Deep Learning Documentation

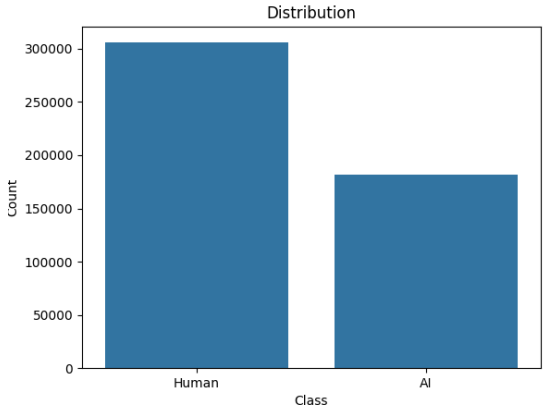
Mihalcea Dragos Stefan, 507

For this project I'm presenting a binary classification task on the *AI vs Human Text* dataset

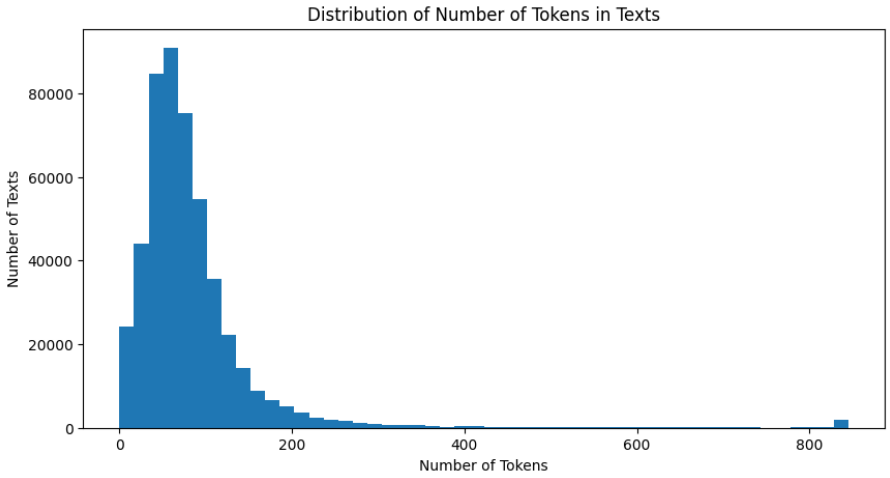
I decided to implement 3 different deep learning models: CNN1D, Autoencoder and BERT with the same text processing but differences in sampling due to longer training time.

Before getting into the models and their hyperparameter tuning, in the next paragraphs I am going to present some observations regarding the dataset itself and some preprocessing I did.

The Dataset:



The dataset contains 487k sample texts with 1/3 being AI generated and 2/3 being Human texts as shown in the picture down below. Because of this imbalance in the dataset, we choose to always sample 50% of each class.



A deeper look into this data after preprocessing reveals that most texts have less than 300 tokens while also exists a group of outliers with more than 800. As we didn't exclude these outliers before training this might be one of the key reasons for some of the model's bad performances.

Text preprocessing:

During preprocessing we firstly extract only the first sentence of each text to reduce the input and then we clean the sentence by removing non-alphanumeric characters, extra spaces, and stopwords.

We then sample a certain amount for each method due to time and hardware limitations in the following way:

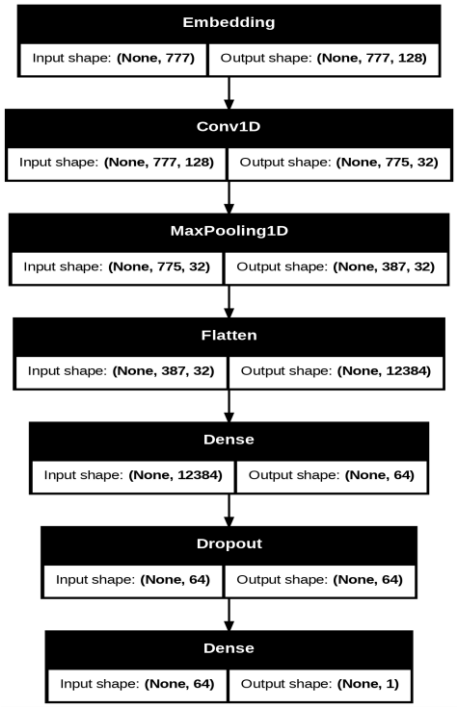
- CNN1D – 300k total samples
- BERT – 25k total samples
- Autoencoder – 5k total samples

After that split the dataset in train 70%, validation 15% and test 15%.

