Abstractive Text Summarization

In the modern Internet age, textual data is ever increasing. We need some way to condense this data while preserving the information and its meaning. We need to summarise textual data for that. Text summarisation is the process of automatically generating natural language summaries from an input document while retaining the important points. It helps in the easy and fast retrieval of information.

Abstractive summarisation is a type of summarization which includes generating new phrases, possibly rephrasing or using words that were not in the original text.

Objective

The primary objective of this experiment is to deploy advanced NLP techniques to generate grammatically correct and insightful summaries for a given articles. To accomplish this, we will test various publicly available transformer models for seq-to-seq modelling and retrain them on the wikihow dataset.

Dataset

Dataset Repo - *News Summary*

Number of sentences	Independent Feature	Target Data Column
4515	ctext	text

The dataset consists of 4515 examples and contains Author_name, Headlines, Url of Article, Short text, Complete Article. I gathered the summarized news from Inshorts and only scraped the news articles from Hindu, Indian times and Guardian. Time period ranges from febrauary to august 2017.

Apporach:

Naturally abstractive approaches are harder. For a perfect abstractive summary, the model has to first truly understand the document and then try to express that understanding in shortform, possibly using new words and phrases.

This is much harder than an extractive summary, requiring complex capabilities like generalisation, paraphrasing and incorporating real-world knowledge. So, to achieve best in class outcomes for the above mentioned task, we are going to fine-tune transformer models.

Transformer is an architecture for transforming one sequence into another one with the help of two parts (Encoder and Decoder), but it differs from the traditional sequence-to-sequence models because it does not imply any Recurrent Networks (GRU, LSTM, etc.).

To understand more about transformers click here.

Choice of Transformers

We are going to fine-tune two transformers and select the best performing model for summarization.

> **Google's T5(Text-To-Text Transfer Transformer):** T5 is an encoder-decoder model pre-trained on a multi-task mixture of unsupervised and supervised tasks and for which each task is converted into a text-to-text format. T5 works well on a variety of tasks out-of-the-box by prepending a different prefix to the input corresponding to each task, e.g.: For translation: Translate English to German.
>
> **Facebook's Bart:** BART is a denoising autoencoder that maps a corrupted document to the original document it was derived from. It is implemented as a sequence-to-sequence model with a bidirectional encoder over corrupted text and a left-to-right autoregressive decoder.

Performance Evaluation Metrics

In the scope of this experiment, we have adopted only content-based methods to measure the performance of summarizer. One of the metric is rouge score.

Rouge Score: ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. It is essentially a set of metrics for evaluating automatic summarisation of texts as well as machine translation. It works by comparing an automatically produced summary or translation against a set of reference summaries (typically human-produced).

It is further divided into different measures depending upon the granularity. More details can be found here

> **Rouge-1** - It refers to overlap of unigrams between the system summary and reference summary.
> **Rouge-2** - It refers to the overlap of bigrams between the system and reference summaries.
> **Rouge-L** – It measures longest matching sequence of words using LCS.

Technologies

- **Programming language**

- **Libraries**

+

Models Performance

Model	Rouge-1	Rouge-2	Rouge-l
T5	0.453173	0.232103	0.418052
Bart	0.431542	0.208504	0.398183

Table of Contents

1. Environment Setup

goto toc

1.1. Install Packages

Install required packages

goto toc

In [1]:
```
!pip install --quiet transformers==4.5.0
!pip install --quiet pytorch-lightning==1.2.7
!pip install --quiet rouge
```

```
     |████████████████████████████████| 2.1 MB 4.2 MB/s
     |████████████████████████████████| 895 kB 39.5 MB/s
     |████████████████████████████████| 3.3 MB 35.1 MB/s
     |████████████████████████████████| 830 kB 4.2 MB/s
     |████████████████████████████████| 269 kB 37.2 MB/s
     |████████████████████████████████| 829 kB 32.3 MB/s
     |████████████████████████████████| 118 kB 47.0 MB/s
     |████████████████████████████████| 234 kB 39.7 MB/s
     |████████████████████████████████| 1.3 MB 44.6 MB/s
     |████████████████████████████████| 142 kB 48.5 MB/s
     |████████████████████████████████| 294 kB 35.5 MB/s
  Building wheel for future (setup.py) ... done
  Building wheel for PyYAML (setup.py) ... done
```

