National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Usama Hanif
Student ID: 22108831

School of Computing
National College of Ireland

Supervisor: Furqan Rustam

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……Usama Hanif……

Student ID: ……x22108831……..……..

Programme: ……Data Analytics…… **Year:** ……2023……..

Module: ……Research Project……,,,,,,,……..……

Lecturer: ……Furqan Rustam……..……

Submission Due Date: ……18/09/2023……..……

Project Title: Research Paper Summarization Using Text-To-Text Transfer Transformer (T5) Model

Word Count: ……1013…… **Page Count:** ……..15……..……..……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……**USAMA HANIF**……..……

Date: ……18/09/2023……..……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Usama Hanif
Student ID: 22108831

1 Introduction

This manual illustrates how to execute and configure the implementation code for the current research project. This document provides specified details about the machine hardware as well as the programs to run. Following the below steps will enable the users to generate summaries of the research papers using the T5, BERT, GPT2, and BART models.

2 System Specification

2.1 Hardware Specification

Following are the hardware specifications of the system that was used to develop the project:

Processor: Intel Core i7 – 9750H
RAM: 16GB
Storage: 500GB
Graphics Card: GTX 1650 4GB
Operating System: Windows 11

2.2 Software Specification

The Google Colab Pro a web-based platform was used to train and evaluate the models and its specification was the following:

Processor: Intel Xeon
Graphics Card: A100 40GB
RAM: 80GB
Storage: 160GB

3 Software Tools

Following are the software tools that were used to implement the project:

3.1 Python

Python programming language was used to develop the project. The main reason to choose Python was its useful libraries for visualization, dataset preparation, and deep learning models. Python was downloaded from the main website[1]. Figure 1 shows the download page of Python's official website.

[1] https://www.python.org/downloads/

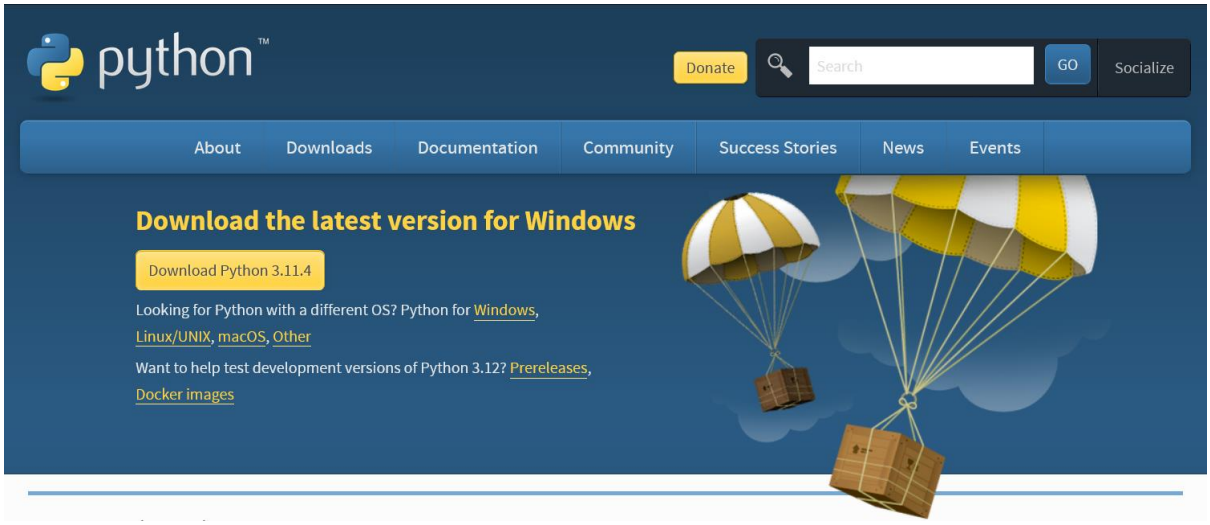

Figure 1: Download page of Python's official website

3.2 Jupyter Notebook

Jupyter Notebook was used as a compiler to run the code as it allows the users to implement all the code in one place and execute the codes in small parts like cells to allow the audience to check the output of each code with ease. Jupyter Notebook was downloaded from its official website[2] and Figure 2 illustrates its download page.