Finally, we tokenize the whole dataset, find vocabulary length and maximum text length and then we pad all texts to reach the maximum length.

CNN 1D Model:

The first model has the following architecture:



This architecture alternates between **convolutional layers** (for feature extraction), **pooling layers** (for downsampling) and **dropout layers** to help prevent overfitting. The **dense layer** at the end combines learned features for final classification.

For this model we also ran a Grid Search with the following parameters over 25 epochs each:

param_grid =

"embedding_size": [32, 64, 128],

"filters": [32, 64, 128],

"dropout_rate": [0.3, 0.5, 0.7],

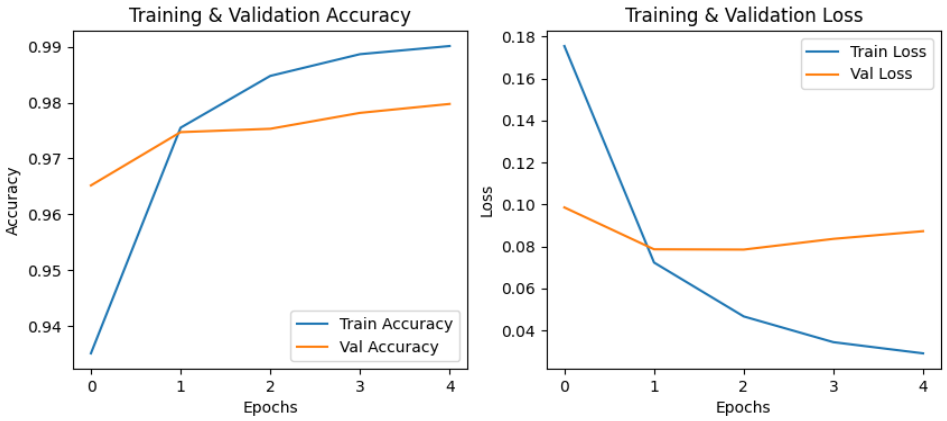
achieving the best validation accuracy of 98.01% with

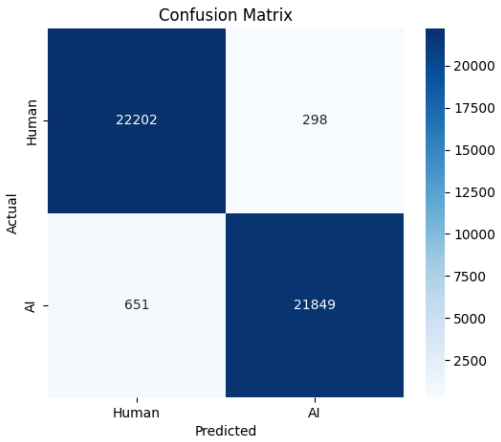
"embedding_size": 128,

"filters": 32,

"dropout_rate": 0.7

This model also returned 97.89% accuracy on test data with the *best parameters.*



