max_pooling1d_2 (GlobalMaxPooling1D)
Input shape: (None, 28, 192)
Output shape: (None, 192)
dropout_4 (Dropout)
Input shape: (None, 192)
Output shape: (None, 192)
dense_4 (Dense)
Input shape: (None, 192)
Output shape: (None, 192)
dropout_5 (Dropout)
Input shape: (None, 192)
Output shape: (None, 192)
dense_5 (Dense)
Input shape: (None, 192)
Output shape: (None, 5)

Model structure

Hyperparameter tuning
The above CNN model structure was derived through an extensive hyperparameter tuning process designed to identify the optimal configuration for this specific text classification task. The tuning process leveraged Keras Tuner's Hyperband algorithm, which efficiently explores the search space by allocating resources to promising configurations while discarding less effective ones.The objective of the tuning process was to maximize validation accuracy. The search space included the following hyperparameters:

Parameters	Search size	Optimal result
Filters in Conv1D Layer	[64,128,192,256]	192

Kernel Size in Conv1D Layer	[3,5,7]	5
Dropout Rates	[0.2,0.3,0.4,0.5]	0.3 and 0.4 for the 1st and 2nd dropout layer
Units in Dense Layer	[128,192,256]	192
Learning Rate	$1x10^{-4}$ to $1x10^{-2}$	$3x10^{-4}$

Model tuning was also applied. The aim of model tuning was to focus on defining the architecture or structure of the neural network. This process explores the high-level design of the model and ensures that the architecture aligns with the problem's requirements. Though after training and evaluating, it gets more difficult and less effective so we consider using the model mentioned above.

Training and Evaluation

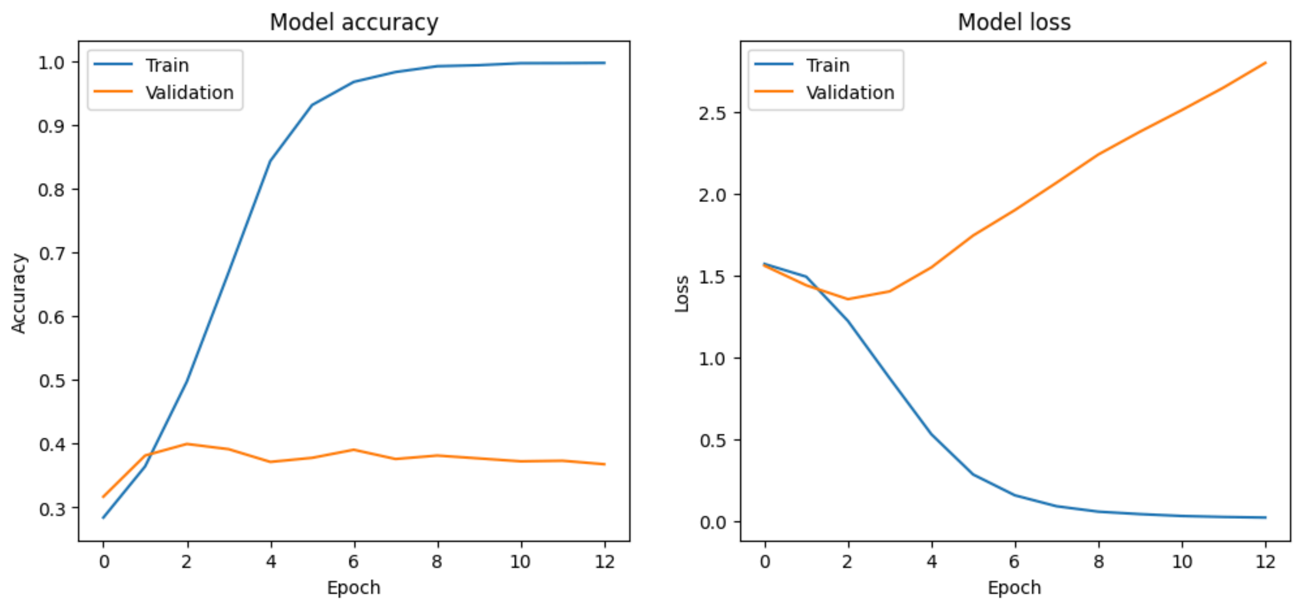
The models were trained on the training dataset with early stopping applied to monitor the validation loss, preventing overfitting. The training process ran for up to 30 epochs with a patience of 10 epochs for early stopping. Input sequences were padded to a uniform length of 32 tokens before being processed, ensuring consistency in input size. Labels were converted into a one-hot format to work with the categorical cross-entropy loss function.

The model utilized Xavier (Glorot Normal) initialization for parameter initialization, with the Adam optimizer at a learning rate of $3x10^{-4}$. Categorical cross-entropy was used as the loss function, with accuracy as the evaluation metric, but we also focus on the F1-score metric as well.

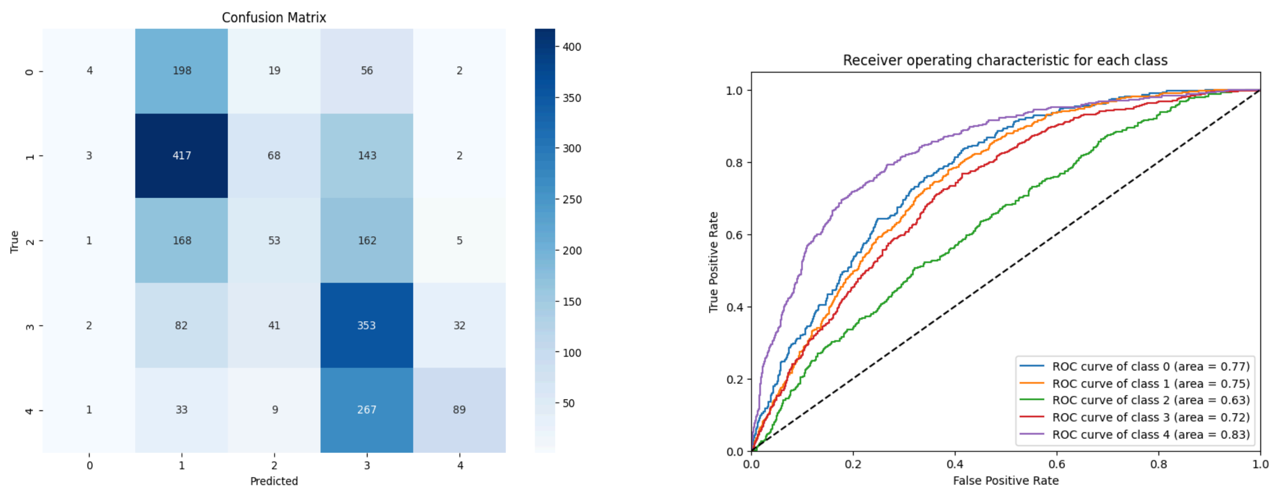
After training the models on the training dataset and monitoring performance on the validation dataset, the final evaluation was conducted using the test dataset. A class weights dictionary value for each class was also applied to further

	precision	recall	f1-score	accuracy
0	0.36	0.01	0.03	
1	0.46	0.66	0.54	
2	0.28	0.14	0.18	
3	0.36	0.69	0.47	
4	0.68	0.22	0.34	
				0.41

Classification report of CNN



Graph: Model loss over epochs



Confusion matrix and ROC curve of CNN

From the above results, it can be seen that the CNN model can predict the negative sentiment better than others with precision of 0.46. The model seems to mistake positive-very positive and negative-very negative sentiment. The accuracy of models in the training set increases over epochs while the validation set doesn't show such property. The loss of the model on the train set does decrease over epochs but the validation loss spikes up after 7 epochs, which indicates overfitting.

2. Long Short Term Memory (LSTM)

Recurrent neural networks (RNNs) are a form of Artificial Neural networks that can memorize arbitrary-length sequences of input patterns by capturing connections between sequential data types. This is suitable for most natural language processing tasks and for this sentiment analysis task specifically as sentences all vary in length and the meaning of a word often is determined by other words surrounding it. In other words, natural sentences, in the context of deep learning, are arbitrary-length sequential data. However, RNN possesses many limitations, one of which is the vanishing/exploding gradient problem that makes it unreliable very quickly as the length of the sequences grows.

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that is capable of learning long-term dependencies in sequence data. LSTM models overcome the limitations of traditional RNNs, such as vanishing gradient problem, by using a unique gating mechanism that controls the flow of information between cells in the network. The LSTM layers take into account not only the word order but also their contextual significance within the sentence. Through this process, the model discerns key patterns residing within the sequences, identifying their correlation with specific emotions.

To further improve the model's ability to capture context, Bidirectional LSTM (BiLSTM) was included in this architecture. While traditional LSTM processes input sequences in one direction (from past to future), BiLSTM processes the sequence in both forward and backward directions. This bidirectional approach allows the model to capture dependencies from both the past and the future within a sequence, which is particularly important for tasks like sentiment analysis where understanding the full context of a sentence is crucial. For example, in sentiment analysis, the meaning of a word can depend on both the preceding and following words, and BiLSTM enables the model to consider both directions simultaneously. This results in improved accuracy and a better understanding of the underlying patterns in the data, leading to more accurate sentiment predictions.