1.2. Load Dependencies

Import required packages

goto toc

In [2]:

```
import os
import torch
import numpy as np
import pandas as pd
import seaborn as sns
from rouge import Rouge
from pathlib import Path
from tqdm.auto import tqdm
from termcolor import colored
import pytorch_lightning as pl
from transformers import AdamW
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from pytorch_lightning.callbacks import ModelCheckpoint
from pytorch_lightning.loggers import TensorBoardLogger
from transformers import T5ForConditionalGeneration, T5TokenizerFast
from transformers import BartForConditionalGeneration, BartTokenizerFast

# Display plot inline
%matplotlib inline
sns.set()

# Set pytorch-lightning seed
pl.seed_everything(42)

if torch.cuda.is_available():
  print('__CUDNN VERSION:', torch.backends.cudnn.version())
  print('__Number CUDA Devices:', torch.cuda.device_count())
  print('Active CUDA Device: GPU', torch.cuda.current_device())
  print('Available devices ', torch.cuda.device_count())
  print('Current cuda device ', torch.cuda.current_device())
```

```
Global seed set to 42
__CUDNN VERSION: 7605
__Number CUDA Devices: 1
Active CUDA Device: GPU 0
Available devices  1
Current cuda device  0
```

In [3]:

```
# Download dataset from google drive
!gdown --id 17BuuCy6UMTUDMzNSrTSnBvHp_IRoIzj1
```

```
Downloading...
From: https://drive.google.com/uc?id=17BuuCy6UMTUDMzNSrTSnBvHp_IRoIzj1
To: /content/news_summary.csv
11.9MB [00:00, 72.8MB/s]
```

In [3]:

```
## Create a config class

class config:
  num_workers = os.cpu_count()
  n_epochs = 3
  batch_size = 8

  text_token_max_length = 512
  summary_token_max_length = 128

  learning_rate = 0.0001
```

2. Load dataset

Read data from news_summary.csv file using pandas method read_csv().

goto toc

In [5]:

```
# Read dataset using pandas
raw_data = pd.read_csv('./news_summary.csv', encoding='latin-1')
raw_data.head()
```

Out[5]:

	author	date	headlines	read_more
0	Chhavi Tyagi	03 Aug 2017,Thursday	Daman & Diu revokes mandatory Rakshabandhan in...	http://www.hindustantimes.com/india-news/raksh...
1	Daisy Mowke	03 Aug 2017,Thursday	Malaika slams user who trolled her for 'divorc...	http://www.hindustantimes.com/bollywood/malaik...
2	Arshiya Chopra	03 Aug 2017,Thursday	'Virgin' now corrected to 'Unmarried' in IGIMS...	http://www.hindustantimes.com/patna/bihar-igim...
3	Sumedha Sehra	03 Aug 2017,Thursday	Aaj aapne pakad liya: LeT man Dujana before be...	http://indiatoday.intoday.in/story/abu-dujana-...
4	Aarushi Maheshwari	03 Aug 2017,Thursday	Hotel staff to get training to spot signs of s...	http://indiatoday.intoday.in/story/sex-traffic...

Note: We only need *ctext* and *text* so will drop other features.

3. Data Preprocessing

...goto toc

3.1. Data Cleaning

...goto toc

In [6]:

```
# Select required columns
data = raw_data[['ctext', 'text']]

# Rename columns
data.columns = ['text', 'summary']
```

In [7]:

