Article

Performance Study on Extractive Text Summarization Using BERT Models

Shehab Abdel-Salam * and Ahmed Rafea

Computer Science Department, The American University in Cairo, AUC Avenue, Cairo 11835, Egypt; rafea@aucegypt.edu
* Correspondence: shehab_m@aucegypt.edu

Abstract: The task of summarization can be categorized into two methods, extractive and abstractive. Extractive summarization selects the salient sentences from the original document to form a summary while abstractive summarization interprets the original document and generates the summary in its own words. The task of generating a summary, whether extractive or abstractive, has been studied with different approaches in the literature, including statistical-, graph-, and deep learning-based approaches. Deep learning has achieved promising performances in comparison to the classical approaches, and with the advancement of different neural architectures such as the attention network (commonly known as the transformer), there are potential areas of improvement for the summarization task. The introduction of transformer architecture and its encoder model "BERT" produced an improved performance in downstream tasks in NLP. BERT is a bidirectional encoder representation from a transformer modeled as a stack of encoders. There are different sizes for BERT, such as BERT-base with 12 encoders and BERT-larger with 24 encoders, but we focus on the BERT-base for the purpose of this study. The objective of this paper is to produce a study on the performance of variants of BERT-based models on text summarization through a series of experiments, and propose "SqueezeBERTSum", a trained summarization model fine-tuned with the SqueezeBERT encoder variant, which achieved competitive ROUGE scores retaining the BERTSum baseline model performance by 98%, with 49% fewer trainable parameters.

Keywords: extractive summarization; deep learning models; recurrent neural networks; supervised learning; transformers; BERT; DistilBERT; SqueezeBERT

check for updates

Citation: Abdel-Salam, S.; Rafea, A. Performance Study on Extractive Text Summarization Using BERT Models. *Information* **2022**, *13*, 67. https://doi.org/10.3390/info13020067

Academic Editor: Kostas Stefanidis

Received: 24 December 2021
Accepted: 26 January 2022
Published: 28 January 2022

1. Introduction

Automatic text summarization is an active research area that can be defined as the process of extracting important sentences or snippets of a large document and combining them into a short version of the document. Summarizing a text can be both time-efficient and cost-efficient. In terms of time efficiency, the human reader can spend less time reading a document by reading a summarized version that grasps the key points of the document. Newsgroups can use a document summarization tool on several documents to gather the important information of each document that discusses the same topic in a shorter version. In terms of cost efficiency, summarization can be used for compressing the amount of textual data being transferred from a device to another device. It would be beneficial for the user to have the choice of reading a summarized version of a document or an article before deciding to download the whole document or article to read. With the current rate of data growth, it will soon become a necessity to have a tool that generates shorter versions of textual data as a service to the human reader.

The task of automatic text summarization is composed of mainly three phases: the data pre-processing phase, algorithmic processing phase, and post-processing phase.

1.1. Data Pre-Processing Phase

It is the process of cleaning and transforming the raw source document to a more compatible data format prior to summarization. Examples of data pre-processing techniques are:

(1) Removal of noise data found in the document;
(2) Sentence and word tokenization;
(3) Removal of punctuation marks;
(4) Removal of stop words to remove frequently occurring words, such as (a), (an), (the), etc.;
(5) Word stemming, which is the removal of suffixes and prefixes;
(6) Word lemmatization, which is the transformation of a word to its base structure, such as transforming the word "playing" to "play";
(7) Part-of-speech tagging.

1.2. Algorithmic Processing Phase

It is the process of applying an algorithmic approach when generating a summary from the pre-processed document. Algorithmic processing includes either extractive or abstractive summarization, which will be further discussed later in this paper; however, extractive techniques are favored rather than abstractive techniques [1], since the former has shown better performance and is relatively easier to implement.

1.3. Post-Processing Phase

It is the process of applying any data transformation when generating the target summary. This phase can be optional to some approaches as we will cover in the literature.

The objective of this paper is to study the performance of BERT-based models for the extractive summarization task, which will be further explained in Section 3. Section 2 covers the literature of automatic text summarization, including the different approaches used. Section 3 discusses the methodology, which describes our experiments in fine-tuning BERT-based variants, such as DistilBERT and SqueezeBERT, for summarization tasks. Section 4 describes our observations and the results of the experiments conducted. Section 5 concludes our work and describes the possible next steps.