This model produces its prediction in the last layer, which is a softmax layer producing a probability distribution of potential emotions. With the highest-probability emotion selected as its prediction, this model possesses a key advantage in its adaptability to varying sentence lengths due to the inherent properties of LSTM networks. As such, it boasts immense versatility and efficacy in accurately classifying the conveyed emotion.

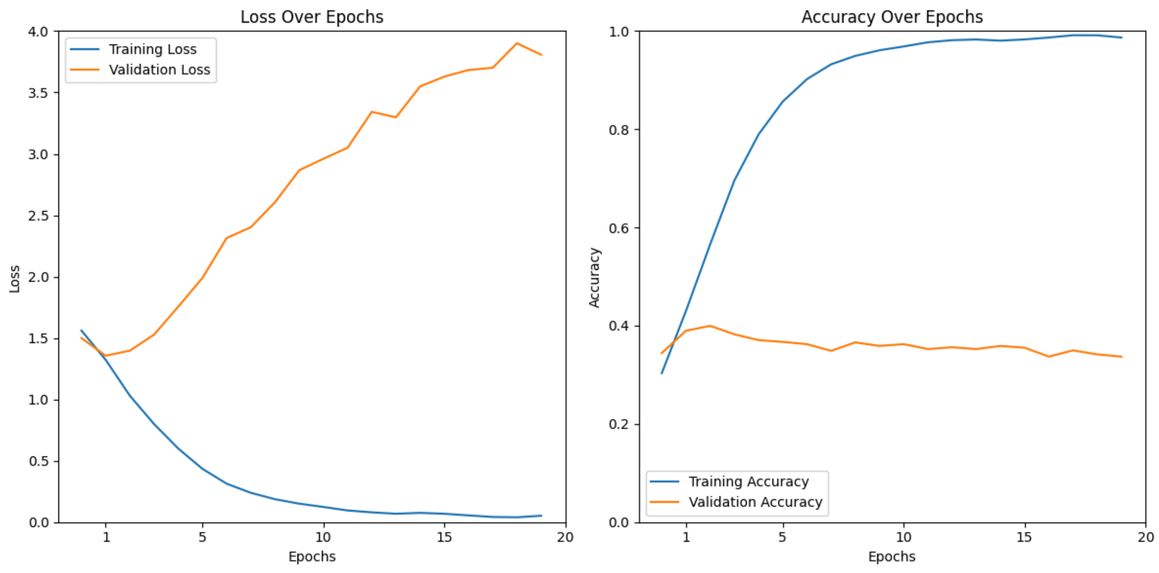
Additional layers were added after experimentations:

1. A densely connected layer was added between the LSTM layer and the final layer. This dense layer refines feature representations for the input sequence produced by the previous layer further by learning higher-level, task-specific features with the help of the ReLU activation function, which introduces non-linearity to handle complex patterns.
2. A dropout layer with the rate of 0.5 to reduce overfitting.

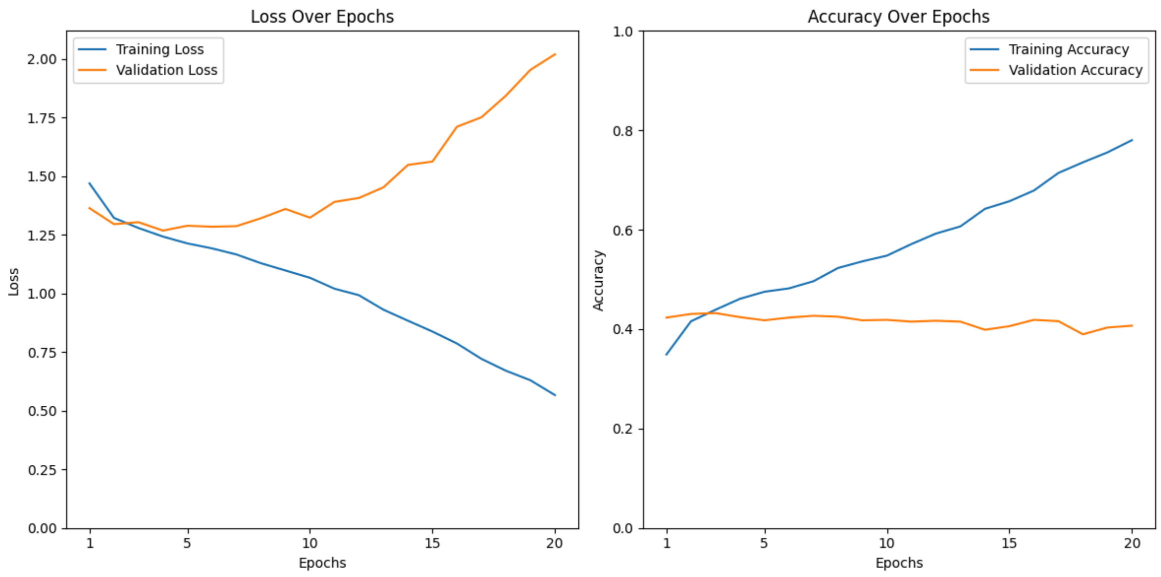
For the purpose of this project, two similar models were trained and compared. The first model contains an embedding layer that learns from scratch, while in the second model that layer is replaced with Google's pre-trained Word2Vec embeddings. Though the first approach produces task-specific and data-specific embeddings, the relatively small size of the dataset may make it less effective, thus creating ways for the general-purpose embeddings that leverage extensive pre-training to produce better results.

Training results show that both approaches to embedding produce models that are subject to overfitting. However, the model with self-learned embedding overfit much more severely and also provide slightly lower accuracy. This could be attributed to the fact that the self-learning approach produces a highly specified embedding that ends up being over-specified. Other factors like experience of our team or the size of the dataset are also considered when explaining the difference in performance between the self-learn approach and the pre-trained Word2Vec approach.

Compare the performance of two embedding approaches				
Name	Accuracy	Precision	F1-score	Recall
Self-learn	0.364	0.364	0.369	0.364
Pre-trained Word2Vec	0.442	0.455	0.415	0.442

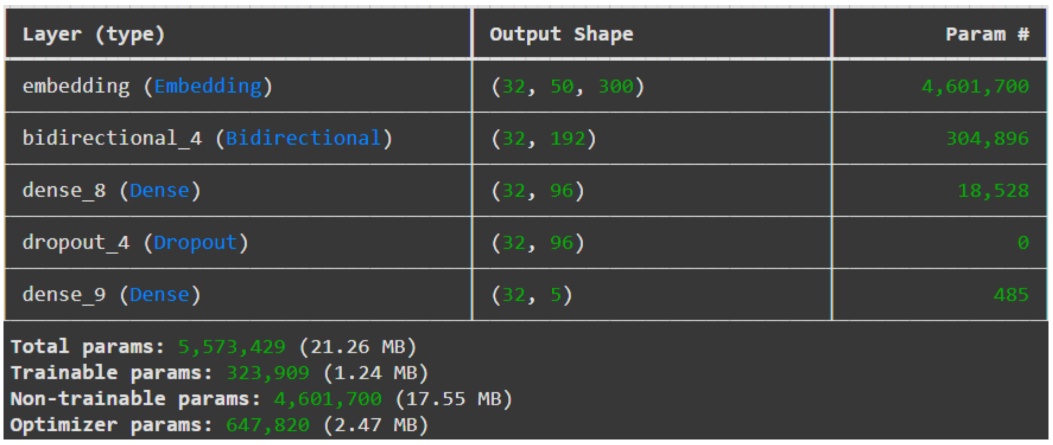


Self-learned embedding training results



Pre-trained Word2Vec embedding training results

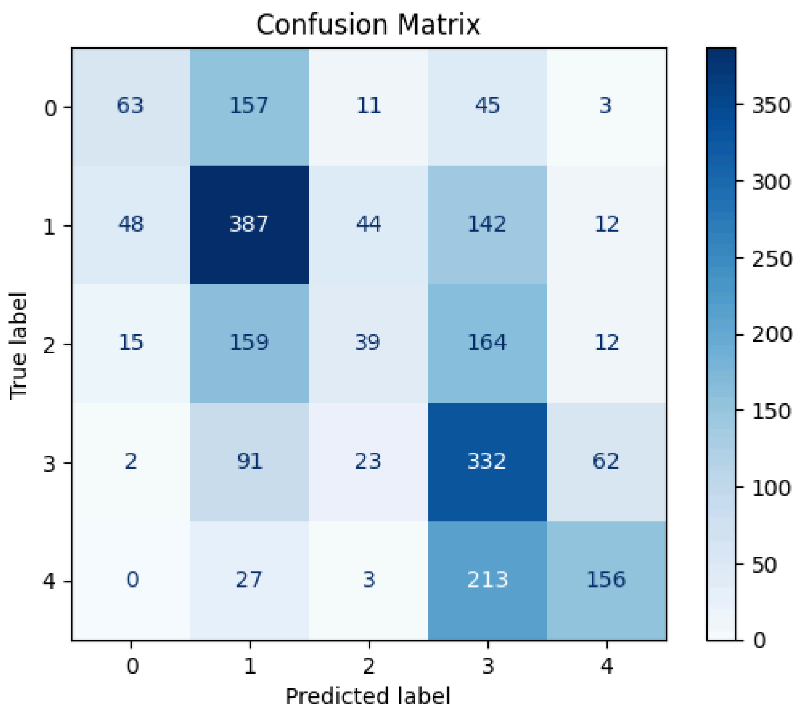
Naturally, the Word2Vec approach was chosen as the main approach for the final model.



