# Romanian Sentence Classification
Mihai Matei 511DS

The project was implemented with the latest tensorflow 2.4 and their own keras implementation using a 1070ti Nvidia graphics card. Pandas was used to pre-process the dataset. My own custom ***matmih*** library was used for plotting, statistics and model building.

The optimizer used was weighted Adam from Google's model garden nlp package https://github.com/tensorflow/models/tree/master/official/nlp since I am using transformer models, specifically Google's Bert EncoderStack implementation of self-attention encoder and it is used to train their NLP models. The initial learning rate is 0.003. The batch size used for training is 32 as in their implementation.

The loss used was sparse crossentropy loss – the default loss used for multiclass classification. The loss was weighed with the dataset class distribution.

Additional data, saved weights can be found at https://github.com/glypher/ml_competition

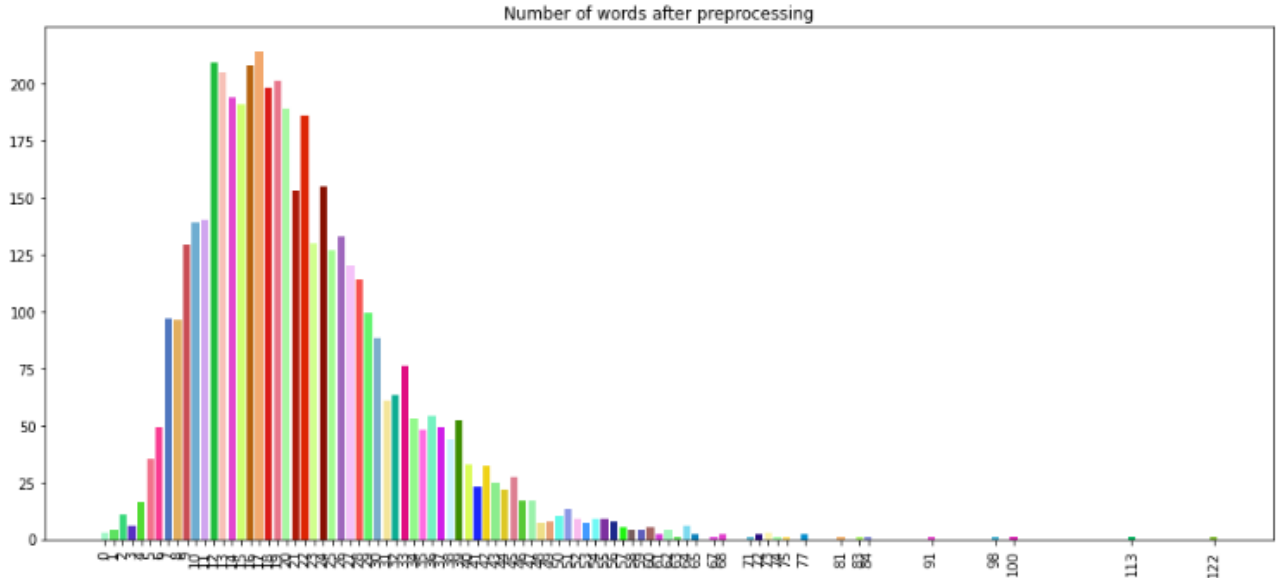
## Model search history

I have tried many approaches to finally implementing a language model and transfer learning classifier trained from scratch using Google's transformer model implementation used in their Bert configuration. Prior to this I used a simple multi dense layer model on the parsed text and achieved an accuracy on the validation data close to 0.7. Then I tried Google's Bert implementation from tensorflow hub using pre-trained cased multi-language model https://tfhub.dev/tensorflow/bert_multi_cased_L-12_H-768_A-12/3 and their preprocessing layer https://tfhub.dev/tensorflow/bert_multi_cased_preprocess/2 by adding a dense 10 class softmax classification layer and achieved 0.84 on the validation original split. But this model only has a maximum of 0.83 on the test set on uploading to the competition. It also had several problems like the sentence token size limitation to 128. Due to the fact that the preprocessing layer had a limited romanian vocabulary the actual pre-processing layer split a romanian word into multiple input tokens so the 128 limit was exceeded by many train sentence samples. I did try to split a larger sentence into smaller sentences but did not optain much better test results.

Hence I moved to implement my own Romanian language model using a simplified version of Bert.

