Improving Scientific Literature Classification: A Parameter-Efficient Transformer-Based Approach

Mohammad Munzir Ahanger
University of Kashmir
Faculty of Applied Sciences, Department of Computer Sciences, Srinagar, India
munzir.scholar@kashmiruniversity.net

M. Arif Wani
University of Kashmir
Faculty of Applied Sciences, Department of Computer Sciences, Srinagar, India
awani@uok.edu.in

Abstract – Transformer-based models have been utilized in natural language processing (NLP) for a wide variety of tasks like summarization, translation, and conversational agents. These models can capture long-term dependencies within the input, so they have significantly more representational capabilities than Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). Nevertheless, these models require significant computational resources in terms of high memory usage, and extensive training time. In this paper, we propose a novel document categorization model, with improved parameter efficiency that encodes text using a single, lightweight, multiheaded attention encoder block. The model also uses a hybrid word and position embedding to represent input tokens. The proposed model is evaluated for the Scientific Literature Classification task (SLC) and is compared with state-of-the-art models that have previously been applied to the task. Ten datasets of varying sizes and class distributions have been employed in the experiments. The proposed model shows significant performance improvements, with a high level of efficiency in terms of parameter and computation resource requirements as compared to other transformer-based models, and outperforms previously used methods.

Keywords: deep learning, document categorization, text classification, scientific literature classification

1. INTRODUCTION

An ever-increasing amount of textual information in the form of research articles, books, conference proceedings, patents, and theses is produced and published every year. PubMed, a biomedical and life science literature search engine, lists more than 30,000 journals and more than 35 million citations [1]. As far back as 2009, the number of published journal articles surpassed 50 million [2] The value and utility of scientific literature depends upon the automatic organization and categorization into different subjects, domains, and themes. Text classification has proven to be an indispensable tool for the organization, curation, and retrieval of such textual data repositories.

Deep learning algorithms such as Convolution neural network (CNN) [3-6] and Recurrent neural network (RNN) [7, 8] based models have been used for text classification. The applications of these supervised deep learning models are numerous and varied, ranging from biometrics such as face recognition [9] and fingerprint recognition [10] to medical science [11, 12] and time series forecasting [13].

Recently more complex, transformer-based models have been applied [14-19]. These models outperform the other simpler models for tasks involving text classification. However, the performance improvements are at the cost of increased model size and complexity. Such complex models are required for good results in tasks such as translation and summarization. However, these models are inefficient when used in comparatively simpler tasks such as text classification. This inefficiency and model complexity result in issues such as higher computational demands, complex fine-tuning, model space complexity, interpretability issues, latency, and data requirements. Besides they may not be suitable for small datasets, consume substantial resources, and lack transparency.

In this study, we address the efficiency and model complexity issues associated with the transformer-based models in dealing with the text classification problem. The architecture choices for the transformer-based model are reconsidered to achieve parameter efficiency for text classification. A new efficient model is proposed and evaluated on the task of Scientific Literature Classification (SLC) and is compared with previously used models.
2. RELATED WORK

Deep learning has been applied in a variety of different applications of supervised learning [20]. Different approaches have been employed in text classification tasks [3-8]. Convolutional Neural network (CNN) based models apply filters of varying sizes on the input text to extract useful features. Such models vary in input representation, the number of CNN layers, and the number and size of filters in each layer. The model Text-CNN [3] trains a CNN over text represented as a matrix comprising pre-trained word vectors. The model uses both fine-tuned as well as pre-trained word embeddings. The authors in [21] demonstrate the advantages of using pre-trained word embeddings in text classification tasks. Multi-group norm constraint CNN [22] uses multiple such word embeddings to improve the performance of CNN-based algorithms. The authors apply the model to text classification tasks such as sentiment analysis, irony detection, and question-type detection. Self-attention mechanisms can be used to achieve improvements for such algorithms [23]. Inspired by the success of large models used in image classification tasks, a deeper CNN network that uses a character-based input text representation is employed by some authors [24, 25]. The model VDCNN [25] uses up to 30 layers to extract features from the input text. For text classification, the use of such deep models is costly and not necessary [26]. Set-CNN employs semantic extension and multi-channel convolution to improve classification performance [27].