Layer (type)	Output Shape	Param #
embedding (Embedding)	(32, 50, 300)	4,601,700
bidirectional_4 (Bidirectional)	(32, 192)	304,896
dense_8 (Dense)	(32, 96)	18,528
dropout_4 (Dropout)	(32, 96)	0
dense_9 (Dense)	(32, 5)	485

```
Total params: 5,573,429 (21.26 MB)
Trainable params: 323,909 (1.24 MB)
Non-trainable params: 4,601,700 (17.55 MB)
Optimizer params: 647,820 (2.47 MB)
```

Architecture of the final model



True label \ Predicted label	0	1	2	3	4
0	63	157	11	45	3
1	48	387	44	142	12
2	15	159	39	164	12
3	2	91	23	332	62
4	0	27	3	213	156

Confusion matrix of the final model

3. BERT

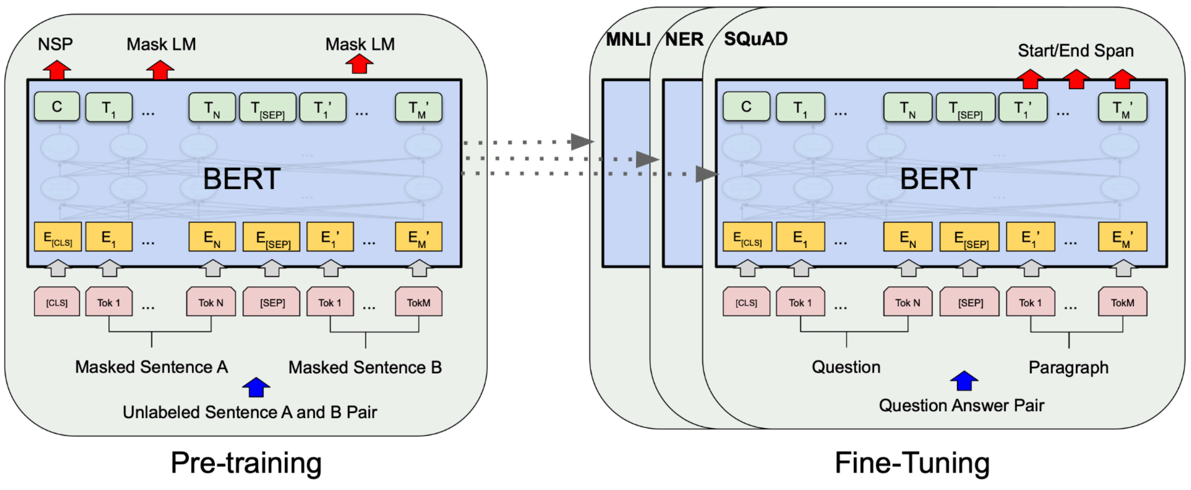
BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained transformer-based language model developed by Google.It revolutionized the field of NLP by introducing a bidirectional training approach, which enables the model to consider the context of a word based on both its left and right surroundings. This contextual understanding significantly improved the accuracy and effectiveness of language understanding tasks.

BERT was trained using the masked language modeling (MLM) and next sentence prediction (NSP) objectives. MLM involves randomly masking some tokens in a sentence and training the model to predict these masked tokens, thereby teaching it to understand the relationships between words in a sentence. NSP, on the other hand, teaches the model to predict whether two sentences logically follow each other, enhancing its ability to understand sentence-level relationships.

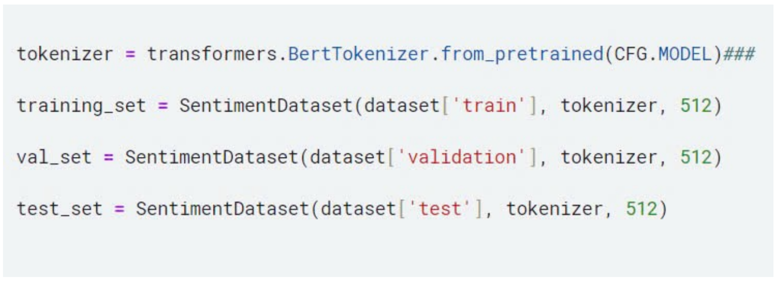
The key innovation in BERT lies in its use of the Transformer architecture, which relies on the self-attention mechanism to process input text in parallel, rather than sequentially. This design allows BERT to capture complex dependencies in text, making it particularly effective for tasks like sentiment analysis, question answering, named entity recognition, and more.

While BERT sets new benchmarks for accuracy on a variety of NLP tasks, its architecture is computationally intensive. With hundreds of millions of parameters, fine-tuning BERT for specific tasks often requires significant computational resources, making it less accessible for smaller-scale applications.

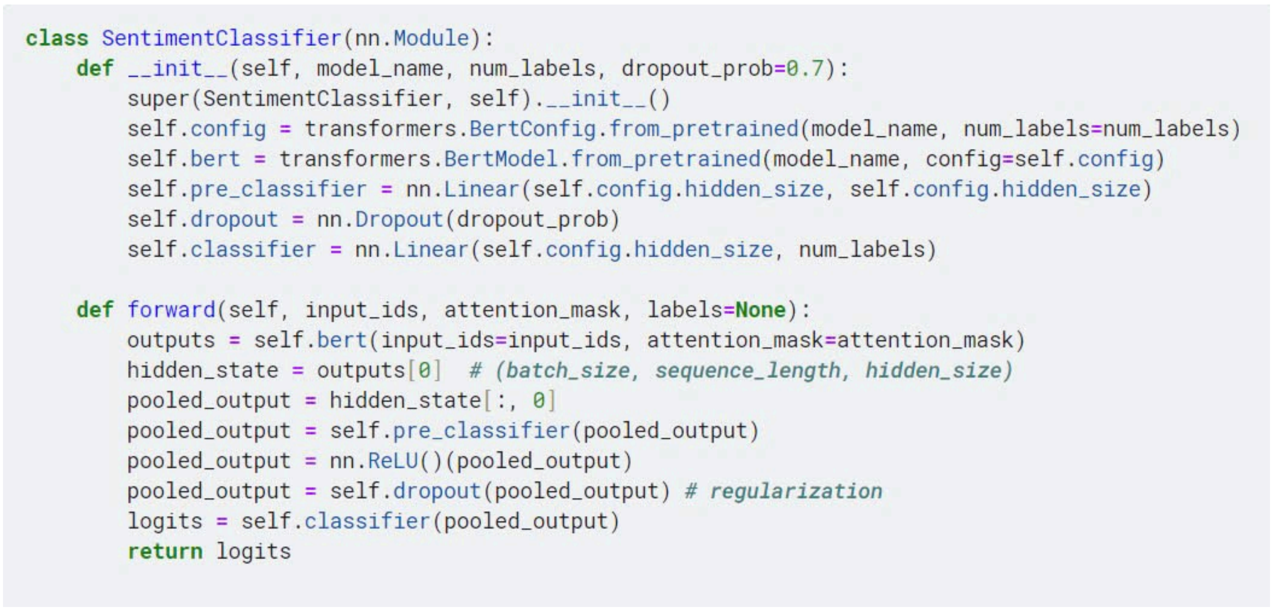
Below is the architecture of BERT:



In this project, we utilize the Bert pretrained model for tokenizing text data, converting tokens to embeddings and processing the input through transformer layers. For fine-tuning, we use two fully connected layers applying a dropout method to do the classification tasks.



```
tokenizer = transformers.BertTokenizer.from_pretrained(CFG.MODEL)###

training_set = SentimentDataset(dataset['train'], tokenizer, 512)

val_set = SentimentDataset(dataset['validation'], tokenizer, 512)

test_set = SentimentDataset(dataset['test'], tokenizer, 512)
```



