AI Project Report
Title: Summarization Models

Submitted By:-

- Prathamesh Chakote (Chakote@pdx.edu)
- Piyush Rajkarne (Rajkarne@pdx.edu)
- Shivraj Khose (Khose@pdx.edu)
- Rasika Satpute (rsatpute@pdx.edu)

About Project:

In our project, we've trained models to summarize large articles and paragraphs. Using a dataset with approximately 4500 rows, we employed T5 and Keras to train the models. We generated predictions and compared them to the text column in the dataset to measure accuracy using ROUGE metrics. For both T5 and Keras, we trained two models each, tweaking the hyperparameters, and compared their results.

Hugging Face Repository:

We trained the model using T5 and Keras :

https://huggingface.co/shivraj221/news-summary-t5-model-2

https://huggingface.co/shivraj221/mt5-small-finetuned-news-summary-kaggle

https://huggingface.co/PrathameshC/transformer-text-summarization-model

About Dataset:

Dataset: https://www.kaggle.com/datasets/sunnysai12345/news-summary/

Dataset Columns and Contents:

Author_name: The name of the author who wrote the article.

Headlines: The headline of the news article.

Url of Article: The URL where the article can be accessed.

Short text: A brief summary of the article.

Complete Article (denoted as ctext): The full text of the article.

We focused on two columns: ctext (Complete Article) and Short text. These were renamed to text and summary respectively for ease of use.

Keras Model :

The Keras Transformer model for text summarization leverages the attention mechanism to generate concise summaries of lengthy texts. By utilizing an encoder-decoder architecture, the Transformer can efficiently process input sequences and produce coherent summaries. The encoder captures the context of the input text, while the decoder generates the summary by focusing on relevant parts of the input. This approach is effective in handling the complexities of natural language, making it a powerful tool for summarizing documents, articles, and other textual content. Keras, with its user-friendly interface and integration with TensorFlow, simplifies the implementation and training of these advanced models.

We performed the following steps while implementing Keras Model:

1. Data Cleaning, Tokenization, and Preprocessing

Data cleaning involves several steps to standardize and prepare the text for model training:

Conversion to Lowercase: All text was converted to lowercase to ensure uniformity.

Accent Removal: Accented characters were converted to their unaccented equivalents using the unidecode library.

Character Filtering: Only specific characters (lowercase letters and punctuation) were retained, removing any non-relevant characters.

Tokenization: The cleaned text was split into tokens (words) to facilitate vocabulary creation and sequence generation.

The cleaned data was then transformed into sequences of integers, where each integer represented a word in the vocabulary. This conversion was necessary for inputting the data into the neural network. Special tokens (<SOS>, <EOS>, <PAD>, <OUT>) were also added to the vocabulary to denote the start and end of sentences, padding, and out-of-vocabulary words.

<u>2. Imbalance or Biases in the Dataset</u>

Length Disparities*:* Significant differences in the lengths of articles and summaries could lead to uneven training. This can be addressed by padding sequences to a uniform length and potentially discarding extremely long or short articles.

Author Bias*:* If a small number of authors contribute a large proportion of the articles, their writing style might dominate the dataset. Ensuring a diverse range of authors can help mitigate this.

Content Bias*:* If certain topics are overrepresented, the model might become biased towards those topics. Balancing the dataset to include a wider range of topics can improve model generalization.

To address these imbalances, we can implement the following solutions:

Stratified Sampling*:* Ensure that the training set includes a representative sample of articles of varying lengths and topics.

***Data Augmentation**:* Generate synthetic data by slightly modifying existing articles to increase diversity.

***Balancing Author Contributions**:* Limit the number of articles per author in the training set to reduce author bias.

We focused on two columns: ctext (Complete Article) and Short text. These were renamed to text and summary respectively for ease of use.

3. Data Cleaning, Tokenization, and Preprocessing

Data cleaning involved several steps to standardize and prepare the text for model training:

Conversion to Lowercase: All text was converted to lowercase to ensure uniformity.

Accent Removal: Accented characters were converted to their unaccented equivalents using the unidecode library.

Character Filtering: Only specific characters (lowercase letters and punctuation) were retained, removing any non-relevant characters.

Tokenization: The cleaned text was split into tokens (words) to facilitate vocabulary creation and sequence generation.

The cleaned data was then transformed into sequences of integers, where each integer represented a word in the vocabulary. This conversion was necessary for inputting the data into the neural network. Special tokens (<SOS>, <EOS>,

<PAD>, <OUT>) were also added to the vocabulary to denote the start and end of sentences, padding, and out-of-vocabulary words.

4. Imbalance or Biases in the Dataset

Exploring the dataset for imbalances or biases is crucial for building a robust model. Potential issues include:

Length Disparities: Significant differences in the lengths of articles and summaries could lead to uneven training. This can be addressed by padding sequences to a uniform length and potentially discarding extremely long or short articles.