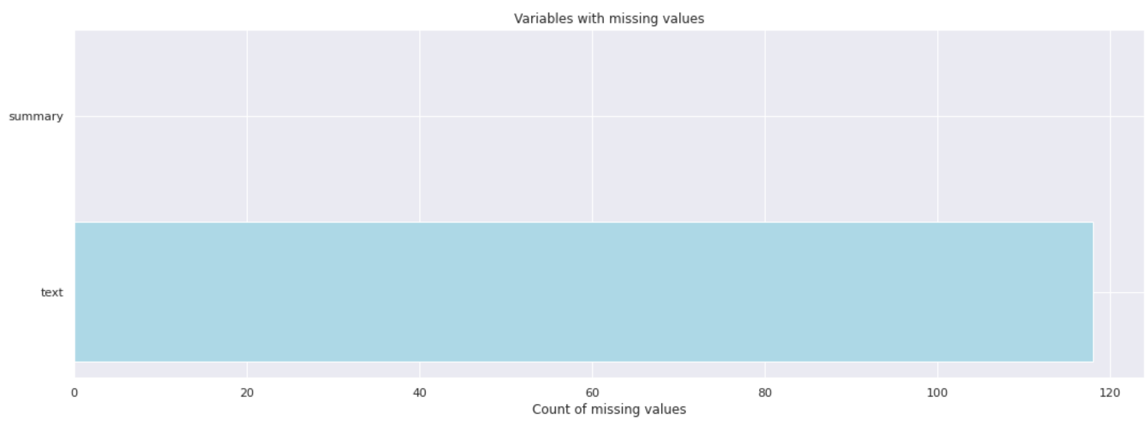
```
# Function to get missing values
def get_missing(data):

    # Create the dataframe
    missing_values = pd.DataFrame()

    # Get list of all columns
    missing_values['Features'] = data.columns.values

    # get the count of missing values
    missing_values['Count'] = data.isnull().sum().values

    # Calculate percentage of missing values
    percentage = data.isna().mean()*100
    missing_values['Percentange'] = percentage.values

    # return the dataframe
    return missing_values
```

In [8]:

```
# Function to plot missing values
def plot_missing(missing_values):
    # Plot missing values

    # Get list of features
    columns = missing_values.Features.values.tolist()

    # Get index's
    ind = missing_values.index.to_list()

    # Create subplots
    fig, ax = plt.subplots(1,figsize=(18, 6))

    # Plot missing values based on count
    rects = ax.barh(ind, missing_values.Count.values.tolist(), color='lightblue
    ax.set_yticks(ind)
    ax.set_yticklabels(columns, rotation='horizontal')
    ax.set_xlabel("Count of missing values")
    ax.set_title("Variables with missing values")
```

In [9]:

```
plot_missing(get_missing(data))
```



In [10]:

```
# Drop missing values
data = data.dropna()
data.head()
```

Out[10]:

	text	summary
0	The Daman and Diu administration on Wednesday ...	The Administration of Union Territory Daman an...
1	From her special numbers to TV?appearances, Bo...	Malaika Arora slammed an Instagram user who tr...
2	The Indira Gandhi Institute of Medical Science...	The Indira Gandhi Institute of Medical Science...
3	Lashkar-e-Taiba's Kashmir commander Abu Dujana...	Lashkar-e-Taiba's Kashmir commander Abu Dujana...
4	Hotels in Mumbai and other Indian cities are t...	Hotels in Maharashtra will train their staff t...

In [11]:

```
# Split dataset into train and validation set
train_df, val_df = train_test_split(data, test_size = 0.1)

print(train_df.shape)
print(val_df.shape)
```

```
(3956, 2)
(440, 2)
```