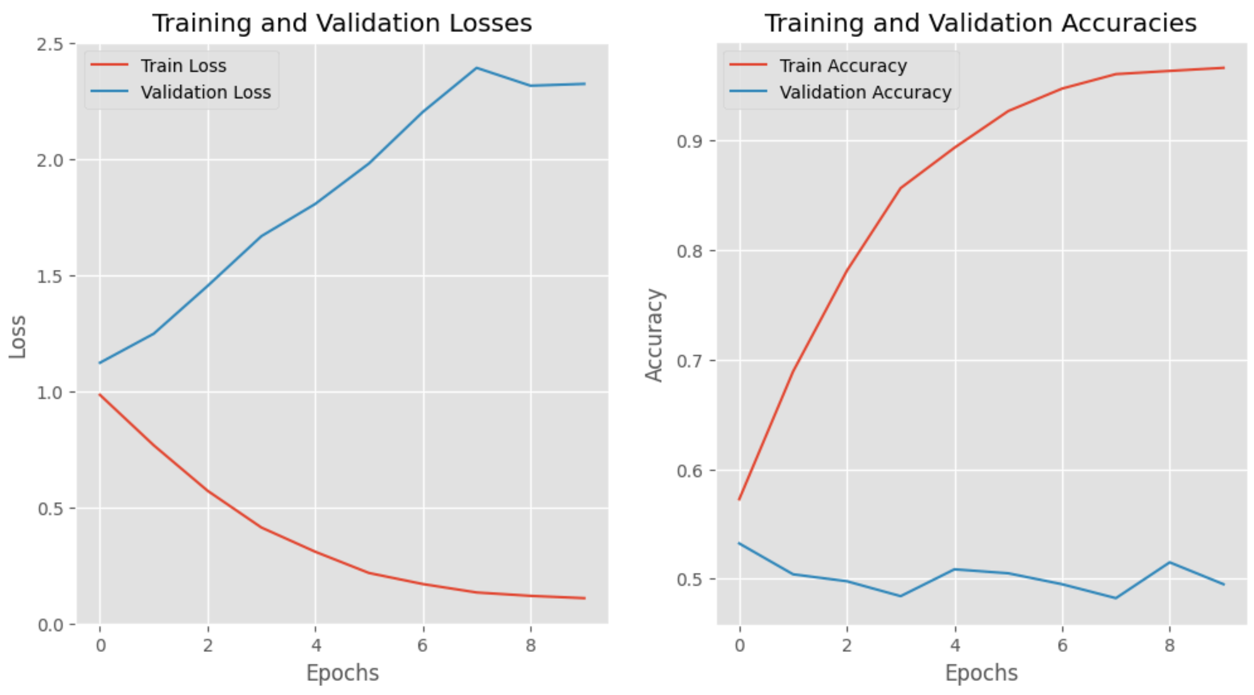
```
class SentimentClassifier(nn.Module):
    def __init__(self, model_name, num_labels, dropout_prob=0.7):
        super(SentimentClassifier, self).__init__()
        self.config = transformers.BertConfig.from_pretrained(model_name, num_labels=num_labels)
        self.bert = transformers.BertModel.from_pretrained(model_name, config=self.config)
        self.pre_classifier = nn.Linear(self.config.hidden_size, self.config.hidden_size)
        self.dropout = nn.Dropout(dropout_prob)
        self.classifier = nn.Linear(self.config.hidden_size, num_labels)

    def forward(self, input_ids, attention_mask, labels=None):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        hidden_state = outputs[0]  # (batch_size, sequence_length, hidden_size)
        pooled_output = hidden_state[:, 0]
        pooled_output = self.pre_classifier(pooled_output)
        pooled_output = nn.ReLU()(pooled_output)
        pooled_output = self.dropout(pooled_output) # regularization
        logits = self.classifier(pooled_output)
        return logits
```

We implement the bert-base-uncased model with the configuration:

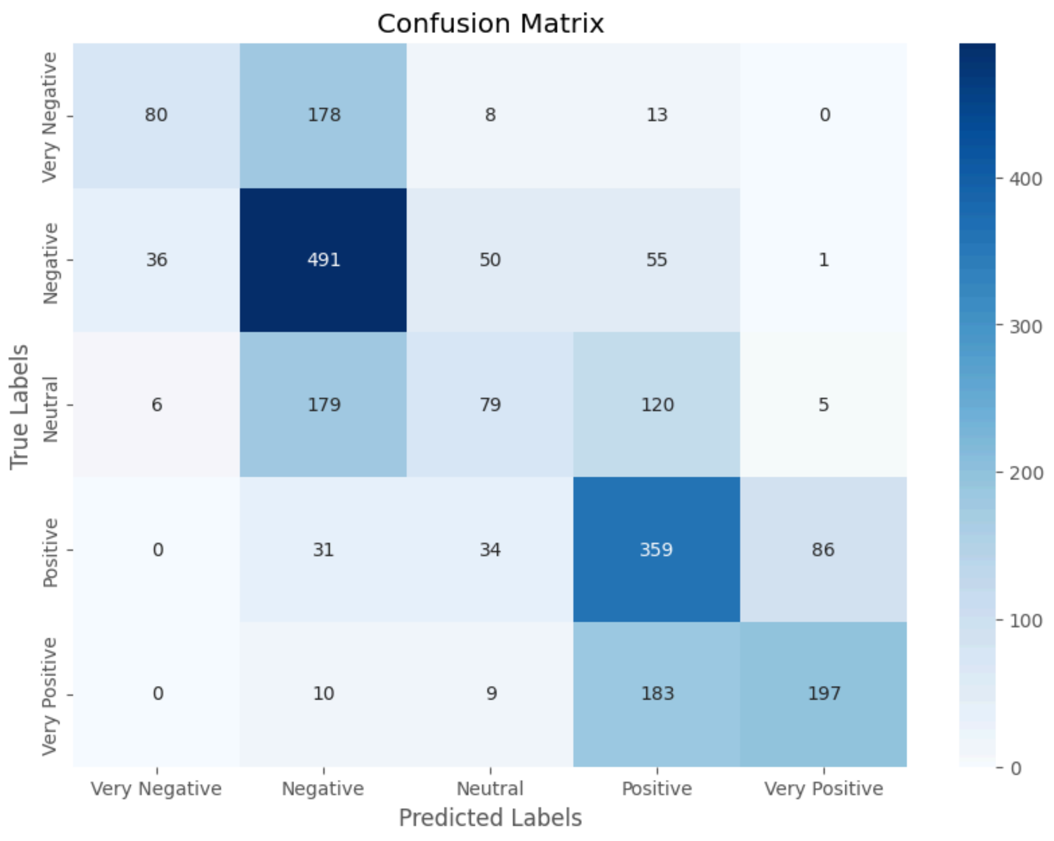
- Model Type: Transformer-based model, based on the BERT architecture.
- Pretrained on: English text.
- Hidden Size: 768
- Number of Layers: 12
- Number of Attention Heads: 12
- Intermediate Size (Feed-forward): 3072
- Max Sequence Length: 512
- Hidden Dropout Probability: 0.1
- Number of Parameters: 11M

We use optuna for hyperparameter tuning. Learning rates are searched from range 1x10^-5 to 3 x 10^-5, with batch size candidates of [16,32,64]. Each trial was run for 5 epochs. After the search, we got the learning rate of 0.00044313179087 and batch size of 16.
The following table shows the best result of our bert model:

	Accuracy	Precision	Recall	F1-Score
10 epochs	0.55	0.55	0.55	0.52



Training and validation loss/accuracy over epochs



True \ Predicted	Very Negative	Negative	Neutral	Positive	Very Positive
Very Negative	80	178	8	13	0
Negative	36	491	50	55	1
Neutral	6	179	79	120	5
Positive	0	31	34	359	86
Very Positive	0	10	9	183	197

Confusion matrix of BERT

The confusion matrix indicates that the model performs well in distinguishing emotions at opposite ends of the spectrum, as there are no misclassifications between very negative and very positive. However, the primary challenge lies in the model's difficulty in differentiating between neutral, positive, and negative sentiments, as well as between positive and very positive or negative and very negative sentences.

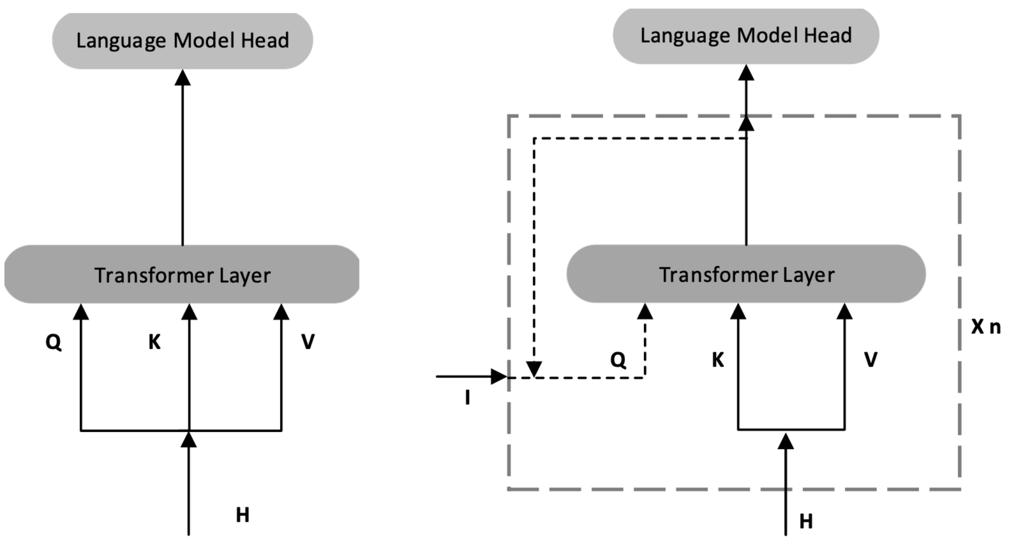
4. DeBERTa

DeBERTa (Decoding-enhanced BERT with Disentangled Attention) is a transformer-based language model that advances the architecture of its predecessor, BERT, by addressing key limitations in its representation and attention mechanisms. While BERT combines word content and positional information into a single vector, DeBERTa takes a more refined approach by separating these components, enabling the model to capture nuanced relationships in text more effectively.

DeBERTa's innovations are designed to improve both contextual understanding and generalization. By disentangling word and positional embeddings, and by introducing an enhanced decoding mechanism during masked language modeling (MLM), DeBERTa achieves state-of-the-art performance on a variety of NLP tasks. These enhancements are particularly valuable for tasks that require fine-grained contextual analysis, such as sentiment classification or syntactic understanding.

