NNTI WS2021 NLP Project Report

Kazi Fozle Azim Rabi
Matriculation Number: 2577734
s8karabi@teams.uni-saarland.de

Rayhanul Islam Rumel
Matriculation Number: 2576541
s8rarume@teams.uni-saarland.de

1 Introduction

Social media platform like Facebook has become one of the most popular online communication medium amongst the Bengali and Hindi speaking people where they share their opinions on about everything (from politics, sports to entertainment etc.). Being a public platform, Facebook users are allowed to share almost anything with minimal restrictions and regulations. Such an unprecedented and unrestricted nature of Facebook also allows its users to express their sentiment and emotions, be it positive or extremely negative. The presence of such text contents from two poles has drawn much attention from Natural Language Processing (NLP) Community. But most of the contributions of the NLP community are focused on English, Spanish, German, French and Portuguese languages. These languages also have pretrained word embeddings and Transformer models that makes it possible to train or fine-tune the pretrained models for new but small corpus of label texts. Around the world, 322 million people use the Hindi language[1] where 228 million people use Bengali [2]. But there is not much work done in terms pretrained embedding matrix or Transformer models. To perform sentiment analysis on texts from these languages, people usually need to train their model from the very beginning.

In this report we share our contributions on training embedding matrix and sentiment analysis classifiers using traditional encoders using LSTM as well as the stateofthe'-art Transformer models for both Bengali and Hindi languages. One of our main focus is to work with these model when we have a relatively small corpus (< 5000 sentences). As encoder-decoder architecture are best suitable for sequence to sequence tasks, we show how to work only with encoder and replace the decoder with a classifer to classify sentiments. We show that selfattention based Transformers as encoders performs greatly compered to traditional LSTM based encoders. We also utilize the power transfer learning and show that for small corpus, one model can benefit by learning about another small corpus.

2 Methodology

In this section, we describe the processing of data, a pre-requisite to feed the data into a model, and the architecture of the model.

2.1 Preprocessing

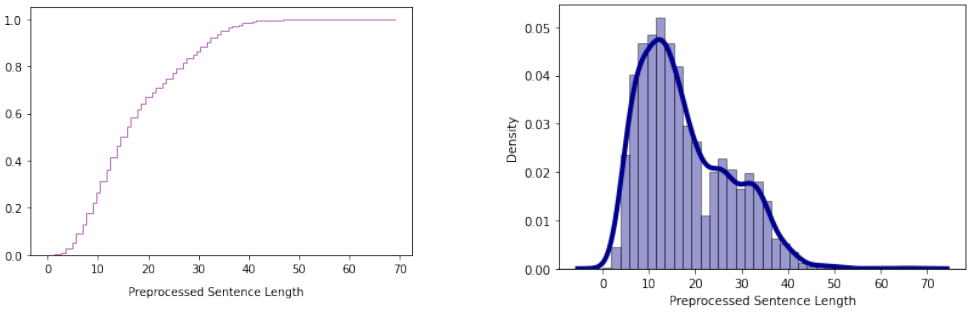
As we deal with text data and we cannot directly use the text to train our model, we are required to preprocess the data. As part of the preprocessing, we first load the HASOC Hindi dataset [3]. We focus on only two columns of the dataset that are text and task_1 (data label).

[1]Hindi language: https://en.wikipedia.org/wiki/Hindi
[2]Bengali languagehttps://en.wikipedia.org/wiki/Bengali_language
[3]Hasoc Hindi Dataset: https://hasocfire.github.io/hasoc/2019/dataset.html

From the dataset, we then tokenize each sentence as a list of words. After that, using the list of Hindi stopwords [4], we remove all the stopwords from the dataset. Then we remove all the usernames and punctuation from the dataset. To remove usernames, we removed all the words that start with '@'. We also manually scan through the dataset and remove all the special characters and symbols for example: … and single characters that hold no meaningfull important information, for example: क, म, ल. For that, we manually translate each single letter that are available in the Hindi dataset to English in order to confirm whether it has any particular meaning. Beside these, we also remove all the URLs from the dataset convert all the texts to lower case. And, lastly we add a new column in the dataset titled preprocessed_sen_len to generate summary statistics and distributions on sentence lengths.