2. Literature Review

Text summarization can be split into two types of summaries: extractive and abstractive. An extractive summary is formed by reusing portions of the original document, such as sentences, and combining them into a summary. To generate an extractive summary, each sentence is ranked based on the most salient information and then reordered together into a summary while preserving the grammatical rules. SumaRuNNer [2] is one of several studies that perform extractive summarization. An abstractive summary is formed by interpreting the original document and generating meaningful sentences in a shorter version. This approach involves more semantic understanding of the document by the model to write a summary in a human-understandable form. An example of such an approach was proposed by Summarist paper which has modules that perform that type of summarization [3].

Throughout the literature, there have been different approaches applied to generate an extractive summary, such as the frequency count-based approach by Luhn [4], or the graph-based approach by TextRank [5], which represents a document as a graph of sentences where the edges between the sentences are connected based on a similarity measurement. LexRank [6] is another graph-based approach based on the concept of eigenvectors. Latent Semantic Analysis, or LSA [7], is a statistical-based approach that tries to find sentences in the document by applying Singular Value Decomposition over document matrix D of size $\mathrm{m} \times \mathrm{n}$, where m is the number of sentences and n is the number of terms. SumBasic [8] is a greedy search approximation approach that uses a frequency-based sentence to set the weights of word probabilities to minimize redundancy.

2.1. Summarization Approaches

In 2016, Cheng and Lapata proposed an attentional encoder–decoder architecture for an extractive single-document summarization trained on the CNN/DailyMail corpus [9]. Their model is based on an encoder–decoder approach, where the encoder learns the representation of sentences and documents while the decoder classifies each sentence based on the encoder's representation using an attention mechanism [10]. They first obtained representation vectors at the sentence level using a single-layer convolutional neural network effectively without long-term dependencies. Afterwards, they built representations for documents using a recurrent neural network that recursively composes sentences [9]. The first step is called the Convolutional Sentence Encoder and the second step is called the Recurrent Document Encoder. The third step is the Sentence Extractor, which is another recurrent neural network that applies the attention mechanism to directly extract salient sentences after reading them. The labeling decision is made with both the encoded document and the previously labeled sentences in mind. After this sequence labeling task, the fourth step is the Word Extractor. This task is responsible for generating the next word in the summary using a hierarchical attention architecture and computes the probability of the next word to be included in the summary [9].

In 2019, Joshi, Eduardo, Enrique, and Laura proposed SummCoder [11], an unsupervised framework for extractive text summarization based on deep auto-encoders.

In their approach, the extractive summarization problem was defined as a sentence selection problem given a document. The determination of which sentence should be included in the summary was based on three metrics:

1. Sentence Content Relevance Metric;
2. Sentence Novelty Metric;
3. Sentence Position Relevance Metric.

This proposed framework ranks the sentences based on the three metrics above, and formulates the problem as follows: given a document D with N sentences $D = (S_1, S_2, \ldots, S_N)$, the sentence I is embedded into a vector V_{Si}, which is the encoder representation computed with the three metrics mentioned before, and then decoded afterwards.

In 2019, Yue Dong et al. proposed a novel method for training neural networks to perform single-document extractive summarization without heuristically generated extractive labels. They call their approach BanditSum, as it treats extractive summarization as a contextual bandit problem where the model receives a document to summarize, referred to as the context, and then chooses a sequence of sentences to include in the summary, referred to as the action [12]. Their paper applies extractive summarization using the policy-gradient reinforcement learning approach. A bandit is a decision-making formalization in which an agent repeatedly chooses one of several actions and receives a reward based on this choice. For the extractive summarization, each document is labeled as the context and each ordered subset of the document's sentences is a different action. The agent is required to learn from a series of actions, determining which one will generate the highest reward. This approach was trained on a CNN/DM dataset, and the authors concluded in their work that BanditSum performs significantly better than other competing approaches when good summary sentences appear late in the source document.

W.S. El-Kassas et al. proposed a graph-based framework in 2020, combining four extractive algorithms called EdgeSumm [13]. EdgeSumm includes graph-based, statistical-based, semantic-based, and centrality-based approaches, which has been reported to achieve performance scores higher than state-of-the-art systems in ROUGE-1 and ROUGE-2. Their proposed approach is meant to solve the summarization problem by introducing a generic framework for summarizing domain-independent documents. It first constructs a text graph model based on the output of the pre-processing step, and then calculates the weights for each node in the graph, which is based on the word frequency besides the occurrence of the word in the title and other important factors. Afterwards, a search graph algorithm is applied to try to find the phrases or edges that link the most frequent words or topics together, and the output of the algorithm is a list of selected edges for each sentence. Afterwards, a candidate summary algorithm is applied, which selects sentences to be included in the summary based on which sentence has at least one edge selected from the candidate edge list from the search graph algorithm output. EdgeSumm was applied on standard datasets, such as DUC2001 and DUC2002, and it outperformed the state-of-the-art text summarization systems for the ROUGE-1 and ROUGE-L metrics in the DUC2002 dataset.