5. Evaluation matrices:

ROUGE (Recall-Oriented Understudy for Gisting Evaluation):

The ROUGE (Recall-Oriented Understudy for Gisting Evaluation) scores are metrics used to evaluate the quality of summaries generated by a model by comparing them to reference summaries.

ROUGE-1 measures the overlap of unigrams (individual words) between the generated summary and the reference summary.

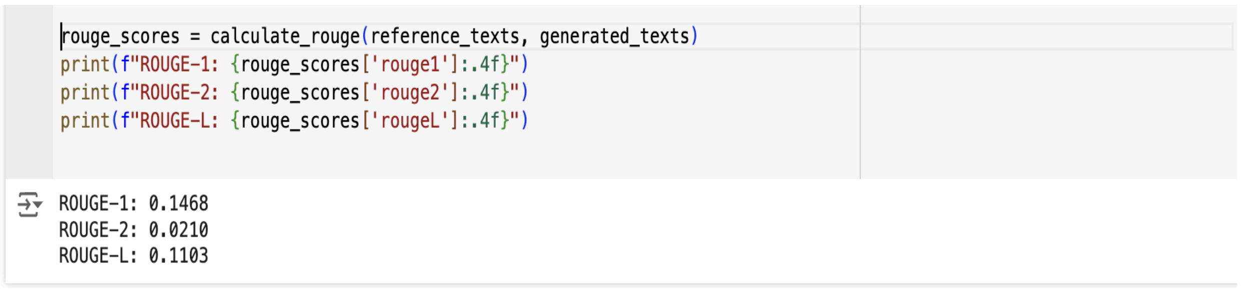
ROUGE-2 measures the overlap of bigrams (pairs of consecutive words) between the generated summary and the reference summary.

ROUGE-L measures the longest common subsequence (LCS) between the generated summary and the reference summary, capturing the ability to maintain the order of the sequences.

RESULTS:

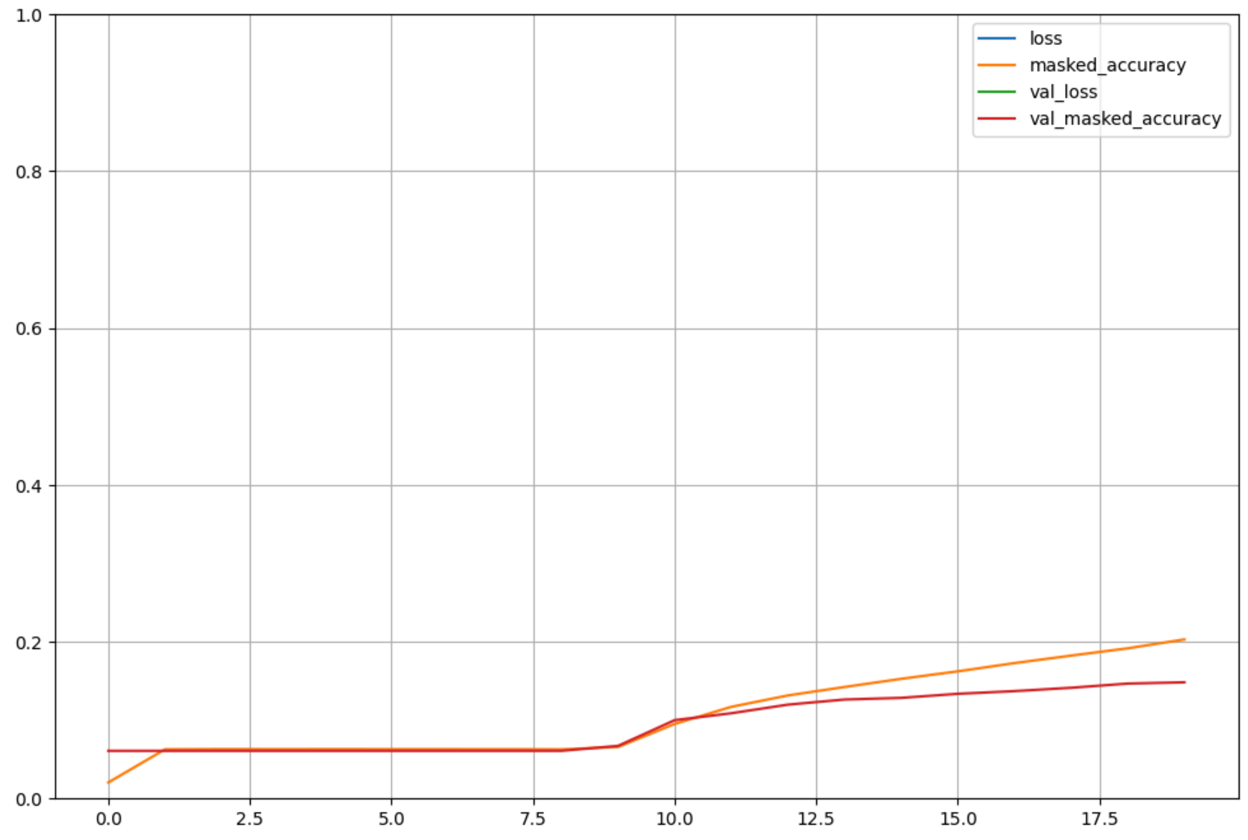
1] In first cycle, we trained the model using following hyperparameters:

```
num_layers = 4
d_model = 128
dff = 512
num_heads = 8
dropout_rate = 0.1
```