For Bengali dataset, the preprocessing is pretty much the same except for removing single characters and digits. In this step, we keep all the single characters that hold a proper meaning. For example, 'ও' (is used as "he/she"), 'ল' (used as "take" in some regions). On the other hand characters for example, 'ক', 'খ'are being eliminated as they hold no specific meaning.In addition to that, as the original Bengali dataset contains a huge information divided into 7 categories take a chunk of the original dataset of similar distributions (in terms of label) as to the Hindi dataset. We also make sure that the chunk is a good representative of the 7 available categories. This helps us to have an unbiased Bengali dataset.

For both datasets, we do not remove any emojis as they contain valuable information on sentiment.



(a) CDF

(b) Probability distribution with histogram

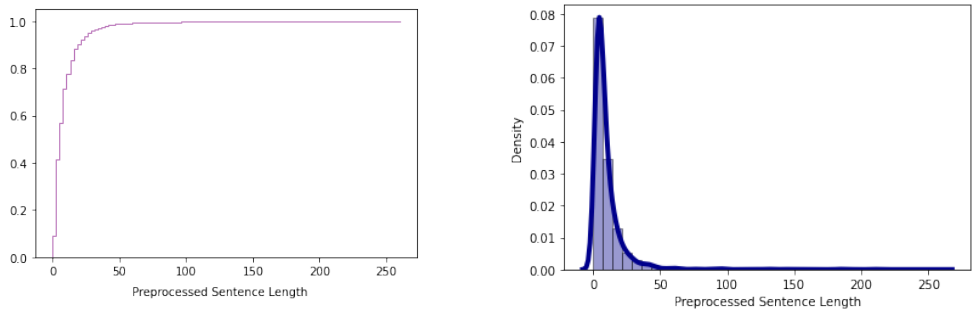
Figure 1: Hindi Sentence length Distributions

2.2 Embedding

Word embeddings are dense vectors of a much smaller dimensionality than most types of vectors. The distance and orientation of the vectors represent the semantic relationships between words. There are different approaches available but we use skip-gram to create words pairs. The pipeline is documented below.

As classifiers do not directly interact with words, we first translate words to numbers before using them in a classifier. One way to do this is to use onehot encoding. Unfortunately, one-hot encoding doesn't hold a lot of information as an onehot vector contains mostly zeros for all the cells except one cell. Word embeddings are calculated in a unique way. Each word is placed in a multi-dimensional space.

[4]Hindi Stop-words: `https://github.com/stopwords-iso/stopwords-hi/blob/master/stopwords-hi.txt`



(a) CDF

(b) Probability distribution with histogram

Figure 2: Bengali Sentence length Distributions

We create a vocabulary from the preprocessed data that contains all the unique words. We also sub-sample the words in order to filter out frequently occurred words within a context. For that, we first calculate the relative frequency, $z(w_i)$ of a word in a corpus and decide based on the following formula,

$$P_{keep}(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1\right) \cdot \frac{0.001}{z(w_i)}$$

For each sentence, we choose a word as center and take the context of the word. In this project, our window size is 5 (inspired by Gensim Library[5]) which means we pick 5 words from each side of a center word (if available). We then generate a random number ranged between 0.0 to 1.0. If the score received from the skip-gram formula for the selected context word is greater than the randomly generated number, we keep the word during the sub-sampling. Also, for computational flexibility, we consider the index of the word in the vocabulary for each word instead of the word itself. So, when we talk about the center word or the context word, we simply refer to the index of the word in the vocabulary.

2.2.1 Word2Vec Model

Mikolov et al. suggested Word2Vec [1], a collection of models for representing distributed word representations in a corpus. Word2Vec (W2V) is a text-to-vector conversion algorithm that takes a text corpus as input and returns a vector representation for each word.

We design a simple two layer linear network to define our Word2Vec model. In the first layer, input is the size of the vocabulary as one hot vector and the output is equal to the embedding size. We choose 300 [1] as the embedding size. The second layer takes the output of the first layer as input and outputs a vector of the size of the vocabulary. Finally we apply the logsoftmax function to convert the output to a onehot vector. We use Adam as the optimizer with a learning rate of 0.001 and Negative Log Likelihood as the loss function.

