CSC413 Project Report:
Comparative Study on the Performances of Various NLP Neural Network Types (CNN, RNN, and Transformer) on News Article Category Classification Task

Dohyun Kim, Gurmanjot Singh, Yuxuan Wang

Due: April 19, 2024

1 Introduction

1.1 Motivation

We, modern human beings, are living in the Information Age, a time period in which access and use of information is a key component of human civilization. Due to increased value of the information, humans have been gathering and storing a huge volume of information since the rise of the information age, and accessing those enormous amounts of valuable information became an essential everyday task for all individuals living in this era. However, obtaining information that is relevant to a specific individual becomes much harder because of the massive volume of the information that is beyond human imagination. One way to make various types of information more accessible is categorizing the information. Information categories such as business, entertainment, politics, sport, and technology allow individuals to collect information they want/need easily and quickly. Unfortunately, not all information in the media is labeled with their topics.

Natural Language Processing (NLP) neural networks, specifically for text classification in this case, can resolve the problem of unlabeled information, by correctly labeling them with appropriate categories. One popular source of text data is news articles, and we will be focusing on classifying news articles into five categories: business, entertainment, politics, sport, and technology. This will assist people to navigate through vast pool of information and to reach the information they were looking for. To extend the study even farther, we will compare the performances of three types of neural networks that are commonly used for language modeling: Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Transformer. We will examine various types of RNN as well, which are simple RNN, Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU).

1.2 Prior Work

Many researchers tried to evaluate and compare the performances of various types of neural networks, in an attempt to find the optimal neural network model for certain tasks. Similar to our research, many scientists focused on the text classification tasks and compared diverse types of neural networks that are commonly used in natural language processing tasks.

Soyalp, Alar, Ozkanli, and Yildiz (2021) compared the book category classification performances of LSTM model, CNN, and Transformer, while using Word2Vec embedding for all models. They used 5 criteria to quantify the performances of three models, which are accuracy, loss, precision, recall, and F1-score. According to all 5 criteria, Transformer showed the best performance of 91.6% accuracy, and LSTM showed the second-best performance of 88.4% accuracy, and CNN showed the worst performance of 87.7% among them. Moreover, there were active studies, specifically on news article classification tasks which is what we are aiming to investigate in this report.

Saigal and Khanna (2020) used different sets of models to classify news articles into their multi-classes. They compared three models which are Least-Squares Support Vector Machine (LS-SVM), Twin Support Vector Machine (TW-SVM), and Least-Squares Twin Support Vector Machine (LS-TW-SVM).

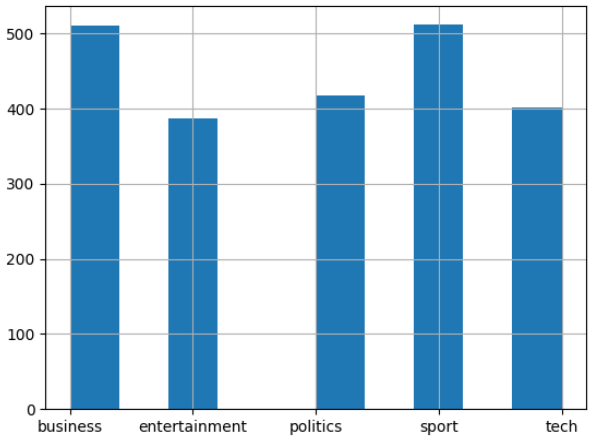

Figure 1: Histogram of the frequencies for the various categories of articles present in our collection

The resulting accuracies of LS-SVM, TW-SVM, and LS-TW-SVM were 83.33%, 90.10%, and 92.96%, respectively, which suggests that LS-TW-SVM is the best model for multi-category news classification tasks.

Sharma, Shakya, Joshi, and Panday (2021) conducted similar research, but their research has more similarity with our research as they also examined the performances of RNN and CNN in news article classification tasks. Three models they compared were CNN, a combination of CNN and GRU, and Hierarchical Attention Network (HAN). The average F1-score of CNN was about 0.8667, the average F1-score of a combination of CNN and GRU was about 0.8753, and the average F1-score of HAN was about 0.8991. Their study suggests that HAN is the optimal model for news article classification tasks among those three models.