## Text pre-processing

The data was first pre-processed by removing invalid (empty) records from the train set and each sentence (sample) was lowered, tokenized, stemmed using Romanian Snowball stemmer, the punctuation was removd and a vocabulary of word->id was build for all of the data (train+validation+test). This way I constructed a Romanian language composed of only the stemmed words found in all of the sentence. The samples were transformed to word identifier prior to feeding them as input to the models. I ploted the resulted word histogram for all sentences:



## Romanian Language model

To achieve better accuracy results I started to implement my own custom transformer language model using Google's model gardern implementation of the self-attention transformer: https://github.com/tensorflow/models/blob/master/official/nlp/transformer/transformer.py

The configuration for the EncoderStack used uses 2 self-attention transformer layers (Bert uses 12) and the other configuration used were taken from the Google Bert implementation (12 attention heads, with 756 filters) folowed by dense feed forward layers of size 3072.

To train the model I used the unsupervised approach described in the Bert paper. For all of the text preprocessed data (train+validation+test) 15% of each sample is masked and the supervised training method applies cross-entropy loss to predict the masked words. This way in an unsupervised manner according to the sentence class label the model tries to predict the missing words.

To do this first I constructed a tensorflow dataset that for each sample (text sentence) choosed 15% of the words and masked them and for 10% or the masked words (1.5% of the words) replaced them with a random words. Additionaly for each sentence a sample_weight was constructed that contained 1 for each masked word and 0 for the other such that the cross entropy loss only took into account the prediction of the masked words and ignored the other.

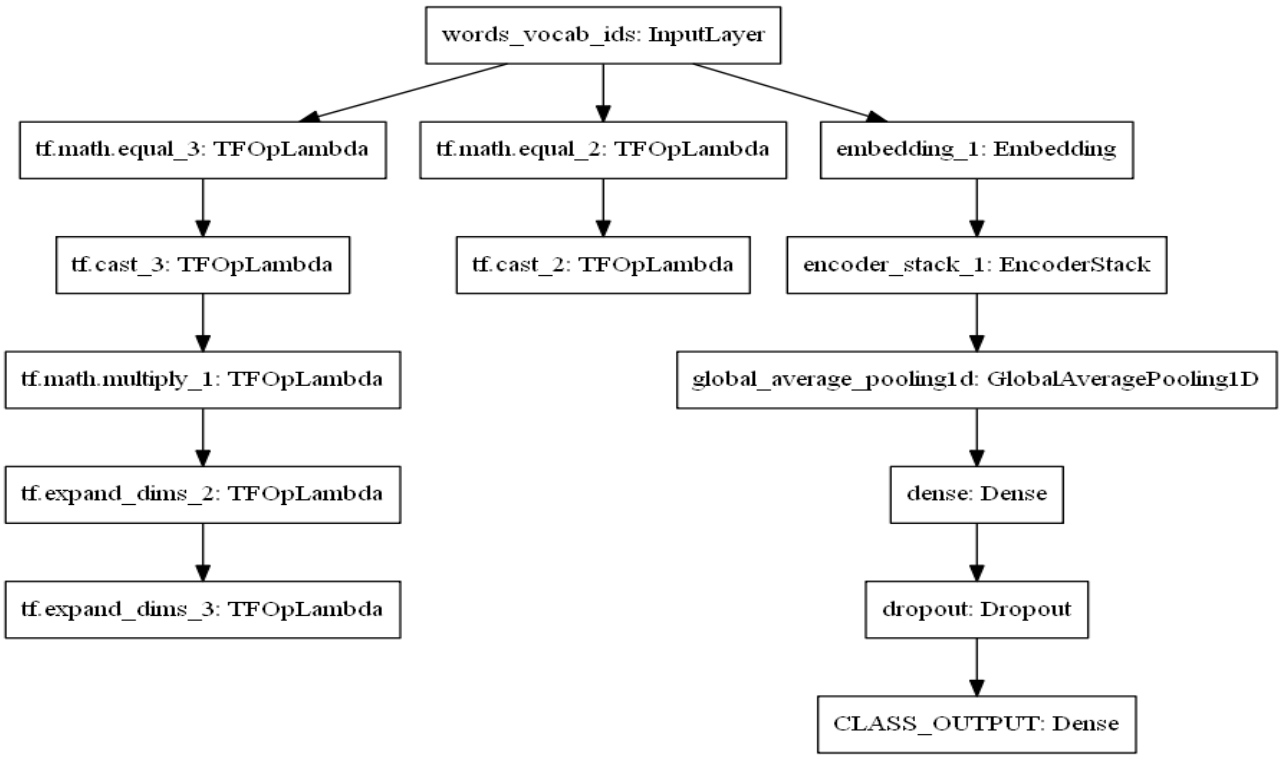
The model contained a trainable keras Embedding layer of size 756, same as the transformer number of filters, so that the model could learn a word embedding of size 756. This a default embedding scheme for NLP deep learning models. The sentence length was set to 128 so that batch can be used and sentences was padded with 0 (the default embedding layer mask word id) until 128. Then an EncoderStack as described above was added folowed by the last classification layer.

