ABSTRACTIVE TEXT SUMMARIZATION USING TRANSFORMER MODELS

PROJECT REPORT

submitted by

NIZAMUDHEEN T. I.
Register No: TVE20CSAI13

to

the APJ Abdul Kalam Technological University
in partial fulfillment of the requirements for the award of the Degree

of

Master of Technology
in
Computer Science and Engineering
with specialization in

Artificial Intelligence

Department of Computer Science and Engineering
College of Engineering Trivandrum
Thiruvananthapuram

SEPTEMBER 2022

DECLARATION

I undersigned hereby declare that the project report “Abstractive Text Summarization Using Transformer Models”, submitted for partial fulfillment of the requirements for the award of degree of Master of Technology of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by me under supervision of Prof.Anurenjan P R. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place : Trivandrum

Date : 11/09/2022 NIZAMUDHEEN T. I.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COLLEGE OF ENGINEERING TRIVANDRUM

CERTIFICATE

This is to certify that this project report entitled **“Abstractive Text Summarization Using Transformer Models”** submitted by **Nizamudheen T. I.**, to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Computer Science and Engineering with specialization in Artificial Intelligence** is a bonafide record of the project work carried out by her under our guidance and supervision. This report in any form has not been submitted to any other University of Institute for any purpose.

Prof. Anurenjan P. R.
Assistant Professor
(Project Guide)
Dept. of Electronics &
Communication Engg.

Dr. Sumesh Divakaran
Professor
(Head of the Department)
Dept. of Computer
Science & Engg.

Prof. Shine S
Associate Professor
(Project Coordinator)
Dept. of Computer
Science & Engg.

Dr. Suresh Kumar E
Professor
(Project Coordinator)
Dept. of Electronics &
Communication Engg.

Prof. Santhosh K.S
Assistant Professor
(Project Coordinator)
Dept. of Mechanical
Engg.

ACKNOWLEDGEMENT

It is with great enthusiasm and spirit I am bringing out this project report. I also feel that this is the right opportunity to acknowledge the support and guidance that came in from various stages during the course of completion of this phase my work. Above all, I owe my gratitude to the Almighty for showering abundant blessing upon us.

I extend my sincere gratitude to our principal, **Dr. Suresh Babu V** for his support in the development of this course. I also express my gratitude towards the Head of the Computer Science and Engineering Department, **Dr. Sumesh Divakaran** for granting us the liberty to work on this project.

I wish to express my profound gratitude to my guides, **Prof. Anurenjan P R, Dr. Sreeni K. G.**, Department of Electronics and Communication Engineering for their amenable guidance which became the backbone for my work to make it a success, and also to the project Co-coordinators **Prof. Shine S**, Associate Professor, Department of Computer Science and Engineering, **Dr. Suresh Kumar E**, Professor, Department of Electronics and Communication Engineering, and **Prof. Santhosh K.S**, Assistant Professor, Department of Mechanical Engineering for the encouragement given by them.

I also acknowledge my gratitude to other members of faculty in the Department of Computer Science and Engineering, Department of Mechanical Engineering and Department of Electronics and Communication Engineering. I also acknowledge my gratitude to my family and friends for their whole hearted cooperation and encouragement.

Nizamudheen T I

ABSTRACT

There is a significant growth of textual data in digital form in today's technology period, and it is constantly multiplying. Using efficient systems is required when dealing with large amounts of textual data. This problem can solve by using automatic summarization systems. As a result, working on the architecture of existing automatic summarization systems and innovating them to make them capable of fulfilling the importance of constantly rising data based on user needs becomes increasingly important.

This project aims to develop an efficient automatic text summarization system using recent attention-based transformer models. The pre-trained transformer models PEGASUS, BART and T5 are used to implement the automatic abstractive text summarization system. The pre-trained models are fine-tuned using CNN/DailyMail dataset. Before that, the dataset was preprocessed and tokenised accordingly. The automatic abstractive text summarization model performance is evaluated using the ROUGE measure. This study mainly focused on the hyperparameter tuning of these models. For that, the number of epochs, learning rate and batch size of the model are used. The results are studied.

Key Words - Natural Language Processing, Deep learning, Text Summarization.

TABLE OF CONTENTS

ACKNOWLEDGEMENT **i**

ABSTRACT **ii**

LIST OF FIGURES **v**

LIST OF TABLES **vi**

ABBREVIATIONS **vii**

1 INTRODUCTION **1**

1.1 Natural Language Processing 1

1.2 Text Summarization 1

1.3 The need for text summarization 2

1.4 Types of text summarization 2

1.4.1 Extractive Text summarization 2

1.4.2 Abstractive Text summarization 4

1.5 Motivation 5

2 EXISTING WORK **6**

3 PROBLEM DEFINITION AND OBJECTIVE **9**

3.1 Problem Statement 9

3.2 Objective 9

4 METHODOLOGY 10

4.1 Design 10

4.2 Dataset 11

4.2.1 Data Fields 12

4.2.2 Data Split 12

4.3 Model architectures 12

4.3.1 T5 12

4.3.2 PEGASUS 17

4.3.3 BART 19

4.4 Hyperparameters 21

4.4.1 Number Epochs 22

4.4.2 Learning rate 22

4.4.3 Batch Size 22

4.4.4 Number of Beams 23

4.5 ROUGE metric 23

5 IMPLEMENTATION 25

5.1 Tools and Libraries 25

5.1.1 TensorFlow 25

5.1.2 Keras 25

5.1.3 PyTorch 25

5.1.4 Transformers 26

5.1.5 NLTK 26

5.1.6 Pandas 26

5.1.7 Google Colaboratory 26

5.2 Implementation steps 27

5.2.1 Installing and importing libraries 27

5.2.2 Dataset Preprocessing 27

5.2.3 Model Fine-tuning 28

6 RESULTS 30

7 CONCLUSIONS AND FUTURE WORKS 33

LIST OF FIGURES

1.1 Text summarization example 2

1.2 Text summarization Categories 3

1.3 Exractive Text Summarization 3

1.4 Abstractive Text Summarization 4

4.1 Methodology for implementing Abstractive text summarization . . . 10

4.2 Model Design 10

4.3 Sample instances from the dataset 11

4.4 Text-To-Text Transfer Transformer-Tasks 12

4.5 Model Structures of T5 transformer model 15

4.6 Corrupting Spans in T5 transformer model 15

4.7 Corrupting Strategies in T5 transformer model 16

4.8 Gap Sentence Generation 17

4.9 Masked Language Model in PEGASUS 18

4.10 Combined Training in PEGASUS 18

4.11 BART Architecture 19

4.12 Transformations used for noising the input in the BART model . . . 19

4.13 ROUGE-N Example 23

4.14 ROUGE-N Example 24

4.15 ROUGE-N Example 24

5.1 Example of installing Package using pip 27

5.2 Example of importing of packages 27

5.3 Example of tokenization 28

5.4 Sample fine-tuning result 28

5.5 Fine-Tuning details 29

5.6 Summary generating function 29

6.1 T5-base summary generation 30

6.2 PEGASUS summary generation 31

6.3 BART-large summary generation 31

6.4 DBART summary generation 32

LIST OF TABLES

4.1 CNN/DailyMail Dataset Split 12

6.1 T5 model Abstractive Summarization results 30

6.2 PEGASUS model Abstractive Summarization results 31

6.3 BART model Abstractive Summarization results 32

6.4 DistlBART model Abstractive Summarization results 32

6.5 Comparison of different Abstractive text summarization models . . 32

ABBREVIATIONS

AI	Artificial Intelligence
BART	Bidirectional Auto-Regressive Transformers
BERT	**B**idirectional **E**ncoder **R**epresentation from **T**ransformers
C4	Colossal Clean Crawled Corpus
DBART	Distilled Bidirectional Auto-Regressive Transformers
DUC	Document Understanding Conferences
EOS	end of sentence
GPT	General Purpose Transformer
GPU	Graphics Processing Unit
GSG	Gap Sentences Generation
LCS	longest common subsequence
LM	Language Model
LSTM	Long Short Term Memory
ML	Machine Learning
MLM	Masked Language Model
NLG	Natural Language Generation
NLP	**N**atural **L**anguage **P**rocessing
NLTK	Natural Language Tool Kit
OOV	Out Of Vocabulary
PEGASUS	Pre-training with Extracted Gap-sentences for Abstractive Summarization
RNN	Recurrent Neural Network
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
T5	Text-to-Text-Transfer-Transformer
UNK	unknown token

CHAPTER 1

INTRODUCTION

In this chapter, it deals with the notion of Natural Language Processing (NLP) and Text summarization.The importance of Abstractive text summarization is specified and also mentioned different summarization methods.

1.1 Natural Language Processing

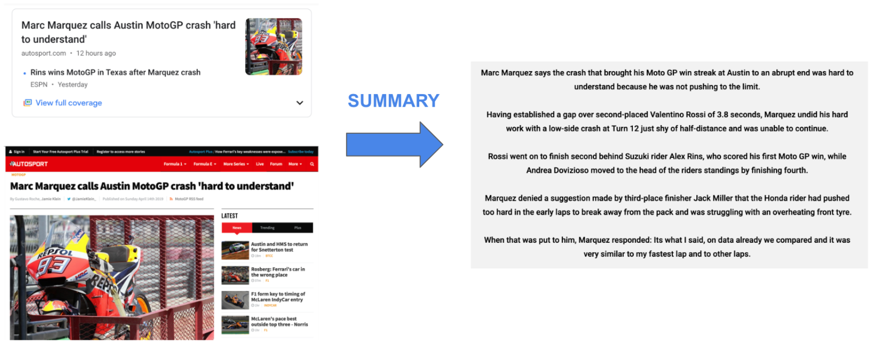
The topic of artificial intelligence known as NLP (Natural Language Processing) focuses on how computers interact with human languages, particularly how to design computers to process and analyse massive volumes of natural language data. The most challenging NLP tasks are those where the final product is a complete new text rather than just a single label or value (such as Classification and Regression) (like Translation, Summarization and Conversation).

Computers can now comprehend natural language just like people do thanks to NLP. Natural language processing use artificial intelligence to take real-world information, process it, and make sense of it in a way that a computer can comprehend, regardless of whether the language is spoken or written. Computers have reading programmes and microphones to gather audio, much as people have various sensors like ears to hear and eyes to see. Computers have a programme to process their various inputs, just as humans have a brain to do so. The input is eventually translated into computer-readable code during processing.

1.2 Text Summarization

Text Summarization [1] Text data summarising is a strategy for producing a succinct and accurate summary of extensive text data while concentrating on the key passages that provide valuable information and preserving the overall meaning.

The goal of automatic text summarising is to reduce long papers to their essential content. Manual text summarization might be time-consuming and expensive. Before creating the necessary summary sentences, machine learning algorithms may be trained to understand documents and recognise the portions that carry key facts and information. An example of a news story that has been placed into a machine learning algorithm to produce a summary is shown in the image below.


Marc Marquez calls Austin MotoGP crash 'hard to understand'
autosport.com • 12 hours ago
- Rins wins MotoGP in Texas after Marquez crash
ESPN • Yesterday

Marc Marquez calls Austin MotoGP crash 'hard to understand'
By Gustavo Roche, Jamie Klein | @JamieKlein_ | Published on Sunday April 14th 2019 | MotoGP RSS feed

LATEST
- Austin and HMS to return for Snetterton test
- Rosberg: Ferrari's car in the wrong place
- F1 form key to timing of McLaren IndyCar entry
- McLaren's pace best outside top three - Norris

Marc Marquez says the crash that brought his Moto GP win streak at Austin to an abrupt end was hard to understand because he was not pushing to the limit.

Having established a gap over second-placed Valentino Rossi of 3.8 seconds, Marquez undid his hard work with a low-side crash at Turn 12 just shy of half-distance and was unable to continue.

Rossi went on to finish second behind Suzuki rider Alex Rins, who scored his first Moto GP win, while Andrea Dovizioso moved to the head of the riders standings by finishing fourth.

Marquez denied a suggestion made by third-place finisher Jack Miller that the Honda rider had pushed too hard in the early laps to break away from the pack and was struggling with an overheating front tyre.