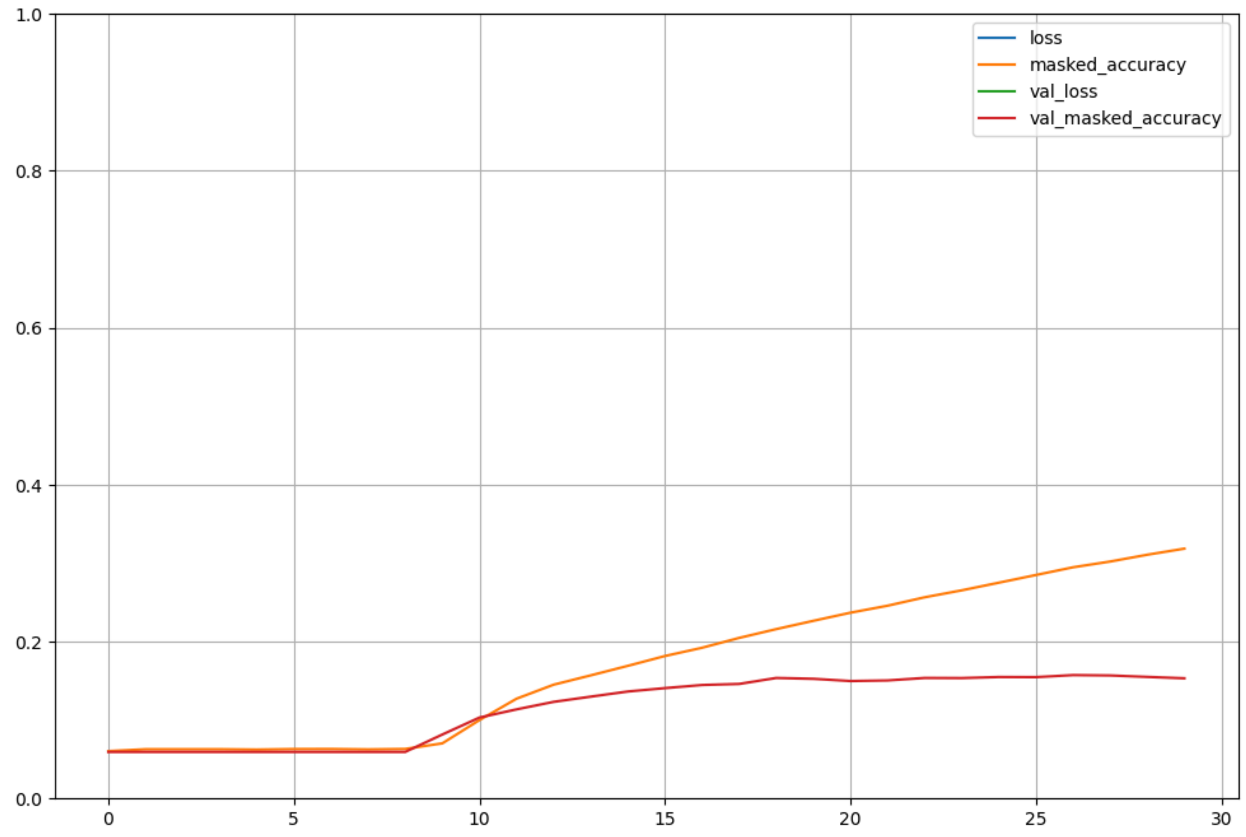
```
rouge_scores = calculate_rouge(reference_texts, generated_texts)
print(f"ROUGE-1: {rouge_scores['rouge1']:.4f}")
print(f"ROUGE-2: {rouge_scores['rouge2']:.4f}")
print(f"ROUGE-L: {rouge_scores['rougeL']:.4f}")
```