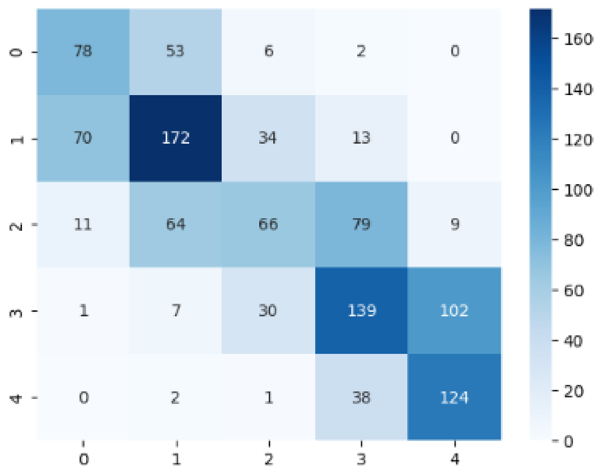
Disentangled attention. Unlike BERT where each word in the input layer is represented using a vector which is the sum of its word (content) embedding and position embedding, each word in DeBERTa is represented using two vectors that encode its content and position, respectively, and the attention weights among words are computed using disentangled matrices based on their contents and relative positions, respectively. This is motivated by the observation that the attention weight of a word pair depends on not only their contents but their relative positions. For example, the dependency between the words "deep" and "learning" is much stronger when they occur next to each other than when they occur in different sentences.

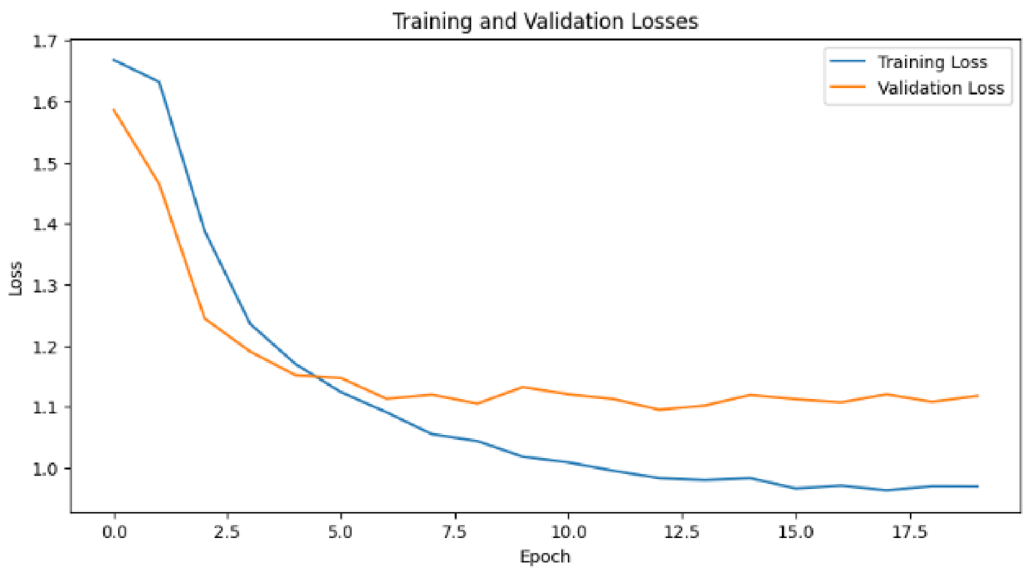
Enhanced mask decoder. Like BERT, DeBERTa is pre-trained using masked language modeling (MLM). MLM is a fill-in-the-blank task, where a model is taught to use the words surrounding a mask token to predict what the masked word should be. DeBERTa uses the content and position information of the context words for MLM. The disentangled attention mechanism already considers the contents and relative positions of the context words, but not the absolute positions of these words, which in many cases are crucial for the prediction. Consider the sentence "a new store opened beside the new mall" with the italicized words "store" and "mall" masked for prediction. Although the local contexts of the two words are similar, they play different syntactic roles in the sentence. (Here, the subject of the sentence is "store" not "mall," for example.) These syntactical nuances depend, to a large degree, upon the words' absolute positions in the sentence, and so it is important to account for a word's absolute position in the language modeling process. DeBERTa incorporates absolute word position embeddings right before the softmax layer where the model decodes the masked words based on the aggregated contextual embeddings of word contents and positions.



(a) BERT decoding layer (b) Enhanced Mask Decoder

Structure of DeBERTa



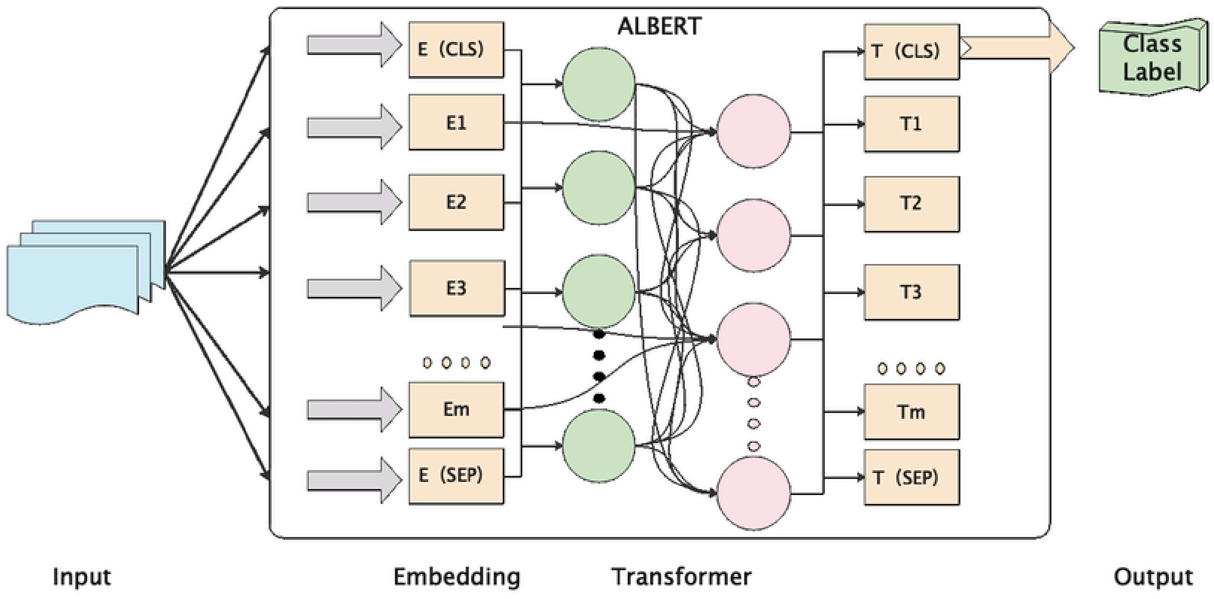


5. ALBERT (A little BERT)

ALBERT (A little BERT) is a Transformers pretrained model proposed by Google AI researchers, introduced in 2019 for self-supervised learning of language representations following the original BERT model. The model was trained on the English language using a masked language modelling (MLM) objective. The main goal of ALBERT is to improve the training and results of BERT architecture with a smaller number of parameters.

ALBERT builds upon the BERT architecture but introduces key innovations to reduce the number of parameters and computational overhead while maintaining high performance on natural language processing tasks. These modifications make ALBERT an efficient alternative to BERT, particularly for large-scale applications with limited computational resources.

ALBERT's architecture is as follows:



ALBERT architecture

There are 3 main points that differs ALBERT from BERT:

1) Factorized Embedding Parametrization: In BERT, the size of the embedding matrix grows linearly with the size of the vocabulary and hidden dimension. This may lead to a large number of parameters, especially with large vocabularies. On the other hand, ALBERT factorizes the embedding layer into two smaller matrices:
   +) A vocabulary embedding matrix: vocabulary size x embedding size.

   +) A projection hidden matrix: embedding size x hidden size.

   This reduces the number of parameters significantly while retaining the model's representation capability.

2) Parameter sharing across Layers: In BERT,each Transformer layer has its own set of parameters, leading to redundancy as the same semantic transformations are applied in multiple layers. While

ALBERT shares parameters across all Transformers layers including self-attention parameters and feedforward network parameters.

3) Sentence Order Prediction (SOP) objective: BERT uses a Next Sentence Prediction (NSP) objective to model inter-sentence relationships,while ALBERT replaces NSP with Sentence Order Prediction (SOP). It detects whether two consecutive segments of text are in the correct order or swapped. This forces the model to focus on coherence and logical sentence order, improving downstream performance.

To sum up, ALBERT has reduced model size, faster training and a comparable performance to the original BERT. When applying to a relatively small dataset of SST-5, we hope that the model can perform competitively against BERT with smaller training time.