Recurrent neural network-based models have been used in text classification tasks [7, 28]. Enhanced recurrent models such as BLSTM-2DCNN try to combine the best features of CNN and RNNs to obtain a better input representation [7, 8].

Hybrid models that combine CNN and RNN in different ways have been proposed, as those used in [29, 30, 31]. These models use a CNN layer to extract useful features followed by an RNN layer to obtain an enriched contextual representation. The authors in [29] use BiGRU (Bidirectional Gated Recurrent unit), a gated RNN while [31] uses LSTM (Long Short-Term Memory). An alternate approach is to use RNN followed by CNN. This approach has been employed by [7, 32]. The authors in [7] use two-dimensional max-pooling to the output of LSTM. The model BLSTM-C uses two layers of LSTM followed by a layer of CNN [32]. In such models, the LSTM layer outputs token representations based on the previously seen tokens. This representation is input into the convolutional layer(s) for feature extraction. Similar models are used in [33, 34, 35]. The authors in [36] propose enhancing RNN models by modifying the activation functions used.

To address the problems associated with CNN and RNN-based methods, specifically, the lack of interpretability and intuition, Attention-based models were proposed in [37]. Attention-based models have been employed for machine translation [37], visualizing important parts of a text [38]. HAN (Hierarchical Attention Net-
Here, $x_i$ is the dense word vector for the $i$th word in the input, $w_i$. $w_0$ is the word embedding matrix initialized using GloVe [44].

The positional embedding learns to encode the order of tokens in the input text as a linear transformation of the position within the input text as shown in (Eq. 2).

$$x_i^{\text{pos}} = w_i^T \cdot \text{pos}$$

Here, $x_i^{\text{pos}}$ is the positional embedding vector corresponding to the $i$th word in the input, $w_i$. $w_0$ is the positional embedding matrix. The positional embedding matrix is initialized as used in [15].

The two embeddings are combined to obtain a better, position-aware word embedding. The combined word embeddings are obtained by calculating the sum of the corresponding word and positional embeddings as shown in (Eq. 3).

$$x_i = x_i^{\text{word}} + x_i^{\text{pos}}$$

Here, $x_i$ is the combined word vector.

### 3.2.2. Light Weight encoder block

The proposed model approach uses a single efficient and lightweight encoder block to obtain token representations using the mechanism of self-attention. We project the word embeddings, to three different dense vector spaces using three projections – Query, Key, and Value which are linear transformations of the combined word vector (Eq. 3). This is shown in (Eq. 4), (Eq. 5), and (Eq. 6).

$$q_i = W_q \times x_i^T$$

$$k_i = W_k \times x_i^T$$

$$v_i = W_v \times x_i^T$$

Here, $v_i$ is the key vector corresponding to the $i$th word in the input text, $x_i$ is the combined word embedding vector and $W_v$ is the value projection matrix.

The encoder outputs an attention-based representation with the capability to attend to a specific piece of information from a potentially infinitely large context. The scaled dot product ($a_{ij}$) of a query vector ($q_i$) (representing the word being encoded) with a key vector ($k_j$) (representing another word) is treated as the attention score assigned to the keyword when interpreting the query word as shown in (Eq. 7).

$$a_{ij} = q_i^T \cdot k_j$$

$$\text{score}(q_i, k_j) = \frac{a_{ij}}{\sqrt{n}}$$

Here, $q_i$ is the query word (one that is being encoded) and $k_j$ is a keyword (one of the words in the input) whose attention score against $q_i$ is to be calculated. $n$ is the length of the query and key vectors.