2 Methods

2.1 Data

Our models will perform a classification task on a collection of BBC news articles of five different subjects–business, entertainment, politics, sport, and tech. This data was collected by and used by D. Greene and Cunningham when discussing a clustering problem for articles, and can be accessed in a .csv file from a public Kaggle post by Gültekin.

The data is structured in a way so that when the .csv is read to a dictionary, we get coupled lists of category, title, and content strings from each article. We convert the categories to their respective numbers from 0-4 for classification and combine the titles and content together for each article so the classification accounts for the title as well for the prediction. We can count the frequencies of each category present in our data from this part, given in Figure 1. Note that the business and sport sections are over-represented by a good proportion compared to the other three.

To clean the data, we wish to eliminate stopwords (i.e., 'a', 'or', and so on) that are redundant and are not relevant to the task of classification, which should also reduce the length of the sequences tremendously. We replace these words with just spaces in the sequence, the group of which we then zip to the categories. For the details regarding the code implementation, refer to the 'Exploratory Code Analysis' section.

We tokenize the texts using a basic English tokenizer, which we then convert into GloVe indices. This will be our representation for the data used by all the models in this paper. We separate the 2225 data set into a training, validation, and test set, with distribution $3 : 1 : 1$, after shuffling. These also have their own distributions of data, from when we use them, given in Figure 2.

To properly batch the training data during training, we pad the sequences up to the largest sequence length. The details for all this are provided in the 'Coding' section of our code. Most of this pre-processing is inspired by Choubey's article on multiclass text classification.

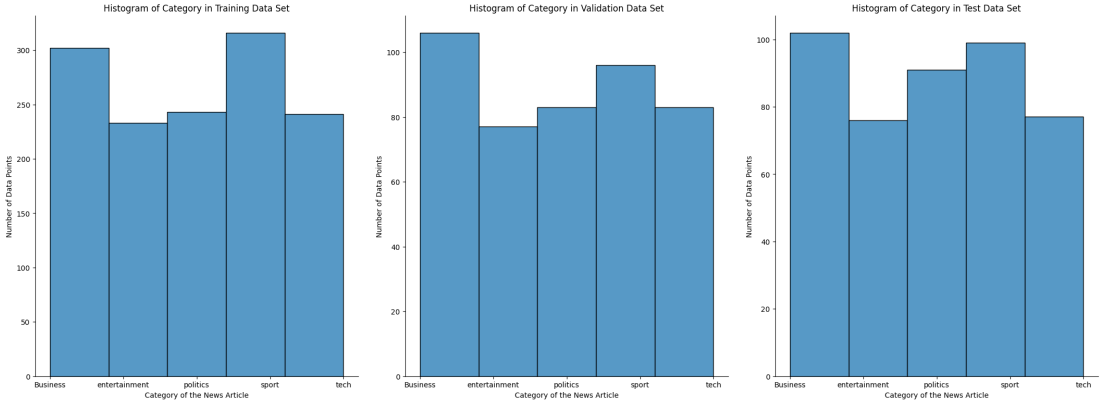

Figure 2: Histograms for training, validation, and test set respectively.

2.2 Model Structures

The following sections will describe the architecture implementation for the models and training method. All of this is done in the file provided and can be assumed to run on Colab via the CPU. We will use the same training schema for each model that we test. As a classification problem, the training objective will be to maximize the accuracy of the model in providing output probabilities that closely match the correct classification of the input. For this, we will use the cross entropy loss criterion.

We will also use the Adam optimizer as it provides one of the faster methods of model convergence with few trade-offs. A batch size of 5 generally seemed to keep the model learning fast and prevented memory leakage if done on a CPU. We compared the choice of two learning rates–0.001 and 0.0001–on each model. We also decided that 15 epochs of training would be sufficient to assess the model's viability, considering how fast it learned over that period. We found the learning rate that maximizes performance is different for each model. Before describing the model architectures specifically, note that each model uses a frozen GloVe embedding of the aforementioned data representation.