When that was put to him, Marquez responded: Its what I said, on data already we compared and it was very similar to my fastest lap and to other laps.

Figure 1.1: Text summarization example

1.3 The need for text summarization

The current expansion of non-structured textual data in the digital sphere necessitates the creation of automatic text summary technologies that make it simple for users to draw conclusions from them. We now have immediate access to vast volumes of information. However, the majority of this data is unnecessary, trivial, and might not communicate the intended message.For instance, if you are searching for certain information in a news item online, you may need to sift through the post's content and spend a lot of time eliminating the extraneous material before finding what you are looking for. Therefore, it is increasingly important to utilise computerised text summarizers that can extract useful information while excluding inessential and irrelevant material. Implementing summarising may make papers easier to read, cut down on the time spent looking for information, and make it possible to put more information into a given space.

1.4 Types of text summarization

In Natural Language Processing, there are primarily two methods for summarising texts: extraction-based text summarization and abstraction-based text summary.

1.4.1 Extractive Text summarization

In extraction-based summarising, a selection of words that best capture the text's key ideas are selected, then they are concatenated to create a summary. Imagine it as

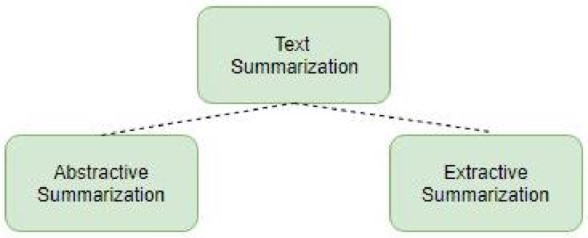


Figure 1.2: Text summarization Categories

a highlighter that only picks out the key details from a source text. Extractive summarization in machine learning often entails weighting the crucial parts of phrases and utilising the findings to produce summaries.Using a scoring algorithm, extractive summarization selects phrases from the source text to create a meaningful summary. This technique involves highlighting key passages in the text, clipping them out, and then piecing the information back together to create a streamlined version.

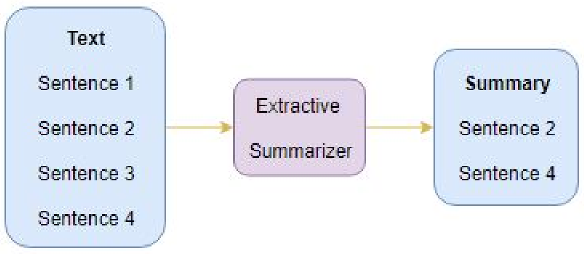


Figure 1.3: Exractive Text Summarization

As a result, they are solely dependent on phrase extraction from the source text. As it is simpler and produces summaries that are naturally grammatical, extractive summarization has received the majority of attention in today's summarising research. Additionally, extractive summaries include the key phrases from the input, which may be a single document or a collection of documents.

An extractive summarization system's usual flow is as follows:

1. Creates an intermediary representation of the input text with the goal of identifying material that is salient. In most cases, it operates by calculating TF metrics for each sentence in the supplied matrix.

2. Scores the sentences based on the representation, giving each one a number that indicates the likelihood that it will be included in the summary.

3. Generates an overview based on the top k phrases. Latent semantic analysis

(LSA) has been used in several research to pinpoint sentences that are crucial in terms of meaning.

1.4.2 Abstractive Text summarization

Abstractive summarization methods aim to produce summary by interpreting the text using advanced natural language techniques to generate a new, shorter text that conveys the most important information from the original document. This requires rephrasing sentences and incorporating information from the full text to generate summaries similar to what a human-written abstract typically does. In actuality, a good abstractive summary is linguistically fluid and contains the input's essential information. They are not limited, then, to merely picking and choosing from the original text.

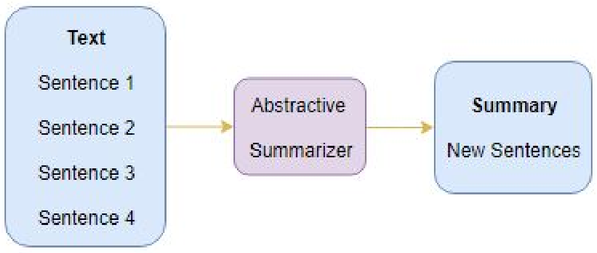


Figure 1.4: Abstractive Text Summarization

Abstractive techniques benefit from current advancements in deep learning. Abstractive approaches capitalise on the recent success of the sequence to sequence models since it can be seen as a sequence mapping problem where the source text should be transferred to the target summary. A neural network reads the text, encodes it, and then creates the target text in these models, which have an encoder and a decoder.

In general, creating abstract summaries is a difficult process that requires advanced language modelling and is more difficult than data-driven alternatives like sentence extraction. In spite of recent advancements employing neural networks motivated by the development of neural machine translation and sequence to sequence models, they are still far from achieving human-level quality in summary creation.

1.5 Motivation

In this era, technology is growing a lot; hence, the amount of data has also increased tremendously. Text data is increasing day by day to be very authentic. It is continuously multiplying in textual data in digital form. Automatic summarising systems make life easier. To successfully handle large amounts of text data on time, we need automatic text summarization systems. These technologies aim to generate accurate summaries. Thorough, concise, fluent, and capable of preserving all relevant information provided on a topic. Some A search is one of the uses of text summarization. Engine snippets are created as a result of a document search and news websites that produce condensed news in the form of headlines to facilitate browsing.This study mainly focuses on abstractive text summarization using pre-trained transformers and studies their performance with respect to some hyper parameters.

CHAPTER 2

EXISTING WORK

This chapter contains the related works of Abstractive text summarization in deep learning. Several methods to date approach state of the art in the field of Abstractive text summarization. This chapter discusses the application of such methods or approaches and their pros and cons.

It begins by noting the work of Facebook AI researchers [2], who suggested a model for phrase summarization. A neural network with attention is used in the model. In conjunction with the model, the researchers applied a generation method, which assisted the system in producing accurate summaries. In another paper, they later upgraded their model with the conditional RNN [3] architecture [4]. They used a convolutional attention-based encoder to assist the decoder focus on relevant input words at each stage of summary synthesis. The new approach outperformed prior work with the Gigaword corpus and DUC 2004. Scaling the abstractive summarising system to generate paragraph-level summaries, efficient alignment, and consistency in output creation are among the problems highlighted in [5].

In a different study [6] on abstractive summarisation in Chinese language text, the researchers initially used a microblogging service to construct a dataset known as LCSTS (Large-scale Chinese Short Text Summarisation). This collection of shorter Chinese texts with condensed outlines makes up a sizable corpus. The introduction of a summary generation model utilising RNNs was then done using this corpus. The usage of hierarchical RNNs for summary generation and study of unusual word difficulties is one of the recommendations made by the researchers for future work.

The neural encoder-decoder is the foundation of IBM's [7] model for abstractive text summarization, which also integrates attention. Embeddings of words, morphemes, and phrases are the input. In addition, bidirectional RNNs [8] and LSTM [9] are employed. The same framework in [6] serves as the basis for their work, although it adds more cutting-edge models that tackle important issues with abstractive summarization. The encoder-decoder model with attention and a broad vocabulary trick makes up the fundamental model. In order to tackle the difficulty of capturing significant concepts in a topic, a feature-rich encoder is used. The handling of terms not in the vocabulary is modelled using a switching generator/pointer technique. For lengthy papers, basic sentences must also be recorded in addition to keywords. Hierarchical attention can help with this.

According to [10], pre-existing abstractive summaries Systems have two issues: they sometimes output repetitious output, and they produce erroneous factual details. This study provides an abstractive summarization architecture that enhances the conventional attention-based seq2seq model with a hybrid pointer generator and coverage. The hybrid pointer-generator can duplicate Words can be selected from the input text by pointing, ensuring accurate information reproduction while preserving the possibility of using the generator to create new words. Although proper information is produced, this nevertheless repeats itself by using a coverage method that retains a record of every list of previously compiled information. ROUGE ratings are used to assess the model using the CNN/Daily Mail dataset [7].

Study [11] suggests another hybrid design handles the issue of sluggish speed and incorrect encoding of lengthy documents. This hybrid architecture includes an extractor module and an abstractor network abridged by policy-based reinforcement learning. The model's performance has improved. Use the ROUGE [12] and METEOR [13] scores on the CNN/Daily Mail corpus [7]. The bidirectional LSTM-RNN is the output of the extractor module, which is based on a convolutional model. The pointer network is then trained using a second LSTM-RNN. The encoder-aligner-decoder framework used in the abstractor module is enhanced by a copy mechanism to handle terms that are no longer in the lexicon.

A unique network architecture was proposed by Google researchers [14]. They referred to as the transformer, which was solely based on attentional processes. Machine translation tasks experiments showed that it trained much faster and produced better results than competing architectures. The transformer is the first sequence transduction model that may decide on the input and output representations without regard to the sequence's recurrence or convolution.

The researchers in [15] were curious to find out if knowledge is beneficial from a pre-trained language model summary of an abstract text. Therefore, they used specific requirements for the generator, decoder, and encoder of a neural model based on a transformer for the bidirectional Transformer Encoder Representations) [16] language model. They observed a substantial encoder improvement. And decoder, but not with the generator's conditioning. Additionally, the authors added two-dimensional convolutional self-attention to the encoder's early layers. On CNN/Daily Mail [7] datasets, the models were compared to the latest and most sophisticated models. In order to translate the model between English and German, it was additionally trained on the German SwissText dataset.The self-attention convolutional model performed worse than the BERT [16] based model. The TF-IDF (Term Frequency-Inverse Document Frequency) based extractive approach addresses the issue of long document summarising and employs the BERT's capacity to forecast the subsequent sentence to improve the accuracy and reliability of the produced summaries. The system became more reliable and effective because of this strategy.

In the paper [18], they proposed an abstractive summarisation system. Utilises the nice food reviews dataset from Amazon. At first, data The dataset went for preprocessing, which transforms It modifies raw data to be appropriate for deep learning models/algorithms. This procedure involves dividing text, Lemmatizing, deleting stop words, inserting contractions, counting the amount of the vocabulary, adding word embedding, and unique tokens like EOS and UNK. Model seq2seq has a bidirectional [8] LSTM [9] encoder integrated into it. a common LSTM decoder: The study's limitations, including the authors, mention that an accurate summary can only generate short input text. The model becomes inconsistent when it is used as an input when extensive text.

CHAPTER 3

PROBLEM DEFINITION AND OBJECTIVE

The chapter deals with the problem statement of the project. The area of interest is Natural Language Processing. The different objectives are also specified in this section.

3.1 Problem Statement

Design and implement Abstractive Text Summarization using Transformer models

3.2 Objective

1. To design seq-to-seq language models to generate a summary from input text.

2. To fine-tune the models using Dataset.

3. To evaluate and study model performance.

CHAPTER 4

METHODOLOGY

This chapter deals with the methodology of the project flow.It discusses with the design, dataset, architecture and performance evaluation metrics of the model.

4.1 Design

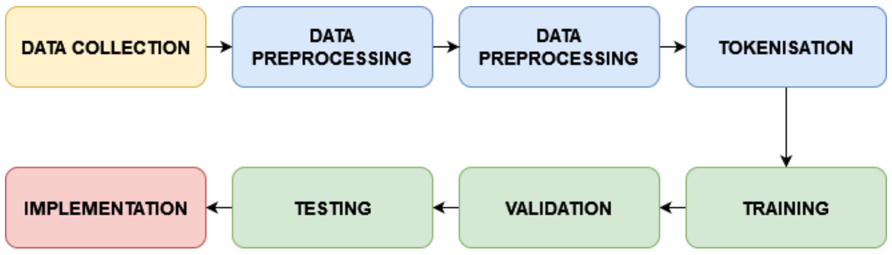


Figure 4.1: Methodology for implementing Abstractive text summarization

The proposed model is for generating abstractive summaries from input documents, here we use three different models for generating summaries from input documents.The models are T5, BART, and PEGASUS. They are all pre-trained on a large corpus, and we were only fine-tuning them using CNN/Dailymail Dataset. Before passing the dataset to the deep learning models, we will preprocess and tokenise the data. Then the dataset will split into train/test/validation sets. We use a training split of the data for the fin-tuning, which has input and output values. After fine-tuning, we will analyse the performance by doing hyperparameter tuning. For the hyperparameter tuning, we used the Number of epochs, batch size, learning rate and Number of beams.