Next, softmax is applied over scores for all key vectors for a query vector. This scaled set of scores ($a_{ij}$) is used to scale each value vector $v_i$. The scaled value vectors are added together to obtain $z_i$, the representation of the $i$th word in the input text. This operation is shown in (Eq. 9).

$$z_i = \sum_j (a_{ij} \otimes v_j)$$

We use residual connections [45] across the attention mechanism in the proposed model, allowing gradients to travel through them directly. The unattended representations ($\tilde{z}_i$) are added to the representation obtained as a result of the attention mechanism ($z_i$) to obtain the input for the next step, layer normalization ($n_i \in \mathbb{R}^{\text{embedding_dim}}$). This is shown in (Eq. 10).

$$n_i = z_i + \tilde{z}_i$$

The proposed model uses layer normalization as a means of regularization to reduce training time [46]. This can be represented as shown in (Eq. 11).
\[ p_i = \ln N_i; \theta_i (n_i) \] (11)

The normalized word representations are input to two successive fully connected neural network layers to generate the attention head output. The output vectors obtained from multiple attention heads are concatenated together and are subjected to a linear transformation layer to output a combined multiheaded word representation. A residual connection is used that bypasses the output i.e., \( p_i \) is added to \( r_i \) to obtain a combined output \( s_i \) as shown in (Eq. 12).

\[ s_i = p_i + r_i \] (12)

3.2.3. Classification Block

The combined output from all the heads is passed to a single linear transformation layer, followed by a normalization layer that generates the normalized vectors \( t_r \) as shown in (Eq. 13).

\[ t_i = \ln N_{Y_z; \theta_z} (W_0 \times s_i^T) \] (13)

Finally, the average overall positions represent the entire text as shown in (Eq. 14).

\[ u = \frac{\sum^n t_i}{n} \] (14)

The feature vector, \( u \), is finally input into a softmax activated fully connected layer of neurons whose outputs are treated as class probabilities as shown in (Eq. 15) and (Eq. 16).

\[ v = W_{out} \times u^T \] (15)

\[ \text{out}_i = \text{softmax}(v_i) = \frac{e^{v_i}}{\sum^m e^{v_j}} \] (16)

4. EXPERIMENTS AND RESULTS

The proposed model was applied to ten different datasets. For comparison, seven other deep-learning-based text classification models were also tested on the task. The following subsections describe the datasets and the model evaluation method used.

4.1. DATASETS

In this study, we used ten datasets comprising abstracts of scientific papers. Three Web of Science (WOS) datasets created and used by [6] were employed. In addition to these, seven new SLC datasets were created and used. We used Python libraries like Urllib, Lxml, and Beautiful Soup to obtain data. The datasets contain labeled abstracts with their associated categories and subcategories.

The three WOS datasets [6] include abstracts from 46958 publications and are categorized into 134 categories and 7 domains. The three COR datasets are derived from the ArXiv metadata repository released by Cornell University. The ArXiv dataset was gathered from the ArXiv [47]. The collection is divided into seven domains and contains 146 areas. The Nature dataset was gathered from the scientific paper repository - Nature [48]. It is divided into eight domains and contains 102 areas. The Springer dataset comprises metadata about 116230 published papers available from Springer [49]. The collection is divided into 24 domains and contains 117 areas. The Wiley dataset was gathered from the Wiley Online Library [50]. There are 494 areas in the collection, which are divided into 74 categories.