2.3 Sentiment Analysis With Encoder

In this section, we describe how we have utilized Encoder, global attention and transfer learning to perform sentiment analysis on both Hindi and the Bengali datasets.

2.3.1 Hindi Hate Speech Prediction

After preprocessing and creating the vocabulary, we take each sentence and replace the words with their indices with the help of an already created dictionary, word_to_index. word_to_index dictionary contains all the words of the vocabulary and their corresponding index in the vocabulary that we use to replace all the words in a sentence with their relevant indices. The length of sentences in a corpus are not same but we want to have each sentence to be of same length. Therefore, we consider a hyperparameter that is sentence length. We choose 32

[5]Gensim Library`https://radimrehurek.com/gensim`

Table 1: Some Hypermaters of EncoderCNN model

Batch Size	4
Initial Learning Rate	1e-4
Criterion	BCELoss
Optimizer	Adam
L2 Regularization Factor Value	1e-6

to be the value of sentence length, as we can see from Figure 1, almost 90% of the sentences contain less than 32 words. For each sentence, we check whether the length of a sentence is greater or less than 32. We pad the sentence at the end with the padding index that is less than 32 in length and truncate sentences that contain more than 32 indices.

Our initial model consists of an Encoder - built using LSTM - and a few Convolutions, Pooling and fully connected layers. The state of the art NLP models utilizes the power of encoder-decoder architecture[2]. But they are well suited for Sequence to Sequence production tasks[3] like machine translation. As our task is predicting if a sentence is hate-speech or not, the complete encoder-decoder architecture was of no use. Hence, we only use the encoder part as it is good at encoding an entire sentence to a context vector/matrix.

For Hindi dataset, we use our pretrained Hindi embedding matrix as the first layer and the output of it then passes through a bi-directional LSTM in the second layer. These both layers work as an encoder in our case. Hindi embedding works as word level encoding where LSTM works as sentence level encoding. We take the encoded sentence and pass it to first of the three 2D convolutional layers. We use 2 layers of max pooling after each of the first two convolutional layers. Each convolutional layer passes the output to the next layer through an ReLU activation call. The output of the last convolutional layer is then passed through a fully connected linear layer. As we are expecting the result to be a binary probability distribution, we use sigmoid function on the result that we observe from the fully connected layer.

Although LSTM suffers less compared to plain RNN from forgetting earlier information, the final hidden state still does not carry all the information. Bachrach et al. [4] suggest instead of passing the final hidden state, to consider all the hidden states and pass them to the next convolutional layer. Although this increases the computational time and space complexity, it enables the model to work with global attention.

Table 1 contains information on some of the hyperparameters that were used to train the model for this task. Along with these we also use 0.01 as gamma to change the learning rate after some predefined epochs, and early stopping to overcome overfitting the data.

2.3.2 Bengali Hate Speech Prediction

For the Bengali hate speech prediction, from preprocessing to using model and defining hyperparameters, everything is exactly the same as sentiment analysis for the Hindi dataset except, instead of using Hindi embedding in the first layer of the model, we use Bengali embedding. We take three different routes to do the sentiment analysis using the Bengali dataset.

At the beginning, after training a model using the Hindi dataset, we freeze the learnable parameters of the model and only change the first layer from Hindi embedding to Bengali embedding. Then using the 10 percent of the Bengali dataset, we test our model. In this approach, we do not train our model.

The second approach also uses a pretrained Hindi model. We again change the embedding layer with the Bengali embedding and then instead of freezing rest of the model parameters, we continue to train the model with the Bengali train set and then test it with the Bengali test set.

Finally, instead of taking a trained model, we consider a fresh model and train it using only the Bengali train set. After the successful completion of training, we test the model with the

Bengali test set.For all the models, we separate 10% data for validation and and based on the validation loss, we decide if we should early stop or continue to prevent overfitting.