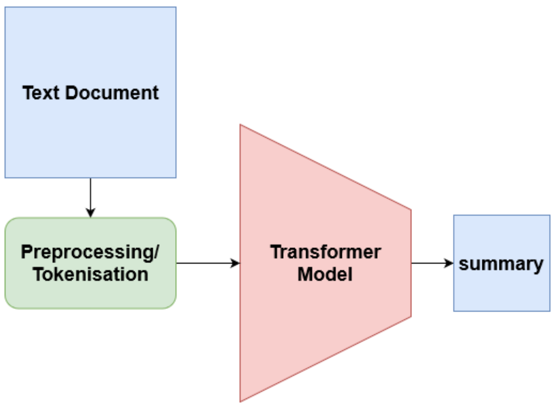


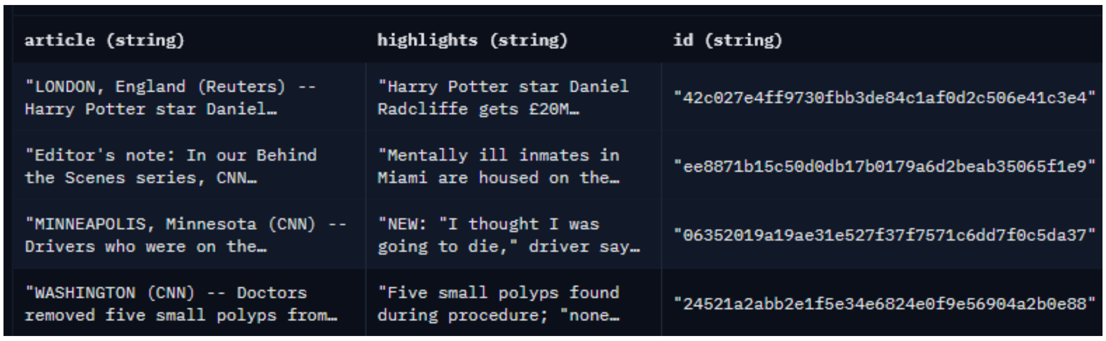
Figure 4.2: Model Design

The input sequence of abstractive summary training texts, which has both text document and its summary, is given to the model after text pre-processing & tokenization. After enough fine-tuning of the model, the test dataset, which has only text document, is given to the model to predict the summary.

4.2 Dataset

The English-language CNN / DailyMail Dataset comprises over 300,000 unique news stories published by CNN and the Daily Mail staff members. Although the initial version was developed for machine reading, comprehension, and abstractive question answering, the current version enables extractive and abstractive summarization.

The dataset comes in three prior versions. Version 1.0.0 delivered roughly 313k unique articles and almost 1M Cloze-style questions to go with the articles in order to assist supervised neural methods for machine-reading and question responding with a significant amount of accurate natural language training data. The dataset was restructured in versions 2.0.0 and 3.0.0 to facilitate summarization rather than question-answering. Version 3.0.0 further offered a non-anonymized version of the data, although both earlier versions had been preprocessed to swap out named entities with labelled unique identifiers. Highlight sentences and news items that make up the data.



article (string)	highlights (string)	id (string)
"LONDON, England (Reuters) -- Harry Potter star Daniel…	"Harry Potter star Daniel Radcliffe gets £20M…	"42c027e4ff9730fbb3de84c1af0d2c506e41c3e4"
"Editor's note: In our Behind the Scenes series, CNN…	"Mentally ill inmates in Miami are housed on the…	"ee8871b15c50d0db17b0179a6d2beab35065f1e9"
"MINNEAPOLIS, Minnesota (CNN) -- Drivers who were on the…	"NEW: "I thought I was going to die," driver say…	"06352019a19ae31e527f37f7571c6dd7f0c5da37"
"WASHINGTON (CNN) -- Doctors removed five small polyps from…	"Five small polyps found during procedure; "none…	"24521a2abb2e1f5e34e6824e0f9e56904a2b0e88"

Figure 4.3: Sample instances from the dataset

The articles are utilised as the context in the data's question-answering setup, and entities are gradually hidden in the highlight sentences to create Cloze-style questions. The model's task is identifying which context item has been concealed in the highlight. Finally, the highlighted sentences are combined to summarise the article in the setting for summarising and written in the CNN articles between April 2007 and April 2015. Articles published in The Daily Mail were created between June 2010 and April 2015.

4.2.1 Data Fields

- id: This field will contain a string containing the SHA1 hash of the URL where the story was retrieved, formatted in hexadecimal.
- article: This field will have a string with the news article's body.
- highlights: There will be a string in this field that contains the article's highlights in the author's own words.

4.2.2 Data Split

There are three splits in the CNN/DailyMail dataset: train, validation, and test. The dataset's statistics for Version 3.0.0 are listed below.

Table 4.1: CNN/DailyMail Dataset Split

Data Split	**Number of instances in the split**
Train	*287113*
Test	*13368*
Validation	*11490*

4.3 Model architectures

4.3.1 T5

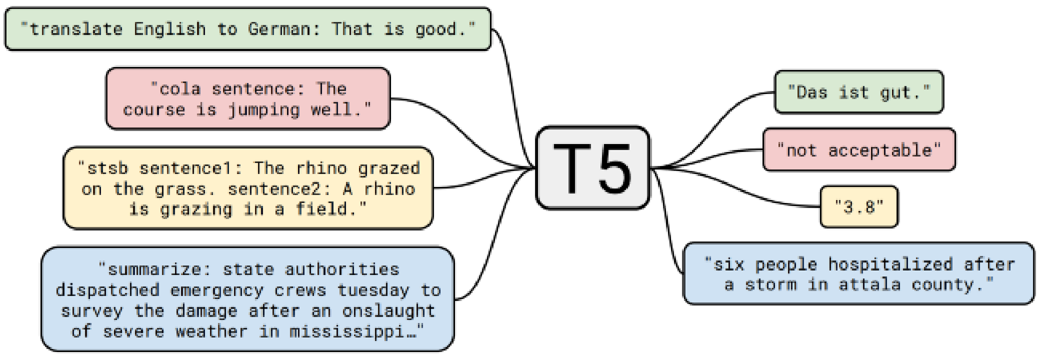

"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Figure 4.4: Text-To-Text Transfer Transformer-Tasks

This article will discuss Google's cutting-edge T5[20] Text-to-Text Transfer Transformer Model. It was first mentioned earlier this year in the publication "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." This research essentially surveys contemporary transfer learning methods for language comprehension and afterwards suggests a unified framework that aims to incorporate all linguistic issues into a text-to-text format. We will understand this strategy more along the way. Additionally, the authors have made available a brand-new dataset termed C4-Colossal Clean Crawled Corpus-to aid in their research.

T5: Text-to-Text-Transfer-Transformer paradigm suggests restructuring all NLP tasks into a standard text-to-text format where the input and output are always text strings. As can be seen in the accompanying animation, which accepts text input from the left for various NLP jobs and outputs the text for each one, formatting allows one T5 model to be used for several tasks. In the following parts, we will learn more about how the model was trained. First, let us talk about the pre-training data for the model. Its designation by the authors is C4 (Colossal Clean Crawled Corpus). It is the cleaned version of the Common Crawl dataset and is about 700GB in size. The writers have mentioned deduplicating, deleting code lines, and extracting only English text as cleaning methods. A top-notch pre-processed English language corpus is also available for download from them. Additionally, the pre-trained T5 model on C4 achieves cutting-edge outcomes on numerous NLP benchmarks while remaining adaptable enough to be optimised for various significant downstream applications.

i The Text-to-Text Framework

Take into account the illustration of a BERT[17]-style architecture that is pre-trained on a Masked language model and Next Sentence Prediction target before being refined on downstream tasks like classification. Here, we individually adjust several pre-trained model instances for a variety of downstream tasks. Contrarily, the text-to-text framework advises applying the same model, loss function, and hyperparameters to all NLP applications. This method produces the "text" equivalent of the anticipated result after modelling the inputs to enable the model to recognise a task.

ii C4- Colossal Clean Crawled Corpus

Pre-training language models using sizable unlabeled datasets is a cliché. One such dataset is the typical crawl. It is acquired via scraping web pages, skipping over any HTML markup. Each month, it generates about 20TB of scraped data. However, Common Crawl has much useless content, such as menus, error warnings, and duplicate sentences. Additionally, there is a substantial amount of content that is irrelevant to our jobs, such as objectionable language, placeholder text, or source codes.

The authors used the Common Crawl scrape from April 2019 for C4 after being cleaned up using the following filters:

- Only keeping sentences that have proper terminal punctuation.
- In addition, any page that contains words on the "List of Dirty, Naughty, Obscene, or Otherwise Bad Words" is removed.
- By filtering out every line that contains the word JavaScript, warnings of the "JavaScript must be enabled" variety are eliminated.
- Removed are any pages that included placeholder text like "lorem ipsum."
- To display the source code, any pages with curly brackets ("") on them are removed (since curly braces appear in many well-known programming languages).
- Three-sentence stretches are taken into account when eliminating duplicates. The same three sentences are cleaned out of any repetitions.
- Last but not least, a lengthy detection is used to remove any pages not identified as English because the downstream tasks are mainly for the English language.

As a result, a dataset of 750GB was created, which is noticeably larger than most pre-training datasets and includes comparatively uncontaminated text.

iii Model

The suggested concept modifies an encoder-decoder transformer[14] in terms of architecture (like applying Layer Normalization before a sub-block and then adding the initial input to the sub-block output, also known as pre-norm). The model configuration is also comparable to the BERT[16] base.

iv Training Approach

The transformer model T5 uses a slightly modified version of the Masked Language Model as its training aim, which is the same as that of BERT. Bidirectional models include Masked Language Models. The representation of the word is taken from both the

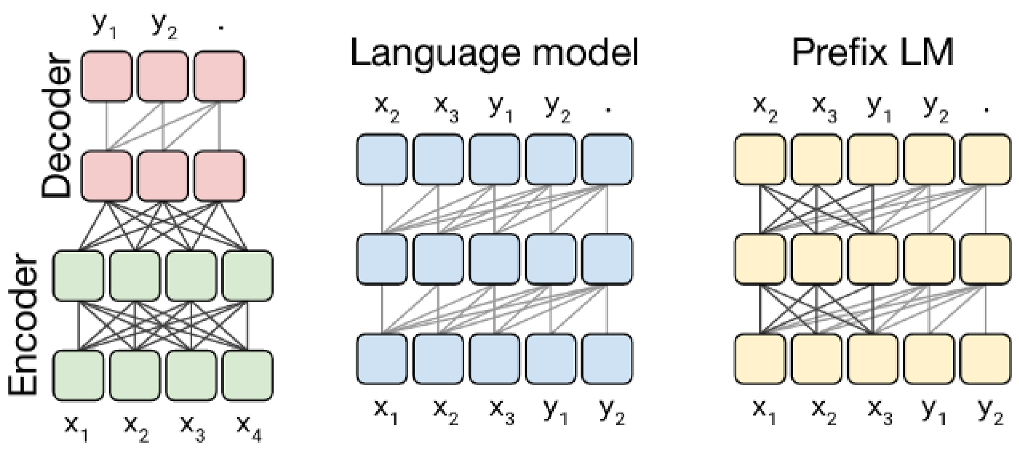


Figure 4.5: Model Structures of T5 transformer model

left and the appropriate context at any moment t. Unlike BERT, which utilises a Mask token for each phrase, T5 uses a single Mask keyword to replace many consecutive tokens. The accompanying figure shows how adding perturbations converts the original text into input and output pairs. The targets were created to produce a sequence, as opposed to BERT, that tries to output one word (itself) through the final feed-forward and softmax at the output level because the end goal is to have trained a model that inputs text and outputs text.