Implementation

In the context of this project, we implement the pretrained model from the transformers library

```
from transformers import AlbertTokenizer, AlbertForSequenceClassification
```

Importing the ALBERT pretrained model

```
tokenizer = AlbertTokenizer.from_pretrained("albert-base-v2")
model = AlbertForSequenceClassification.from_pretrained("albert-base-v2", num_labels=5)
```

Tokenizer and model

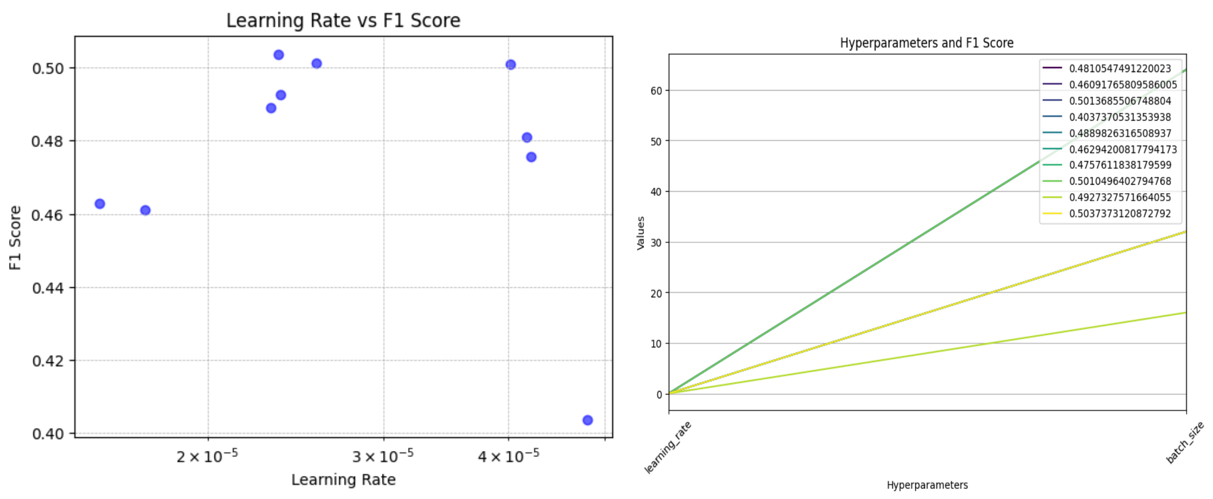
We implement the ALBERT-base-v2 model with the following configuration:

- 12 repeating layers
- 128 embedding dimension
- 768 hidden dimension
- 12 attention heads
- 11M parameters

Hyperparameter tuning is a critical step in optimizing the performance of machine learning models. For ALBERT, we focused on identifying the best combination of **learning rate** and **batch size**, two key hyperparameters that significantly impact the training dynamics and final performance of the model. The primary goal of the hyperparameter tuning process was to maximize the weighted F1 score on the validation dataset from the SST-5 task. This metric was chosen because it provides a balanced measure of precision and recall across all classes, making it particularly suitable for sentiment classification, where class imbalance may occur. We use *optuna* to make our hyperparameter tuning process possible. Learning rates are searched from range 1x10^-5 to 5 x 10^-5, with batch size candidates of [16,32,64]. Each trial was run for 3 epochs and validation was performed at the end of each epoch to track performance.

```
best_training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=3,
    per_device_train_batch_size=best_params["batch_size"],
    per_device_eval_batch_size=best_params["batch_size"] * 2,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=best_params["learning_rate"],
    load_best_model_at_end=True,
    logging_dir="./logs",
    logging_steps=100,
    fp16=True,
    save_total_limit=2,
    dataloader_num_workers=4,
)
```

Hyperparameter tuning process results



Results of the hyperparameter tuning process

After conducting the trials, the study identified the following optimal hyperparameters

Best Hyperparameters: {'learning_rate': 2.3532960037330013e-05, 'batch_size': 32}

The best model achieved a strong performance on the validation dataset, with a notable improvement in the weighted F1 score compared to default hyperparameter settings. Using the best hyperparameters identified during tuning, the model was trained for 20 epochs to fully leverage the benefits of optimized learning settings. Validation performance was closely monitored, and the final model was saved for downstream evaluation on the test set. After 20 epochs, the training loss seems to be too small around 0.01 so we stop there. Results for training after 20 epochs

	Accuracy	Precision	Recall	F1-Score
20 epoch	0.5049954	0.5095450	0.5049954	0.496458

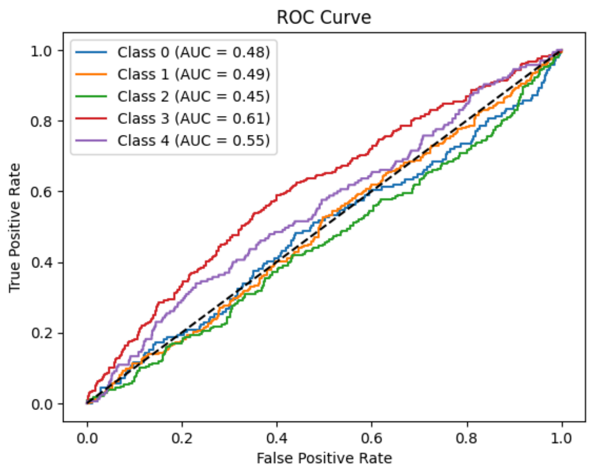
Results of albert-base-v2 after hyperparameter tuning

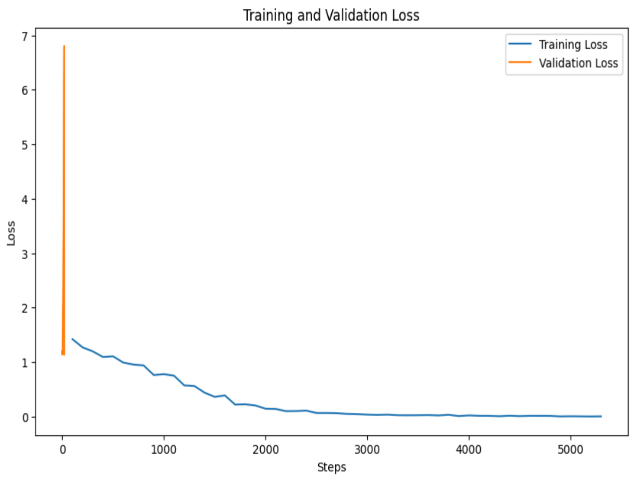
The model “albert-large v2” an extended version of albert-base was also examined the results

	Accuracy	Precision	Recall	F1-Score
20 epoch	0.416893	0.481523	0.416893	0.3691776

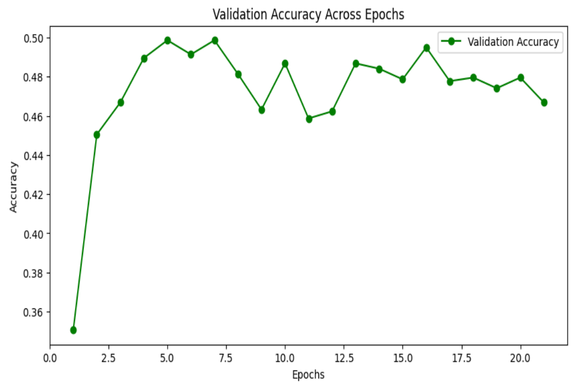
Results of albert-large-v2 after hyperparameter tuning

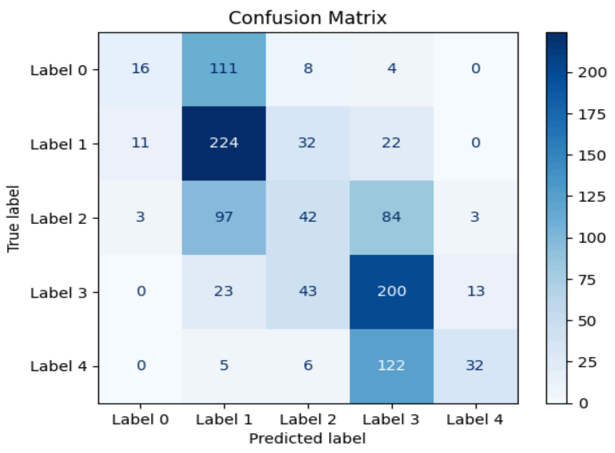
Obviously the albert-base-v2 is the go-to model due to faster training time and slightly better results, more analysis is done on this model