3.2. Build Dataset Class

...goto toc

In [12]:

```
class NewsSummaryDataset(Dataset):
    def __init__(self,
                 data : pd.DataFrame,
                 tokenizer,
                 text_max_token_len : int = config.text_token_max_length,
                 summary_max_token_len : int = config.summary_token_max_length)

      self.tokenizer = tokenizer
      self.data = data
      self.text_max_token_len = text_max_token_len
      self.summary_max_token_len = summary_max_token_len

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index : int):
        data_row = self.data.iloc[index]



        # Encode text
        text = data_row['text']
        text_encoding = self.tokenizer(text,
                                       max_length = self.text_max_token_len,
                                       padding = "max_length",
                                       truncation = True,
                                       return_attention_mask = True,
                                       add_special_tokens = True,
                                       return_tensors = "pt")



        # Encode summary
```

```
        summary = data_row['summary']
        summary_encoding = self.tokenizer(summary,
                                          max_length = self.summary_max_token_len,
                                          padding = "max_length",
                                          truncation = True,
                                          return_attention_mask = True,
                                          add_special_tokens = True,
                                          return_tensors = "pt")

        # Replace 0's with -100 to let the transformer understand
        labels = summary_encoding['input_ids']
        labels[labels == 0] = -100

        return dict(
            text = text,
            summary = summary,
            text_input_ids = text_encoding['input_ids'].flatten(),
            text_attention_mask = text_encoding['attention_mask'].flatten(),
            labels = labels.flatten(),
            labels_attention_mask = summary_encoding['attention_mask'].flatten(
        )
```

3.3. Build Lightining Data Module

...goto toc

In [13]:

```
class NewsSummaryDataModule(pl.LightningDataModule):
    def __init__(self,
                 train_df: pd.DataFrame,
                 val_df : pd.DataFrame,
                 tokenizer,
                 batch_size: int = config.batch_size,
                 text_max_token_len : int = config.text_token_max_length,
                 summary_max_token_len : int = config.summary_token_max_length
                 ):

        super().__init__()

        self.train_df = train_df
        self.test_df = val_df

        self.tokenizer = tokenizer
        self.batch_size = batch_size

        self.text_max_token_len = text_max_token_len
        self.summary_max_token_len = summary_max_token_len

    def setup(self, stage = None):

        # Build train dataset from custom dataset
        self.train_dataset = NewsSummaryDataset(
            self.train_df,
            self.tokenizer,
            self.text_max_token_len,
            self.summary_max_token_len
        )

        # Build validation dataset from custom dataset
        self.test_dataset = NewsSummaryDataset(
            self.test_df,
            self.tokenizer,
            self.text_max_token_len,
```

```
            self.summary_max_token_len
        )

    def train_dataloader(self):
        # Convert dataset to dataloader and return
        return DataLoader(self.train_dataset,
                          batch_size = self.batch_size,
                          shuffle = True,
                          num_workers = config.num_workers)

    def val_dataloader(self):
        # Convert dataset to dataloader and return
        return DataLoader(self.test_dataset,
                          batch_size = self.batch_size,
                          shuffle = False,
                          num_workers = config.num_workers)

    def test_dataloader(self):
        # Convert dataset to dataloader and return
        return DataLoader(self.test_dataset,
                          batch_size = self.batch_size,
                          shuffle = False,
                          num_workers = config.num_workers)
```

4. Model Development

...goto toc

In [4]:

```
# General class for summarization
class NewsSummaryModel(pl.LightningModule):
  def __init__(self, model):

    super().__init__()
    self.model = model# T5ForConditionalGeneration.from_pretrained(MODEL_NAME,

  def forward(self, input_ids, attention_mask, decoder_attention_mask, labels =
      output = self.model(
          input_ids,
          attention_mask = attention_mask,
          labels = labels,
          decoder_attention_mask = decoder_attention_mask
      )

      return output.loss, output.logits

  def checkType(self, input_ids, attention_mask, labels, labels_attention_mask,
    if type(input_ids) != torch.Tensor or attention_mask != torch.Tensor or lab
        raise ValueError(f"\n\ninput_ids : {input_ids} \n\nattention_mask:{atte

  def training_step(self, batch, batch_idx):
    input_ids = batch['text_input_ids']
    attention_mask = batch['text_attention_mask']
    labels = batch['labels']
    labels_attention_mask = batch['labels_attention_mask']


    loss, outputs = self(
      input_ids = input_ids,
      attention_mask = attention_mask,
      decoder_attention_mask = labels_attention_mask,
      labels = labels
```

```
    )

    self.log("train_loss", loss, prog_bar = True, logger = True)

    return loss

  def validation_step(self, batch, batch_idx):
    input_ids = batch['text_input_ids']
    attention_mask = batch['text_attention_mask']
    labels = batch['labels']
    labels_attention_mask = batch['labels_attention_mask']

    #self.checkType(input_ids, attention_mask, labels, labels_attention_mask, m

    loss, outputs = self(input_ids = input_ids,
      attention_mask = attention_mask,
      labels = labels,
      decoder_attention_mask = labels_attention_mask
    )

    self.log("val_loss", loss, prog_bar = True, logger = True)

    return loss

  def test_step(self, batch, batch_idx):
    input_ids = batch['text_input_ids']
    attention_mask = batch['text_attention_mask']
    labels = batch['labels']
    labels_attention_mask = batch['labels_attention_mask']

    #self.checkType(input_ids, attention_mask, labels, labels_attention_mask, m

    loss, outputs = self(input_ids = input_ids,
      attention_mask = attention_mask,
      labels = labels,
      decoder_attention_mask = labels_attention_mask
    )

    self.log("test_loss", loss, prog_bar = True, logger = True)

    return loss

  def configure_optimizers(self):

      return AdamW(self.parameters(), lr = config.learning_rate)
```