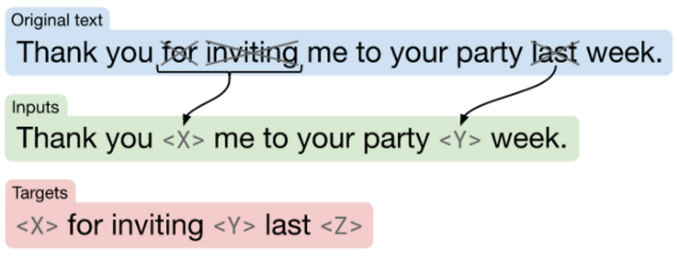

Original text
Thank you ~~for inviting~~ me to your party ~~last~~ week.
Inputs
Thank you <X> me to your party <Y> week.
Targets
<X> for inviting <Y> last <Z>

Figure 4.6: Corrupting Spans in T5 transformer model

As part of the pre-training, the model was trained on the C4[22] corpus with the same goal. Then, it was adjusted for various tasks, including sentence similarity, summarization, and language translation. Fine-tuning was carried out by displaying to the model the I/O text pairs with task-specific prefix-text appended to each input. By limiting the model's generation scope, adding a prefix, for instance, allowed it to adjust its weight for a specific task and would only provide the required output for that task alone. The goal, training method, and decoding process are all the same for all tasks. The authors assert that they have not discovered any instances in which the model has become confused and has produced an unexpected result or the predicted result of a different task. They also modelled regression tasks like phrase similarity as a text creation target, which was pretty exciting. Finally, they constructed a number between 0 and 5 with 0.2 quantization, limiting the model to numbers with a 0.2 difference to narrow the range of real numbers.

The authors tested and tuned hyperparameters extensively for a variety of workloads. The tuning is depicted in the diagram below at various levels.

- **Pre-training Style**: They experimented with Deshuffling & denoising objective, BERT-style Masked Language Model objective, and conventional auto-regressive language modelling objective. They discovered that the BERT[16] style (missing context prediction) was the best option for pre-training the model.
- **Corruption Scheme**: Three different corruption strategies were tested: masking a random word, masking a span of words (more than one consecutive word), and removing a word from the input. Both I/O are text strings given the task type at hand. They found that corrupting the span worked best.
- **Corruption Rate**: They experimented with various corruption rates and discovered that all of them performed essentially the same, with 15% performing marginally better.
- **Corruption Length**: They also experimented with various corruption span lengths and discovered that the model performed poorly the more extended the span length, which seems to be accurate given that a span length of one sentence would require the model to construct text from scratch, allowing it the freedom to be highly variable.

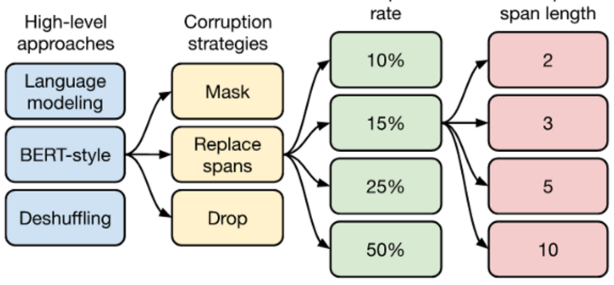


Figure 4.7: Corrupting Strategies in T5 transformer model

4.3.2 PEGASUS

This section will explain Google AI called "PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization", published in ICML 2020.PEGASUS[23] carries out the seq2seq architecture like any other sequence transduction task. Nevertheless, the new aspect of this architecture is its self-supervised pre-training goal.

i Gap Sentences Generation (GSG): Self-Supervised Objective for Summarization

The newest hot trend in deep learning is self-supervised learning. It essentially ends data's reliance on labelled samples and opens up a new pool of unlabeled data for training. Google AI Blog's self-supervised pre-training in PEGASUS The fundamental presumption underlying this aim is that the closer the pre-training self-supervised target is to the ultimate downstream task, the better the performance will be in fine-tuning. As a result, whole sentences are "masked" in PEGASUS so that the model can be taught to predict them. The authors acknowledge that this endeavour appears almost insurmountable, even for humans. However, such training results in a greater comprehension for creating phrases that include a sample of the original content, confirming their assertion. The name of this activity is Gap Sentence Generation (GSG). The writ-

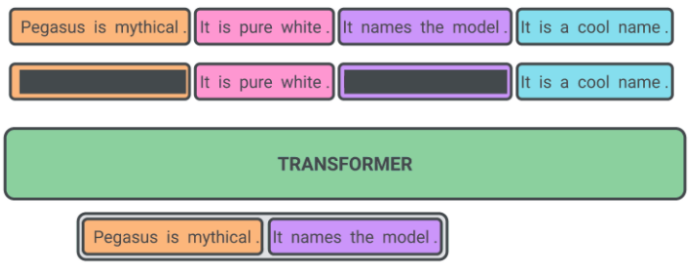


Figure 4.8: Gap Sentence Generation

ers also claim that it is better to mask the most significant sentences in the manuscript. This is accomplished by identifying sentences that, using a metric called ROUGE, are most similar to the entire document (usually used to evaluate the summary quality in summarization tasks).

ii Masked Language Model (MLM)

In the MLM[16] pre-training objective, the model randomly masks words from the sequences and predicts these masked words using other words from the sequence. This

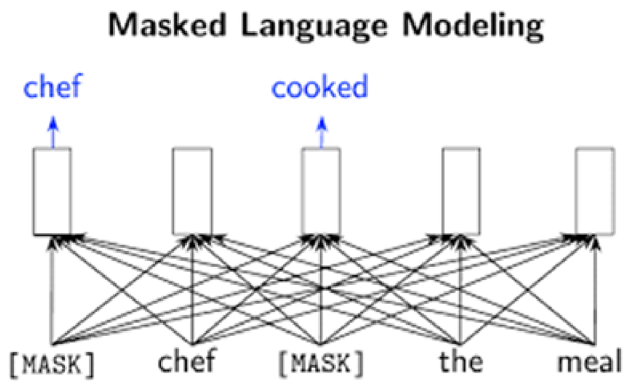


Figure 4.9: Masked Language Model in PEGASUS

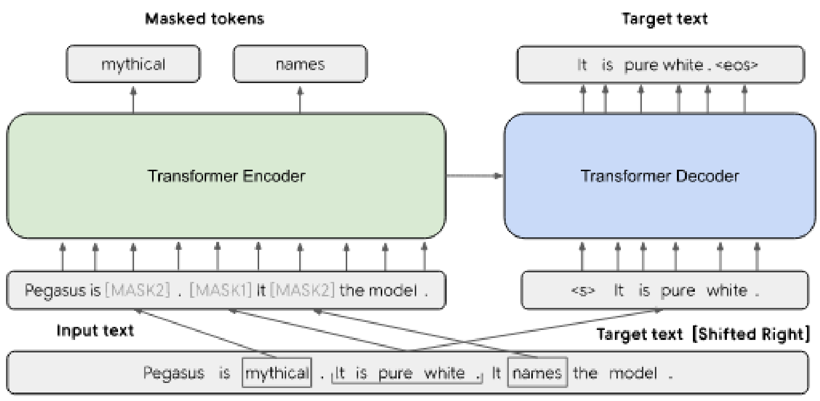

Masked tokens
mythical
names
Target text
It is pure white . <eos>
Transformer Encoder
Transformer Decoder
Pegasus is [MASK2] . [MASK1] It [MASK2] the model .
<s> It is pure white .
Input text
Target text [Shifted Right]
Pegasus is mythical . It is pure white . It names the model .

Figure 4.10: Combined Training in PEGASUS

idea can interpret the GSG task as a document-level MLM. Same as in the BERT, the model chooses 15% of the tokens in the input text, and the chosen tokens are either:

1. Replaced 80% of the time by a mask token [MASK2]
2. Replaced 10% of the time by a random token
3. Left unmodified 10% of the time

The Transformer encoder is trained using MLM as the only pre-training objective or in conjunction with GSG[25]. When fine-tuning downstream tasks by Rothe et al., the Transformer decoder shares all parameters with the encoder when MLM is the only pre-training aim .

The PEGASUS Transformer is pre-trained using a combination of the two methods covered earlier. Fig 4.12 is an example that simultaneously uses GSG and MLM as pre-training objectives. There are three sentences at the beginning. As target generating text, [MASK1] is used to mask one sentence (GSG). The other two sentences are still present in the input, but [MASK2] has arbitrarily disguised some words (MLM).

4.3.3 BART

A transformer model developed by Facebook AI research, called Bidirectional Auto-Regressive Transformers (BART), combines the decoder-only model GPT from OpenAI and the encoder-only model BERT from Google. The BERT requires knowledge of the entire input sequence and is adequate for downstream tasks like categorization. However, the generated word should only depend on previously generated words. Hence it is not ideal for generation tasks where this is required. On the other hand, General Purpose Transformer (GPT) adopts a unidirectional auto-regressive strategy when coming. It works well for text production but not so well for tasks like categorization that call for knowledge of the entire sequence.

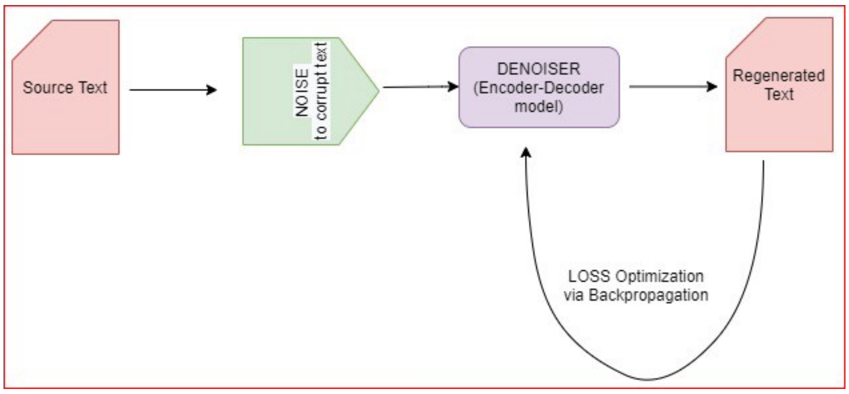


Figure 4.11: BART Architecture

BART is a self-supervised auto-encoder that first takes a source text with noise added to it as input and then uses a language model to predict how to replace the corrupted tokens to reconstruct the original text. The model performs best for Natural Language Generation (NLG) tasks. However, comprehension tasks can also benefit from its use.

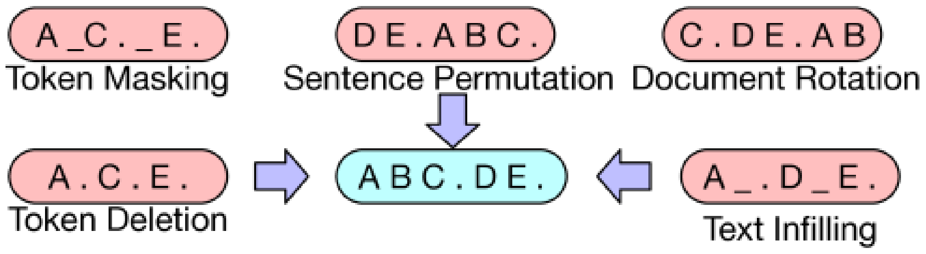


Figure 4.12: Transformations used for noising the input in the BART model

In the BART study, a transformer-based Seq2Seq model is used, and each layer of the decoder cares for the last hidden layer of the encoder. The model attempts to denoise distorted source text by producing the original text from the decoder. It can be viewed as a Seq2Seq model that has been altered to function as an auto-encoder. The architecture's use of GELU[17] rather than the RELU[19] activation layer is noteworthy. It does not use a feed-forward network at the top as BERT does for word prediction. Furthermore, BART performs better for tasks requiring language production and utilizes just 10% more parameters than analogous BERT-based architecture. The initialization of the architecture's parameters as a normal distribution is N(0.00,0.02).

1. **Masking of Tokens**: [MASK] tokens are used to mask a sample of random tokens.

2. **Token deletion**: After some randomly selected tokens have been sampled and eliminated (a task similar to masking), the model assigns a new token to take their place.

3. **Infilling of Tokens**: From the Poisson distribution, several text spans (a collection of connected tokens) are selected, and each span is substituted by a masked token [MASK].

4. **Permutation of Sentences:**Random sentence rearranging is known as sentence permutation.