ResNet (CNN with residual connections): The first model we consider is a residual convolutional neural network, which has a general structure of a convolutional network (convolution layers stacked with pooling and other normalization layers, with a classification network at the end) plus residual connections between them. Particularly, we use ResNet-50, a pretrained residual network with 50 layers, consisting of a convolution, max pooling, then 49 convolution blocks with skip connections, and then a final average pooling layer and a fully connected layer for a final output of some given size, which can be visualized in Figure 3. We modify the first convolutional layer to allow one channel inputs and the last fully connected layer so the output is just of size 5. The embedded GloVe representation is reshaped via a bilinear interpolation layer into a 244×244 input that can be accepted by ResNet. The implementation is in the 'Convolutional Neural Network' section of the code.

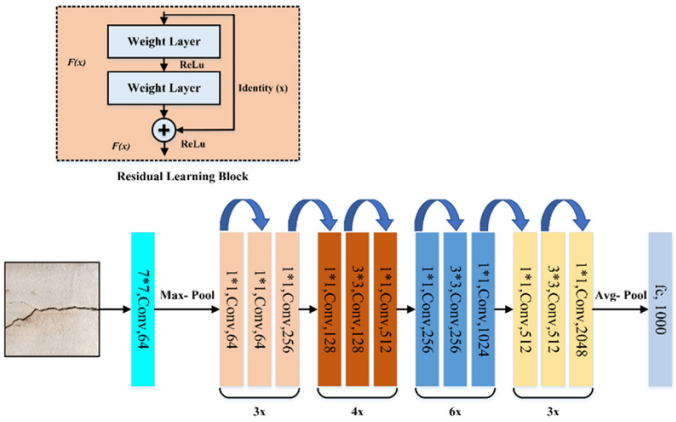

Figure 3: Diagram depicting the layers of ResNet-50 without adjustments (ref. Ali pg. 1688).

RNN (simple, LSTM, and GRU): The next set of models are recurrent type models, using RNN structure. That is, we pass the embedded sequence one-by-one into a layer which produces a hidden state, which is then also fed with the next sequence element back into the layer. These can be stacked up to any number of layers, which we can choose to be normal RNN, LSTM, or GRU structures. The unfolded structure is shown in Figure 4. We set the number of layers to 3 and the hidden layer size (inside the recurrent node) to be 64. We then take the last hidden state outputted and apply a fully connected layer that takes it to 5 numbers. The details of this implementation are provided in the 'Recurrent Neural network' section of the code.

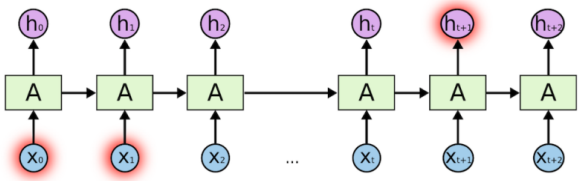

Figure 4: Depiction of an unfolded recurrent type network, with some sequence (x_n) being fed into a black box layer A (i.e., GRU, LSTM, or simple) to produce hidden states (h_n) .

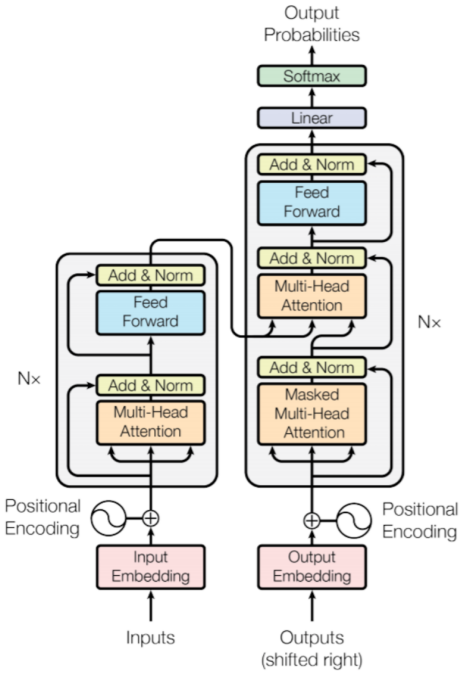

Figure 5: Depiction of full transformer model. We only use the encoder on the left (ref. Ankit).