4.1. T5 Transformer

...goto toc

In [16]:

```
# T5 transfromer
config.t5_model_path = "t5-base"

# T5 Fast Tokenizer
config.t5_tokenizer = T5TokenizerFast.from_pretrained(config.t5_model_path)

# T5 pretrained model
config.t5_pretrained_model = T5ForConditionalGeneration.from_pretrained(config.
```

In [17]:

```
# Create datamodule for T5 model
t5_data_module = NewsSummaryDataModule(train_df, val_df, config.t5_tokenizer, b
```

In [18]:

```
# Build the T5 model as pytorch-lightning
t5_model = NewsSummaryModel(config.t5_pretrained_model)
```

In [19]:

```
# Create custom Model Checkpoint
t5_checkpoint_callback = ModelCheckpoint(
    dirpath = "t5_checkpoints",
    filename = "t5-best-checkpoint",
    save_top_k = 1,
    verbose = True,
    monitor = "val_loss",
    mode = "min",
)

# Create tensorboard logger
t5_logger = TensorBoardLogger("t5_lightning_logs", name = "t5-news-summary")
```

In [20]:

```
# Build the trainer
t5_trainer = pl.Trainer(
    logger = t5_logger,
    checkpoint_callback = t5_checkpoint_callback,
    max_epochs = config.n_epochs,
    gpus = 1,
    progress_bar_refresh_rate = 30
)
```

```
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
```

In [21]:

```
# Fit the model
t5_trainer.fit(t5_model, t5_data_module)
```

```
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name  | Type                       | Params
-----------------------------------------------------
0 | model | T5ForConditionalGeneration | 222 M
-----------------------------------------------------
222 M     Trainable params
0         Non-trainable params
222 M     Total params
891.614   Total estimated model params size (MB)
```

```
Epoch 0, global step 494: val_loss reached 1.42206 (best 1.42206), saving model
to "/content/t5_checkpoints/t5-best-checkpoint.ckpt" as top 1
```

```
Epoch 1, global step 989: val_loss reached 1.40367 (best 1.40367), saving model
to "/content/t5_checkpoints/t5-best-checkpoint.ckpt" as top 1
```

```
Epoch 2, global step 1484: val_loss reached 1.40129 (best 1.40129), saving mode
l to "/content/t5_checkpoints/t5-best-checkpoint.ckpt" as top 1
```

Out[21]: 1

In [22]:

```
%reload_ext tensorboard
%tensorboard --logdir ./t5_lightning_logs
```

In [27]:

```
def summarize(text, trained_model, tokenizer):
  text_encoding = tokenizer(
      text, max_length = 512, padding = "max_length", truncation = True,
      return_attention_mask = True, add_special_tokens = True,
      return_tensors = "pt"
  )

  generated_ids = trained_model.model.generate(
      input_ids = text_encoding['input_ids'],
      attention_mask = text_encoding['attention_mask'],
      max_length = 150,
      num_beams = 2,
      repetition_penalty = 2.5,
      length_penalty = 1.0,
      early_stopping = True
  )

  predictions = [
   tokenizer.decode(gen_id, skip_special_tokens=True, clean_up_tokenization_spa
   for gen_id in generated_ids
  ]

  return "".join(predictions)
```