5. **Rotation of Documents**: A token is evenly and randomly selected from it, and the document is rotated around that token.

Combining BERT for encoding the corrupted source text and GPT for recreating the original text is recommended by predicting the masked tokens to get good performance for such applications. Furthermore, to enhance the efficiency of downstream tasks like Natural Language Generation and comprehension tasks, a seq2seq-based architecture termed BART is required.

BART can be fine-tuned to bear impressive performance over different downstream tasks:

- Sequence Classification: The pre-trained BART is employed, and the top hidden states of the decoder are used as a meaningful input representation of sequence and in a new multi-class classifier. The final representation of the decoder's output.

- Token Classification: The pre-trained BART is employed, and the top hidden states of the decoder, which produce the final representation of the decoder's output, are used to classify the tokens by providing a meaningful representation of each word.

- Sequence Generation: As the decoder outputs a sequence that contains information copied from the original input sequence, it is comparable to denoising the pre-training target. It can be applied to summarise information and respond to questions by generating sequences.

- Automated Translation A novel encoder is introduced that accepts the source sequence as input and assumes the entire pre-trained encoder-decoder is a decoder that will generate the target sequence. Only the new encoder's parameters are learned in the first stage, where the encoder tries to get knowledge about alignment between the input and target sequences during training, while the parameters of the previously trained architecture are locked. The second step is learning the entire design in fewer iterations.

4.4 Hyperparameters

A mathematical model called a machine learning model has several parameters that must be learned from the data.We can fit the model's parameters by using pre-existing data to train a model.On the other hand, hyperparameters are a different class of parameters that cannot be directly learned through regular training.Usually, they are fixed before the start of the programme itself. These variables indicate the model's fundamental characteristics, such as its complexity or how quickly it should pick up new information.For this study, we conducted hyperparameter tuning in the different transformer models mentioned earlier. The hyperparameters used in this study will explain below.

4.4.1 Number Epochs

The number of epochs, a hyperparameter, controls how many times the learning algorithm will run through the whole training dataset.At the end of one epoch, the internal model parameters have been updated for each sample in the training dataset. One or more batches make up an epoch.

4.4.2 Learning rate

The "learning rate" or "step size" refers to how frequently the weights are updated during training[20]. In particular, the learning rate is a hyperparameter that can be customised and used to train neural networks. Its value is typically small and positive, between 0.00 and 1.00. The learning rate specifies how quickly the model adapts to the challenge. Given the more minor changes to the weights made with each update, lower learning rates necessitate more training epochs, whereas more considerable learning rates produce quick changes and necessitate fewer training epochs. While a learning rate that is too modest can stall the process, one that is too great can lead to the model converging too soon to an unsatisfactory result. Carefully choosing the learning rate is challenging when developing deep learning neural networks. However, it might be the model's most crucial hyperparameter.

4.4.3 Batch Size

The batch size[20] is a hyperparameter that tells how many samples must be processed before the internal model parameters are updated. Consider a batch as a for-loop making predictions while iterating over one or more samples. The predictions are compared to the anticipated output variables after the batch, and an error is calculated. The update procedure is employed from this error to enhance the model, for example, by moving along the error gradient. One or more batches can be created from a training dataset. When all training samples are combined into a single batch, the learning algorithm is known as batch gradient descent. Stochastic gradient descent is the name of the learning algorithm when the batch size is one sample. When the batch number is greater than one sample and less than the size of the training dataset, the learning algorithm is known as mini-batch gradient descent.

4.4.4 Number of Beams

A hyperparameter in the text generation tasks is the number of beams. The beam search algorithm[23] will use it. In each timestep, the beam search algorithm chooses several options for an input sequence based on conditional probabilities. For example, beam Width, often known as the number of beams B, is a characteristic that affects the number of many choices. The beam search chooses B best alternatives with the highest probability as the most likely candidates for the time step at each time step.

4.5 ROUGE metric

Recall-Oriented Understudy for Gisting Evaluation is referred to as ROUGE[19]. It is a collection of measures for rating machine translations and artificial text summarization. It operates by contrasting a summary or translation generated automatically with a set of reference summaries (typically human-produced).

The metric is mainly focusing on Recall.The recall divides the total number of n-grams in the reference by the number of overlapping n-grams discovered in both the model output and the reference. The granularity of texts being compared between system summaries and reference summaries can be considered ROUGE-N, ROUGE-S, and ROUGE-L.

- **ROUGE-N**: Measures the overlap of unigrams, bigrams, trigrams, and higher order n-grams.

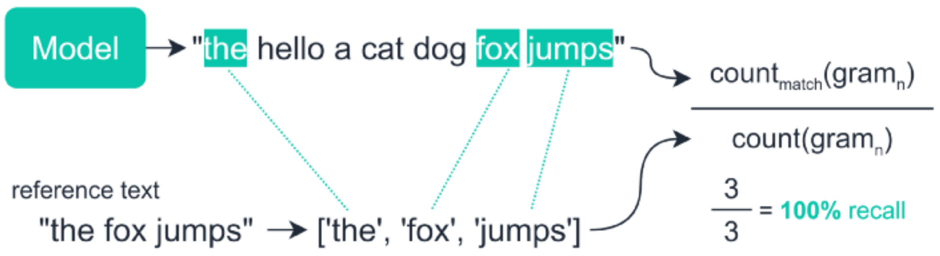


Figure 4.13: ROUGE-N Example

- **ROUGE-L**: uses longest common subsequence-(LCS) to measure the most extended matched word sequence. LCS has the benefit of only requiring in-sequence matches that accurately reflect sentence-level word order, as opposed to consecutive matches. You do not require a predetermined n-gram length because it automatically incorporates the longest common n-grams that occur in sequence.

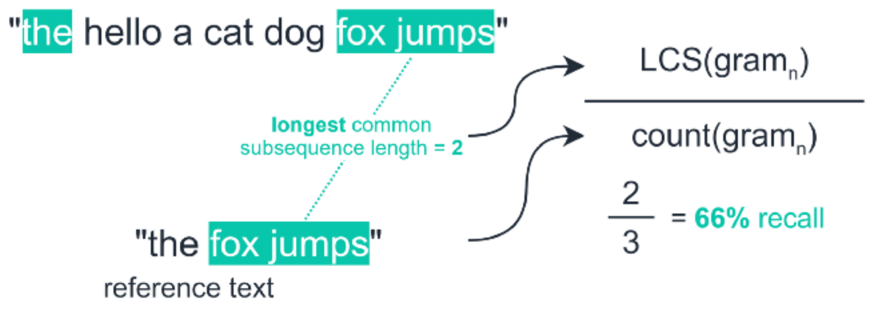


Figure 4.14: ROUGE-N Example

- **ROUGE-S**: Any word pair in a phrase that allows for arbitrary gaps is in order. Another name for this is skip-gram concurrence. For instance, the skip-bigram method assesses word pairings' overlap when there are no more than two spaces between the words. As an illustration, the skip-bigrams for "cat in the hat" would be "cat in, cat the cat hat, in the, in the hat, the hat."

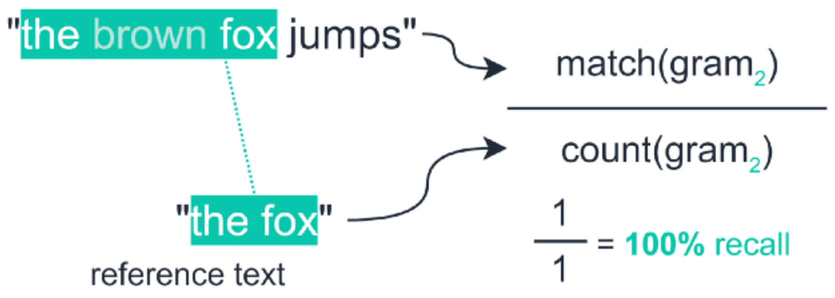


Figure 4.15: ROUGE-N Example

CHAPTER 5

IMPLEMENTATION

In this chapter, the implementation details are given. The modules, libraries, software required are mentioned here. The implementation code is executed in Python language and all the modules and libraries used are supported by Python.

5.1 Tools and Libraries

5.1.1 TensorFlow

A complete open-source machine learning platform is called TensorFlow. It has a vast, adaptable ecosystem of resources from the community, libraries, and tools. It enables developers to quickly create and deploy ML-powered apps and researchers to advance the state-of-the-art in ML. To undertake machine learning and deep neural network research, the Google Brain team, a group of researchers and engineers within Google's Machine Intelligence Research division, created TensorFlow. The approach, however, is sufficiently all-encompassing to be helpful in a wide range of other disciplines.

5.1.2 Keras

A Python library with open-source software called Keras offers an API for artificial neural networks. Keras provide the TensorFlow library interface. As well as various tools to make working with image and text data easier, Keras includes numerous implementations of widely used neural-network building blocks like layers, objectives, activation functions, and optimizers. This helps to simplify the coding required to create deep neural networks. The GitHub problems page and a Slack channel serve as community support forums; the source is available on GitHub.

5.1.3 PyTorch

An open-source machine learning (ML) framework called PyTorch is built using the Torch library and the Python programming language. One of the most popular platforms for deep learning research is this one. The framework is designed to hasten the transition from research prototyping to implementation. Like NumPy, PyTorch uses tensor computations accelerated by graphics processing units (GPU). Tensors are multidimensional arrays that may be worked with and modified via APIs. The PyTorch framework supports over 200 different mathematical operations.

5.1.4 Transformers

Transformers is a Python package by Huggingface for implementing NLP Tasks, especially for using pre-trained NLP models. Transformers offers APIs that make it easy to download and effectively train modern pre-trained models. Utilizing pre-trained models can cut down on time spent training a model from scratch and lower systems' carbon footprint and computation costs.

5.1.5 NLTK

A platform for creating Python programmes that operate with human language data for applications involving statistical natural language processing (NLP) is called the Natural Language Toolkit (NLTK).

It includes text processing libraries for tokenization, parsing, classification, stemming, tagging, and semantic reasoning. Along with a cookbook and a book that describes the concepts underlying the basic language processing tasks that NLTK offers, it also contains visual demonstrations and sample data sets.

5.1.6 Pandas

One of the popular Python libraries for data analysis and machine learning activities is called Pandas. It is constructed on top of the multi-dimensional array-supporting Numpy library. In Python, Pandas is one of the most widely used data wrangling tools. It integrates well with various data science modules, and it is frequently included in all Python distributions, including those sold by commercial vendors like ActiveState's ActivePython and those that come with your operating system.

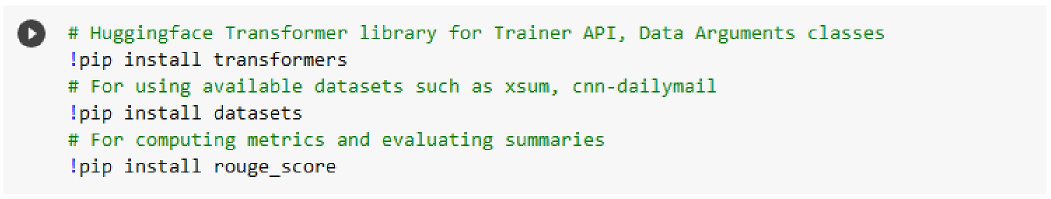
5.1.7 Google Colaboratory

In terms of AI research, Google is reasonably active. For example, TensorFlow, an artificial intelligence framework, and Colaboratory, a development platform, were both created by Google. TensorFlow is freely available, and Google has made Colaboratory available to everyone. As a result, Colaboratory is known as Google Colab or just Colab. The utilisation of GPU is another appealing feature that Google provides for developers. Colab is free and supports GPU. By making it available to the public, its software may become the de facto method for teaching data science and machine learning in universities. It might also have a long-term goal of developing a user base for the pay-per-use Google Cloud APIs. Whatever the cause, Colab's debut has made it simpler to learn about and create machine learning applications.