The language model tried to do self prediction of the sentence so the last layer contained a [128x6004] softmax output so that each word of the 128 max words would be mapped by the softmax to a probability that is one of the 6004 vocabulary words. The custom loss constructed the cross entropy loss as the sum of the loss only for the masked word, hence the model would somehow learn what word was missing from the sentence. This is the clever unsuprevised approach used by Bert.

The total training time took around 8 hours on an Amazon P3 spot machine and the best accuracy model weights were saved . The will be loaded to do fine tuning of the classification model: https://github.com/glypher/ml_competition/blob/main/weights_nlp/transformer_weights.save

## Classification model

To learned language model weights were loaded into the transfer learning model:

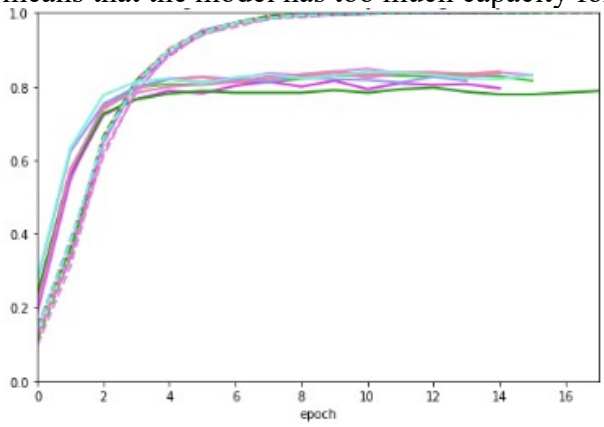


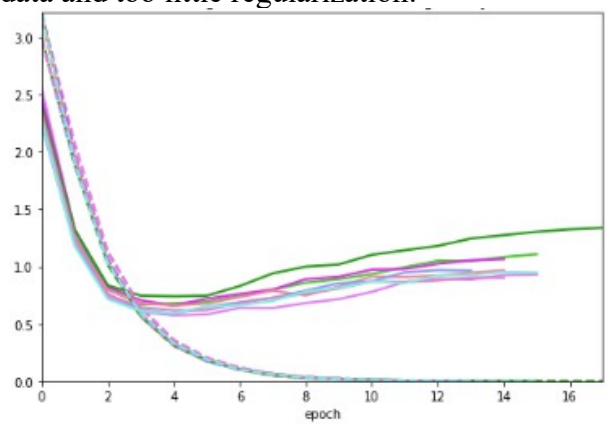
The same model with the trained loaded weights was used except that the last dense layer was removed and some additional layers were added:

LanguageModel (Embedding + EncoderStack(2layers))
GlobalAveragePooling1d
Dense 128 + RELU + Dropout 0.3
Dense 10 + Softmax

The model was trained on the original train-validation split and a 7 fold stratified split on the train+validation data using the same AdamW optimizer with 0.003 learning rate, 32 batch size for a maximum of 60 epochs with early stopping when validation accuracy did not increase for 5 epochs.