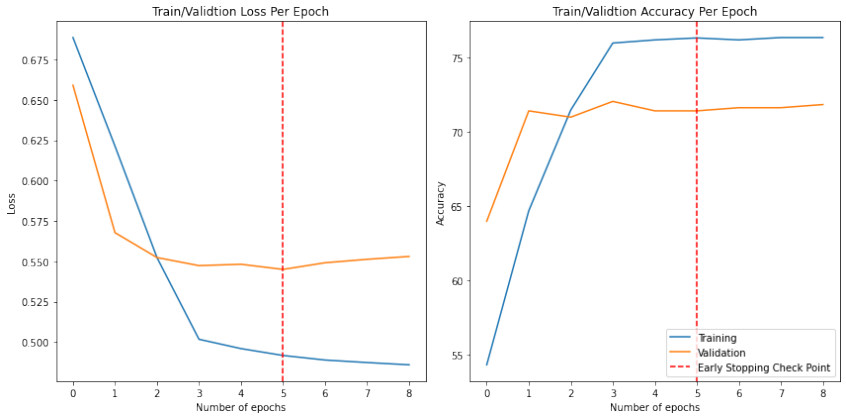


Figure 3: Bengali Encoder CNN Loss and Accuracy (With Hindi Weights)

2.4 Sentiment Analysis With Transformer

While Enocders built on top of LSTM layers with global attention somewhat solve the long-term dependency problem, they fail to determine the correlation of single words of a single sentence[5]. To overcome this issue and also to improve the long-term dependency in sequence to sequence conversion tasks, Vaswani et al. proposes Transformers[5] - that solves long-term dependency problems by creating connections between words within a sentence. This connection between words of a sentence allows the Transformer model to create a representation of a sentence that is better than the context vector/matrix produced by a LSTM based sequential encoders. This mechanism is known as self-attention or intra-attention. The Transormer models are very popular for their sequence to sequence conversion where one transformer works as an encoder and a slightly different one works as a decoder. The encoder focuses on input sentence's self-attention where the decoder also focuses on the encoder-decoder attention (correlation between input and output sequence) beside focusing on output sentence's self-attention. The transformer model has one additional benefit than other sequential encoder-decoder model as the Transformers are able to process a sequence at once rather than at different time steps. As mentioned earlier, we only use the encoder Transformer to get a better representation of the sentence as our task is not a sequence to sequence one, hence the decoder is not required.

The Transformer encoder mainly consist of two layers: i) one multi-head (parallelized) self-attention layer, ii) one fully connected layer for each position. The encoder takes a positionally-embedded (context aware) sentence as input and produces a different representation of same shape. To positionally embed a sentence, we follow the novel approach[5] suggested by Vaswani et al. After multiplying each word with the embedding matrix (like Word2Vec), we add the output of a sinusoidal function to uniquely and deterministically represent the positions of each words within a sentence. This is important as any sequence passes through the Transormer encoder simultaneously, it does not have any sense of the word orders as oppose to RNN/LSTM where these model take a sequence in word by word.

Table 2: Some Hypermaters of TransformerCNN model

Batch Size	1
Initial Learning Rate	1e-4
Criterion	BCELoss
Optimizer	Adam
L2 Regularization Factor Value	1e-6
Number of Multihead Self-Attention Head	2
Number of Neurons in Transformer's Linear Layer	50
Convolution Layer 1	kernel size = (1, 21)
Convolution Layer 2	kernel size = (1, 25)
Convolution Layer 3	kernel size = (1, 58)
Number of Fully |Connected Layers	2
Last Layer Activation Function	Sigmoid