In 2019, Yang Liu proposed a fine-tuned BERT summarization approach called BERTSum [14]. BERT is a stack of pre-trained transformer encoders that can better understand the textual data through attention mechanisms, making it a contextualized language model that can be used for various downstream tasks. In his methodology, he used BERT to output representation for each sentence, since BERT was trained as a masked-language model, and then he modified the input sequence and embeddings of BERT to make extractive summarization possible. Figure 1 shows the complete architecture of BERTSum model, in which he inserted a classification token [CLS] before each sentence and a [SEP] token after each sentence, and then introduced interval segment embedding to distinguish between different sentences in the segment embedding layer.

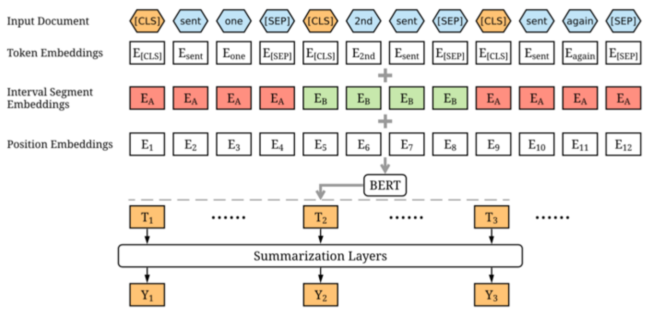
Input Document
[CLS]
sent
one
[SEP]
[CLS]
2nd
sent
[SEP]
[CLS]
sent
again
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{sent} E_{one} $E_{[SEP]}$ $E_{[CLS]}$ E_{2nd} E_{sent} $E_{[SEP]}$ $E_{[CLS]}$ E_{sent} E_{again} $E_{[SEP]}$
Interval Segment Embeddings
E_A E_A E_A E_A E_B E_B E_B E_B E_A E_A E_A E_A
Position Embeddings
E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10} E_{11} E_{12}
BERT
T_1 T_2 T_3
Summarization Layers
Y_1 Y_2 Y_3

Figure 1. BERTSum Architecture.

The output of the BERT layer is the contextualized embeddings, which steps into several summarization-specific layers to capture document-level features for extracting summaries. The author calculates the final predicted score Y for each sentence S_i from 1 to N, where N is the number of sentences, and then computes the binary classification entropy as the loss function to classify whether S_i should be included in the summary or not [14].

In 2020, Ming Zhong et al. presented a paper that tackled extractive text summarization as a text matching problem, called MatchSum [15]. Their approach formulated the problem as a semantic text matching problem in which a source document and candidate summaries will be matched in a semantic space, and the candidate summary closest to the reference summary in that space will be selected as the output summary. The authors have conducted experiments on five datasets which demonstrate the effectiveness of the matching framework, and they believe the power of this matching-based summarization framework has not been fully exploited yet.

2.2. ROUGE Evaluation Algorithm

Generating a summary does not have an absolute correct answer. Each summary generated by a human reader differs from another human reader based on what he or she perceives as important information to include. In text summarization, an evaluation metric to use is Recall-Oriented Understudy for Gisting Evaluation, or in short, ROUGE [16]. Rouge-N is an N-gram recall measure between the human summary and the generated summary. It measures the overlap between the golden or human summary and the generated summary. There are different variants of ROUGE metric depending on how many overlapped words are measured. Examples are, unigram (ROUGE-1), bigram (ROUGE-2), trigram (ROUGE-3), or longest common sequence of words (ROUGE-L).

3. Methodology

This section describes the objective of each experiment to study the summarization performance when fine-tuning the model with variants of the BERT encoder [17]. This section also discusses the different architectures that have been trained and evaluated for extractive summarization. We use BERTSum as our baseline model and modified the source code, which is available in the BERTSum paper [18]. Here is our modified source code, used for conducting the following experiments below, which can be found at https://github.com/ShehabMMohamed/PreSumm, accessed on 25 November 2021.