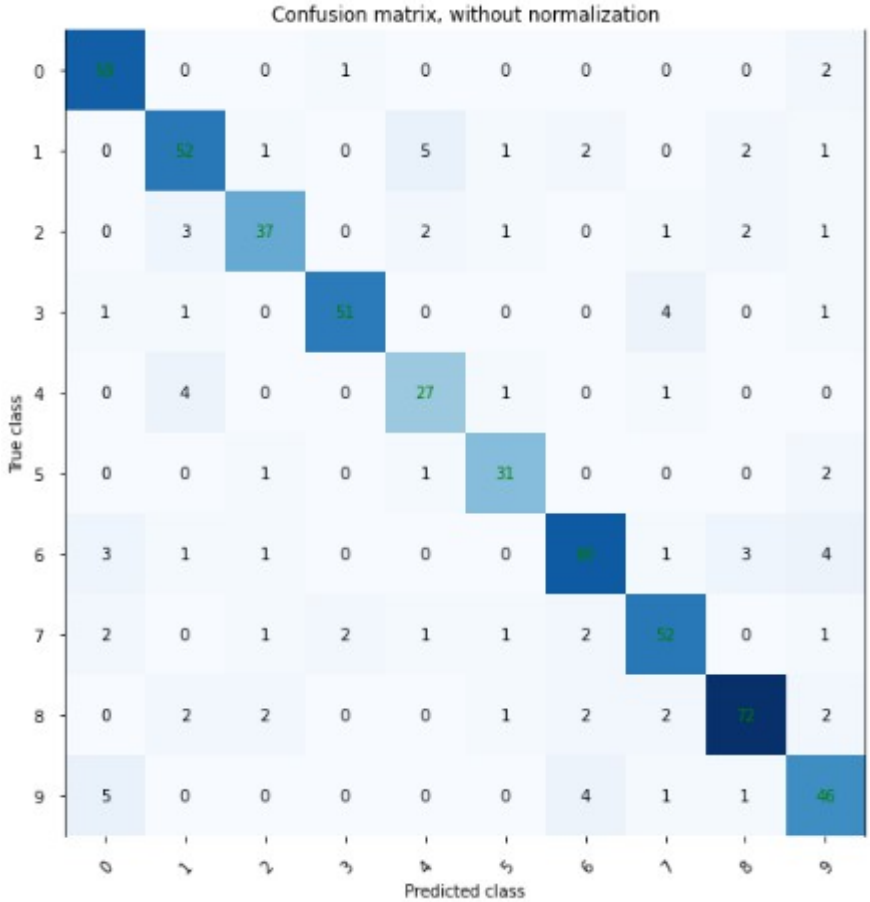
The validation accuracy found was 0.8484 on the original train-validation split with 0.8314, 0.8154, 0.8404, 0.7968, 0.8264, 0.8378, 0.8401 respectively on the kfold split. Plots of the training and validation accuracy and loss clearly show that the mode is overfitting after 5 epochs which means that the model has too much capacity for the data and too little regularization:



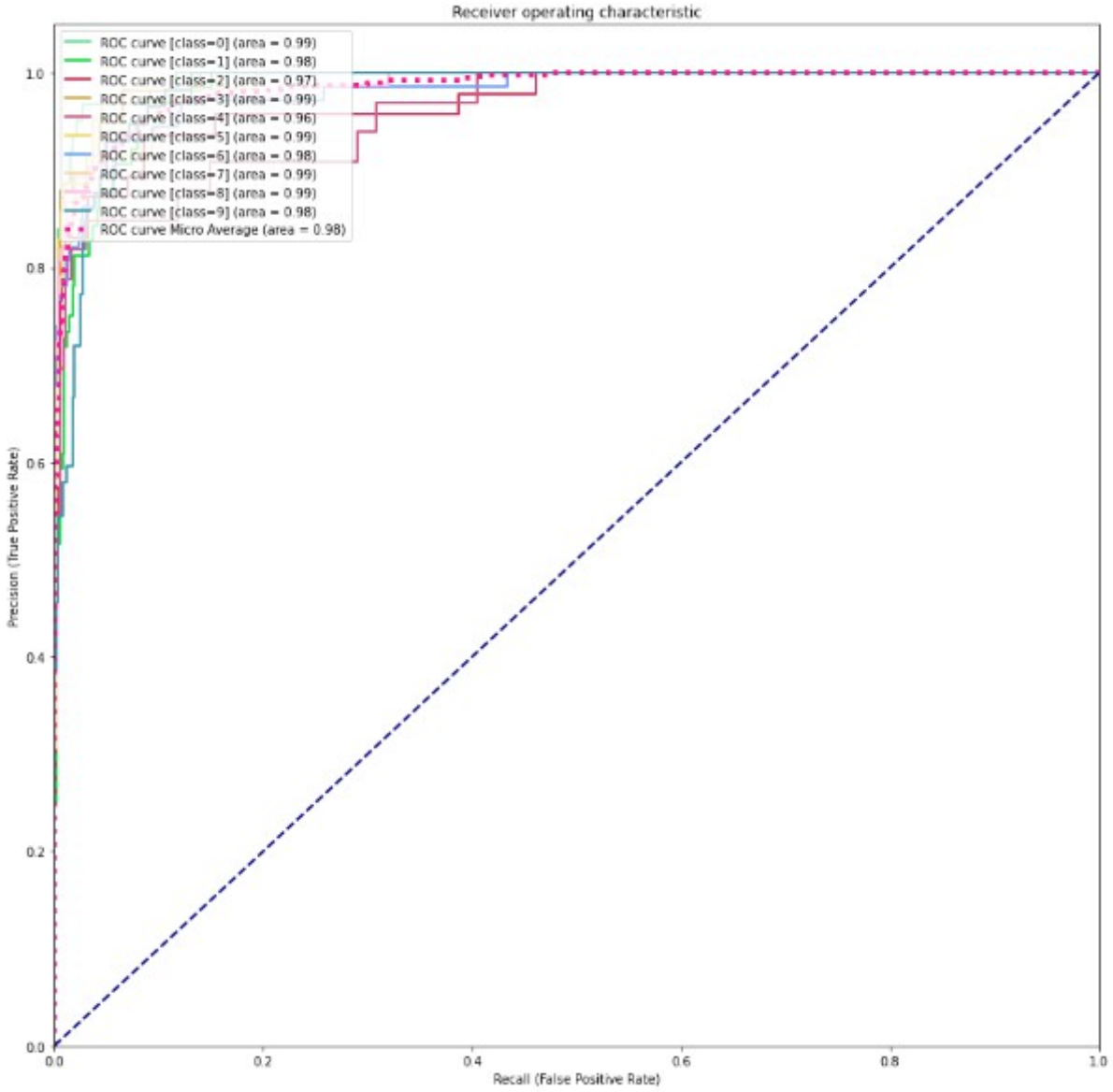




**Original Validation Set metrics**

As specified the original validation accuracy for the transfer learning model was **0.8484.** Confusion matrix, ROC curve and predicted class histogram was plotted as well.



The ROC probability curve for differenciating the classes show that class 4 is the one that is worst but the accuracy is quite high with an AUC greater then 0.96:

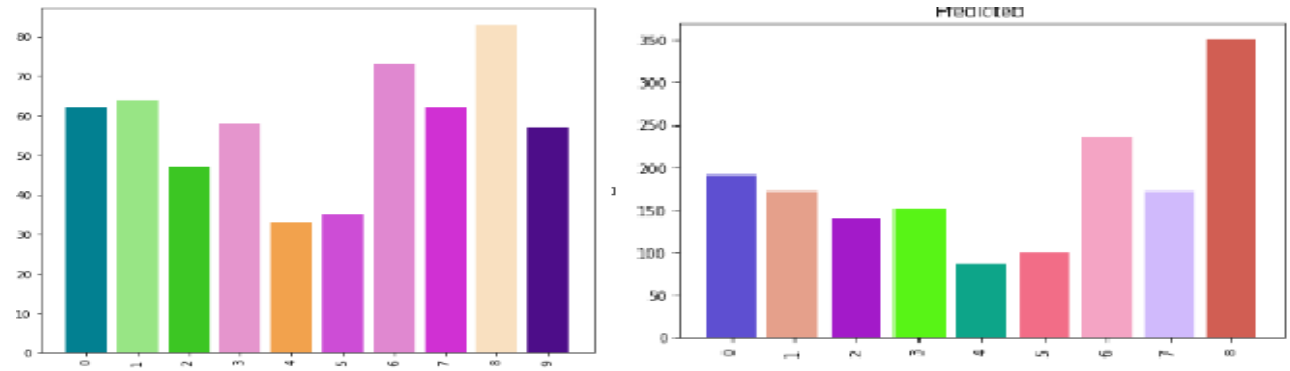



## Ensemble Model

To further improve the final overall test accuracy an ensemble model was done using the predictions of all the above 8 trained models using majority voting. The accuracy on all the data was 1.0 since all models see different part of all of the data.

## Test Data Results

The ensemble model was submitted in the competition and obtained 0.85 accuracy. Also the predicted test class histogram was plotted for each model to compare it to the validation histogram since I assumed that their distribution should be the same:



## Conclusion

Overal I am not happy with my results. I do not admit that I understood the transformer model only its principle but it works better than a simple multilayer perceptron and a lstm model. Perhaps traing the original language model on a much larger dataset and not doing stemming and other pre-processing, keeping punctuation would yield a better result.