5.2 Implementation steps

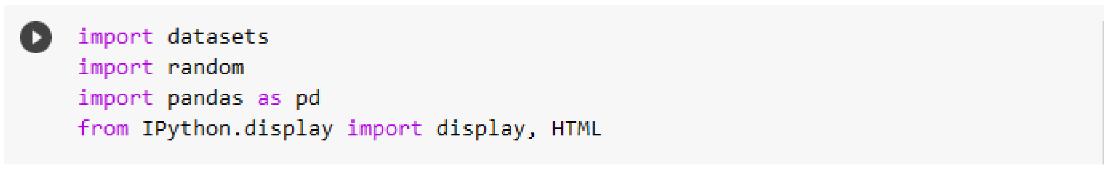
5.2.1 Installing and importing libraries

The above python libraries can be installed using the python-pip library, which is already preinstalled while installing python. The python libraries can be installed using the '!pip install' command in the jupyter notebook or using 'pip install' in Linux's Terminal or Window's Command prompt. After installing packages by calling the 'import the-package-name' can make access to the package.Example of installing and importing of packages are given below 5.1 & 5.2



```
# Huggingface Transformer library for Trainer API, Data Arguments classes
!pip install transformers
# For using available datasets such as xsum, cnn-dailymail
!pip install datasets
# For computing metrics and evaluating summaries
!pip install rouge_score
```

Figure 5.1: Example of installing Package using pip

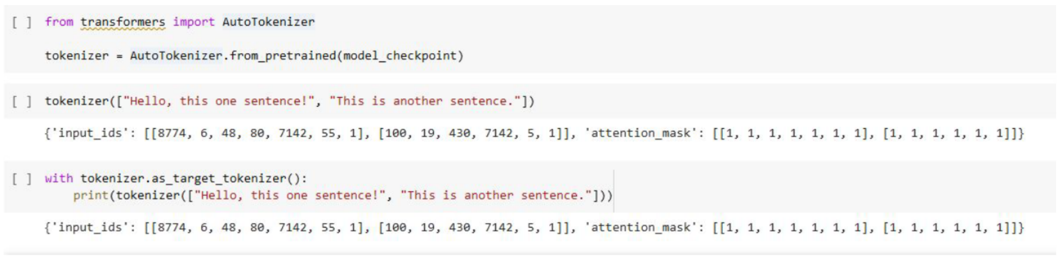


```
import datasets
import random
import pandas as pd
from IPython.display import display, HTML
```

Figure 5.2: Example of importing of packages

5.2.2 Dataset Preprocessing

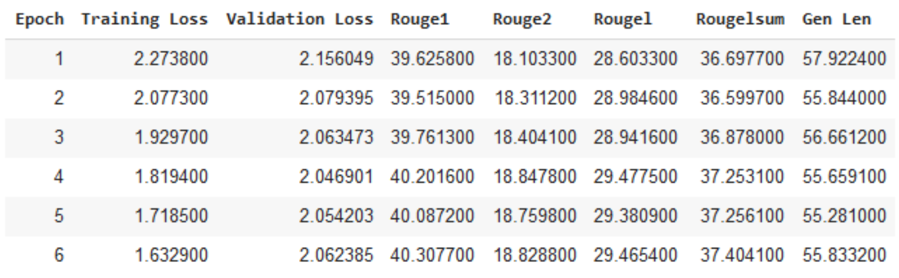
The dataset collected from the Kaggle website and dataset consists of three columns/features id, article and highlight. Removed id feature because it has a unique value in each entry. While exploring the two other features, many unwanted characters exist, which will not help in the training/finetuning process. So the dataset was cleaned from extra spaces, digits, and characters and replaced with all contractions using their parent word. after data cleaning, the dataset needed to tokenize for finetuning the pre-trained transformer model. for the tokenization, used an auto tokenizer function from hugging face. By only passing the model name and dataset, the function automatically tokenize according to the model needs.



```
[ ] from transformers import AutoTokenizer

    tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)

[ ] tokenizer(["Hello, this one sentence!", "This is another sentence."])

    {'input_ids': [[8774, 6, 48, 80, 7142, 55, 1], [100, 19, 430, 7142, 5, 1]], 'attention_mask': [[1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1]]}

[ ] with tokenizer.as_target_tokenizer():
        print(tokenizer(["Hello, this one sentence!", "This is another sentence."]))

    {'input_ids': [[8774, 6, 48, 80, 7142, 55, 1], [100, 19, 430, 7142, 5, 1]], 'attention_mask': [[1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1]]}
```

Figure 5.3: Example of tokenization

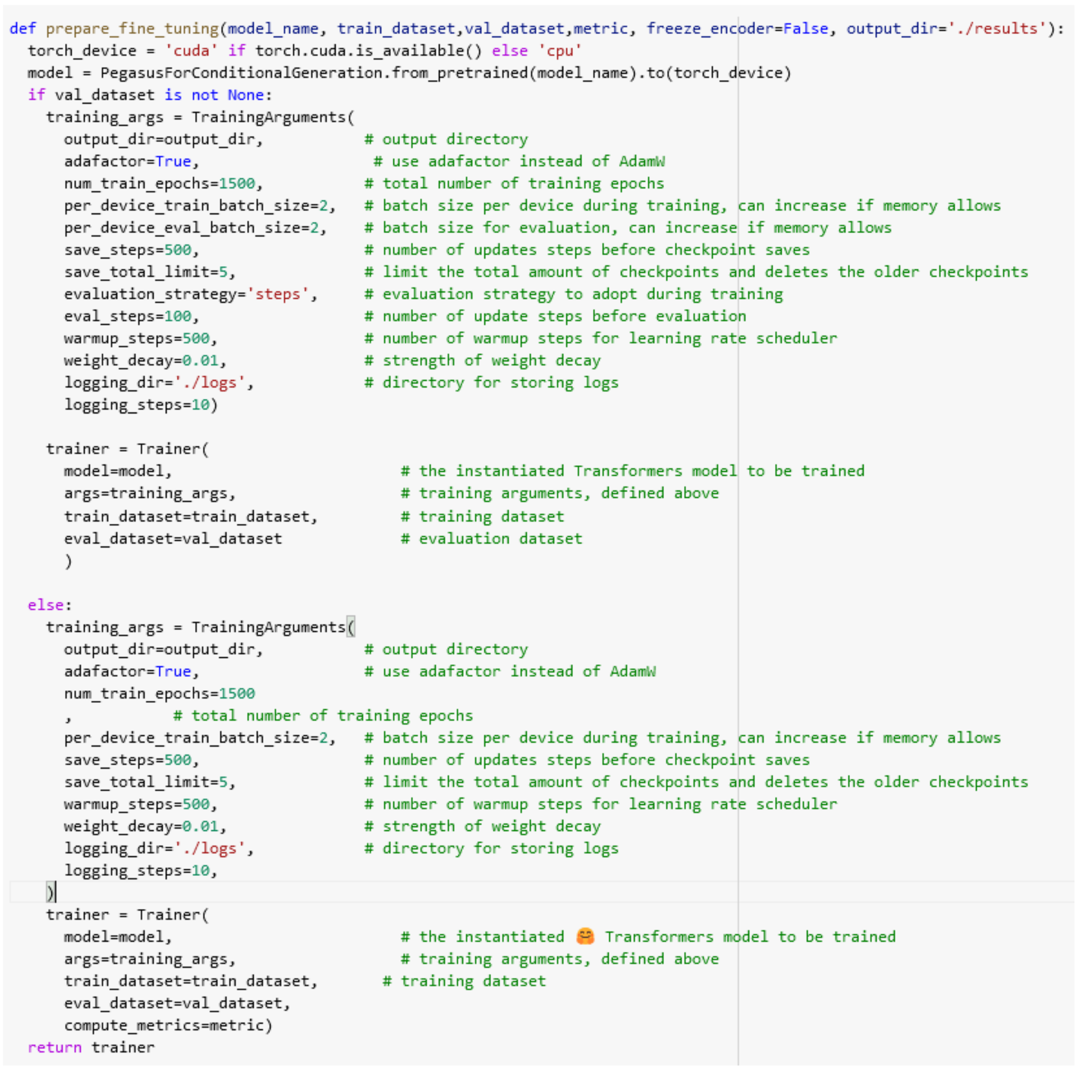
5.2.3 Model Fine-tuning



Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
1	2.273800	2.156049	39.625800	18.103300	28.603300	36.697700	57.922400
2	2.077300	2.079395	39.515000	18.311200	28.984600	36.599700	55.844000
3	1.929700	2.063473	39.761300	18.404100	28.941600	36.878000	56.661200
4	1.819400	2.046901	40.201600	18.847800	29.477500	37.253100	55.659100
5	1.718500	2.054203	40.087200	18.759800	29.380900	37.256100	55.281000
6	1.632900	2.062385	40.307700	18.828800	29.465400	37.404100	55.833200

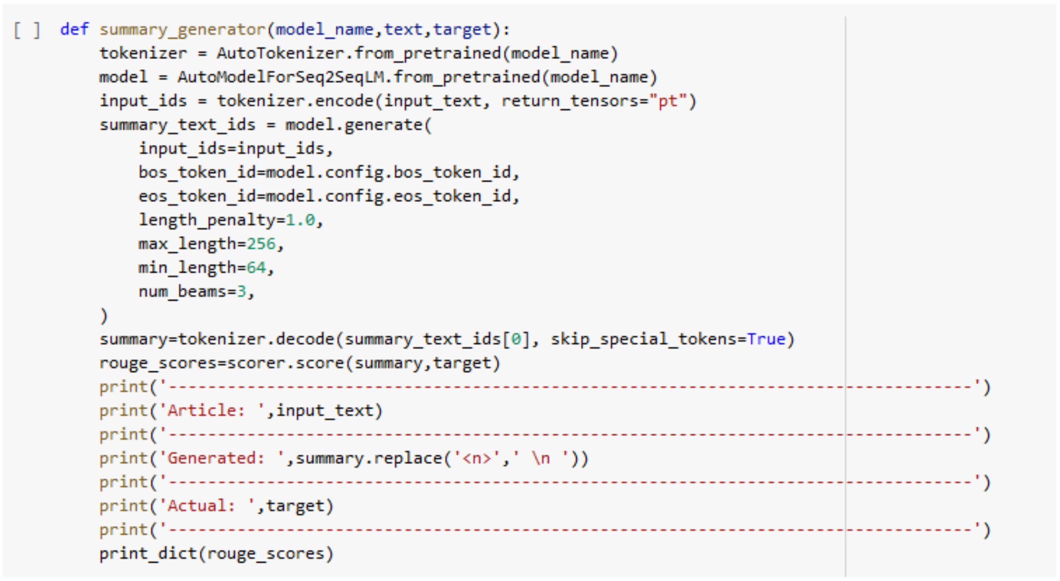
Figure 5.4: Sample fine-tuning result

We need the pre-trained model and its pre-trained weights for finetuning pre-trained transformer models. Model and its model weights fetched using the huggingface's transformers package. After getting the model and its pre-trained model weights, we need to specify the training arguments. for that, we use the TrainingArguments function. we can pass the required parameters into the TrainingArguments function.after setting the training parameter, we need to finetune the model. for that, we use the Trainer function from the transformer model. inside the Trainer function, we have to pass the Training Arguments, train validation dataset and metric for the evaluation.



```python
def prepare_fine_tuning(model_name, train_dataset,val_dataset,metric, freeze_encoder=False, output_dir='./results'):
  torch_device = 'cuda' if torch.cuda.is_available() else 'cpu'
  model = PegasusForConditionalGeneration.from_pretrained(model_name).to(torch_device)
  if val_dataset is not None:
    training_args = TrainingArguments(
      output_dir=output_dir,           # output directory
      adafactor=True,                   # use adafactor instead of AdamW
      num_train_epochs=1500,           # total number of training epochs
      per_device_train_batch_size=2,   # batch size per device during training, can increase if memory allows
      per_device_eval_batch_size=2,    # batch size for evaluation, can increase if memory allows
      save_steps=500,                  # number of updates steps before checkpoint saves
      save_total_limit=5,              # limit the total amount of checkpoints and deletes the older checkpoints
      evaluation_strategy='steps',     # evaluation strategy to adopt during training
      eval_steps=100,                  # number of update steps before evaluation
      warmup_steps=500,                # number of warmup steps for learning rate scheduler
      weight_decay=0.01,               # strength of weight decay
      logging_dir='./logs',            # directory for storing logs
      logging_steps=10)

    trainer = Trainer(
      model=model,                         # the instantiated Transformers model to be trained
      args=training_args,                  # training arguments, defined above
      train_dataset=train_dataset,         # training dataset
      eval_dataset=val_dataset             # evaluation dataset
      )

  else:
    training_args = TrainingArguments(
      output_dir=output_dir,           # output directory
      adafactor=True,                  # use adafactor instead of AdamW
      num_train_epochs=1500
      ,          # total number of training epochs
      per_device_train_batch_size=2,   # batch size per device during training, can increase if memory allows
      save_steps=500,                  # number of updates steps before checkpoint saves
      save_total_limit=5,              # limit the total amount of checkpoints and deletes the older checkpoints
      warmup_steps=500,                # number of warmup steps for learning rate scheduler
      weight_decay=0.01,               # strength of weight decay
      logging_dir='./logs',            # directory for storing logs
      logging_steps=10,
    )
    trainer = Trainer(
      model=model,                         # the instantiated 🤗 Transformers model to be trained
      args=training_args,                  # training arguments, defined above
      train_dataset=train_dataset,       # training dataset
      eval_dataset=val_dataset,
      compute_metrics=metric)
  return trainer
```

