NLP Project
summarization

Wiam ADNAN wiam.adnan@centrale-casablanca.ma
Aymane Hanine aymane.hanine@centrale-casablanca.ma
Salma KHMASSI salma.khmassi@centrale-casablanca.ma
Hermann Agossou hermann.agossou@centrale-casablanca.ma
Zakaria Taouil zakaria.taouil@centrale-casablanca.ma

June 24, 2023

1 Introduction

Automatic text summarization is the task of condensing a piece of text into a shorter version, while still retaining its key information and meaning. One particular type of summarization is abstract summarization, which involves condensing an abstract into two or three sentences. This task is of great importance in many fields, including scientific research, where abstracts are often used to provide a summary of research papers.

In recent years, significant progress has been made in developing machine learning models for automatic summarization, leveraging advances in natural language processing and deep learning. However, summarization remains a challenging task, as it requires models to understand the meaning and structure of the input text, and to generate summaries that are both accurate and readable.

In this project, we explore different approaches to automatic summarization, focusing on the use of pre-trained models and fine-tuning techniques. We use a dataset of news articles as our main experimental data, and evaluate the performance of different models using a set of standard evaluation metrics, including ROUGE and BLEU. Our goal is to gain insights into the strengths and limitations of different summarization approaches, and to identify strategies for improving the quality of the generated summaries.

We explored three different approaches to automatic summarization. The first approach involved using a pre-trained summary model that had high performance on the dataset, followed by fine-tuning a lower-performing model on a subset of the data. The resulting model was then evaluated on a held-out test set. The second approach involved fine-tuning any model on a different dataset,

followed by applying the fine-tuned model to our own dataset to generate summaries. The third approach involved using a pre-trained extractive model to generate summaries, and then fine-tuning an abstractive model on those summaries.

2 Evaluation metrics

We have used ROUGE-1, ROUGE-2, ROUGE-L, and BLEU as evaluation metrics, in this paragraph we will explain how they are calculated in the context of text summarization:

- **ROUGE-1** = (number of overlapping unigrams) / (total number of unigrams in the reference summary)

  To calculate ROUGE-1, we first tokenize both the generated summary and the reference summary into unigrams (i.e., single words). We then count the number of unigrams that are shared by both summaries and divide by the total number of unigrams in the reference summary. This gives us a measure of how many words in the generated summary are also present in the reference summary at the unigram level.ù

- **ROUGE-2** = (number of overlapping bigrams) / (total number of bigrams in the reference summary)

  To calculate ROUGE-2, we first tokenize both summaries into bigrams (i.e., pairs of adjacent words). We then count the number of bigrams that are shared by both summaries and divide by the total number of bigrams in the reference summary. This gives us a measure of how many pairs of adjacent words in the generated summary are also present in the reference summary.

- **ROUGE-L** = (length of the longest common subsequence) / (total number of words in the reference summary)

  To calculate ROUGE-L, we first compute the longest common subsequence (LCS) between the generated summary and the reference summary. The LCS is the longest sequence of words that appear in the same order in both summaries. We then divide the length of the LCS by the total number of words in the reference summary. This gives us a measure of how much of the reference summary is captured by the generated summary at the subsequence level.

- **BLEU** = expmin(0,1-n/r) * (product of precision scores for n-grams)

  To calculate BLEU, we first tokenize both summaries into n-grams of various lengths (typically 1 to 4). We then calculate the precision of each set of n-grams in the generated summary by counting the number of n-grams that are present in the reference summary and dividing by the total number of n-grams in the generated summary. We then compute the geometric mean of these precision scores and multiply by a brevity penalty term, which penalizes summaries that are too short. The brevity penalty is calculated as expmin(0,1-n/r), where n is the length of the generated summary and r is the length of the reference summary. This gives us a measure of how many n-grams in the generated summary are also present in the reference summary, while taking into account the length of the summaries.