Table 1 describes the features of the datasets used in the study. These datasets vary in the number of domains, training and testing samples, mean number of words and characters per sample, and vocabulary size. These characteristics can impact the performance of text classification models trained on these datasets. The larger and more diverse the dataset, the better the model’s performance is likely to be. Table 2 lists the classes within each dataset.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Domains</th>
<th>Number of Abstracts</th>
<th>Training samples</th>
<th>Testing samples</th>
<th>Words/sample (Mean)</th>
<th>Chars/sample (Mean)</th>
<th>Vocabulary Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>WOS5376</td>
<td>3</td>
<td>5736</td>
<td>4588</td>
<td>1148</td>
<td>209.03</td>
<td>1386.13</td>
<td>42306</td>
</tr>
<tr>
<td>WOS11967</td>
<td>7</td>
<td>11967</td>
<td>8017</td>
<td>3950</td>
<td>201.43</td>
<td>1340.19</td>
<td>57875</td>
</tr>
<tr>
<td>WOS46985</td>
<td>7</td>
<td>46985</td>
<td>31479</td>
<td>15506</td>
<td>205.27</td>
<td>1375.76</td>
<td>125968</td>
</tr>
<tr>
<td>ArXiv</td>
<td>7</td>
<td>40060</td>
<td>32048</td>
<td>8012</td>
<td>148.27</td>
<td>978.67</td>
<td>112452</td>
</tr>
<tr>
<td>Nature</td>
<td>8</td>
<td>49782</td>
<td>24891</td>
<td>24891</td>
<td>175.20</td>
<td>1206.56</td>
<td>84228</td>
</tr>
<tr>
<td>Springer</td>
<td>24</td>
<td>116230</td>
<td>92984</td>
<td>23246</td>
<td>167.92</td>
<td>1128.41</td>
<td>22254</td>
</tr>
<tr>
<td>Wiley</td>
<td>74</td>
<td>179953</td>
<td>143962</td>
<td>35991</td>
<td>170.26</td>
<td>1150.62</td>
<td>113354</td>
</tr>
<tr>
<td>COR-8883</td>
<td>3</td>
<td>8883</td>
<td>7106</td>
<td>1777</td>
<td>135.70</td>
<td>889.15</td>
<td>28836</td>
</tr>
<tr>
<td>COR-61033</td>
<td>4</td>
<td>61033</td>
<td>48826</td>
<td>12207</td>
<td>127.54</td>
<td>782.714</td>
<td>52053</td>
</tr>
<tr>
<td>COR-233962</td>
<td>6</td>
<td>233962</td>
<td>187169</td>
<td>46793</td>
<td>148.24</td>
<td>971.305</td>
<td>172954</td>
</tr>
</tbody>
</table>
### Table 2. Classes in datasets

<table>
<thead>
<tr>
<th>Number Labels</th>
<th>Number</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArXiv, Comp Sci, Economics, EE&amp;SS, Math, Physics, Q Biology, Q Finance</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Wiley, Economics, Chaotic Dynamics, Algebraic Geometry</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>COR-61033, HEP - Experiment, HEP - Lattice, Nuclear Exp, Quantitative Finance</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

### 4.2. EXPERIMENTAL SETUP

This subsection describes the experimental setup used in this study. The experiments were performed using Python v3.9.1 [51] using Keras v2.3.1 [52] API for Tensorflow 2.0 [53]. Python libraries such as the lxml v4.6.4 [54], and beautiful soup v4.10.0 [55] were utilized to acquire some of the datasets from websites such as ArXiv, Wiley, Springer, and Nature.

The datasets were cleaned by filtering out special characters. Tokenization was then done using a dictionary size of 20000. A constant length of 250 words was ensured using padding and truncation as needed. The datasets underwent a random shuffle to eliminate any ordering bias, and subsequently, a standard (80-20) rule was used to split them into training and testing subsets. In the proposed model, we utilized the Adam optimizer, with a learning rate of 0.01 and a batch size of 16. All the models were trained for 20 epochs. (Fig. 2) shows the training graphs for the proposed model on two different datasets.

### 4.3. RESULTS

We compare the performance of the proposed model with previously applied deep learning-based text classification methods based on classification accuracy percentages. This subsection presents the results of the experiments conducted to assess the efficacy of the proposed model and to compare its performance against other deep learning-based models that have previously been applied to the task of SLC.

Table 3 compares the accuracy of the proposed model with state of art models on seven different datasets. The performance can be depicted visually as shown in (Fig. 3). Model training times are listed in Table 4.