In [28]:

```
## Perform predictions on test set
sample = val_df.iloc[:100, :]

test_texts = sample['text'].values.tolist()
actual_summaries = sample['summary'].values.tolist()

predicted_summaries = [summarize(text, t5_model, config.t5_tokenizer) for text
```

```
/usr/local/lib/python3.7/dist-packages/torch/_tensor.py:575: UserWarning: floor
_divide is deprecated, and will be removed in a future version of pytorch. It c
urrently rounds toward 0 (like the 'trunc' function NOT 'floor'). This results
in incorrect rounding for negative values.
To keep the current behavior, use torch.div(a, b, rounding_mode='trunc'), or fo
r actual floor division, use torch.div(a, b, rounding_mode='floor'). (Triggered
internally at  /pytorch/aten/src/ATen/native/BinaryOps.cpp:467.)
  return torch.floor_divide(self, other)
```

In [29]:

```
def get_rouge_scores(actual_summaries, predicted_summaries):

  rouge = Rouge()
  scores = rouge.get_scores(actual_summaries, predicted_summaries)

  for i in tqdm(range(len(scores))):
    for metrics_ in scores[i].keys():
      scores[i][metrics_] = scores[i][metrics_]['f']

  return scores
```

```
scores = get_rouge_scores(actual_summaries, predicted_summaries)
```

In [30]:

```
t5_scores_df = pd.DataFrame(columns=["rouge-1", "rouge-2", "rouge-l"])
t5_scores_df = t5_scores_df.from_dict(scores)
t5_scores_df
```

Out[30]:

	rouge-1	rouge-2	rouge-l
0	0.571429	0.355140	0.549451
1	0.574468	0.318584	0.553191
2	0.514286	0.406780	0.514286
3	0.541667	0.347826	0.520833
4	0.365591	0.160714	0.344086
...	...	...	...
95	0.652632	0.432432	0.652632
96	0.407767	0.119658	0.368932
97	0.516129	0.310345	0.451613
98	0.490196	0.186441	0.392157
99	0.495238	0.306452	0.400000

100 rows × 3 columns

In [31]:

```
print(f"Rouge-1 : {t5_scores_df['rouge-1'].mean()}")
print(f"Rouge-2 : {t5_scores_df['rouge-2'].mean()}")
print(f"Rouge-l : {t5_scores_df['rouge-l'].mean()}")
```

```
Rouge-1 : 0.45317305080246073
Rouge-2 : 0.23210318060306115
Rouge-l : 0.4180516141961908
```

In [32]:

```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [34]:

```
!cp "/content/t5_checkpoints" -r "/content/drive/MyDrive/Abstractive Text Summa
```

In [35]:

```
!cp "/content/t5_lightning_logs" -r "/content/drive/MyDrive/Abstractive Text Su
```

4.2. Bart Transformer

...goto toc

In [15]:

```
# Bart transfromer
config.bart_model_path = "facebook/bart-base"
```

```
# T5 Fast Tokenizer
config.bart_tokenizer = BartTokenizerFast.from_pretrained(config.bart_model_pat

# T5 pretrained model
config.bart_pretrained_model = BartForConditionalGeneration.from_pretrained(con
```

In [16]:
```
# Create datamodule for bart model
bart_data_module = NewsSummaryDataModule(train_df, val_df, config.bart_tokenize
```

In [17]:
```
# Build the T5 model as pytorch-lightning
bart_model = NewsSummaryModel(config.bart_pretrained_model)
```

In [18]:
```
# Create custom Model Checkpoint
bart_checkpoint_callback = ModelCheckpoint(
    dirpath = "bart_checkpoints",
    filename = "bart-best-checkpoint",
    save_top_k = 1,
    verbose = True,
    monitor = "val_loss",
    mode = "min",
)