Figure 5.5: Fine-Tuning details



```python
[ ]  def summary_generator(model_name,text,target):
         tokenizer = AutoTokenizer.from_pretrained(model_name)
         model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
         input_ids = tokenizer.encode(input_text, return_tensors="pt")
         summary_text_ids = model.generate(
             input_ids=input_ids,
             bos_token_id=model.config.bos_token_id,
             eos_token_id=model.config.eos_token_id,
             length_penalty=1.0,
             max_length=256,
             min_length=64,
             num_beams=3,
         )
         summary=tokenizer.decode(summary_text_ids[0], skip_special_tokens=True)
         rouge_scores=scorer.score(summary,target)
         print('---------------------------------------------------------------------------------')
         print('Article: ',input_text)
         print('---------------------------------------------------------------------------------')
         print('Generated: ',summary.replace('<n>',' \n '))
         print('---------------------------------------------------------------------------------')
         print('Actual: ',target)
         print('---------------------------------------------------------------------------------')
         print_dict(rouge_scores)
```

Figure 5.6: Summary generating function

CHAPTER 6

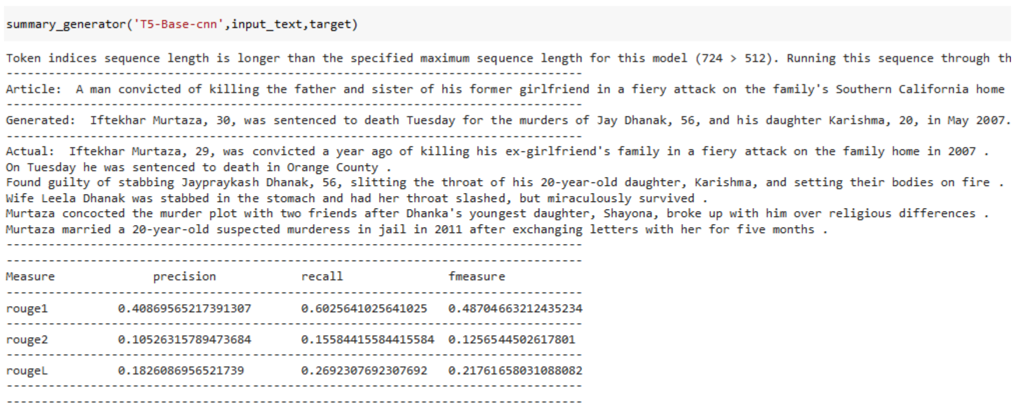
RESULTS

In this chapter the prediction results are discussed. The result is predicted on the test dataset.

Table 6.1: T5 model Abstractive Summarization results

MODEL	TIME	EPOCH	BATCH	T-Loss	Beam	R1	R2	LR
BASE	$4:37$	1	2	2.31	3	26.83	11.69	$2e-5$
BASE	$42:23$	3	3	1.21	3	26.91	11.97	$1e-6$
SMALL	$2:44$	1	8	1.79	3	24.54	11.81	$1e-6$
SMALL	$8:05$	3	8	1.81	5	24.50	11.79	$2e-5$

The experiments are carried out in the different T5 models. Two variants of the T5 model (T5 base model and T5 small model) are analyzed. For the T5 base model, batch size of 2, learning rate of 2e-5, and for single epoch, the training loss and validation loss are comparatively high. On the other hand, the rogue one and rogue two scores are low. T5 small variant with single epoch and three epochs are also analyzed. The losses are less for T5 small models compared to the T5 base with a single epoch.

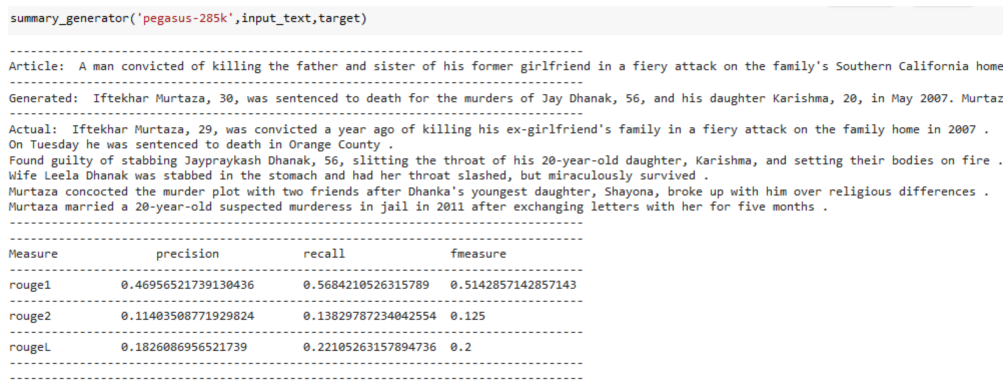


```
summary_generator('T5-Base-cnn',input_text,target)

Token indices sequence length is longer than the specified maximum sequence length for this model (724 > 512). Running this sequence through th
--------------------------------------------------------------------------------
Article:  A man convicted of killing the father and sister of his former girlfriend in a fiery attack on the family's Southern California home
--------------------------------------------------------------------------------
Generated:  Iftekhar Murtaza, 30, was sentenced to death Tuesday for the murders of Jay Dhanak, 56, and his daughter Karishma, 20, in May 2007.
--------------------------------------------------------------------------------
Actual:  Iftekhar Murtaza, 29, was convicted a year ago of killing his ex-girlfriend's family in a fiery attack on the family home in 2007 .
On Tuesday he was sentenced to death in Orange County .
Found guilty of stabbing Jaypraykash Dhanak, 56, slitting the throat of his 20-year-old daughter, Karishma, and setting their bodies on fire .
Wife Leela Dhanak was stabbed in the stomach and had her throat slashed, but miraculously survived .
Murtaza concocted the murder plot with two friends after Dhanka's youngest daughter, Shayona, broke up with him over religious differences .
Murtaza married a 20-year-old suspected murderess in jail in 2011 after exchanging letters with her for five months .
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Measure             precision           recall              fmeasure
--------------------------------------------------------------------------------
rouge1         0.40869565217391307    0.6025641025641025    0.48704663212435234
--------------------------------------------------------------------------------
rouge2         0.10526315789473684    0.15584415584415584   0.1256544502617801
--------------------------------------------------------------------------------
rougeL         0.182608695652173 9    0.2692307692307692    0.21761658031088082
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Figure 6.1: T5-base summary generation

However, the T5 base model having a batch size of 3 and learning rate 1e-6, gave the least training loss for a running epoch of 3. The rogue scores are also comparatively good for this experimental setup. The total time taken is 42.23 hours for the training of this model.

In the Pegasus model, two experiments are carried out with the same batch size but different epochs. The learning rate used is 2e-5. For epoch 3, the model gave comparatively less training loss. There is a slight change in training loss and rogue scores in two experimental setups. The time taken for training is three times in 3 epochs than that of a single epoch, and the results are less varied.

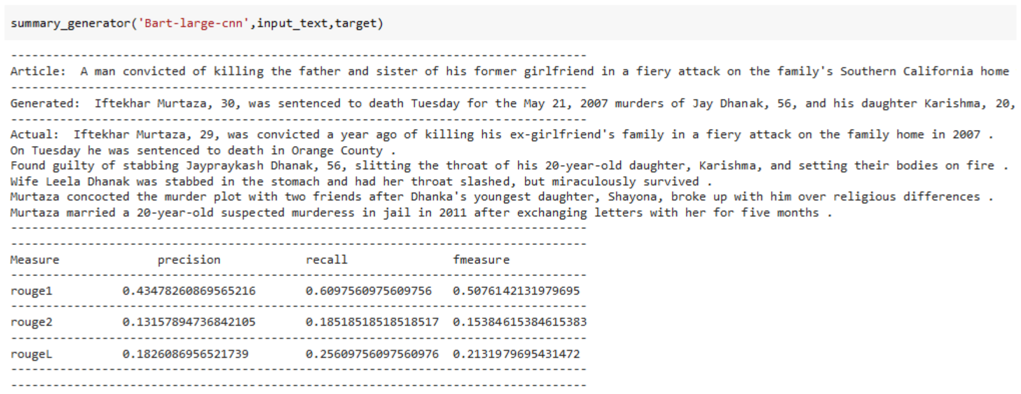


```
summary_generator('pegasus-285k',input_text,target)

---------------------------------------------------------------------------
Article:  A man convicted of killing the father and sister of his former girlfriend in a fiery attack on the family's Southern California home
---------------------------------------------------------------------------
Generated:  Iftekhar Murtaza, 30, was sentenced to death for the murders of Jay Dhanak, 56, and his daughter Karishma, 20, in May 2007. Murtaz
---------------------------------------------------------------------------
Actual:  Iftekhar Murtaza, 29, was convicted a year ago of killing his ex-girlfriend's family in a fiery attack on the family home in 2007 .
On Tuesday he was sentenced to death in Orange County .
Found guilty of stabbing Jaypraykash Dhanak, 56, slitting the throat of his 20-year-old daughter, Karishma, and setting their bodies on fire .
Wife Leela Dhanak was stabbed in the stomach and had her throat slashed, but miraculously survived .
Murtaza concocted the murder plot with two friends after Dhanka's youngest daughter, Shayona, broke up with him over religious differences .
Murtaza married a 20-year-old suspected murderess in jail in 2011 after exchanging letters with her for five months .
---------------------------------------------------------------------------
---------------------------------------------------------------------------
Measure              precision              recall                fmeasure
---------------------------------------------------------------------------
rouge1         0.46956521739130436       0.5684210526315789    0.5142857142857143
---------------------------------------------------------------------------
rouge2         0.11403508771929824       0.13829787234042554   0.125
---------------------------------------------------------------------------
rougeL         0.1826086956521739        0.22105263157894736   0.2
---------------------------------------------------------------------------
---------------------------------------------------------------------------
```

Figure 6.2: PEGASUS summary generation

Table 6.2: PEGASUS model Abstractive Summarization results

MODEL	TIME	EPOCH	BATCH	T-Loss	BEAMS	R1	R2	LR
PEGASUS	31 : 20	1	3	1.559	3	41.6	18.7	$2e-5$
PEGASUS	105 : 52	3	3	1.553	3	41.8	18.87	$2e-5$



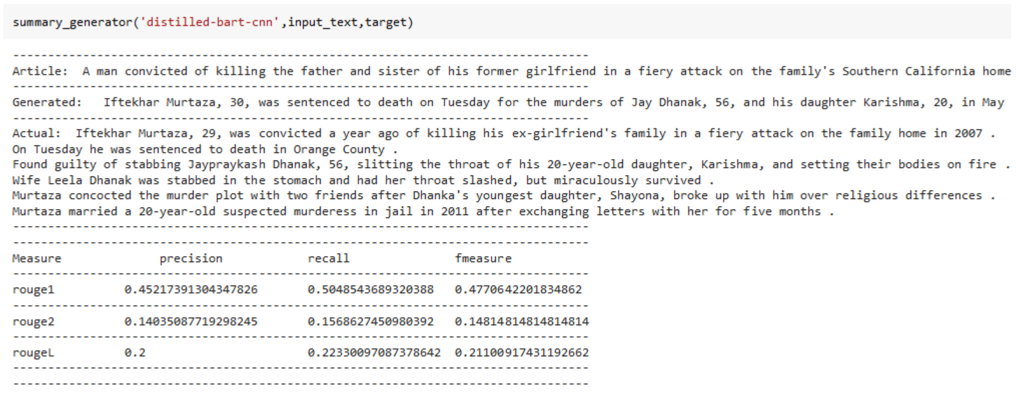
```
summary_generator('Bart-large-cnn',input_text,target)