3.1. Experiment: Exploring DistilBERT Model

The objective of this experiment is to explore DistilBERT, a compressed variant of BERT [16], which is used to reduce the computation cost of training an extractive summarizer, and produce a summary while maintaining the model's performance to be deployed on low-resource devices. The objective is to modify BERTSum architecture to replace the BERT-base encoder with the DistilBERT encoder, re-train the model, and record the ROUGE scores. This experiment ran on the available GPU resource on Google Colab notebook.

DistilBERT aims to leverage knowledge distillation during the pre-training phase and show that it is possible to reduce the size of a BERT model by 40% while retaining 97% of its language understanding capabilities and being 60% faster. Knowledge distillation was first introduced by Bucila et al. in 2006 [19–21], which is a compression technique in which a compact model called "the student" is trained to reproduce the behavior of a larger model called "the teacher".

The architecture of DistilBERT is introduced as the student that has the same general architecture as BERT the teacher, except for the following changes:

1. The token-type embeddings and the pooler are moved;
2. The number of layers is reduced by a factor of 2.

3.2. Experiment: Exploring SqueezeBERT Model

The objective of this experiment is to explore SqueezeBERT, a model proposed as an efficient network [22,23] that reduces the computationally expensive training of complex architectures, such as BERT. The SqueezeBERT architecture uses grouped convolutions as a computation replacement instead of the fully connected layers for the Q, K, V, and FFN layers in the BERT encoder. This paper discusses an observation that grouped convolutions have yielded significant speedups for computer vision networks. They demonstrate a replacement of several operations in self-attention layers with grouped convolutions and proposed this novel network architecture called SqueezeBERT which reportedly runs 4.3$\times$ faster than BERT-base. In this experiment, we modify BERTSum to replace the BERT-base encoder with SqueezeBERT, train the model, and record the ROUGE scores. This experiment runs on the available GPU resource by Google Colab notebooks.

4. Experiments & Results

4.1. Training Fine-Tuned DistilBERT Summarizer

This experiment shows an effort to explore the DistilBERT model for extractive summarization tasks. The model was trained using the adam optimizer with a learning rate 2×10^{-3}, with a batch size of 3000, and a 10% dropout. Table 1 shows the ROUGE scores of the fine-tuned summarizer trained on 30,000 epochs on the CNN/DM dataset. Every 10,000 epochs, we save the model weights and generate the ROUGE-1, ROUGE-2, and ROUGE-L validation scores. The performance of the model before 10,000 epochs was relatively low, so we limited saving the model weights by starting with the 10,000 as our first checkpoint. The results in Table 1 record the output of the evaluation algorithm, where the summary is evaluated by computing the value of precision, recall, and F-score. Average R represents the average weight of recall, Average P represents the average weight of precision, and Average F represents the average weight of the F-measure. The training time taken for a DistilBERT summarizer was around 25 min per 1000 checkpoints on a Google GPU session.

Table 1. Training DistilBERT on 3 checkpoints.

DistilBERT Summarizer	Average R Average P Average F		
	ROUGE-1	**ROUGE-2**	**ROUGE-L**
Model at 10,000	0.54046 0.37102 0.42535	0.24811 0.17079 0.19527	0.49321 0.33920 0.38856
Model at 20,000	0.52756 0.37355 0.42238	0.24105 0.17152 0.19324	0.48198 0.34193 0.38631
Model at 30,000	0.50242 0.36837 0.41015	0.22208 0.16356 0.18153	0.45819 0.33656 0.37443

During the training phase, the model generated a relatively higher score at checkpoint 10,000 when compared to the same model of different checkpoints. By computing the average of ROUGE-1, ROUGE-2, and ROUGE-L, the DistilBERT summarizer generated competitive results that are slightly less when compared with the BERT-base model in Table 2. DistilBERT has a ROUGE-1 score lower than BERT-base by 1.6%, a ROUGE-2 score lower than BERT-base by 3.5%, and ROUGE-L score lower than BERT-base by 1.9%. However, the number of parameters with the DistilBERT model is less than the BERT model by ~36%, which is directly proportional to the training time needed for these models. The DistilBERT summarizer was trained on a single GPU for almost 3 days and was able to retain approximately 98% of the baseline model in terms of summary generation performance, with a decrease of ~36% in size.

Table 2. DistilBERT Performance with BERT-baseline.