3 Approach 1:

Abstract summarization is a challenging task that involves condensing lengthy pieces of text into a few concise sentences that capture the essence of the content. In this first paragraph, we present an approach for abstract summarization using pre-trained deep learning models and fine-tuning techniques. Specifically, we use three pre-trained models to summarize a dataset of scientific abstracts related to artificial intelligence in agriculture. We then compare the performance of each model using metrics such as Rouge 1, Rouge 2, Rouge L, and BLEU to identify the best and second-best models. To further improve performance, we fine-tune the summaries of the second-best model on the summaries of the best model. Our approach offers a systematic and rigorous way to generate accurate and effective abstract summaries, which can be useful for applications such as document summarization, information retrieval, and more.

3.1 Used Models :

The three pre-trained models are:

- **Big Bird Pegasus For Conditional Generation** (BBPCG): Big Bird Pegasus is a cutting-edge language model that excels in conditional text generation tasks. Its hybrid architecture, combining Transformer and sequence-to-sequence models, allows it to handle long-form inputs and generate high-quality summaries. It also features an efficient pre-training method that enables it to achieve impressive results on a variety of tasks, including text summarization, language modeling, and machine translation.
- **Pegasus-xsum** (pegasus): is a powerful language model designed for text summarization tasks. It leverages the Transformer architecture to generate high-quality summaries that capture the key information from long-form inputs. Pegasus-Xsum was pre-trained on a large corpus of web texts to learn to generate summaries that are both grammatically correct and semantically accurate. It also features a decoding strategy that incorporates a length penalty to ensure that the generated summaries are neither too long nor too short. As a result, Pegasus-Xsum is able to achieve state-of-the-art performance on benchmark summarization datasets.
- **t5-large-finetuned-xsum-cnn** (t5-large): is a language model that has been fine-tuned on the CNN/Daily Mail summarization dataset using the T5 Transformer architecture. The model is capable of generating high-quality summaries that effectively capture the key information from news articles. T5-Large-Finetuned-Xsum-CNN achieves its impressive results by leveraging a combination of pre-training on a large corpus of texts and fine-tuning on the summarization task. The resulting model is capable of generating summaries that are both accurate and fluent. Its superior performance has been demonstrated on the CNN/Daily Mail dataset and other benchmark summarization datasets.

3.2 Steps for generating summaries :

The summaries are generated following the main steps :

1. Import the model and tokenizer
2. Move the model to a specified device (CPU or GPU), for our case we have worked in Collab, hence all calculations are made using GPU
3. Set batch size and maximum summary length as hyperparameters.
4. Create an empty list summaries to store the generated summaries.
5. Loop through the abstracts in the training dataset in batches of size *batch_size*.
    - Tokenize the batch of abstracts using the tokenizer.
    - Generate summaries for each batch using the specified *max_summary_length* and the model.
    - The generate method of the model takes the encoded input sequences, generates corresponding output sequences using beam search, and returns the resulting tensor of output token ids.
    - Decode the generated token ids into human-readable text summaries using the tokenizer.
    - Add the batch of summaries to the list summaries.

3.3 Results on train dataset :

We started by generating the summaries for the *train_dataset.json*, where we will do the fine tuning. Then we will evaluate the performances on the *test_dataset.json*. The results on the *train_dataset.json* are as follow :

model	ROUGE-1	ROUGE-2	ROUGE-L	BLEU
t5-large	0.169237	0.069559	0.158777	2.746262e-243
BBPCG	0.122082	0.024987	0.113230	7.539752e-214
pegasus	0.229497	0.078772	0.210326	4.516283e-215

Table 1: Evaluation metrics for different models.

The table shows the performance of the three different text summarization models: $t5 - large - finetuned - xsum - cnn$, $BigBirdPegasusForConditionalGeneration\_summaries$, and $pegasus - xsum\_summary$, on the evaluation metrics of ROUGE-1, ROUGE-2, ROUGE-L, and BLEU. From the results, it can be observed that the $pegasus - xsum\_summary$ model achieved the highest scores across all metrics, followed by $t5 - large - finetuned - xsum - cnn$, while

BigBirdPegasusForConditionalGeneration_summaries had the lowest scores. This suggests that the *pegasus − xsum_summary* model may be the most effective model for text summarization tasks compared to the other two models. However, it is important to note that these results are based on a specific dataset and may not necessarily generalize to other datasets or use cases.