Transformer: The last model we implement is the transformer, particularly the encoder component of the transformer. This contains a series of blocks applied to the whole padded sequence, each of which first apply a multihead attention layer (producing the weighted values for each sequence position), and then a feedforward layer which has same dimensional input and output and a bottleneck hidden layer that is applied pointwise to each element of the sequence, which is given as the left piece of the diagram in Figure 5 (the encoder). These produce final weighted values (same dimension as the input sequence embeddings) per sequence position, which we then average so we can apply a final feedforward that takes the embedding size to a dimension 5 space. Since the transformer does not identify any positional information in the input embedding by itself, we add a fixed Fourier (cosine-sine) positional embedding to the original embedding, which adjusts each query given their position. We also use a causal attention mask to avoid the model learning values of current points in the sequence from future points, as well as a mask for the padding we added in the data representation to avoid learning redundancies. We apply a dropout of 0.1 in the embedding and transformer layers. To match the RNN case, we use 3 blocks with a hidden layer size of 64 and 2 attention heads. The details are given in the 'Transformer' section of the code, and many design choices are based off of Melchor's article on transformers.

NOTE Observe that the ResNet model is very different in comparison to the other two types, as the expressivity of a regular CNN of the same number of layers as a RNN or transformer is much lower, especially for text classification problems. We also did not have a particularly formal comparison between the models, we did not have a control that we fixed for each model that decided that they were the same "size" so to speak. However, we did find that the model runtimes were very close to each other, so their computational costs are similar to each other. Practical applications would treat these models the same in the computational cost, and so comparisons in memory usage and performance stats are warranted.

3 Results

3.1 Model Performances

For each model, we will provide the loss-accuracy graphs for the best choice of learning rate between 0.001 and 0.0001, but we will report the final accuracies for both in Table 1.

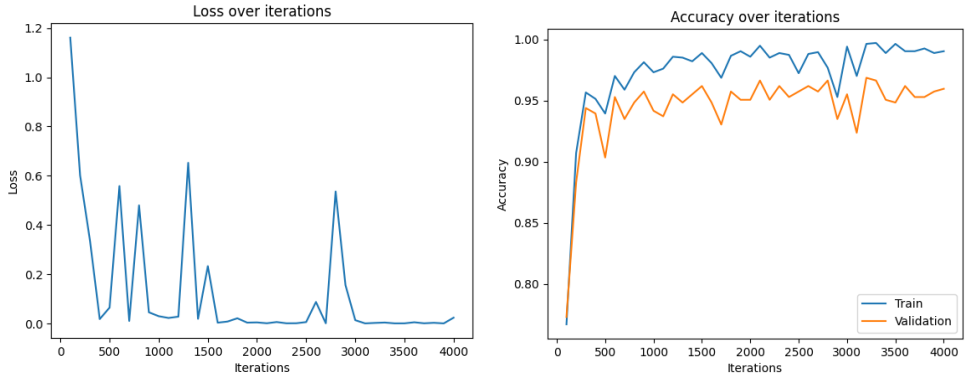

Figure 6: Loss and accuracy of transformer model across training iterations, lr = 0.0001.

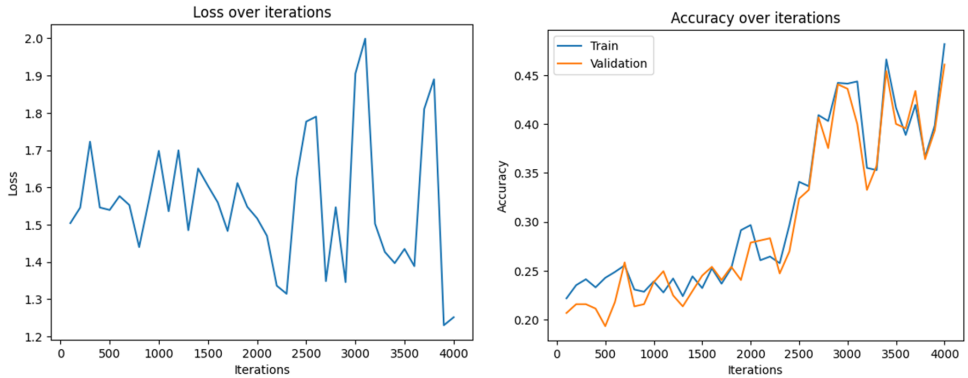

Figure 7: Loss and accuracy of simple RNN model across training iterations, lr = 0.0001.