BERT Models	ROUGE Scores			Params
	ROUGE-1	**ROUGE-2**	**ROUGE-L**	
BERT-base	43.23	20.24	39.63	120.5 M
DistilBERT	42.54	19.53	38.86	77.4 M

4.2. Training Fine-Tuned SqueezeBERT Summarizer

This experiment shows an effort to explore the SqueezeBERT model, a recently developed model based on grouped convolutions instead of attention networks, as an inspiration from the CV research. To run this experiment, I have used the publicly available single GPU provided by Google Colab to train this model, which has taken around ~60 h to complete. Table 3 shows the output of training the model on a total of 30,000 epochs. The hyperparameters of this model training is identical to the DistilBERT experiment using the adam optimizer with a learning rate 2e-3, with a batch size of 3000, and a 10% dropout. The training time taken for a SqueezeBERT summarizer was around 20 min per 1000 checkpoints on a Google GPU session.

By computing the average of ROUGE-1, ROUGE-2, and ROUGE-L, the SqueezeBERT summarizer generated competitive results that were slightly less when compared with the BERT-base model. By observing the SqueezeBERT experiment, the model generated ROUGE scores nearly identical or slightly better than the previous experiment with the DistilBERT model. This will be further discussed in the overall discussion. In Table 4 below, SqueezeBERT has a ROUGE-1 score lower than BERT-base by 1.6%, a ROUGE-2 score lower than BERT-base by 3.35%, and a ROUGE-L score lower than BERT-base by 1.8%. However, the number of parameters with the SqueezeBERT model is less than the BERT model by ~49%, which is directly proportional to the training time needed for these models. One key finding here to notice is that SqueezeBERT summarizer was able to retain approximately 98% of the baseline model in terms of summarization performance, with a decrease of ~49% in size. This experiment has demonstrated high-performance retention when training a summarizer with an architectural size that is almost half the size of the BERT-base model.

Table 3. Training SqueezeBERT on three checkpoints.

SqueezeBERT Summarizer	**Average R Average P Average F**		
	ROUGE-1	**ROUGE-2**	**ROUGE-L**
Model at 10,000	0.52760 0.37796 0.42538	0.24278 0.17427 0.19563	0.48227 0.34612 0.38922
Model at 20,000	0.50638 0.37708 0.41699	0.22715 0.16975 0.18717	0.46294 0.34537 0.38162
Model at 30,000	0.48312 0.36130 0.39832	0.20655 0.15517 0.17044	0.43988 0.32963 0.36306

Table 4. SqueezeBERT Performance with BERT-baseline.

BERT Models	**ROUGE Scores**			**Params**
	ROUGE-1	**ROUGE-2**	**ROUGE-L**	
BERT-base	43.23	20.24	39.63	120.5 M
SqueezeBERT	42.54	19.56	38.92	62.13 M

5. Conclusions

This section covers the conclusion of the experiments conducted in this paper, as well as the contribution of this paper, and the possible next steps for future work.

5.1. Experiment Conclusions

Given the results of the experiments, there is great potential to further adopt computer vision-based techniques on NLP tasks given the performance of the SqueezeBERT summarization model. Given the summarization performance and reduction of the trainable parameters in Table 5, the SqueezeBERT summarizer can be productionized and deployed for real-time summary generation given the compression of the architecture, instead of deploying the original BERT-base summarizer which consists of ~120 million parameters, whereas the proposed model consists of ~62 million parameters.

Table 5. Comprehensive extractive summarization performance.

BERT Models	**ROUGE Scores**			**Params**
	ROUGE-1	**ROUGE-2**	**ROUGE-L**	
BERT-base	43.23	20.24	39.63	120.5 M
DistilBERT	42.54	19.53	38.86	77.4 M
SqueezeBERT	42.54	19.56	38.92	62.13 M