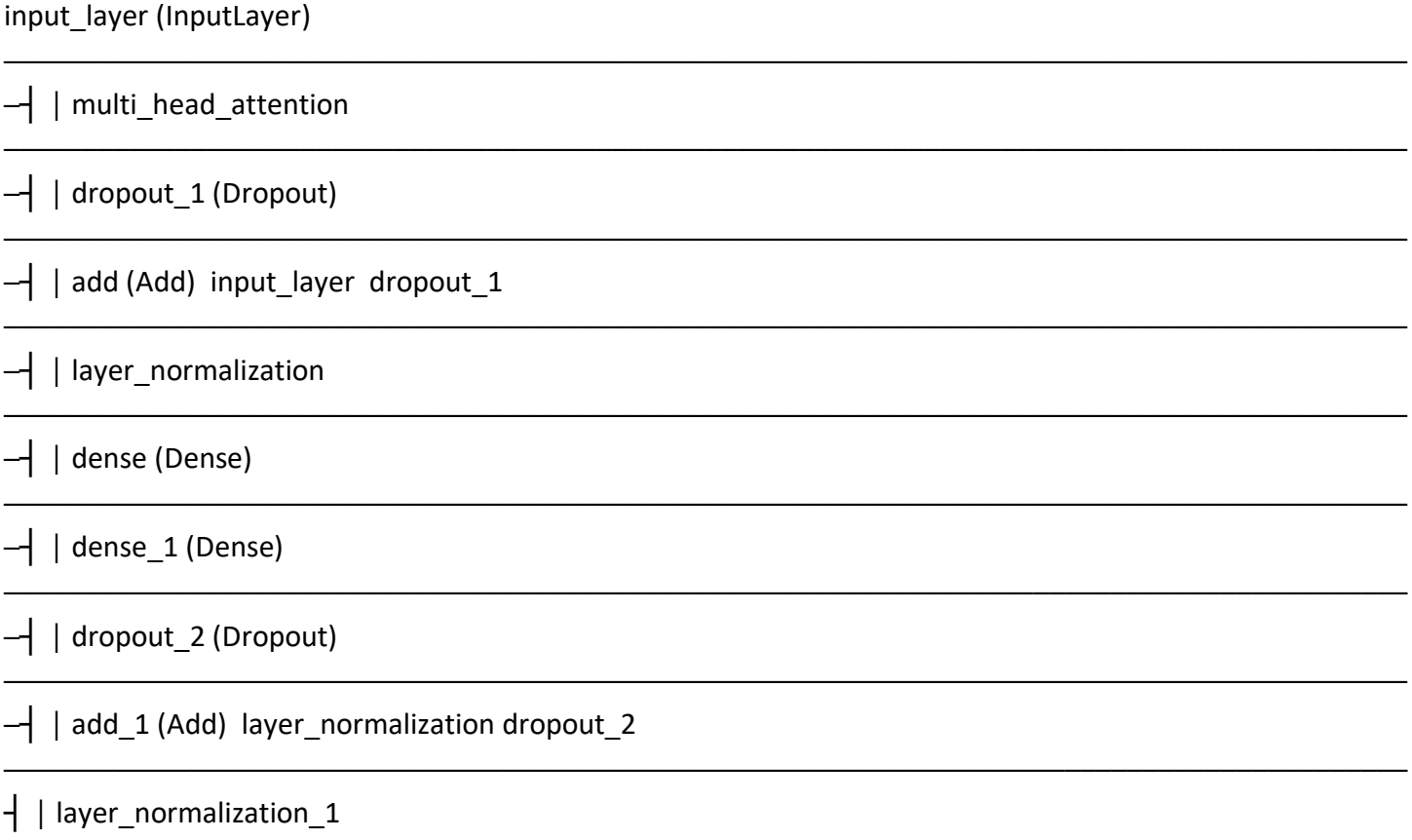


We can also observe in the graphs above that the model is not overfitting and is performing really well in predicting both classes.

BERT Model:

With this model we tried to replicate the architecture of the popular BERT model. The main architecture is looking like this:

And the **TransformerBlock** looks like this:



One thing we did not implement from the original BERT was the Positional Embeddings, here we use only normal embeddings.

For this model we ran a Grid Search with the following parameters over 25 epochs each:

param_grid =

" num_heads ": [2, 4],

"num_blocks ": [2, 4],

" feedforward_dim ": [128, 256],

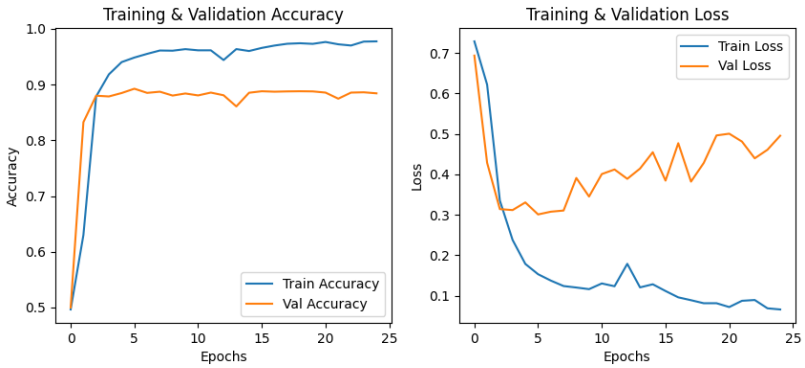
achieving the best validation accuracy of 88.9% with

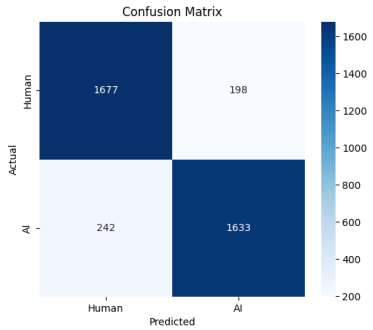
"num_heads": 2,

"num_blocks": 2,

"feedforward_dim": 256

This model also returned 88.26% accuracy on test data with the *best parameters.*





We can observe in the pictures above that even though the model is performing quite well and is representing both classes well, it is overfitting a bit as loss validation is slowly growing after epoch 7.