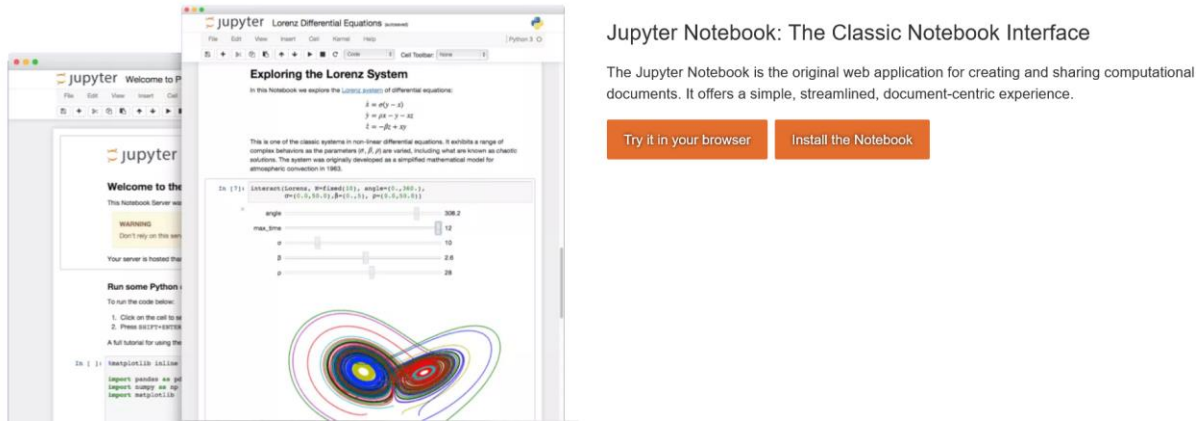

Figure 2: Download page of Jupyter Notebook's official website

4 Project Implementation

Following are the Python packages which were installed using pip and used to implement the project:

- NLTK
- Pandas
- Numpy
- Matplotlib
- Keras
- Tensorflow
- Transformers

[2] https://jupyter.org/

- Datasets
- Rouge-Score
- Huggingface-Hub

```
#installing necessary packages
!pip install transformers==4.20.0
!pip install keras_nlp==0.3.0
!pip install datasets
!pip install huggingface-hub
!pip install nltk
!pip install rouge-score
#Loading necessary packages
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import os
import logging
import nltk
from tensorflow import keras
import keras_nlp
from transformers.keras_callbacks import KerasMetricCallback
from google.colab import drive
from transformers import TFAutoModelForSeq2SeqLM
import matplotlib.pyplot as plt
from tensorflow import keras
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

Figure 3: Necessary Libraries and Packages

Pandas library was used to load and check the dataset as can be seen in Figure 4:

```
data = pd.read_csv('scisumm.csv')
```

```
data
```

	text	summary
0	TnT - A Statistical Part-Of-Speech Tagger Trig...	TnT - A Statistical Part-Of-Speech Tagger\nTri...
1	Mildly Non-Projective Dependency Structures Sy...	Mildly Non-Projective Dependency Structures\nS...
2	Using Corpus Statistics And WordNet Relations ...	Using Corpus Statistics And WordNet Relations ...
3	Automatic Labeling Of Semantic Roles present a...	Automatic Labeling Of Semantic Roles\nWe prese...
4	Generative Models For Statistical Parsing With...	Generative Models For Statistical Parsing With...
...	...	...
1004	Combining Lexical Syntactic And Semantic Featu...	Combining Lexical Syntactic And Semantic Featu...
1005	Similarity of Semantic Relations are at least ...	Similarity of Semantic Relations\nThere are at...
1006	Further Meta-Evaluation of Machine Translation...	Further Meta-Evaluation of Machine Translation...
1007	Soft Syntactic Constraints for Hierarchical Ph...	Soft Syntactic Constraints for Hierarchical Ph...
1008	A Unification Method For Disjunctive Feature D...	A Unification Method For Disjunctive Feature D...

1009 rows × 2 columns

Figure 4: Loading and Checking the Dataset

The dataset was preprocessed by converting the text into lowercase characters which can be seen in the following Figure 5:

```
def preprocess_text(text):
    # Tokenize the text into sentences and words
    sentences = sent_tokenize(text)
    words = [word.lower() for sent in sentences for word in word_tokenize(sent)]

    # Joining the processed words back into a sentence
    processed_text = ' '.join(words)

    return processed_text



# Applying the preprocessing on the 'text' column
data['preprocessed_text'] = data['text'].apply(preprocess_text)

# Display the preprocessed DataFrame
print(data.head())
```

```python
def preprocess_summary(summary):
    # Tokenize the summary into sentences and words
    sentences = sent_tokenize(summary)
    words = [word.lower() for sent in sentences for word in word_tokenize(sent)]