# Create tensorboard logger
bart_logger = TensorBoardLogger("bart_lightning_logs", name = "bart-news-summar
```

In [19]:
```
# Build the trainer
bart_trainer = pl.Trainer(
    logger = bart_logger,
    checkpoint_callback = bart_checkpoint_callback,
    max_epochs = config.n_epochs,
    gpus = 1,
    progress_bar_refresh_rate = 30
)
```

```
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
```

In [20]:
```
# Fit the model
bart_trainer.fit(bart_model, bart_data_module)
```

```
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name  | Type                         | Params
-------------------------------------------------------
0 | model | BartForConditionalGeneration | 139 M
-------------------------------------------------------
139 M     Trainable params
0         Non-trainable params
```

```
139 M     Total params
557.682   Total estimated model params size (MB)
```

```
Epoch 0, global step 494: val_loss reached 1.01568 (best 1.01568), saving model
to "/content/bart_checkpoints/bart-best-checkpoint.ckpt" as top 1
```

```
Epoch 1, step 989: val_loss was not in top 1
```

```
Epoch 2, step 1484: val_loss was not in top 1
```

Out[20]: 1

In [21]:

```
%reload_ext tensorboard
%tensorboard --logdir ./bart_lightning_logs
```

In [23]:

```
def summarize(text, trained_model, tokenizer):
  text_encoding = tokenizer(
      text, max_length = 512, padding = "max_length", truncation = True,
      return_attention_mask = True, add_special_tokens = True,
      return_tensors = "pt"
  )

  generated_ids = trained_model.model.generate(
      input_ids = text_encoding['input_ids'],
      attention_mask = text_encoding['attention_mask'],
      max_length = 150,
      num_beams = 2,
      repetition_penalty = 2.5,
      length_penalty = 1.0,
      early_stopping = True
  )

  predictions = [
   tokenizer.decode(gen_id, skip_special_tokens=True, clean_up_tokenization_spa
   for gen_id in generated_ids
  ]

  return "".join(predictions)

## Perform predictions on test set
sample = val_df.iloc[:100, :]

test_texts = sample['text'].values.tolist()
actual_summaries = sample['summary'].values.tolist()

predicted_summaries = [summarize(text, bart_model, config.bart_tokenizer) for t
```

```
/usr/local/lib/python3.7/dist-packages/torch/_tensor.py:575: UserWarning: floor
_divide is deprecated, and will be removed in a future version of pytorch. It c
urrently rounds toward 0 (like the 'trunc' function NOT 'floor'). This results
in incorrect rounding for negative values.
To keep the current behavior, use torch.div(a, b, rounding_mode='trunc'), or fo
r actual floor division, use torch.div(a, b, rounding_mode='floor'). (Triggered
internally at  /pytorch/aten/src/ATen/native/BinaryOps.cpp:467.)
  return torch.floor_divide(self, other)
```

In [24]:

```
def get_rouge_scores(actual_summaries, predicted_summaries):
```

```
    rouge = Rouge()
    scores = rouge.get_scores(actual_summaries, predicted_summaries)

    for i in tqdm(range(len(scores))):
      for metrics_ in scores[i].keys():
        scores[i][metrics_] = scores[i][metrics_]['f']

    return scores

  scores = get_rouge_scores(actual_summaries, predicted_summaries)
```

In [25]:

```
bart_scores_df = pd.DataFrame(columns=["rouge-1", "rouge-2", "rouge-l"])
bart_scores_df = bart_scores_df.from_dict(scores)
bart_scores_df
```

Out[25]:

	rouge-1	rouge-2	rouge-l
0	0.484211	0.319328	0.484211
1	0.388889	0.208000	0.370370
2	0.568627	0.438596	0.529412
3	0.659091	0.558559	0.636364
4	0.408163	0.071429	0.326531
...	...	...	...
95	0.361702	0.201835	0.361702
96	0.408163	0.196429	0.387755
97	0.517647	0.385321	0.494118
98	0.387755	0.201835	0.346939
99	0.358491	0.100840	0.283019

100 rows × 3 columns

In [26]:

```
print(f"Rouge-1 : {bart_scores_df['rouge-1'].mean()}")
print(f"Rouge-2 : {bart_scores_df['rouge-2'].mean()}")
print(f"Rouge-l : {bart_scores_df['rouge-l'].mean()}")
```

```
Rouge-1 : 0.4315419470197944
Rouge-2 : 0.20850446483227983
Rouge-l : 0.39818282263688576
```

In [27]:

```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [28]:

```
!cp "/content/bart_checkpoints" -r "/content/drive/MyDrive/Abstractive Text Sum
```

In [29]:

```
!cp "/content/bart_lightning_logs" -r "/content/drive/MyDrive/Abstractive Text
```

5. Model Comparision

...goto toc

In [44]:

```
final_results_df = pd.DataFrame(columns = ['Model', "Rouge-1", "Rouge-2", "Roug

temp_list = [dict(zip(final_results_df.columns.values.tolist(), ["T5", bart_sco
temp_list.append(dict(zip(final_results_df.columns.values.tolist(), ["Bart", ba

final_results_df = final_results_df.from_dict(temp_list)
final_results_df
```

Out[44]:

	Model	Rouge-1	Rouge-2	Rouge-l
0	T5	0.453173	0.232103	0.418052
1	Bart	0.431542	0.208504	0.398183

We can see that **T5** Model outperformed **Bart** with higher rouge score.

In [9]:

```
# T5 transfromer
config.t5_model_path = "t5-base"

# T5 Fast Tokenizer
config.t5_tokenizer = T5TokenizerFast.from_pretrained(config.t5_model_path)

# T5 pretrained model
config.t5_pretrained_model = T5ForConditionalGeneration.from_pretrained(config.
```

In [7]:

```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [20]:

```
def summarize(text, trained_model, tokenizer):
  text_encoding = tokenizer(
      text, max_length = 512, padding = "max_length", truncation = True,
      return_attention_mask = True, add_special_tokens = True,
      return_tensors = "pt"
  )

  generated_ids = trained_model.model.generate(
      input_ids = text_encoding['input_ids'],
      attention_mask = text_encoding['attention_mask'],
      max_length = 150,
      num_beams = 2,
      repetition_penalty = 2.5,
      length_penalty = 1.0,
      early_stopping = True
  )

  predictions = [
   tokenizer.decode(gen_id, skip_special_tokens=True, clean_up_tokenization_spa
```

```python
     for gen_id in generated_ids
    ]

    return "".join(predictions)
```

In [6]:

```python
text = "Machine learning is a branch of artificial intelligence (AI) and comput
text
```

Out[6]:

```
'Machine learning is a branch of artificial intelligence (AI) and computer scie
nce which focuses on the use of data and algorithms to imitate the way that hum
ans learn, gradually improving its accuracy. Machine learning is an important c
omponent of the growing field of data science. Through the use of statistical m
ethods, algorithms are trained to make classifications or predictions, uncoveri
ng key insights within data mining projects. These insights subsequently drive
decision making within applications and businesses, ideally impacting key growt
h metrics. As big data continues to expand and grow, the market demand for data
scientists will increase, requiring them to assist in the identification of the
most relevant business questions and subsequently the data to answer them.'
```

In [16]:

```python
## Sample predictions using trained summarizer
model_checkpoint_path = "/content/drive/MyDrive/Abstractive Text Summarization/
trained_model = NewsSummaryModel(config.t5_pretrained_model)
model = trained_model.load_from_checkpoint(model_checkpoint_path)
model.freeze()
```

In [22]:

```python
summarised_text_ = summarize(text, model, config.t5_tokenizer)
summarised_text_
```

Out[22]:

```
'Machine learning is a branch of artificial intelligence (AI) and computer scie
nce which focuses on the use of data and algorithms to imitate the way that hum
ans learn, gradually improving its accuracy. Through statistical methods, algor
ithms are trained to make classifications or predictions, uncovering key insigh
ts within data mining projects. As big data continues to expand and grow, the m
arket demand for data scientists will increase.'
```