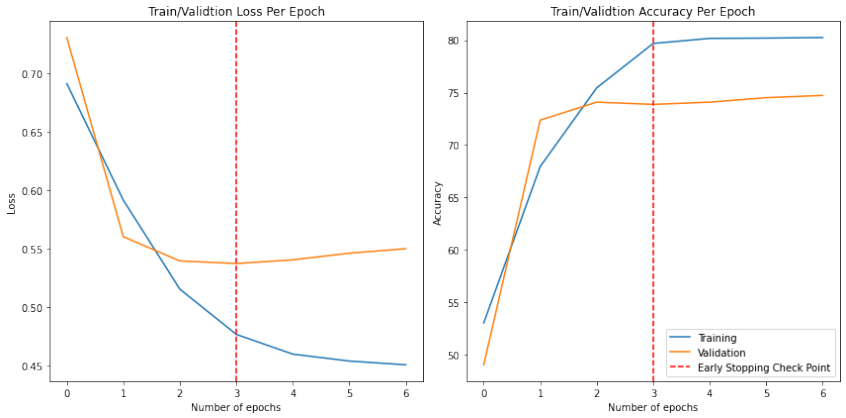


Figure 4: Hindi Encoder with CNN Loss and Accuracy

2.4.1 Hindi Hate Speech Prediction

To improve our Hindi hate speech detection model, we use the self-attention based Transformer unit as the encoder to replace our previous LSTM and global attention based encoder. We pass the Transormer's output to three 2D convolution layers (one after other). Each convolution layer consists of a max-pool and a ReLU activation function. Finally, the output form the third convolution layer goes through two fully connected layers (again ReLUed in between) and finally the model performs Sigmoid activation to do the prediction.

For convolution layer's kernel, we had to select some unusally shaped kernels to get the best out of the models (Also for the EncoderCNN models). As we use 32 as sentence length, and each word is being represented by 300 features, our input shapes (to the CNN layers) were 32×300. Our goal was to pass a compact but most highlighter features of a sentence to the fully connected layers. Very wide ($height = 1$) kernels allowed us to select the most representative features of a word, while the max-pooling layers selected the words that contributed the most.

In this model, we do not use a pretrained Embedding matrix the way we use for LSTM based EncoderCNN model as those embedding matrices were too simple. We know that an embedding matrix's job is to only capture some feature of the words, but the Word2Vec model we trained contains only two linear layers without any non-linear activation such as ReLU. On

the contrary, in the TransformerCNN model, the embedding layer sits behind a really complex Transormer encoder which tries to learn the meaning of words within a sentence followed by 3 convolution, 2 pooling and 2 fully connected layers. So during the back-propagation, not only the intermediate layers learn, as the entire graph is connected, the embedding layer also gets benefited, thus capturing the features of the words more accurately. Hence, we neither use any pretrained embedding layer nor freeze its learning.

Like the previous model, here we again use Adam as the optimizer and BinaryCrossEntropyLoss (BCELoss) as the criterion. As small batch size improves generalization performance and allows a significantly smaller memory footprint [6], our batch size is 1 (stochastic gradient descent). Table 2 provides information on some of the important hyper-parameters and their values that we use in the TransformerCNN model.

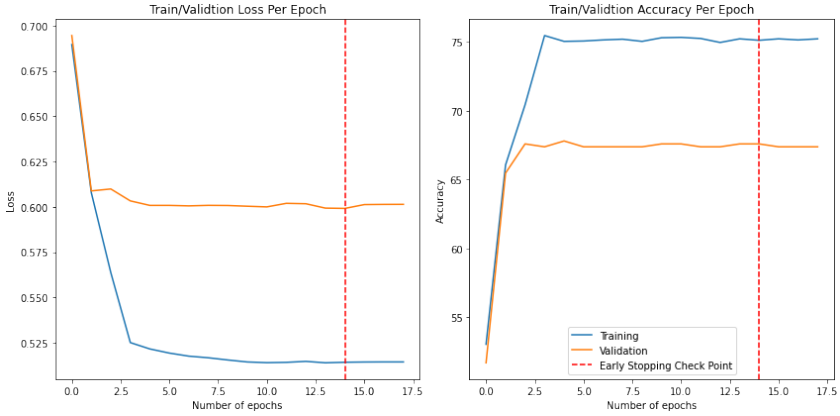


Figure 5: Bengali Encoder CNN Loss and Accuracy (Without Hindi Weights)

2.4.2 Bengali Hate Speech Prediction

For Bengali hate speech prediction, we use exactly the same model architecture along with the hyper-parameter values from Table 2. As we opt not to use our pretrained word embeddings, here we start with training a TransformerCNN model with the Bengali dataset. This also provides us with a new embedding matrix for the words in our vocabulary which then we use to test how the Hindi TransferCNN model generalize to Bengali dataset. For this task, we take the pretrained Hindi TransferCNN model and replace its embedding layer with the new Bengali embedding layer and copy all the other weights. For both of these tasks, we use 10% data for testing and 10% data for validation. Here also the validation loss was used to early stop the training to stop the model from overfitting.