### Table 3. Comparison with previously used models on classification accuracy (%) metrics

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters (M)</td>
<td>25.6</td>
<td>33.7</td>
<td>11.5</td>
<td>16.9</td>
<td>14.35</td>
<td>34.9</td>
<td>11.32</td>
<td>11.9</td>
</tr>
<tr>
<td>WOS-5736</td>
<td>49.46</td>
<td>97.41</td>
<td>88.48</td>
<td>97.07</td>
<td>82.9</td>
<td>96.47</td>
<td>97.82</td>
<td>99.13</td>
</tr>
<tr>
<td>WOS-11967</td>
<td>16.55</td>
<td>92.94</td>
<td>25.45</td>
<td>92.94</td>
<td>67.64</td>
<td>93.52</td>
<td>93.98</td>
<td>96.17</td>
</tr>
<tr>
<td>WOS-40896</td>
<td>31.13</td>
<td>89.33</td>
<td>76.59</td>
<td>88.2</td>
<td>75.76</td>
<td>88.67</td>
<td>90.45</td>
<td>95.71</td>
</tr>
<tr>
<td>ArXiv</td>
<td>26.26</td>
<td>85.35</td>
<td>80.53</td>
<td>83.32</td>
<td>66.11</td>
<td>85.84</td>
<td>84.26</td>
<td>87.11</td>
</tr>
<tr>
<td>Springer</td>
<td>27.7</td>
<td>100</td>
<td>61.6</td>
<td>85.0</td>
<td>63.0</td>
<td>99.98</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Nature</td>
<td>33.79</td>
<td>57.23</td>
<td>51.23</td>
<td>52.21</td>
<td>53.44</td>
<td>60.12</td>
<td>61.83</td>
<td>61.56</td>
</tr>
<tr>
<td>Wiley</td>
<td>8.15</td>
<td>52.04</td>
<td>7.82</td>
<td>62.10</td>
<td>49.50</td>
<td>50.81</td>
<td>43.39</td>
<td>72.24</td>
</tr>
<tr>
<td>COR-8883</td>
<td>88.74</td>
<td>91.64</td>
<td>97.97</td>
<td>91.87</td>
<td>67.24</td>
<td>93.35</td>
<td>92.90</td>
<td>94.25</td>
</tr>
<tr>
<td>COR-61033</td>
<td>90.06</td>
<td>90.66</td>
<td>93.38</td>
<td>92.38</td>
<td>28.16</td>
<td>94.67</td>
<td>92.46</td>
<td>93.41</td>
</tr>
<tr>
<td>COR-233962</td>
<td>87.23</td>
<td>91.93</td>
<td>90.725</td>
<td>92.31</td>
<td>56.44</td>
<td>90.2</td>
<td>91.55</td>
<td>93.37</td>
</tr>
</tbody>
</table>
Fig. 2. Training graphs of the proposed model on two datasets WOS-46985 and Wiley