3.4 Fine tuning :

In this section we will generate summaries from the most performing model (pegasus in this case), in order to fine tune the second most performing model (t5-large). To do so, we have used a dataset that we have called *train_dataset.json*. It contains the abstracts and the corresponding summary from pegasus model.

Fine-tuning the T5-Large model for text summarization involves training the pre-trained model on a specific task of summarizing text. This process involves providing the model with a large dataset of paired input-output text examples, where the input is a longer text document, and the output is a shorter summary of that document. The model learns to generate summaries by adjusting its parameters based on the training examples. The fine-tuning process involves adjusting hyperparameters, such as learning rate and batch size, and optimizing the loss function to improve the model's performance on the summarization task. Once the fine-tuning is complete, the model can be used to generate summaries for new text documents. The T5-Large model has been shown to achieve state-of-the-art results on text summarization tasks when fine-tuned on large and diverse datasets.

Note: Due to computation limitation, we have used a dataset of 300 samples for the fine tunning. In average, it takes about 1.5 minutes to get the summary of one abstract which is 100 hours for 4000 samples in the original dataset for a single model.

3.5 Evaluation and discussion :

To evaluate the ensemble of the models and approaches, we have fixed a test dataset called *test_dataset.json*, where we test all our models and ideas using the 4 metrics. In the table bellow, we can see the results of the 3 models and the fine tunned one:

Model	ROUGE-1	ROUGE-2	ROUGE-L	BLEU
BBPCG	0.128516	0.029043	0.118884	1.241677e-10
Pegasus	0.199311	0.055300	0.186434	5.717189e-215
T5_large	0.186353	0.082153	0.177092	1.226987e-240
T5_fine_tunned_on_Pegasus	0.163148	0.052152	0.147984	2.347712e-235

Table 2: Evaluation metrics for different models on test data

The results show that the model "*pegasus_summaries*" outperforms the other models across all the evaluation metrics. Its Rouge-1 score of 0.199 is significantly higher than the other models, indi-

cating that it does a better job of capturing the most important information in the abstracts. The high Rouge-2 score of 0.055 suggests that the model is also good at capturing important phrases or combinations of words in the text. Its high Rouge-L score of 0.186 shows that it is able to maintain coherence and capture the overall meaning of the text.

The "*t5_large*" model comes in second place in terms of performance, with similar scores to "*pegasus*" on Rouge-2 and Rouge-L, but a slightly lower Rouge-1 score. This suggests that while it may not capture the most important information as well as the best model, it is still capable of generating good summaries that capture the overall meaning of the text.

The "$t5\_fine\_tuned\_on\_pegasus$" model is fine-tuned on summaries generated by the "*pegasus*" model, but its performance is not as good as the second-best model. This suggests that fine-tuning on summaries generated by another model may not always lead to better results.

The "*BBPCG*" model performs significantly worse than the other models on all metrics. This may be because the model is not as well-suited for abstract summarization tasks as the other models or because it was not fine-tuned on a suitable dataset.

Overall, the results suggest that "*pegasus*" is the best model for abstract summarization tasks among the models evaluated. However, it is worth noting that other factors such as the dataset used for training and the specific requirements of the task may also influence the choice of the best model.

Fine-tuning a pre-trained language model on a specific dataset can often lead to improved performance on specific downstream tasks. However, in some cases, like this one, fine-tuning may not lead to the expected improvement in performance. In this case, when we fine-tuned the "*t5_large*" model on the abstract summarization task, its performance was not as good as the pre-trained model, contrary to what was expected. There could be several reasons for this. One possible reason is the small size of the dataset used for fine-tuning, which may not have been sufficient to capture the complexity of the task. Another possible reason is that the dataset used for fine-tuning is different from the dataset used to train the original pre-trained model. In such cases, the pre-trained model may not have learned the relevant features needed for the downstream task, which could limit the effectiveness of fine-tuning.