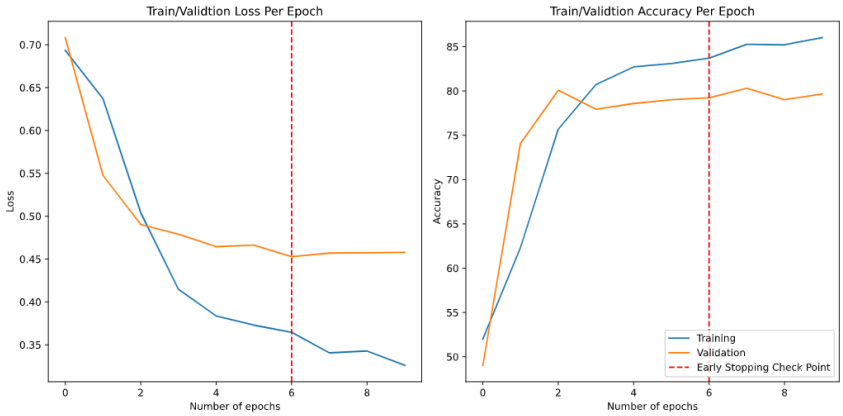
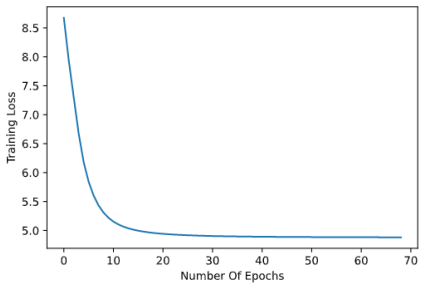


Figure 6: Hindi Transformer CNN Loss and Accuracy

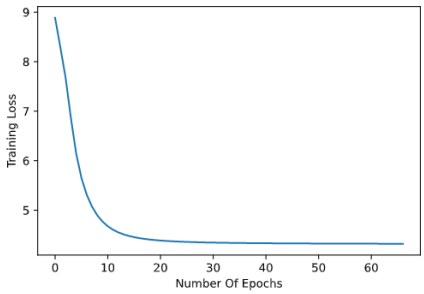
3 Results

In this section, we discuss the results our word embedding and sentiment classification tasks with different models.

3.1 Embedding



(a) Train loss of Hindi word embedding



(b) Train loss of Bengali word embedding

Figure 7: Train loss of Word Embedding

We can see from Figure 7a and 7b the training loss for the Hindi dataset is 4.88 and for Bengali, the train loss is 4.32. The length of Bengali vocabulary is 14583 and Hindi vocabulary is 18171. There were in total 341810 Bengali training pairs and 676040 Hindi training pairs. Even though we use 100 epoch to train the model, as the train loss was not improving, the training stopped after 69 epochs for Hindi dataset and after 62 epochs for Bengali dataset due to intervention of early stopping. As we do not intend to use validation set for this task, we use train loss as for the early stopping criteria. We allow our model to overfit the data as much as possible as it would allow to learn more about the words.As our sentiment classification dataset contains words only from these vocabularies, we don't intend to generalize the model. From our point of view there is no harm of having an overfitted Word2Vec model.

3.2 Sentiment Analysis & Improvement

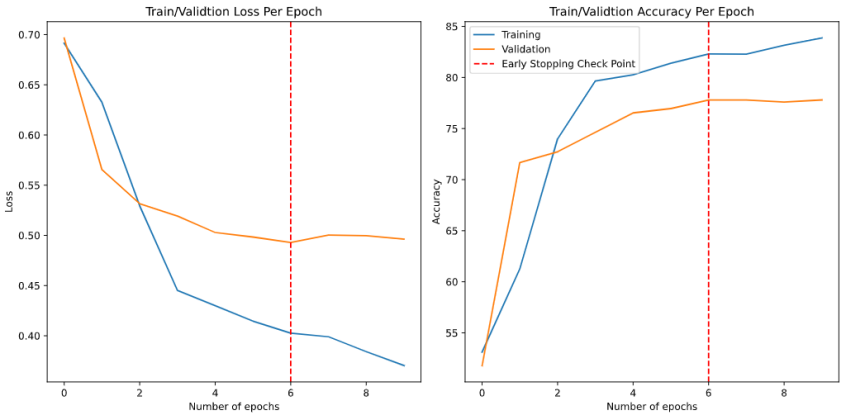


Figure 8: Bengali Transformer CNN Loss and Accuracy

The Hindi EncoderCNN classifier continued for 7 epochs although the the model we use to test the test data uses the states stored at epoch 4 as this model has the smallest validation loss. We use the model states at epoch 4 and epoch 15 for Bengali EncodeCNN on top of Hindi classifier and the new Bengali EncoderCNN classifier respectively, for the same reason. Test accuracy of the Hindi classifier was 72.53%. The same classifier had only 50.85 test accuracy