    # Joinin the processed words back into a sentence
    processed_summary = ' '.join(words)

    return processed_summary



# Applyin the preprocessing on the 'summary' column
data['preprocessed_summary'] = data['summary'].apply(preprocess_summary)

# Display the preprocessed DataFrame
print(data.head())
```

	text	summary	preprocessed_text	preprocessed_summary
0	TnT - A Statistical Part-Of-Speech Tagger Trig...	TnT - A Statistical Part-Of-Speech Tagger\nTri...	tnt - a statistical part-of-speech tagger trig...	tnt - a statistical part-of-speech tagger trig...
1	Mildly Non-Projective Dependency Structures Sy...	Mildly Non-Projective Dependency Structures\nS...	mildly non-projective dependency structures sy...	mildly non-projective dependency structures sy...
2	Using Corpus Statistics And WordNet Relations ...	Using Corpus Statistics And WordNet Relations ...	using corpus statistics and wordnet relations ...	using corpus statistics and wordnet relations ...
3	Automatic Labeling Of Semantic Roles present a...	Automatic Labeling Of Semantic Roles\nWe prese...	automatic labeling of semantic roles present a...	automatic labeling of semantic roles we presen...
4	Generative Models For Statistical Parsing With...	Generative Models For Statistical Parsing With...	generative models for statistical parsing with...	generative models for statistical parsing with...

Figure 5: Preprocessing of the Dataset

After applying preprocessing, the preprocessed columns were saved while others were dropped and the dataset was saved in a CSV format which can be seen in the following Figure 6:

```python
columns_to_drop = ['text', 'summary']
data.drop(columns_to_drop, axis=1, inplace=True)
```

```python
# Saving the Preprocessed dataset
data.to_csv('preprocessed_scisumm.csv', sep=',', index=False, encoding='utf-8')
```

Figure 6: Dropping the columns and saving the preprocessed dataset

The preprocessed dataset was uploaded to the Huggingface website and later loaded into the Google Colab Platform which can be seen in the following Figure 7:

```python
from datasets import load_dataset
#Loading preprocessed dataset from the HuggingFace repository
dataset = load_dataset("usamahanif719/scisumm", split="train")
```

Figure 7: Loading Preprocessed dataset from Huggingface

4.1 Implementation of the T5 Model

Some important variables like learning rate, number of epochs, and model version were declared in Figure 8.

```
#Declaring some important values for model building
MAX_TARGET_LENGTH = 150  # Maximum length of the summary
BATCH_SIZE = 4  # Batch size for training
LEARNING_RATE = 1e-5  # Learning rate for training
EPOCHS = 10  # Maximum number of epochs the model will train

#version of the model that will be used
MODEL = "t5-base"
```

Figure 8: Declaring Important Values

In Figure 9 the google drive was mounted and some folders were created to save checkpoints and models:

```
# Mounting the Google Drive to save model and checkpoints
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#Creating the checkpoints folder in the Google Drive
CHECKPOINT_DIR = '/content/drive/MyDrive/T5_Model_Checkpoints'
```

```
#Checking if the checkpoint folder exist and if not then creating a new one
if not os.path.exists(CHECKPOINT_DIR):
    os.makedirs(CHECKPOINT_DIR)
```

```
#Creating the model folder in the Google Drive to save the models
SAVED_MODEL_DIR = '/content/drive/MyDrive/T5_Trained_Model'
```

```
#Checking if the model folder exist and if not then creating a new one
if not os.path.exists(SAVED_MODEL_DIR):
    os.makedirs(SAVED_MODEL_DIR)
```

Figure 9: Mounting Google Drive and Creating folders

Figure 10 shows the split of the dataset into train and test sets.

```
#Splitting the dataset into train and test sets
dataset = dataset.train_test_split(
    train_size=0.8, test_size=0.2
)
```

Figure 10: Splitting Dataset

In Figure 11 a suitable tokenizer was downloaded for the T5 model

```
from transformers import AutoTokenizer
# Downloading the suitable tokenizer for the base T5 model
tokenizer = AutoTokenizer.from_pretrained(MODEL)
```

Figure 11: Downloading Tokenizer

In Figure 12 the prefix was declared to download the summarization capabilities of the T5 model:

```
# Creating a prefix to download the T5's summarization capabilites
if MODEL in ["t5-small", "t5-base", "t5-large", "t5-3b", "t5-11b"]:
    prefix = "summarize: "
else:
    prefix = ""
```

Figure 12: Declaring the prefix for the model

In Figure 13, a function was created to tokenize the dataset columns, and a map function was used to apply it.