4 Approach 2:

In this second paragraph, we present an approach for abstract summarization using pre-trained deep learning models and fine-tuning techniques. Specifically, we use a pre-trained model a T5 (Text-to-Text Transfer Transformer) that we fine-tuned on XSum, a dataset of news articles from various sources, with their summaries. We used the fine-tuned model to summarize 400 abstracts from our dataset. We then evaluate the performance of the model using metrics such as Rouge 1, Rouge 2, Rouge L, and BLEU.

4.1 Used Model :

Text-to-Text Transfer Transformer (T5-small): T5-small is a pre-trained version of the Text-to-Text Transfer Transformer model with 220 million parameters, smaller than T5-3B and T5-11B. It is pre-trained on a variety of tasks, including unsupervised and supervised tasks, using self-attention to learn contextual representations of input text. T5-small can be fine-tuned on a specific task using a smaller amount of labeled data, and has achieved state-of-the-art performance on natural language processing tasks such as summarization, translation, and question answering. Its flexibility and simple, unified training framework make it suitable for a wide range of text-to-text tasks.

4.2 Dataset for fine-tuning :

Extreme Summarization (Xsum): For our project, we used the XSum dataset (Narayan and Gardent, 2018), which is a collection of news articles from a diverse range of sources. The dataset contains approximately 226,000 articles, each of which has a summary of around one sentence in length. To adapt the dataset for our project, we focused on articles related to the domain of agriculture, selecting a subset of 400 articles for fine-tuning our model.

4.3 Steps for generating summaries :

The summaries are generated following the main steps using the GPU provided on google collab :

1. Import the necessary libraries including Hugging Face Transformers and Datasets.
2. Load and tokenize the dataset using the Datasets library.
3. Download and load the pre-trained T5 model checkpoint from the Hugging Face Model Hub.
4. Define a special data collator for sequence-to-sequence models using DataCollatorForSeq2Seq from the Transformers library: The data colllector creates input ids, attention mask and labels

5. Define the optimizer and compile the model using the Adam optimizer and the built-in loss calculation function.
6. Train the model (learning_rate=2e-5, batch_size=8, max_input_length=1024, min_target_length=150, max_target_length=250) with the training and validation sets and the defined metric function for calculating ROUGE scores. We use only 10% for training and 10% for validation because of hardware and time limitations
7. Generate summaries using the trained model and the summarization pipeline from the Transformers library. The pipeline method takes in the trained model and tokenizer as arguments and uses the summarization pipeline.
8. Use metrics such as blue, rouge-1, rouge-2 and rouge-L to evaluate the results

4.4 Results on train dataset :

We started by generating the summaries on both the *train_dataset.json* and the *test_dataset.json*. Then we will evaluate the performances. The results are as follow :

model	ROUGE-1	ROUGE-2	ROUGE-L	BLEU
T5-small-finetuned-on-Xsum	0.362165	0.217916	0.338231	1.130762e-10

Table 3: Evaluation metrics for the model.

The table shows the performance of the model *t5small* fine-tuned on *X_sum* of the dataset on the evaluation metrics of ROUGE-1, ROUGE-2, ROUGE-L, and BLEU. From the results, it can be observed that the approach using fine-tuning give lower scores than the the first approach using large models. This suggests that the $pegasus - xsum\_summary$ model may be the most effective model for text summarization tasks compared to all models tested till now. However, it is important to note that the model was fine-tuned on 10% of Xsum on only nb_epoch=1 which can be the reason for this performance.