when we tested it with the Bengali test data. As these deep learning models are very data hungry, with only around 4500-5000 data, these models could not generalize well. Around 80% training accuracy even after using a big dropout probability of 0.35 and regularization also indicates towards the scarcity of training data. Test accuracy of the Bengali classifier that was trained using only Bengali training data produced 68.03% on test data. This accuracy is quite less than the one we got from the Hindi classifier. One obvious reason for this is, the randomly sampled data we got from the original Bengali dataset contained only 14583 words in the vocabulary compared to Hindi's 18171. Beside this, as can be seen from Figure 2 and 1, the mean sentence length for Bengali data was around 10 compared to Hindi's almost 18. Their distribution graphs also shows that the Hindi sentences were more varying in lengths while the Bengali sentences were compact near the mean (small standard deviation). But when we train the already trained Hindi classifier with Bengali data the accuracy on the same test data shows exactly 3% improvement. The reason is that while the new Bengali classifier was using random weights initially, the pretrained Hindi classifier already had some knowledge that it learnt from the Hindi dataset and benefited from transfer learning. Figure 4, 3 and 5 show the training and validation loss and accuracy of the EncoderCNN models.

A similar trend was also visible for the different TransformerCNN classifiers. The only difference is that all the test and validation accuracies experienced a significant improvement. The test accuracies of Hindi model were 81.97% and 58.91% (around 11% and 9% improvement) respectively on Hindi and Bengali test datasets. The accuracy of the new Bengali classifier showed more than 10% improvement with an accuracy of 78.18%. Figure 6, 8 and 8 shows the training and validation's loss and accuracy of the TransformerCNN models and Table 3 summarises all the accuracies of all the models.

Table 3: Different Model's Training, Validation and Test Accuracy Summary

Model Description	**Train**	**Validation**	**Test**
EncoderCNN Models			
Hindi	79.69 %	73.88 %	72.53 %
Hindi classifier on Bengali Data	NA	NA	50.85 %
Bengali trained on top of Hindi weights	76.30 %	71.40 %	71.03 %
Bengali trained with only Bengali Data	75.11 %	67.58 %	68.03 %
TransformerCNN Models			
Hindi	83.71 %	79.23 %	81.97 %
Hindi classifier on Bengali Data	NA	NA	58.91 %
Bengali trained with only Bengali Data	81.40 %	76.96 %	78.18 %

4 Conclusion

In this report, we explored different models e.g. LSTM Encoders with global attention, Transformer with self-attention and CNN to analyze sentiments using Hindi and Bengali dataset. For the EncoderCNN model, we train our own Embadding matrix using Bengali and Hindi words. Due to not having sufficiently large dataset, most of our models tries to overfit. As a result, model accuracies for different models that we observe are not significantly good compared to the state-of-the art approaches like BERT [6] as those model are trained on millions of data for several months. However, we show that using transfer learning can improve the performance compared to other similar models and approaches. For future work, as we believe that training a large dataset would allow us to hugely improve the accuracy with of our TransforerCNN model, we plan to train this model on a larger dataset with some fine-tuning of the hyperparameters and evaluate the on different languages.

[6]BERT: `https://arxiv.org/abs/1810.04805`

References

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[2] Wouter Leeftink and Gerasimos Spanakis. Towards controlled transformation of sentiment in sentences, 2019.

[3] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[4] Yoram Bachrach, Andrej Zukov-Gregoric, Sam Coope, Ed Tovell, Bogdan Maksak, Jose Rodriguez, and Conan McMurtie. An attention mechanism for answer selection using a combined global and local view, 2017.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. CoRR, abs/1706.03762, 2017.

[6] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks, 2018.