ROC curve and training/validation loss





Validation accuracy and confusion matrix

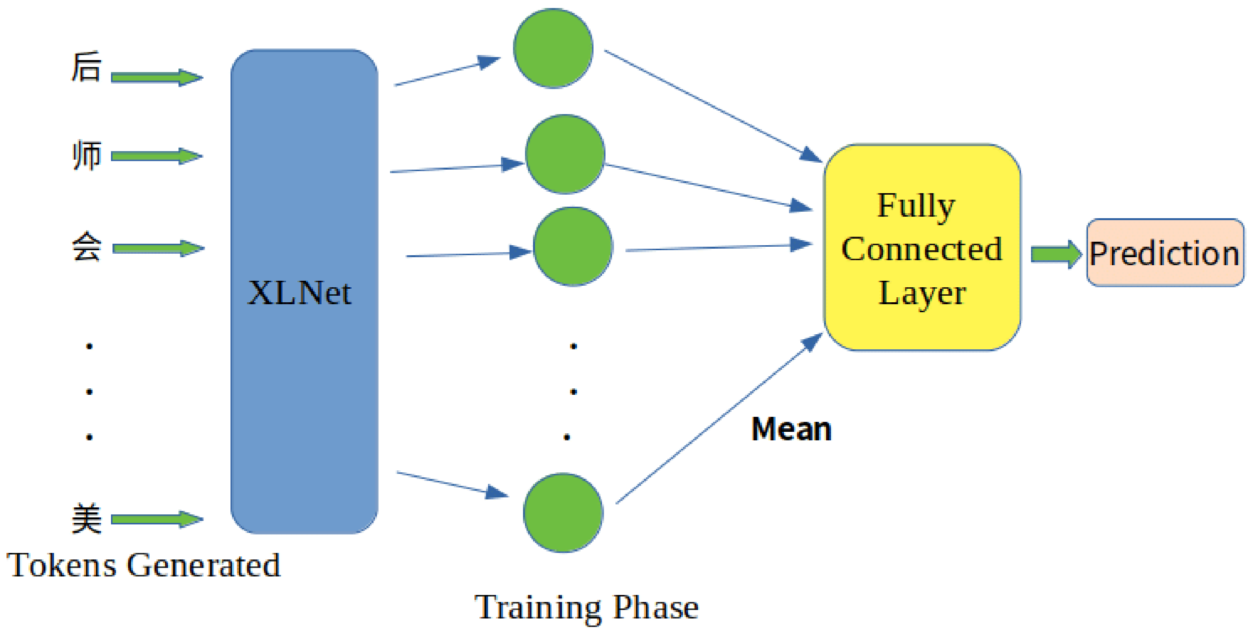
The confusion matrix shows that the model can distinguish opposites emotion pretty well as there is no misprediction of negative - positive or very negative - very positive. The main issue is that the model got mistaken when predicting the neutral - positive - negative and positive - very positive, negative - very negative sentences as it is quite hard to draw a clear line to distinguish those two groups of sentiments.

6. XLNet

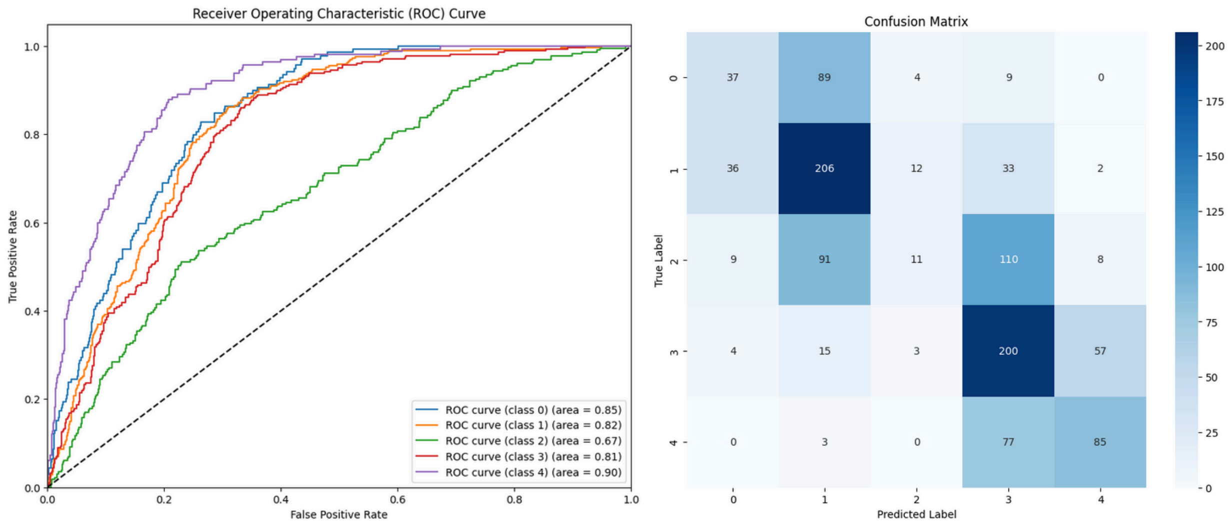
XLNet (eXtreme Language Model Network) is a transformer-based model that extends and improves upon the limitations of BERT through its innovative **permutation language modeling** objective. While BERT relies on masked language modeling (MLM), which predicts randomly masked tokens using their bidirectional context, XLNet introduces a permutation-based approach that considers all possible orders of a given input sequence. This enables the model to capture bidirectional dependencies without the drawbacks of artificially masked tokens, resulting in a more comprehensive understanding of context and relationships in text.

XLNet's architecture is built on **Transformer-XL**, an extension of the original transformer model designed to handle longer sequences through a segment-level recurrence mechanism. This allows XLNet to process and model long-range dependencies more efficiently while maintaining contextual coherence across segments. The model also integrates relative positional encoding, which further enhances its ability to understand relationships between words over extended contexts, a critical feature for complex tasks like sentiment classification or document-level reasoning.

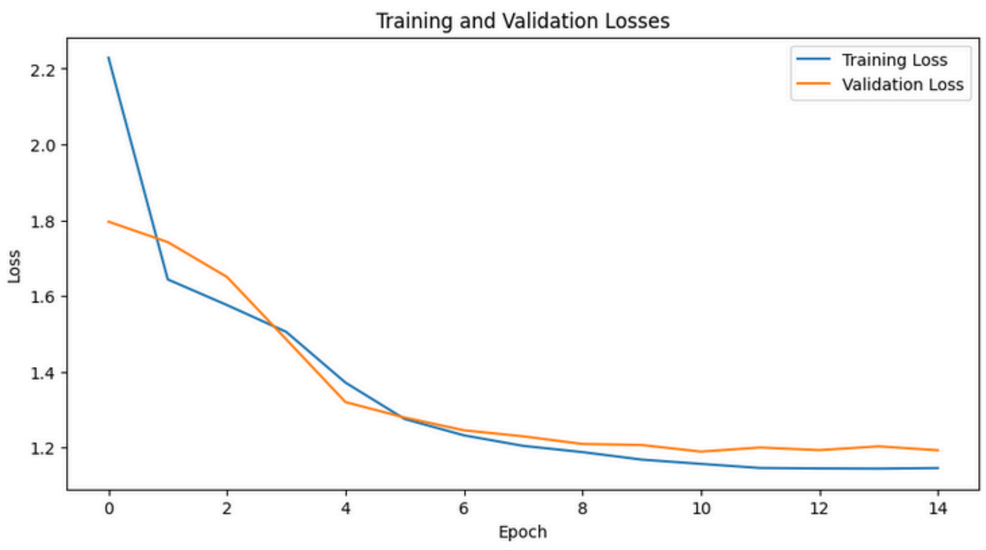
XLNet's unique permutation-based objective ensures that the model can leverage bidirectional context without the pretrain-finetune discrepancy associated with masked tokens in MLM. By combining the strengths of Transformer-XL with a novel training paradigm, XLNet achieves state-of-the-art performance across numerous natural language processing benchmarks, making it particularly effective for tasks that require nuanced understanding of word order and interdependencies.



Architecture of XLNet



ROC curve and confusion matrix



Training and Validation losses over epochs of XLNet

The results of training XLNet on the SST-5 dataset demonstrate the model's ability to capture extreme sentiment classes effectively but reveal challenges in distinguishing between subtle or intermediate sentiment levels. The **ROC curves** show strong performance for the most positive sentiment class (AUC = 0.90), while neutral and adjacent classes exhibit lower AUC values, indicating difficulty in separating fine-grained sentiments. The **confusion matrix** confirms this observation, with significant misclassifications occurring between neighboring sentiment classes. This suggests that the nuanced nature of the SST-5 dataset, where sentiment differences are subtle and context-dependent, poses challenges for even advanced transformer-based models like XLNet.

The **training and validation loss curves** indicate successful model optimization, as both losses steadily decline and converge without clear signs of overfitting. However, the slight gap between the training and validation loss highlights potential challenges, such as overfitting on complex patterns in the training data, insufficient generalization, or the need for better regularization strategies. Additionally, the imbalanced class distribution in SST-5, where neutral or intermediate classes may dominate or overlap semantically, exacerbates the difficulty in training. This project underscores the complexities of fine-grained sentiment analysis, particularly for deep learning models tasked with learning subtle distinctions in natural language.

Part IV: Results and Comparisons

1. Metric evaluation

In the context of this project, we primarily use 4 popular metrics of accuracy, precision, recall and F1-score.

- TP (True Positive): The prediction is correct and the actual value is positive.
- FP (False Positive): The prediction is wrong and the actual value is positive.
- TN (True Negative): The prediction is correct and the actual value is negative.
- FN (False Negative): The prediction is wrong and the actual value is negative.
- $Accuracy$: Measures the proportion of correct predictions made by the mode.

$$\boxed{Accuracy \; = \; \frac{(TP + TN)}{(TP + FP + TN + FN)}}$$

- *Precision*: Measures the proportion of true positive predictions among all positive predictions made by the model,

$$\boxed{Precision \; = \; \frac{TP}{(TP + FP)}}$$

- *Recall*: Measures the proportion of true positive predictions among all actual positive instances

$$\boxed{Recall \; = \; \frac{TP}{(TP + FN)}}$$

- *F1-Score:* Measures the harmonic mean of precision and recall

$$\boxed{F1 \; = \; \frac{2 \times Precision \times Recall}{Precision + Recall}}$$

In the context of our project, we target to gain the highest score in 2 metrics Accuracy and F1 Score. Accuracy not only provides a straightforward measure of the model's performance, gives a clear picture of how well the model is performing overall but also metric for model's benchmark comparisons. F1-Score is used to ensure all classes are appropriately considered, even if they have different representation sizes. It also provides a deeper insight into the model's ability to correctly classify minority and majority classes alike. By targeting high performance in both metrics, we ensure that the model is not only accurate but also fair and effective in handling data.