4.5 Discussion :

1. One limitation of this approach is the small amount of data used for training. Fine-tuning a pre-trained model on only 10% of the X-sum dataset may not be enough to capture all the nuances of the task, which could lead to suboptimal results. Increasing the amount of training data and/or the number of training epochs could potentially improve the performance of the model.
2. Another factor that could impact the quality of the generated summaries is the choice of pre-trained model. While T5 is a powerful language model, there may be other models that are better suited for the summarization task. For example, the use of most larger models like in the other approach could have contributed to its better performance.

3. Additionally, the quality of the input data could also have an impact on the performance of the model. X-sum is a dataset of news articles, which can vary widely in terms of topic, tone, and complexity. Some articles may be more difficult to summarize than others, which could affect the overall performance of the model. Using a dataset more specific to agriculture could potentially improve the performance of the model.
4. Finally, the ROUGE scores used to evaluate the model may not fully capture the quality of the generated summaries. While they provide a quantitative measure of performance, they do not account for factors such as coherence, fluency, and readability. Therefore, it may be important to also perform a qualitative evaluation of the generated summaries, either through manual inspection or through user studies, to get a more complete picture of the model's performance.

5 Approach 3:

5.1 Principle:

Abstractive summarization involves generating a summary in a more human-like way by interpreting and paraphrasing the text to create a shorter, more readable version. This approach requires more sophisticated natural language processing techniques like deep learning and neural networks to understand the meaning of the text and generate new language that accurately conveys the same information. It often produces summaries that are more readable and understandable to humans, but the process is much more difficult, and the results can be less accurate than extractive summarization. On the other hand, the extractive summarization involves selecting and combining the most important sentences or phrases from the original text to create a summary. This approach can be effective when the goal is to provide a concise and accurate representation of the key points in the original text. As these extractive models can be trained with less data and can produce summaries that are easier to evaluate and interpret, we have used such a model to generate the summaries of our train dataset which will be used for the fine tuning of an abstract model.

Here are the steps that we followed:

1. **Choose the extractive model and apply it to the train dataset**: we have chosen as an abstractive extractive model the *bart_large_xsum_samsum*. It's a language model developed by Facebook AI Research that was pre-trained on a large corpus of text data. The model was specifically designed to perform well on the tasks of summarization and question answering. The data on which it has been pre-trained is a combination of the XSUM and the SAMSUM datasets. The first is a collection of news articles from the BBC that have been manually summarized, and the second is a collection of questions and answers based on a set of Wikipedia articles. Generally, the BART model architecture is based on a transformer architecture, which has been shown to be highly effective for a wide range of natural language processing tasks and the BART large XSUM SAMSUM is one of the largest variants of this model with 406 million parameters. We have applied this model to our train dataset which contains 300 abstracts and collected the results (summaries) in the list summary.
2. **Choose the abstractive model and fine tune it with the results of the abstractive extractive model**: we have chosen as an abstractive model the $t5\_large\_finetuned\_xsum$. It refers to a T5 model that has been fine-tuned on the XSum dataset, which is a collection of short news articles with corresponding one-sentence summaries. The "large" in "t5-large" refers to the size of the model, which has 770 million parameters, making it larger and more powerful than the base or small versions of the T5 model. It is also considered as a powerful tool for text summarization tasks and has achieved state-of-the-art performance on several benchmark datasets. We have fine-tuned this model based on the summaries of the first model and we have tested it on our test dataset which contains 100 abstracts.

5.2 Process:

1. Load/install the necessary libraries.
2. Load the dataset train.
3. Load the abstractive extractive model: bart large xsum samsum.
4. Apply the bart large xsum samsum model to the train dataset.
5. Create a new dataset containing the abstracts of the dataset train an their summaries obtained by applying the extractive abstractive model.
6. Tokenize the new dataset.
7. Load the template abstractive: t5 large finetuned xsum.
8. Define the hyperparameters.
9. Fine tune the t5 large finetuned xsum model.
10. Apply the fine tuned model on the test dataset.
11. Calculate the metrics needed to evaluate the fine tuned model.