```
# Creating a function to tokenize the dataset
def preprocess_function(examples):
    inputs = [prefix + doc for doc in examples["preprocessed_text"]]
    model_inputs = tokenizer(inputs, truncation=True, padding=True)

    # Setup the tokenizer for targets
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(
            examples["preprocessed_summary"], max_length= MAX_TARGET_LENGTH, truncation=True, padding=True,
        )

    model_inputs["labels"] = labels["input_ids"]

    return model_inputs
```

```
# Applying the tokenization function to dataset using map function
tokenized_datasets = dataset.map(preprocess_function, batched=True)
```

Figure 13: Applying Tokenization

In Figure 14, seq2seq class was downloaded to cache the data:

```
from transformers import TFAutoModelForSeq2SeqLM
# Downloading the seq2seq class to cashe the data
model = TFAutoModelForSeq2SeqLM.from_pretrained(MODEL)
```

Figure 14: Cache the dataset

In Figure 15, a seq2seq data collator was downloaded and applied to the dataset

```
from transformers import DataCollatorForSeq2Seq
# Downloading the datacollator of seq2seq class to pad and embedding the data
data_collator = DataCollatorForSeq2Seq(tokenizer, model=model, return_tensors="tf")
```

```
# Applying the datacollator to both the train and test set
train_dataset = tokenized_datasets["train"].to_tf_dataset(
    batch_size=BATCH_SIZE,
    columns=["input_ids", "attention_mask", "labels"],
    shuffle=True,
    collate_fn=data_collator,
)
test_dataset = tokenized_datasets["test"].to_tf_dataset(
    batch_size=BATCH_SIZE,
    columns=["input_ids", "attention_mask", "labels"],
    shuffle=False,
    collate_fn=data_collator,
)
generation_dataset = (
    tokenized_datasets["test"]
    .shuffle()
    .select(list(range(200)))
    .to_tf_dataset(
        batch_size=BATCH_SIZE,
        columns=["input_ids", "attention_mask", "labels"],
        shuffle=False,
        collate_fn=data_collator,
    )
)
```

Figure 15: Applying the data collator to the dataset

In Figure 16, the T5 model was compiled and trained for 10 epochs:

```
# Optimizng and compiling the data
optimizer = keras.optimizers.Adam(learning_rate=LEARNING_RATE)
model.compile(optimizer=optimizer)