2.Model Selection

Model selection is a crucial step in ensuring that the most effective and efficient model is used for this sentiment analysis task.

Loss function: We chose to use Cross-entropy loss function not only because it is one of the most used loss functions, but it proves to be a suitable choice for this task, as it provides great insight in how the model works with different labels

$$\boxed{H = \sum p(x) log(p(x))}$$

Cross entropy loss function formula

Weight initialization: Xavier. The model weights are initialized using the **Xavier Initialization** method (also known as Glorot Initialization). This technique is commonly used in neural networks and helps to address the vanishing or exploding gradient problem during the training process. Xavier initialization ensures that the weights are scaled appropriately, so the signal propagates effectively throughout the network.

Optimizer: AdamW. We employ the **AdamW (Adam with Weight Decay)** optimizer to train the model. AdamW is an extension of the Adam optimizer that decouples weight decay from the optimization step, making it more suitable for tasks where regularization through weight decay is important. AdamW adapts the learning rate for each parameter based on estimates of first and second moments of the gradients, allowing the model to converge faster and with greater stability.

Regularization: Dropout. To prevent overfitting and enhance the generalization capability of the model, we incorporate **Dropout** as a form of regularization. Dropout is applied during training, particularly in the attention layers and the feedforward layers of the transformer architecture. This technique randomly disables a fraction of neurons during each forward pass, forcing the model to learn more robust features. By preventing the model from relying too heavily on any single neuron, dropout helps improve its ability to generalize to unseen data.

3. Results and Comparisons

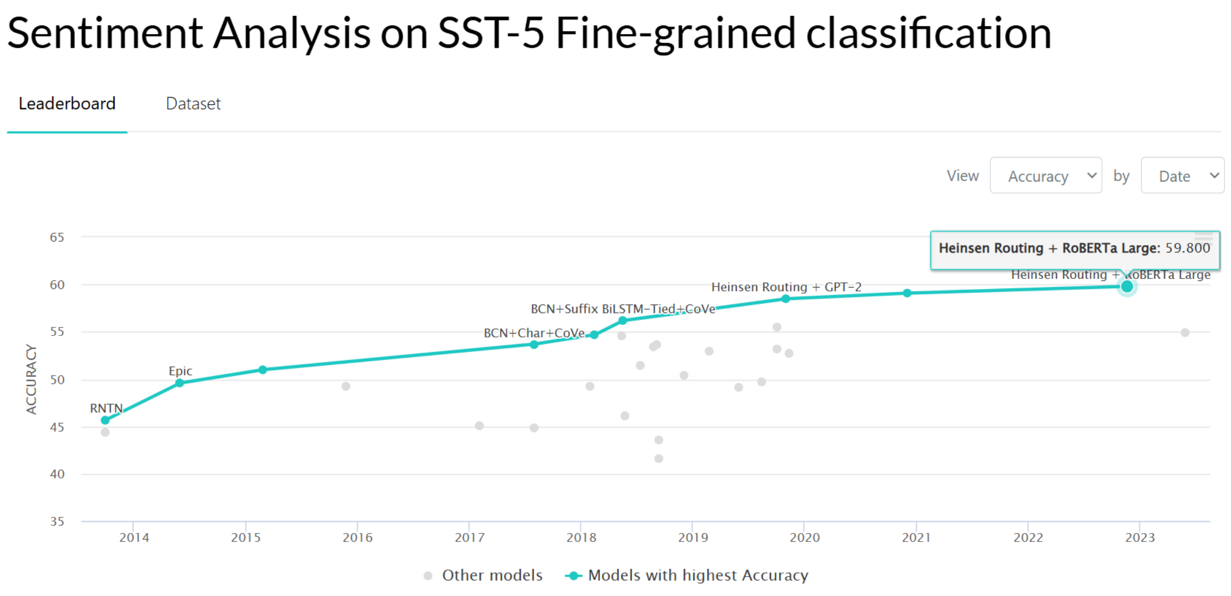
The following table shows the final results of our approaches:

Model	Accuracy	Precision	Recall	F1
CNN	0.4145	0.4347	0.4145	0.3618
LSTM	0.4420	0.4550	0.4420	0.4150
BERT	0.5500	0.5500	0.5500	0.5200
deBerta	0.5377	0.5059	0.5377	0.5133
ALBERT	0.5049	0.5095	0.5048	0.4964
Xlnet	0.4896	0.4664	0.4896	0.4388

From the table of results, it is obvious that the choice of model significantly impacts the performance of sentiment analysis. Among the models evaluated, **BERT** demonstrated the highest overall performance, achieving the best balance across Accuracy (0.55), Precision (0.55), Recall (0.55), and F1-score (0.52). This indicates that BERT effectively captures contextual information in text, making it well-suited for sentiment analysis tasks. The **deBERTa** model also performed competitively, achieving an F1-score of 0.5133, slightly lower than BERT but still robust. This suggests that transformer-based architectures generally excel in tasks requiring nuanced text understanding.In contrast, the **CNN** model had the lowest performance across all metrics, with an Accuracy of 0.29 and an F1-score of 0.324. This indicates that CNNs, while effective in tasks such as image recognition, may not be the most suitable for sentiment analysis due to their limited ability to capture long-range dependencies in text. The **LSTM** model performed moderately, with an Accuracy of 0.442 and an F1-score of 0.415, showcasing its ability to handle sequential data better than CNN but falling short compared to transformer-based models. **ALBERT** and **XLNet** showed similar performance, with ALBERT slightly outperforming XLNet in terms of F1-score (0.496 compared to 0.4388). However, both models lagged behind BERT and deBERTa, possibly due to differences in architecture or training strategies.

Overall, the results suggest that transformer-based models, particularly BERT, are the most effective for sentiment analysis in this evaluation.

It should be noted that the accuracy of the SST-5 dataset to our knowledge is not relatively high. The following is the results of some SOTA models (State-of-the-art models) which are publicly shown on paperswithcode website:



Credit: results of SOTA models on paperswithcode

It can be seen that the model which has the highest accuracy on the SST-5 model has an accuracy of 59.800. This serves as a comparison of our approaches to some widely publicly known models.

4. Future applications

While the current model demonstrates promising results for sentiment analysis, there is still ample room for further refinement and enhancement. Future work can explore several avenues to improve the model's performance and broaden its applications:

+)Exploring Larger Models: While the current model leverages convolutional layers for feature extraction, the incorporation of more sophisticated architectures like Transformer-based models (e.g., BERT, GPT, or ALBERT) may provide richer contextual representations of the input text.

+)Transfer Learning: Leveraging transfer learning by fine-tuning pre-trained models on domain-specific datasets could improve the model's performance in specialized applications.

+)Fine-Tuning Hyperparameters: The current study focused on optimizing key hyperparameters, such as learning rate and batch size. However, further fine-tuning other hyperparameters, such as regularization strength, dropout rates, and layer architecture, could yield even more robust models. Techniques such as Bayesian optimization or more advanced search strategies could be employed to explore a broader range of configurations.

+)Real-Time Applications: There is a significant opportunity for applying sentiment analysis models in real-time applications, such as monitoring customer feedback, social media sentiment, or live event sentiment tracking. Implementing the model in a real-time feedback loop could enhance decision-making processes for businesses or public organizations.

In conclusion, while this model serves as a strong foundation for sentiment analysis, there are several exciting directions for future improvements. By leveraging more advanced architectures, expanding the model's capabilities, and incorporating domain-specific knowledge, we can continue to enhance its accuracy, generalization, and applicability to a wide range of tasks and industries.

5. References

1) J. Li, "Fine-Grained Sentiment Analysis with a Fine-Tuned BERT and an Improved Pre-Training BERT," 2023 IEEE International Conference on Image Processing and Computer Applications

(ICIPCA), Changchun, China, 2023, pp. 1031-1034, doi: 10.1109/ICIPCA59209.2023.10257673. keywords: {Deep learning;Analytical models;Sentiment analysis;Network topology;Image processing;Transfer learning;Text categorization;Fine-grained sentiment analysis;natural language understanding;deep learning;BERT;Transformer},
2) Brian Cheang, “Language Representation Models for Fine-Grained Sentiment Classification”, Wei, David Kogan, Howey Qiu, Masud Ahmed
3) Z. Ding, Y. Qi and D. Lin, "Albert-based sentiment analysis of movie review," 2021 4th International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE), Changsha, China, 2021, pp. 1243-1246, doi: 10.1109/AEMCSE51986.2021.00254.
4) Y. Zhang, C. Liu and W. Liu, "A Study on Sentiment Analysis of Movie Reviews based on ALBERT-TextCNN-HAN," 2023 3rd International Symposium on Computer Technology and Information Science (ISCTIS), Chengdu, China, 2023, pp. 758-763, doi: 10.1109/ISCTIS58954.2023.10212999.