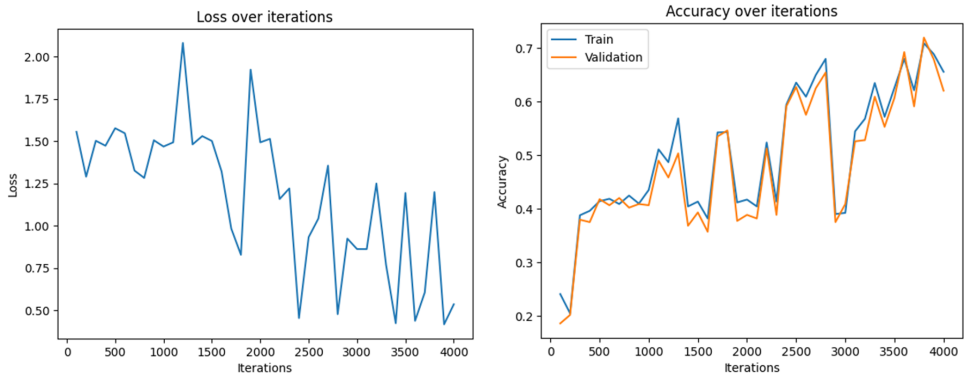

Figure 8: Loss and accuracy of LSTM RNN model across training iterations, lr = 0.001.

Among the evaluated models, the GRU achieved the highest validation accuracy of 0.9775 and a test accuracy of 0.9730. The transformer secured the second best performance, with the same test accuracy as GRU but wth a slightly lower validation accuracy. The third best was the ResNet, showcasing substantial but not leading effectiveness. Lastly, the LSTM and the standard RNN recorded the lowest accuracies, demonstrating a comparative shortfall in performance.

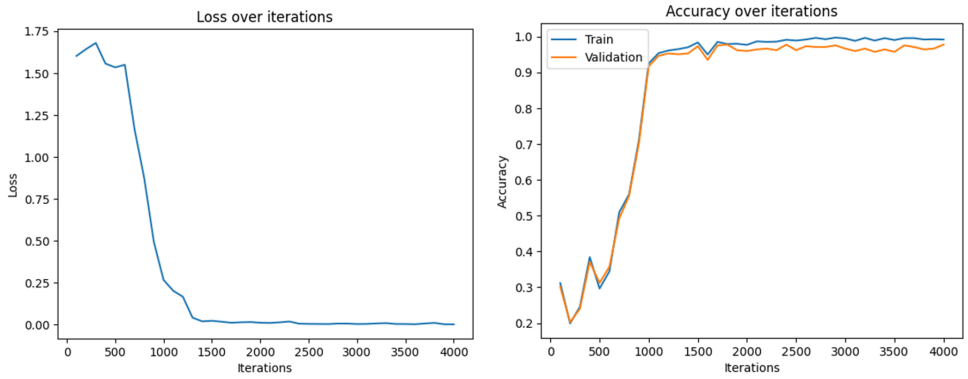

Figure 9: Loss and accuracy of GRU RNN model across training iterations, lr = 0.001.

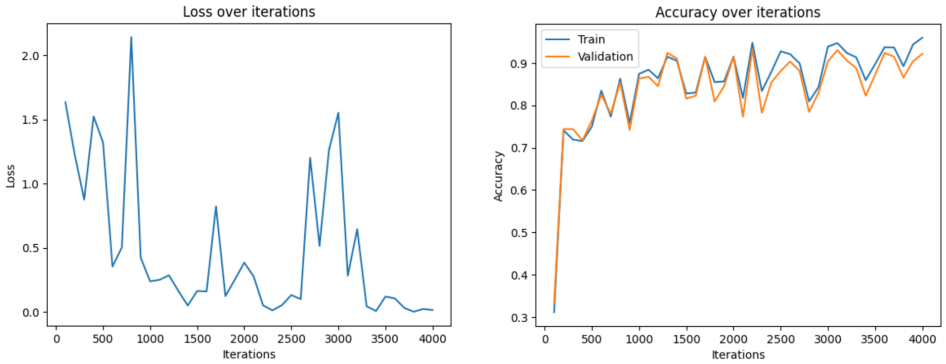

Figure 10: Loss and accuracy of ResNet50 CNN model across training iterations, lr = 0.001.