# Training the model on train set
model.fit(
    train_dataset, validation_data=test_dataset, epochs=EPOCHS
)
```

```
Epoch 1/10
202/202 [==============================] - 233s 806ms/step - loss: 1.7974 - val_loss: 0.7629
Epoch 2/10
202/202 [==============================] - 156s 773ms/step - loss: 0.7863 - val_loss: 0.6890
Epoch 3/10
202/202 [==============================] - 155s 770ms/step - loss: 0.7233 - val_loss: 0.6595
Epoch 4/10
202/202 [==============================] - 156s 771ms/step - loss: 0.6789 - val_loss: 0.6363
Epoch 5/10
202/202 [==============================] - 155s 770ms/step - loss: 0.6472 - val_loss: 0.6193
Epoch 6/10
202/202 [==============================] - 155s 770ms/step - loss: 0.6202 - val_loss: 0.6053
Epoch 7/10
202/202 [==============================] - 156s 771ms/step - loss: 0.6006 - val_loss: 0.5903
Epoch 8/10
202/202 [==============================] - 156s 771ms/step - loss: 0.5847 - val_loss: 0.5817
Epoch 9/10
202/202 [==============================] - 155s 770ms/step - loss: 0.5687 - val_loss: 0.5731
Epoch 10/10
202/202 [==============================] - 156s 774ms/step - loss: 0.5538 - val_loss: 0.5736
```

Figure 16: T5 model training

Figure 17 shows the loss graph of the trained T5 model:

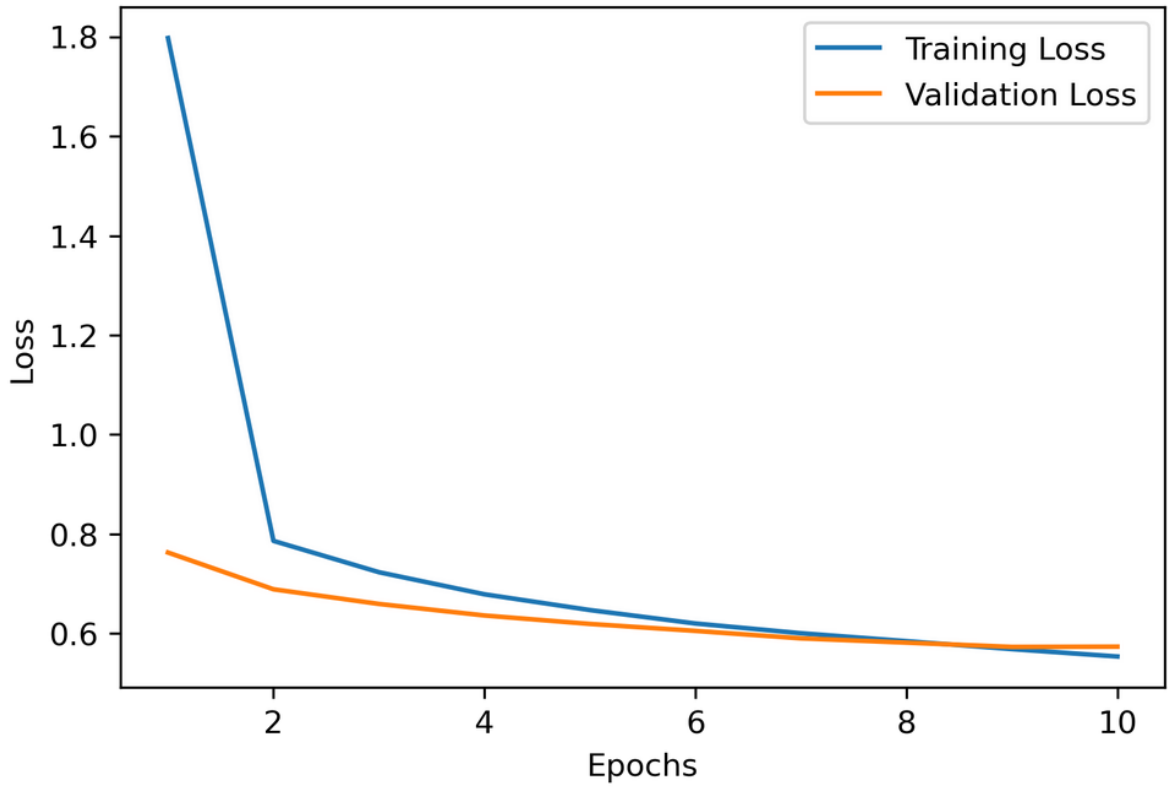

Figure 17: Loss graph of the trained T5 model

Figure 18 shows the code for generating the summaries using the trained T5 model:

```
from transformers import pipeline

summarizer = pipeline("summarization", model=model, tokenizer=tokenizer, framework="tf")

summarizer(
    raw_datasets["test"][2]["preprocessed_text"],
    max_length=MAX_TARGET_LENGTH,
)
```

```
[{'summary_text': 'introducing cross-lingual word clusters for direct transfer of linguistic structure it has been establi
shed that incorporating word cluster features derived from large unlabeled corpora can significantly improve prediction of
linguistic structure . while previous work has focused primarily on english , we extend these results to other languages ,
along two dimensions : first , they hold true for a number of languages . second , and more interestingly , a system for i
nducing crosslingual clusters and we show that by augmenting direct-transfer systems , the relative error of delexicalized
dependency recognizers , can be reduced by up to 26 % .'}]
```

Figure 18: Generating Summary

Figure 19 shows the evaluation of the T5 model using ROUGE scores:

```
from nltk.translate.bleu_score import sentence_bleu
from rouge_score import rouge_scorer
# Generating the ROUGE scores of the generated summary
scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)
reference_summary = dataset["test"][0]["preprocessed_summary"]  # The reference summary for evaluation
generated_summary = summarizer(dataset["test"][0]["preprocessed_text"], max_length=MAX_TARGET_LENGTH, truncation=True,
                               return_tensors="pt")[0]
```

```
generated_summary_token_ids = generated_summary['summary_token_ids']
generated_summary_text = tokenizer.decode(generated_summary_token_ids, skip_special_tokens=True)

scores = scorer.score(generated_summary_text, reference_summary)
print(scores)
```