---------------------------------------------------------------------------
Article:  A man convicted of killing the father and sister of his former girlfriend in a fiery attack on the family's Southern California home
---------------------------------------------------------------------------
Generated:  Iftekhar Murtaza, 30, was sentenced to death Tuesday for the May 21, 2007 murders of Jay Dhanak, 56, and his daughter Karishma, 20,
---------------------------------------------------------------------------
Actual:  Iftekhar Murtaza, 29, was convicted a year ago of killing his ex-girlfriend's family in a fiery attack on the family home in 2007 .
On Tuesday he was sentenced to death in Orange County .
Found guilty of stabbing Jaypraykash Dhanak, 56, slitting the throat of his 20-year-old daughter, Karishma, and setting their bodies on fire .
Wife Leela Dhanak was stabbed in the stomach and had her throat slashed, but miraculously survived .
Murtaza concocted the murder plot with two friends after Dhanka's youngest daughter, Shayona, broke up with him over religious differences .
Murtaza married a 20-year-old suspected murderess in jail in 2011 after exchanging letters with her for five months .
---------------------------------------------------------------------------
---------------------------------------------------------------------------
Measure              precision              recall                fmeasure
---------------------------------------------------------------------------
rouge1         0.43478260869565216       0.6097560975609756    0.5076142131979695
---------------------------------------------------------------------------
rouge2         0.13157894736842105       0.18518518518518517   0.15384615384615383
---------------------------------------------------------------------------
rougeL         0.1826086956521739        0.25609756097560976   0.2131979695431472
---------------------------------------------------------------------------
---------------------------------------------------------------------------
```

Figure 6.3: BART-large summary generation

BART large and BART base variants are used for the analysis. For the BART base model, with the same batch size and beam width of 3, single and three epochs are analyzed. Better performance is obtained as the epochs increase. Four experiments are carried out in BART large model by changing different parameters. For the batch size of 3, the epoch of 6 and the learning rate of 2e-5, BART large model gave less training loss. The validation loss is also less in this scenario. The ROUGE-1 score of 43.0 and ROUGE-2 score of 20.2, the highest among all scenarios, are obtained. The total training time for this experimental set-up is 127.38 hours.

Table 6.3: BART model Abstractive Summarization results

MODEL	TIME	EPOCH	BATCH	T-Loss	BEAMS	R1	R2	LR
BASE	$4:51$	1	12	1.87	3	40.40	18.73	$2e-5$
BASE	$20:19$	3	12	1.553	3	42.56	18.47	$2e-5$
LARGE	$16:32$	1	3	1.68	3	40.33	18.87	$2e-5$
LARGE	$127:38$	6	3	1.09	5	43.00	20.2	$2e-5$
BASE	$14:35$	1	3	1.7994	5	42.73	19.66	$5e-5$
BASE	$53:46$	6	8	1.3537	5	42.81	19.96	$5e-5$



```
summary_generator('distilled-bart-cnn',input_text,target)
```

```
--------------------------------------------------------------------------------
Article:  A man convicted of killing the father and sister of his former girlfriend in a fiery attack on the family's Southern California home
--------------------------------------------------------------------------------
Generated:   Iftekhar Murtaza, 30, was sentenced to death on Tuesday for the murders of Jay Dhanak, 56, and his daughter Karishma, 20, in May
--------------------------------------------------------------------------------
Actual:  Iftekhar Murtaza, 29, was convicted a year ago of killing his ex-girlfriend's family in a fiery attack on the family home in 2007 .
On Tuesday he was sentenced to death in Orange County .
Found guilty of stabbing Jaypraykash Dhanak, 56, slitting the throat of his 20-year-old daughter, Karishma, and setting their bodies on fire .
Wife Leela Dhanak was stabbed in the stomach and had her throat slashed, but miraculously survived .
Murtaza concocted the murder plot with two friends after Dhanka's youngest daughter, Shayona, broke up with him over religious differences .
Murtaza married a 20-year-old suspected murderess in jail in 2011 after exchanging letters with her for five months .
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Measure            precision           recall              fmeasure
--------------------------------------------------------------------------------
rouge1        0.45217391304347826    0.5048543689320388   0.4770642201834862
--------------------------------------------------------------------------------
rouge2        0.14035087719298245    0.1568627450980392   0.14814814814814814
--------------------------------------------------------------------------------
rougeL        0.2                    0.22330097087378642  0.21100917431192662
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Figure 6.4: DBART summary generation

distilBART model with four different epochs with beam width three is analyzed. For all the experiments in distilBART, the learning rate 2e-5 is used. Batch sizes of 24 and 32 are used in the experimental set-up. In the distilBART model, the training and validation loss is less for a batch size of 32 and an epoch of 12. However, the rogue-1 and rogue-2 scores are also high in this case. The total training time is 25.34 hours.

Table 6.4: DistlBART model Abstractive Summarization results

MODEL	TIME	EPOCH	BATCH	T-Loss	BEAMS	R1	R2	LR
DBART	$3:44$	1	24	3.47	3	39.51	18.29	$2e-5$
DBART	$9:57$	6	24	1.63	3	40.30	18.82	$2e-5$
DBART	$5:09$	3	32	1.96	3	40.20	18.84	$2e-5$
DBART	$25:34$	12	32	1.31	3	40.25	18.89	$2e-5$

Table 6.5: Comparison of different Abstractive text summarization models

MODEL	**ROGUE 1**	**ROUGE 2**
BRIO	47.78	23.55
GLM-XXLarge	23.55	21.4
PEGASUS	44.7	21.4
BART-Large	44.17	21.47
DCA	43.00	20.2
rnn-ext + RL	41.69	18.72

CHAPTER 7

CONCLUSIONS AND FUTURE WORKS

The proposed model uses a pre-trained transformer model for abstractive text summarization. The proposed model is used for generating text summaries of CNN news reports, and it is tested on new data. Implemented abstractive text summarization system using three different pre-trained transformer models. The models are pretty good at generating a summary from a text document. The system implemented using BART-large performed well and gave higher R1 and R2 scores compared to other models and variants. It outperforms existing Abstractive models like RNN, LSTM, Bi-LSTM, BERT, and Transformer in generating summaries from the text document. The model can generate text summaries like a human-written from document. However, the model is not optimal compared with recent models like BRIO and GLM; BRIO and GLM models perform much better than our proposed models due to large pre-training corpora and pre-training objectives.

The model can be re-trained using a large dataset and by increasing the complexity of language objective tasks to improve results are considered future works. Combining PEGASUS and BART language objectives can bring good results in abstractive text summary generation.

REFERENCES

[1] A. Nenkova and K. McKeown, “Automatic summarization,” in Proc. 49th Annu. Meeting Assoc. Comput. Linguistics, Hum. Lang. Technol., Jun. 2011.

[2] A. M. Rush, S. Chopra, and J. Weston, “A neural attention model for abstractive sentence summarization,” in Proc. Conf. Empirical Methods Natural Lang. Process., 2015, pp. 379-389.

[3] J. L. Elman, “Finding structure in time,” Cognit. Sci., vol. 14, no. 2,pp. 179-211, Mar. 1990.

[4] S. Chopra, M. Auli, and A. M. Rush, “Abstractive sentence summarization with attentive recurrent neural networks,” in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol., 2016, pp. 93-98.

[5] A. M. Rush, S. Chopra, and J. Weston, “A neural attention model for abstractive sentence summarization,” in Proc. Conf. Empirical Methods Natural Lang. Process., 2015, pp. 379-389.

[6] B. Hu, Q. Chen, and F. Zhu, “LCSTS: A large scale chinese short text summarization dataset,” in Proc. Conf. Empirical Methods Natural Lang. Process., 2015, pp. 1967-1972.

[7] R. Nallapati, B. Zhou, C. dos Santos, C. Gulcehre, and B. Xiang, “Abstractive text summarization using sequence-to-sequence RNNs and beyond,” in Proc. 20th SIGNLL Conf. Comput. Natural Lang. Learn.,2016, pp. 280-290.

[8] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” IEEE Trans. Signal Process., vol. 45, no. 11, pp. 2673-2681, 1997.

[9] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Comput., vol. 9, no. 8, pp. 1735-1780, Nov. 1997

[10] A. See, P. J. Liu, and C. D. Manning, “Get to the point: Summarization with pointer-generator networks,” in Proc. 55th Annu. Meeting Assoc.Comput. Linguistics (Long Papers), vol. 1, 2017, pp. 1073-1083.

[11] R. Nallapati, B. Zhou, C. dos Santos, C. Gulcehre, and B. Xiang, “Abstractive text summarization using sequence-to-sequence RNNs and beyond,” in Proc. 20th SIGNLL Conf. Comput. Natural Lang. Learn.,2016, pp. 280-290.

[12] Y.-C. Chen and M. Bansal, “Fast abstractive summarization with reinforce-selected sentence rewriting,” in Proc. 56th Annu. Meeting Assoc. Comput. Linguistics (Long Papers), vol. 1, 2018, pp. 675-686.

[13] C. Lin, “ROUGE: A package for automatic evaluation of summaries,” in Proc. Jpn. Circulat. J., Conf., vol. 34, 2004, p. 8.

[14] S. Banerjee and A. Lavie, “METEOR: An automatic metric for MT evaluation with improved correlation with human judgments,” in Proc. 2ndWorkshop Stat. Machine Transl., 2007, pp. 65-72.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez,L. Kaiser, and I. Polosukhin, “Attention is all you need,” in Proc. Adv. Neural Inf. Process. Syst., NIPS, 2017, pp. 5998-6008.

[16] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol., 2019, pp. 4171-4186.

[17] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol., 2019, pp. 4171-4186.

[18] A. K. M. Masum, S. Abujar, M. A. I. Talukder, A. K. M. S. A. Rabby, and S. A. Hossain, “Abstractive method of text summarization with sequence to sequence RNNs,” in Proc. 10th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT), Jul. 2019, pp. 1-5.

[19] M. Sanad. (2019). A Comprehensive Guide to Build your own Language Model in Python. [Online].

[20] S. Ruder, “An overview of gradient descent optimization algorithms,” in DBLP Computer Science Bibliography, 2016.

[21] R. Khandelwal. (2019). Overview of Different Optimizers for Neural Networks. Retrieved From. [Online].

[22] K. Jing and J. Xu, “A survey on neural network language models,” 2019, arXiv:1906.03591. [Online].

[23] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, “A neural probabilistic language model,” J. Mach. Learn. Res., vol. 3, pp. 1137-1155, Feb. 2003

[24] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” in Proc. 1st Int. Conf. Learn. Represent. (ICLR), 2013, pp. 1-12.

[25] S. Chopra, M. Auli, and A. M. Rush, “Abstractive sentence summarization with attentive recurrent neural networks,” in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol., 2016, pp. 93-98.

[26] Scikit-learn.org. 2022. scikit-learn: machine learning in Python — scikit-learn 1.1.1 documentation. [online] Available at: <https://scikit-learn.org/stable/index.html> [Accessed 11 May 2022].

[27] Z. Huang, W. Xu, and K. Yu, “Bidirectional lstm-crf models for sequence tagging,” arXiv e-prints, 2015.

[28] Q. Chen, X. Zhu, Z. Ling, S.Wei, and H. Jiang, “Distraction-based neural networks for document summarization,” in Proc. 25th Int. Joint Conf. Artif. Intell., 2016. [Online].

[29] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, “Pre-trained models for natural language processing: A survey,” Sci. China Technol. Sci., vol. 63, no. 10, pp. 1872-1897, Oct. 2020, doi: 10.1007/s11431-020-1647-3.