Table 4. Comparison with training times with previously used models (in seconds)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>TextCNN</th>
<th>MGNCNN</th>
<th>CharCNN</th>
<th>RCNN</th>
<th>VDCNN</th>
<th>HDLtex-CNN</th>
<th>HDLtex-RNN</th>
<th>sBERT</th>
</tr>
</thead>
<tbody>
<tr>
<td>WOS5736</td>
<td>73.152</td>
<td>3631.1</td>
<td>158.12</td>
<td>501.49</td>
<td>269.139</td>
<td>140.4653</td>
<td>166.1034</td>
<td>157.2688</td>
</tr>
<tr>
<td>WOS11967</td>
<td>106.231</td>
<td>5498.2</td>
<td>221.28</td>
<td>716.46</td>
<td>405.123</td>
<td>206.5897</td>
<td>708.5794</td>
<td>286.5306</td>
</tr>
<tr>
<td>WOS46985</td>
<td>333.494</td>
<td>10805.6</td>
<td>771.904</td>
<td>2644.6</td>
<td>1267.18</td>
<td>482.8745</td>
<td>2948.638</td>
<td>604.8373</td>
</tr>
<tr>
<td>arXiv</td>
<td>341.412</td>
<td>15226.</td>
<td>450.240</td>
<td>771.23</td>
<td>687.927</td>
<td>330.3308</td>
<td>831.7495</td>
<td>500.1951</td>
</tr>
<tr>
<td>Nature</td>
<td>243.161</td>
<td>10844.7</td>
<td>386.278</td>
<td>572.34</td>
<td>407.972</td>
<td>271.8696</td>
<td>639.7405</td>
<td>361.6975</td>
</tr>
<tr>
<td>Springer</td>
<td>776.393</td>
<td>34626.14</td>
<td>880.364</td>
<td>2096.1</td>
<td>788.691</td>
<td>549.7363</td>
<td>2251.08</td>
<td>721.5509</td>
</tr>
<tr>
<td>Wiley</td>
<td>1222.11</td>
<td>54054.59</td>
<td>1077.36</td>
<td>3002.8</td>
<td>8974.24</td>
<td>1845.265</td>
<td>3309.169</td>
<td>1333.135</td>
</tr>
<tr>
<td>COR-8883</td>
<td>94.99683</td>
<td>4236.737</td>
<td>186.051</td>
<td>457.67</td>
<td>555.342</td>
<td>127.8164</td>
<td>696.8934</td>
<td>179.0955</td>
</tr>
<tr>
<td>COR-61033</td>
<td>464.651</td>
<td>20722.85</td>
<td>670.703</td>
<td>1324.9</td>
<td>1819.57</td>
<td>434.1656</td>
<td>906.9122</td>
<td>527.6306</td>
</tr>
<tr>
<td>COR-233962</td>
<td>1509.796</td>
<td>67334.97</td>
<td>1773.456</td>
<td>4780.1</td>
<td>8909.46</td>
<td>1510.261</td>
<td>2864.248</td>
<td>1493.209</td>
</tr>
</tbody>
</table>

Fig. 3. Classification accuracy performance of different models

5. DISCUSSION

This section provides an analysis and discussion of the experimental results, comparing and contrasting the performance of the various models under consideration. From the experimental results, it follows that models such as TextCNN capture local n-gram features and ignore long-range dependencies within the input, resulting in poor performance on the task. Models such as HDLtex CNN use a deeper network architecture with a wider range of filter sizes. Although this improves the results, the models still do not capture long-range dependencies within the text.

Models like VDCNN and CharCNN detect character n-grams rather than words and phrases within the text. The models need to be deeper, which increases the time required to train and infer. Models such as RCNN treat text as a sequence of tokens and attempt to capture long-range dependencies within the text by finding contextual representations. This is limited by the problem of vanishing gradients. HDLtex RNN uses LSTM to maintain an internal cell state and a system of gates to maintain information over an extended number of timesteps. These models are slow to train and infer because of their sequential nature.
Transformer-based models make it possible to attend to a potentially infinitely long context while encoding a given position within a text. These models, however, suffer from a large parameter space, leading to inefficiency in tasks like text classification.

The proposed model employs a single, parameter-efficient encoder block. The parameter efficiency achieved facilitates faster training and utilization. Moreover, the model significantly reduces resource requirements, making it feasible to train and deploy in low-resource environments. The proposed model outperforms previously used methods on the task.

6. CONCLUSION

Text classification, particularly in scientific literature, holds significant importance. While Transformer-based models have revolutionized NLP, applying them to text classification results in parameter-inefficient models with considerable space and time complexities. The proposed model addresses these issues by re-evaluating the architectural choices of Transformer-based models while prioritizing parameter efficiency. As a result, the model surpasses the performance of previously employed deep learning-based methods in the task of SLC, with only a fraction of the parameter space required by Transformer-based models like BERT. However, it is essential to recognize that our approach’s applicability extends beyond scientific literature classification. To gauge its full potential and limitations, future research should involve its evaluation in other related tasks such as sentiment analysis, toxicity detection, and similar domains. Assessing the model’s performance in diverse contexts will provide a comprehensive understanding of its capabilities and guide further refinements.

7. REFERENCES


[52] Keras, keras.io/api (accessed: 2020)