We have trained the BERT-baseline and used it as a benchmark for what are known as “compressed models”, such as DistilBERT and SqueezeBERT. Post-training, these summarization models all retain performance levels above >90% of the baseline model, which has around 120.5 million parameters to train. The first experiment was an attempt to introduce a distillation-based BERT model that has a lower number of parameters by ~35%, and when testing the model it retained ~98% of the BERT model. The second experiment was to train a more efficient model called SqueezeBERT, which replaced all the attention layers with grouped convolutional layers for efficient computation. This experiment yielded an interesting observation, namely that SqueezeBERT maintains the same performance as DistilBERT with unigrams, and has slightly better performances with bi-grams and longest common sequences. SqueezeBERT retains 98% of the performance of the baseline model, with 49% fewer parameters (consisting of 62.13 million parameters instead of the baseline model which has 120.5 million parameters). SqueezeBERT is a good candidate for training a summarizer nearly half the size of the original model with minimal downgrades in summarization performance. It also gave a key observation, namely that by using efficient networks inspired by computer vision literature [24–26], such as grouped convolutional layers, it can improve NLP downstream tasks, and for this paper, it improved the summarization task by reducing the training time while retaining performance. This observation concludes the result of the performance study of BERT-based variants that is aimed at expanding the literature in the summarization task.

5.2. Contribution

Given the outcomes of the experiments in the methodology, there is a potential productionized version of the SqueezeBERT extractive summarizer from the results recorded above. SqueezeBERT has fewer parameters than DistilBERT by approximately 20% and yields the same ROUGE-1 score, while yielding slightly higher ROUGE-2 and ROUGE-L scores. Although SqueezeBERT and DistilBERT produce slightly lower scores in the BERT-baseline model, SqueezeBERT has an advantage of having less training time and fewer parameters than the baseline model by ~48.44%, which is close to half of the BERT-baseline model's trained parameters. Table 6 shows the comprehensive comparison of the two compressed models with the BERT-baseline model by recording the performance retention of each ROUGE metric and the decrease percentage of the total parameters of the baseline model.

Table 6. Performance comparison with BERTSum baseline model.

	Performance Retention % ROUGE-1	Performance Retention % ROUGE-2	Performance Retention % ROUGE-L	Parameters Reduction %
DistilBERT	98.4	96.49	98.05	35.77
SqueezeBERT	98.4	96.64	98.2	48.44

5.3. Future Work

These experiments were conducted on a single GPU resource from Google Colab, and although it was sufficient for the experiments above, there is some room for additional work, such as hyperparameter tuning these fine-tuned models, to generate better summarization performance. Another possible future work is to train these models on domain-specific datasets and produce an extractive summarizer dedicated to specific use cases, such as a medical or academic extractive summarizer. Further possible future work is exploring the potential of fine tuning the SqueezeBERT model for abstractive summarization instead of extractive summarization, and reporting on the performance results if there are any significant findings. To further reduce the size of the pre-trained model, there is also room for adopting model compression techniques, such as quantization and pruning.

Quantization is a family of techniques which aims to reduce the number of bits required to store each parameter and/or activation in a neural network, while at the same time maintaining the accuracy of that network. This technique has been applied in different NLP studies [27,28].

Pruning aims to directly eliminate certain parameters from the network while also maintaining accuracy, thereby reducing the storage and potentially computational cost of that network; for an application of this NLP, this research demonstrates an application of pruning [28].