Autoencoder Model:

For this model we also show 2 architectures:

- The autoencoder itself:

```
input_text = Input(shape=(max_length,))

embedded = Embedding(vocab_size, embedding_size)(input_text)

encoded = LSTM(lstm_size, activation='tanh',
return_sequences=False)(embedded) encoded = BatchNormalization()(encoded)

latent = RepeatVector(max_length)(encoded)


decoded = LSTM(lstm_size, activation='tanh', return_sequences=True)(latent)
output = TimeDistributed(Dense(vocab_size, activation="softmax"))(decoded)
```

- The classification network:

```
encoded_input = Input(shape=(max_length,))

 x = Dense(128, activation="relu")(encoded_input)

x = Dropout(dropout_rate)(x)

 x = Dense(64, activation="relu")(x)

classifier_output = Dense(1, activation="sigmoid")(x)
```

For this model we ran a Grid Search with the following parameters over 25 epochs each:

param_grid =

"embedding_size": [256, 512],

" lstm_size ": [128, 256],

"dropout_rate": [0.3, 0.5],

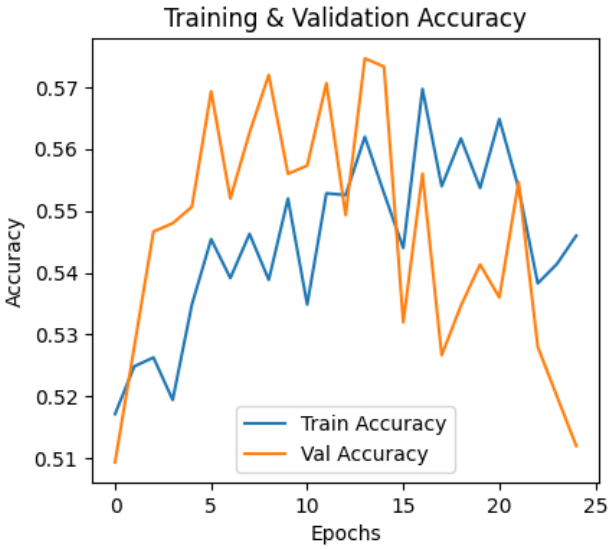
achieving the best validation accuracy of 56.13% with

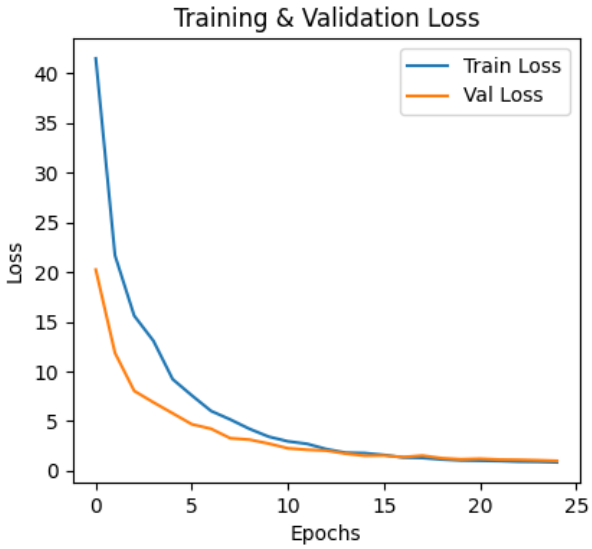
"embedding_size": 256,

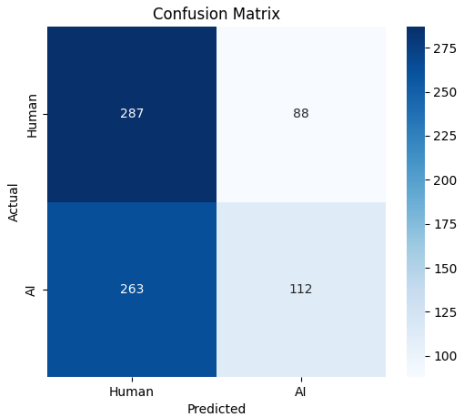
"lstm_size": 256,

"dropout_rate": 0.5,

This model also returned 53.2% accuracy on test data with the *best parameters.*







As observed in the pictures above but also in the results this method performed poorly without being able to learn relevant features. Moreover, it seems the classifier is predicting most examples as Human. This could be due to several factors like model complexity, insufficient data. One thing I observed was that after I trained the autoencoder and used it encoder to process the inputs for the networks number of unique generated vectors was low starting for 6 to 40 on 700 total samples.