Qualitatively, the GRU seems to train the smoothest up to maximum performance, while the transformer and ResNet both still show some noise. On the other hand, the transformer seems to overfit a bit by 4% to the training based on the difference from the validation, even though the best validation is much closer. The other remaining models do not show this behaviour, but are much more erratic and tend to not cleanly improve, which costs them accuracy. Although the graphs are not shown, the learning rate cases that were omitted seem to have little to no growth and have random changes in the loss and the accuracy, which is indicative of either the learning rate causing big jumps and so exploding the error (too large for RNN and transformer), or being so small that the noise in the batch gradient descent dominates the improvement. This could be fixed by generally applying a learning rate scheduler to decay the learning rate, so the learning rate is acceptable for models that need the boost at the start, and later on in training for fine-tuning and helping the models that prefer smaller learning rates.

Table 1: Best validation accuracies and final test accuracies for each model from training.

Acc.s/Models	**Transformer**		**GRU**		**LSTM**		**RNN**		**ResNet**	
Learning rate	0.001	0.0001	0.001	0.0001	0.001	0.0001	0.001	0.0001	0.001	0.0001
Best Val. Acc.	0.5213	0.9685	0.9775	0.4787	0.7191	0.5393	0.4337	0.4607	0.9056	0.9303
Final Test. Acc.	NA	0.9730	0.9730	0.5101	0.6315	0.3865	0.3528	0.4517	0.8876	0.9124

NOTE For the transformer, we didn't finish the training for 0.001 due to time and RAM constraints and we found that it was actually performing worse over the training, suggesting that its loss is super-sensitive to changes and requires a smaller learning rate. The validation is in the first quarter of the training period–the test accuracy computation caused a crash for that runtime session as we were using a different computer than for 0.0001, which prevented us from getting a test accuracy for that session, but it is expected to be low, since both training and validation was low for that case. It is advised to check this by training the code for this learning rate.

4 Discussion

4.1 Limitations

There are several general limitations in the project. First of all, the number of news articles for each category is not balanced, so the model might be biased toward categories with more data points. And since the paper focuses more on the comparison between the models and the limitations of hardware such as GPU, the hyperparameters are not tuned carefully. Only learning rates are tuned because of the long runtime. Thus, the performance might not be optimal.

There are also some limitations specific to each model. For the ResNet, the project uses pre-trained ResNet50 due to the limitation of hardware. The ResNet model could achieve higher accuracy with larger model such as ResNet152, but it requires large computational cost and takes very long time to run.

The RNN models only contain very minimal layer structure, which consists one dropout layer, RNN/LSTM/GRU layer, and one fully-connected layer at the end. The performance could be improved if more complex structures were used.

The same limitation as RNN could be said for the transformer, but the usage of memory scaled exponentially with the number of blocks and hidden layer size.

4.2 Conclusion

Our result suggests that the performances vary significantly by type of neural network and hyperparameters. Overall, the transformer model showed the best performance, in terms of the accuracy. The runtime is controlled, and the runtime of each model is roughly the same. Surprisingly, GRU model's accuracy is almost as high as the transformers, given that the structure of GRU is quite simple compared to the transformer. The accuracy of transformers is 6.6% higher than Resnet, and 54% higher than LSTM. The performance of the standard RNN is the worst of them, the accuracy of transformers is 115% higher than the standard RNN. The final rank of the final test performance for each model from the best to the worst is: Transformers, GRU (RNN), ResNet (CNN), LSTM (RNN), and standard RNN.

However, as mentioned above, the models have limitations in complexity, so the performance might be changed if larger models were used or the models were tuned differently. It may be of interest to discuss how scaling and tuning the models changes their performance.