```
{'rouge1': Score(precision=0.7130434782608696, recall=0.9879518072289156, fmeasure=0.8282828282828283), 'rouge2': Score(pr
ecision=0.7017543859649122, recall=0.975609756097561, fmeasure=0.8163265306122448), 'rougeL': Score(precision=0.7130434782
608696, recall=0.9879518072289156, fmeasure=0.8282828282828283)}
```

Figure 19: ROUGE scores of the T5 model

Figure 20 shows the evaluation of the T5 model using the BLEU score

```
from nltk.translate.bleu_score import sentence_bleu

reference = [reference_summary.split()]
candidate = generated_summary_text.split()

# Calculating the BLEU score
bleu_score = sentence_bleu(reference, candidate)
print(bleu_score)
```

```
0.47019530867801906
```

Figure 20: BLEU score of the T5 model

In Figure 21 the trained model is saved into Google Drive.

```
# Saving the Trained model to the Google Drive
model.save(SAVED_MODEL_DIR)
```

Figure 21: Saving the model

4.2 Implementation of the BERT Model

The BERT model's tokenization was implemented as same as was done for T5. Figure 22 shows the variables declared for the BERT model,

```
#Declaring some important values for model building
BATCH_SIZE = 8  # Batch size for training the model
LEARNING_RATE = 2e-5  # learning rate for training the model
EPOCHS = 50  # Maximum number of epochs

#version of the model that will be used
MODEL_CHECKPOINT = "bert-base-cased"
```

Figure 22: Values for the BERT model

Figure 23 shows that the BERT model was trained for 50 epochs.

```
Epoch 39/50
101/101 [==============================] - 129s 1s/step - loss: 4.1581 - val_loss: 3.8918
Epoch 40/50
101/101 [==============================] - 128s 1s/step - loss: 4.1501 - val_loss: 3.8773
Epoch 41/50
101/101 [==============================] - 129s 1s/step - loss: 4.1433 - val_loss: 3.9414
Epoch 42/50
101/101 [==============================] - 129s 1s/step - loss: 4.1467 - val_loss: 3.8978
Epoch 43/50
101/101 [==============================] - 129s 1s/step - loss: 4.1267 - val_loss: 3.9841
Epoch 44/50
101/101 [==============================] - 128s 1s/step - loss: 4.1298 - val_loss: 3.9413
Epoch 45/50
101/101 [==============================] - 129s 1s/step - loss: 4.1429 - val_loss: 3.9716
Epoch 46/50
101/101 [==============================] - 129s 1s/step - loss: 4.0946 - val_loss: 3.9017
Epoch 47/50
101/101 [==============================] - 129s 1s/step - loss: 4.1241 - val_loss: 3.9064
Epoch 48/50
101/101 [==============================] - 129s 1s/step - loss: 4.1130 - val_loss: 3.8991
Epoch 49/50
101/101 [==============================] - 129s 1s/step - loss: 4.1034 - val_loss: 3.8612
Epoch 50/50
101/101 [==============================] - 129s 1s/step - loss: 4.1059 - val_loss: 3.9722
```

Figure 23: Training of the BERT model

Figure 24 shows the loss graph of the BERT Model.

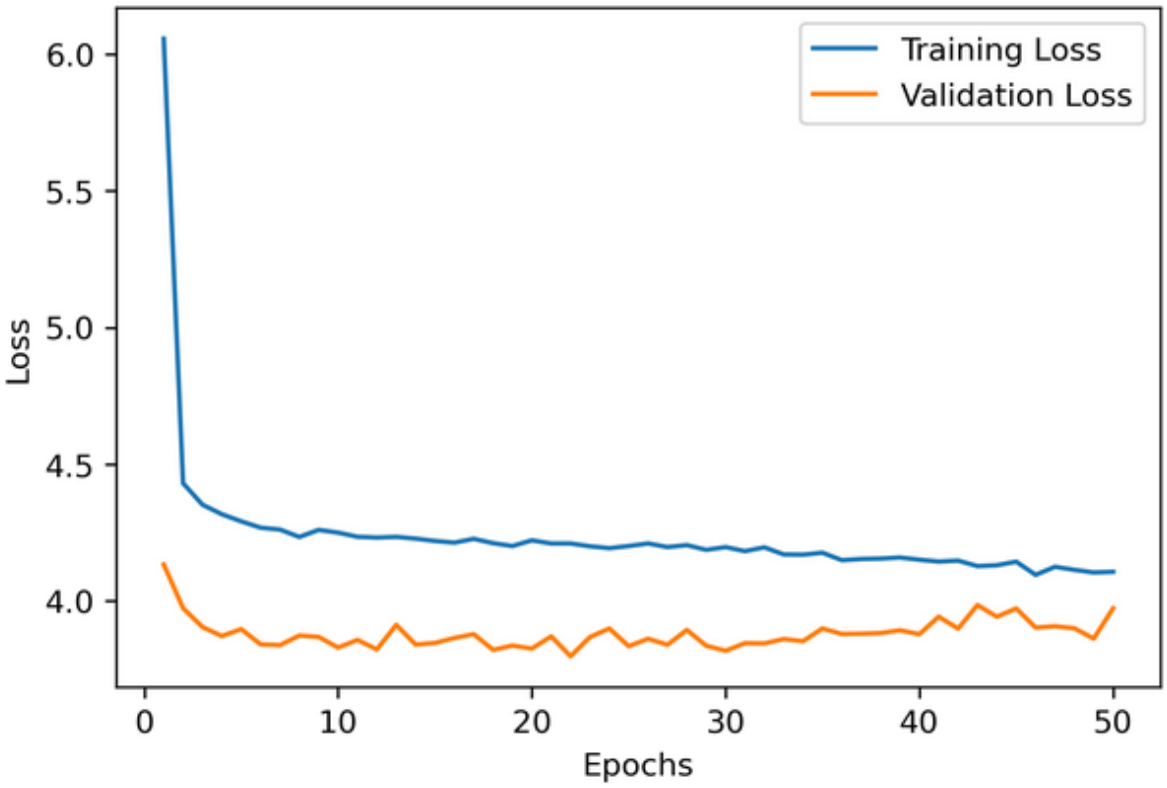

Figure 24: Loss Graph of the BERT model

Figure 25 shows the ROUGE scores of the trained BERT model.