Author Contributions: Conceptualization, S.A.-S. and A.R.; methodology, S.A.-S. and A.R.; software, S.A.-S. and A.R.; validation, S.A.-S. and A.R.; formal analysis, S.A.-S. and A.R.; investigation, S.A.-S.; resources, S.A.-S. and A.R.; data curation, S.A.-S. and A.R.; writing—original draft preparation, S.A.-S. writing—review and editing, S.A.-S. and A.R.; visualization, S.A.-S. and A.R.; supervision, A.R.; project administration, S.A.-S. and A.R.; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: Shehab Abdel-Salam would like to acknowledge Ahmed Rafea for the guidance and knowledge that he provided throughout the length of this paper and for giving the inspiration to research in this domain and write such a paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Gambhir, M.; Gupta, V. Recent automatic text summarization techniques: A survey. *Artif. Intell. Rev.* **2016**, *47*, 1–66. [CrossRef]
2. Nallapati, R.; Zhai, F.; Zhou, B. Summarunner: A Recurrent Neural Network-Based Sequence Model for Extractive Summarization of Documents. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; pp. 3075–3081.
3. Hovy, E.; Lin, C.Y. *Automated Text Summarization in SUMMARIST*; MIT Press: Cambridge, MA, USA, 1999; pp. 81–94.
4. Luhn, H.P. The Automatic Creation of Literature Abstracts. *IBM J. Res. Dev.* **1958**, *2*, 159–165. [CrossRef]
5. Mihalcea, R.; Tarau, P. TextRank: Bringing Order into Texts. In Proceedings of the Empirical Methods on Natural Language Processing, Barcelona, Spain, 1 July 2004; pp. 404–411.
6. Erkan, G.; Radev, D.R. LexRank: Graph-based Lexical Centrality as Salience in Text Summarization. *J. Artif. Intell. Res.* **2004**, *22*, 457–479. [CrossRef]
7. Steinberger, J.; Jezek, K. Using Latent Semantic Analysis in Text Summarization and Summary Evaluation. In Proceedings of the 7th International Conference on Information System Implementation and Modeling, New York, NY, USA, 23–24 April 2004; pp. 93–100.
8. Nenkova, A.; Vanderwende, L. The Impact of Frequency on Summarization. In *Technical Report, Microsoft Research (MSR-TR-2005-101)*; Columbia University: New York, NY, USA, 2005.
9. Cheng, J.; Lapata, M. Neural Summarization by Extracting Sentences and Words. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Berlin, Germany, 7–12 August 2016; Association for Computational Linguistics: Stroudsburg, PA, USA, 2016.
10. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention is all you need. In Proceedings of the Conference on Neural Information Processing Systems (NeurIPS), Long Beach, CA, USA, 4–9 December 2017.
11. El-Kassas, W.S.; Salama, C.R.; Rafea, A.A.; Mohamed, H.K. EdgeSumm: Graph-based framework for automatic text summarization. *Inf. Process. Manag.* **2020**, *57*, 102264. [CrossRef]
12. Lin, C.Y. Rouge: A Package for Automatic Evaluation Of Summaries. In Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004), Barcelona, Spain, 25–26 July 2004; pp. 25–26.
13. Liu, Y.; Lapata, M. Text Summarization with Pretrained Encoders. *arXiv* **2019**, arXiv:1908.08345.
14. Zhong, M.; Liu, P.; Chen, Y.; Wang, D.; Qiu, X.; Huang, X. Extractive Summarization as Text Matching. ACL Anthology. 2020. Available online: https://www.aclweb.org/anthology/2020.acl-main.552/ (accessed on 9 August 2021).
15. Joshi, A.; Fidalgo, E.; Alegre, E.; Fernández-Robles, L. SummCoder: An unsupervised framework for extractive text summarization based on deep auto-encoders. *Expert Syst. Appl.* **2019**, *129*, 200–215. [CrossRef]
16. Sun, S.; Cheng, Y.; Gan, Z.; Liu, J. Patient knowledge distillation for bert model compression. *arXiv* **2019**, arXiv:1908.09355.
17. Iandola, F.N.; Shaw, A.E.; Krishna, R.; Keutzer, K.W. SqueezeBERT: What Can Computer Vision Teach NLP about Efficient Neural Networks? *arXiv* **2020**, arXiv:2006.11316.
18. VSanh, i.; Debut, L.; Chaumond, J.; Wolf, T. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *arXiv* **2020**, arXiv:1910.0110v4. Available online: https://arxiv.org/pdf/1910.01108v4.pdf (accessed on 12 July 2021).

19. Dong, Y.; Shen, Y.; Crawford, E.; van Hoof, H.; Cheung, J.C.K. Banditsum: Extractive summarization as a contextual bandit. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Brussels, Belgium, 31 October–4 November 2018.
20. Bucila, C.; Caruana, R.; Niculescu-Mizil, A. Model compression. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), Philadelphia, PA, USA, 20–23 August 2006.
21. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the 2018 Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UH, USA, 18–22 June 2018.
22. Hinton, G.E.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. *arXiv* **2015**, arXiv:1503.02531.
23. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: Alexnet-level accuracy with 50$\times$ fewer parameters and <0.5 mb model size. *arXiv* **2016**, arXiv:1602.07360.
24. Howard, G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
25. Tan, M.; Le, Q.V. EfficientNet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International Conference on Machine Learning (ICML), Long Beach, CA, USA, 10–15 June 2019.
26. Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. HuggingFace's Transformers: State-of-the-Art Natural Language Processing. arXiv.org. 2020. Available online: https://arxiv.org/abs/1910.03771 (accessed on 6 November 2021).
27. Zafrir, O.; Boudoukh, G.; Izsak, P.; Wasserblat, M. Q8BERT: Quantized 8bit bert. *arXiv* **2019**, arXiv:1910.06188.
28. Sanh, V.; Wolf, T.; Rush, A.M. Movement pruning: Adaptive sparsity by fine-tuning. *arXiv* **2020**, arXiv:2005.07683.