To take one more step farther, the comparison can be extended to different classifying problems such as images or time series, and can be used to combine the best of both worlds for the classification problem that hasn't been discussed much, like video classification tasks. Deeper study for such classification is needed to enrich our understanding on the neural networks' classification ability and optimal choice of models for such tasks.

5 Work Allocation

- Dohyun Kim:
  - Conducting data cleaning and processing (removing stop words)
  - Conducting Exploratory Data Analysis (EDA) (checking for data imbalance and generating histograms of news article category for entire data, train data, validation data, and test data sets)
  - Implementing GloVe embedding
  - Implementing general functions for training (`collate_batch`, `accuracy`, and `train_model`)
  - Implementing RNN models (Simple RNN, LSTM, and GRU)
  - Training and collecting the loss and accuracy curves for the RNN models, as well as test accuracies (for Results)
  - Formatting LaTeX file for the report
  - Writing 'Introduction' and 'References' sections of the report
  - Contributed to 'Discussion' section of the report
- Gurmanjot Singh:
  - Implementing Transformer and `PositionEmbedding` used in the embedding component of the transformer
  - Training and collecting the loss and accuracy curves for the Transformer model, as well as test accuracy (for Results)
  - Writing 'Methods' section of the report, particularly the 'Data' and 'Model Structures' subsections
  - Added all diagrams, tables and graphs throughout report
  - Contributed to Results section of the report
- Yuxuan Wang:
  - Implementing Residual CNN (ResNet) model
  - Training and collecting the loss and accuracy curves for the ResNet CNN model, as well as test accuracy (for Results)
  - Training part of the transformers model at learning rate = 0.001
  - Writing 'Results' and 'Discussion' sections of the report

6 References

- Ali, L., Alnajjar, F., Jassmi, H. A., Gocho, M., and Khan, W., & Serhani, M. (2021). Performance Evaluation of Deep CNN-Based Crack Detection and Localization Techniques for Concrete Structures. *Sensors*, 21, 1688. https://doi.org/10.3390/s21051688

- Ankit, U. (2022, June 28). *Transformer neural networks: A step-by-step breakdown.* Builtin. https://builtin.com/artificial-intelligence/transformer-neural-network

- Choubey, V. (2021, August 22). *Multiclass text classification using deep learning.* Medium. https://medium.com/analytics-vidhya/multiclass-text-classification-using-deep-learning-f25b4b1010e5

- Melchor, D. (2021). *A detailed guide to Pytorch's nn.Transformer() module.* Medium. https://towardsdatascience.com/a-detailed-guide-to-pytorchs-nn-transformer-module-c80afbc9ffb1

- Greene, D., & Cunningham, P.(2006). Practical solutions to the problem of Diagonal Dominance in Kernel Document Clustering. *Proc. International Conference on Machine Learning.* http://mlg.ucd.ie/files/publications/greene06icml.pdf

- Gültekin, H. (2020). *BBC News Archive.* Kaggle. https://www.kaggle.com/datasets/hgultekin/bbcnewsarchive

- Gültekin, H. (2021). *Categorizing News on the Website using NN (BBC).* Kaggle. https://www.kaggle.com/code/hgultekin/categorizing-news-on-the-website-using-nn-bbc

- Saigal, P., & Khanna, V. (2020). Multi-category news classification using Support Vector Machine based classifiers. *SN Applied Sciences, 2.* https://doi-org.myaccess.library.utoronto.ca/10.1007/s42452-020-2266-6

- Sharma, P., Shakya, A., Joshi, B., & Panday, S. P. (2021). Hierarchical multi label classification of news articles using RNN, CNN and HAN. *ICT with Intelligent Applications, 1*, 499-506. https://doi-org.myaccess.library.utoronto.ca/10.1007/978-981-16-4177-0_50

- Soyalp, G., Alar, A., Ozkanli, K., & Yildiz, B. (2021). Improving text classification with Transformer. *2021 6th International Conference on Computer Science and Engineering*, 707-712. https://doi.org/10.1109/UBMK52708.2021.9558906