```
generated_summary_token_ids = generated_summary['summary_token_ids']
generated_summary_text = tokenizer.decode(generated_summary_token_ids, skip_special_tokens=True)

scores = scorer.score(generated_summary_text, reference_summary)
print(scores)
```

```
{'rouge1': Score(precision=0.7535545023696683, recall=0.4162303664921466, fmeasure=0.5362563237774031), 'rouge2': Score(pr
ecision=0.5238095238095238, recall=0.2887139107611549, fmeasure=0.3722504230118444), 'rougeL': Score(precision=0.606635071
0900474, recall=0.33507853403141363, fmeasure=0.43170320404721757)}
```

Figure 25: ROUGE scores of the BERT model

Figure 26 shows the BLEU score of the Trained BERT model.

```
from nltk.translate.bleu_score import sentence_bleu

reference = [reference_summary.split()]
candidate = generated_summary_text.split()

# Calculating the BLEU score
bleu_score = sentence_bleu(reference, candidate)
print(bleu_score)
```

```
0.2134688280180426
```

Figure 26: BLEU score of the BERT model

4.3 Implementation of GPT2 Model

In Figure 27 the pre-trained gpt2 model was declared

```
#Declaring some important values
MAX_TARGET_LENGTH = 200  # Maximum length of the output by the model
# version of the model that will be used
MODEL_CHECKPOINT = "gpt2"
```

Figure 27: Declaring GPT2 model

Figure 28 shows that the GPT2 tokenizer was downloaded and Figure 29 shows that a function was created to tokenize the dataset and a map function was used to apply the tokenization.

```
from transformers import GPT2Tokenizer
# Downloading the GPT2 tokenizer for the model
tokenizer = GPT2Tokenizer.from_pretrained(MODEL_CHECKPOINT)
# Add a new padding token to the tokenizer
tokenizer.add_special_tokens({'pad_token': '[PAD]'})
```

Figure 28: Downloading GPT2 tokenizer

```
# Creating a function to tokenize the dataset
def preprocess_function(examples):
    inputs = [doc for doc in examples["preprocessed_text"]]
    model_inputs = tokenizer(inputs, padding="max_length", truncation=True, return_tensors="tf")
    print("Input IDs shape:", model_inputs["input_ids"].shape)
    print("Attention Mask shape:", model_inputs["attention_mask"].shape)

    # Calculating the maximum target length dynamically based on the batch
    max_target_length = max(len(summary) for summary in examples["preprocessed_summary"])
    max_target_length = min(max_target_length, MAX_TARGET_LENGTH)
    labels = tokenizer(
        examples["preprocessed_summary"],
        padding="max_length",
        max_length=max_target_length,
        truncation=True,
        return_tensors="tf",
    )["input_ids"]

    model_inputs["labels"] = labels

    return model_inputs
```

```
# Applying the tokenization function to dataset using map function
tokenized_datasets = dataset.map(preprocess_function, batched=True)
```

Figure 29: Applying the tokenization to the dataset

The data collatoring was applied as same as was done in the T5 model. Figure 30 shows the ROUGE scores of the GPT2 model.