5.3 Results on train dataset :

We started by generating the abstracts for the *train_dataset.json*, where we will do the fine tuning. Next, we will evaluate the performance of the fine-tuned abstractive model on the *train_dataset.json*. The results on the $train_dataset.json$ are as follows:

model	ROUGE-1	ROUGE-2	ROUGE-L	BLEU
Extractive model *bart_large_xsum_samsum*	0.356324	0.244907	0.350222	1.701539e-222
Abstractive model *t5_large_finetuned_xsum*	0.156297	0.044861	0.1397918	3.610569e-239

Table 4: Evaluation metrics for different models.

The table shows the performance of two different models on four evaluation metrics commonly used in text summarization: ROUGE-1, ROUGE-2, ROUGE-L, and BLEU.

The first model is an extractive model called "*bart_large_xsum_samsum*", and the second model is an abstractive model called "*t5_large_finetuned_xsum*".

In terms of performance, the extractive model outperforms the abstractive model on all four evaluation metrics. Specifically, the extractive model achieves ROUGE-1, ROUGE-2, and ROUGE-L scores of 0.356, 0.245, and 0.350, respectively, while the abstractive model achieves scores of 0.156, 0.045, and 0.140, respectively. The extractive model also achieves a higher BLEU score of 1.701539e-222 compared to the abstractive model's score of 3.610569e-239.

5.4 Finetuning the abstractive model:

Extractive summarization methods identify and select important sentences or phrases from the source document and stitch them together to form a summary. These summaries tend to be more factually accurate and can capture the main ideas of the source document. By providing these extractive summaries as inputs to the abstractive model during fine-tuning, the abstractive model can learn from the important sentences and phrases identified by the extractive model and generate more accurate and informative summaries. This approach can help the abstractive model to focus on the most important information while avoiding errors or irrelevant details.

5.5 Results on test dataset:

model	ROUGE-1	ROUGE-2	ROUGE-L	BLEU
Abstractive model	0.166340	0.049878	0.148912	5.442148e-235
Finetuned Abstractive model	0.141666	0.037907	0.128724	3.023488e-241

Table 5: Evaluation metrics for different models.

The performance of the fine-tuned abstractive model was evaluated using standard evaluation metrics, including ROUGE-1, ROUGE-2, ROUGE-L, and BLEU. The results showed that the fine-tuned model had lower scores across all metrics compared to the pre-trained model. This is a common challenge when fine-tuning models, as the process involves balancing the benefits of pre-trained knowledge with the need to adapt to a new domain or task. In this case, it's possible that the pre-trained knowledge in the original model was not fully compatible with the extractive summaries used during fine-tuning, resulting in a decrease in performance.

Another possible reason for the decrease in performance could be the quality of the extractive summaries used during fine-tuning. Extractive summarization methods are not perfect, and the summaries they produce can sometimes miss important information or include irrelevant details. Using lower quality extractive summaries during fine-tuning could lead to the abstractive model learning from incorrect or incomplete information, leading to a decrease in performance.

6 Conclusion

In conclusion, the three approaches presented in this text describe different methods for abstract summarization using pre-trained deep learning models and fine-tuning techniques. The first approach focuses on using three large pre-trained models and fine-tuning the second-best model on the best model's summaries to further improve performance. The second approach uses a single pre-trained model fine-tuned on a small subset of news articles in the agriculture domain. The third approach uses an abstractive-extractive hybrid approach to generate summaries by first using an extractive model to generate summaries and then fine-tuning an abstractive model on these summaries.

While the three approaches have their own strengths and limitations, the evaluation metrics used suggest that the first approach achieved the highest scores on all metrics, followed by the third and second approaches. However, it is important to note the metrics used (bleu, rouge1, rouge2, and rougeL) do not always capture the quality of the generated summary or its coherence.

Overall, these approaches provide useful insights into the different methods that can be used for abstract summarization, and how pre-trained deep learning models and fine-tuning techniques can be leveraged to improve performance. These methods have potential applications in document summarization, information retrieval, and other areas where summary generation is required.