```
generated_summary_token_ids = generated_summary['summary_token_ids']
generated_summary_text = tokenizer.decode(generated_summary_token_ids, skip_special_tokens=True)

scores = scorer.score(generated_summary_text, reference_summary)
print(scores)
```

```
{'rouge1': Score(precision=0.9132231404958677, recall=0.2822477650063857, fmeasure=0.43121951219512195), 'rouge2': Score(p
recision=0.8298755186721992, recall=0.2557544757033248, fmeasure=0.3910068426197458 3), 'rougeL': Score(precision=0.8595041
32231405, recall=0.2656449553001277, fmeasure=0.4058536585365854)}
```

Figure 30: ROUGE scores of the GPT2 Model

Figure 31 shows the BLEU score of the GPT2 model.

```
from nltk.translate.bleu_score import sentence_bleu

reference = [reference_summary.split()]
candidate = generated_summary_text.split()

# Calculating the BLEU score
bleu_score = sentence_bleu(reference, candidate)
print(bleu_score)
```

```
0.19936258437373938
```

Figure 31: BLEU score of the GPT2 Model

4.4 Implementation of the BART Model

Figure 32 shows the implementation of the pre-trained BART model.

```
# version of the model declared
MODEL_CHECKPOINT = 'facebook/bart-base'

from transformers import BartTokenizer, TFBartForConditionalGeneration

# Importing and initializing the BART tokenizer
tokenizer = BartTokenizer.from_pretrained(MODEL_CHECKPOINT)
# Add a new padding token to the tokenizer
tokenizer.add_special_tokens({'pad_token': '[PAD]'})

# applying the preprocess_function for BART tokenizer and dynamic padding
def preprocess_function(examples):
    inputs = [doc for doc in examples["preprocessed_text"]]
    model_inputs = tokenizer(inputs, padding="max_length", truncation=True, return_tensors="tf")
    print("Input IDs shape:", model_inputs["input_ids"].shape)
    print("Attention Mask shape:", model_inputs["attention_mask"].shape)

    # Calculating the maximum target length dynamically based on the batch
    max_target_length = max(len(summary) for summary in examples["preprocessed_summary"])
    max_target_length = min(max_target_length, MAX_TARGET_LENGTH)
    labels = tokenizer(
        examples["preprocessed_summary"],
        padding="max_length",
        max_length=max_target_length,
        truncation=True,
        return_tensors="tf",
    )["input_ids"]

    model_inputs["labels"] = labels

    return model_inputs
tokenized_datasets = dataset.map(preprocess_function, batched=True)
# Initializing the BART model
model = TFBartForConditionalGeneration.from_pretrained(MODEL_CHECKPOINT)
```

Figure 32: Implementation of the BART model

The data collatoring was implemented as same as done for the T5 model. Figure 33 shows the ROUGE scores of the BART model.

```
generated_summary_token_ids = generated_summary['summary_token_ids']
generated_summary_text = tokenizer.decode(generated_summary_token_ids, skip_special_tokens=True)

scores = scorer.score(generated_summary_text, reference_summary)
print(scores)
```

```
{'rouge1': Score(precision=0.7469879518072289, recall=0.6078431372549019, fmeasure=0.6702702702702701), 'rouge2': Score(pr
ecision=0.6219512195121951, recall=0.504950495049505, fmeasure=0.5573770491803278), 'rougeL': Score(precision=0.6867469879
518072, recall=0.5588235294117647, fmeasure=0.6162162162162163)}
```

Figure 33: ROUGE scores of the BART model

Figure 34 shows the BLEU score of the BART model.

```
from nltk.translate.bleu_score import sentence_bleu

reference = [reference_summary.split()]
candidate = generated_summary_text.split()

# Calculating BLEU score
bleu_score = sentence_bleu(reference, candidate)
print(bleu_score)
```

```
0.3852117445736123
```

Figure 34: BLEU score of the BART model