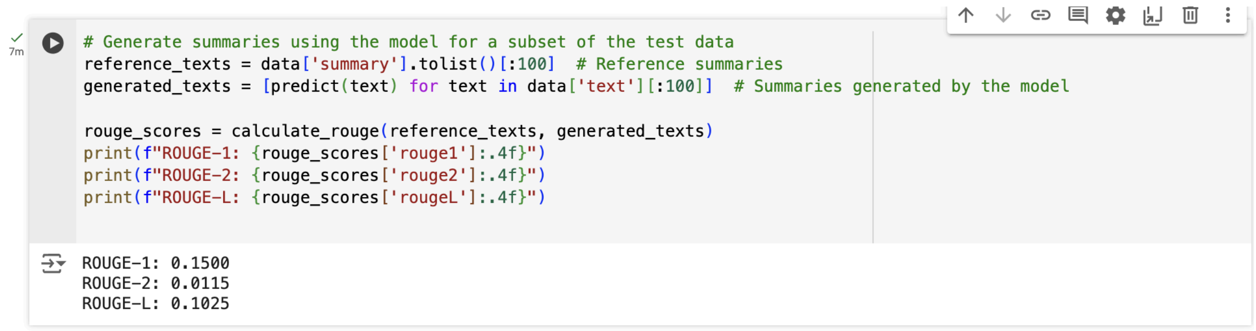
```
ROUGE-1: 0.1468
ROUGE-2: 0.0210
ROUGE-L: 0.1103
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
loss
masked_accuracy
val_loss
val_masked_accuracy

2] In second cycle, we tweaked the hyperparameters as following:

```
# Define Hyperparameters
BATCH_SIZE = 128
BUFFER_SIZE = len(encoder_inp)
steps_per_epoch = len(encoder_inp) // BATCH_SIZE
embedding_dim = 512
units = 1024
vocab_inp_size = len(vocab)
vocab_tar_size = len(vocab)
```





```
# Generate summaries using the model for a subset of the test data
reference_texts = data['summary'].tolist()[:100]  # Reference summaries
generated_texts = [predict(text) for text in data['text'][:100]]  # Summaries generated by the model

rouge_scores = calculate_rouge(reference_texts, generated_texts)
print(f"ROUGE-1: {rouge_scores['rouge1']:.4f}")
print(f"ROUGE-2: {rouge_scores['rouge2']:.4f}")
print(f"ROUGE-L: {rouge_scores['rougeL']:.4f}")
```

```
ROUGE-1: 0.1500
ROUGE-2: 0.0115
ROUGE-L: 0.1025
```

<u>**Overall Interpretation**</u>

The graph shows that the training loss decreases and the masked accuracy increases, indicating that the model is learning effectively from the training data.

The validation loss decreases initially but then increases slightly, while the validation masked accuracy increases and then plateaus. This suggests that the model might be starting to overfit the training data, as it performs slightly worse on the validation data after a certain point.

The slight divergence between training and validation metrics towards the end of training suggests that early stopping might be useful to prevent overfitting and ensure better generalization.

In summary, the graph indicates that while the model is learning and improving on the training data, it may start to overfit after a certain number of epochs, as evidenced by the flattening of validation accuracy and the slight increase in validation loss.

T5 Model :

The T5 model is implemented by exploiting the capabilities of the Hugging Face transformers library. This library provides a comprehensive toolkit for working with transformer-based models, including seamless integration, pre-trained model access, and speed up training and evaluation workflows. At the center of the implementation is the T5 model, a versatile transformer architecture capable of handling a variety of natural language processing tasks, including text summarization.

Through a series of steps, the code loads the pre-trained T5 model, configures training parameters, and put together the training and evaluation processes. It tokenizes the input data, sets up training arguments, defines evaluation metrics, and trains the model using the provided datasets. The trained model's performance is evaluated using ROUGE metrics, a standard measure for assessing summarization quality. Once trained, the model is pushed to the Hugging Face Hub for wider accessibility and sharing. The implementation showcases the ease and efficiency of employing state-of-the-art transformer models like T5 for text summarization tasks, facilitated by the transformers library's comprehensive functionality and ecosystem support.

The following steps were performed for the implementation of T5 model

1. Installing Libraries

The code begins with installing necessary libraries, including **rouge_score** **transformers[torch]**, **accelerate**, **numpy**, **pandas** and various modules from **transformers**.

2. Preprocessing Data:

We define functions to calculate the length of headlines and texts and then print out the mean lengths. The data is split into training, validation, and test sets, and then converted into the Dataset format.

3. Tokenization:

The data is tokenized using the loaded tokenizer with defined maximum input and target lengths.

4. Baseline Evaluation:

The code defines a function to create a baseline summary by selecting the first three sentences of each article and evaluates the baseline using ROUGE metrics.

5. Model Training Configuration:

Hugging Face Hub login is performed.

The pre-trained T5 model and data collator are loaded.

Training arguments are defined, specifying the output directory, learning rate, batch size, number of epochs, and other parameters.

6. Metrics Computation:

A function is defined to compute ROUGE metrics for evaluation.

7. Training:

The model is trained using the `Seq2SeqTrainer` class with specified arguments.

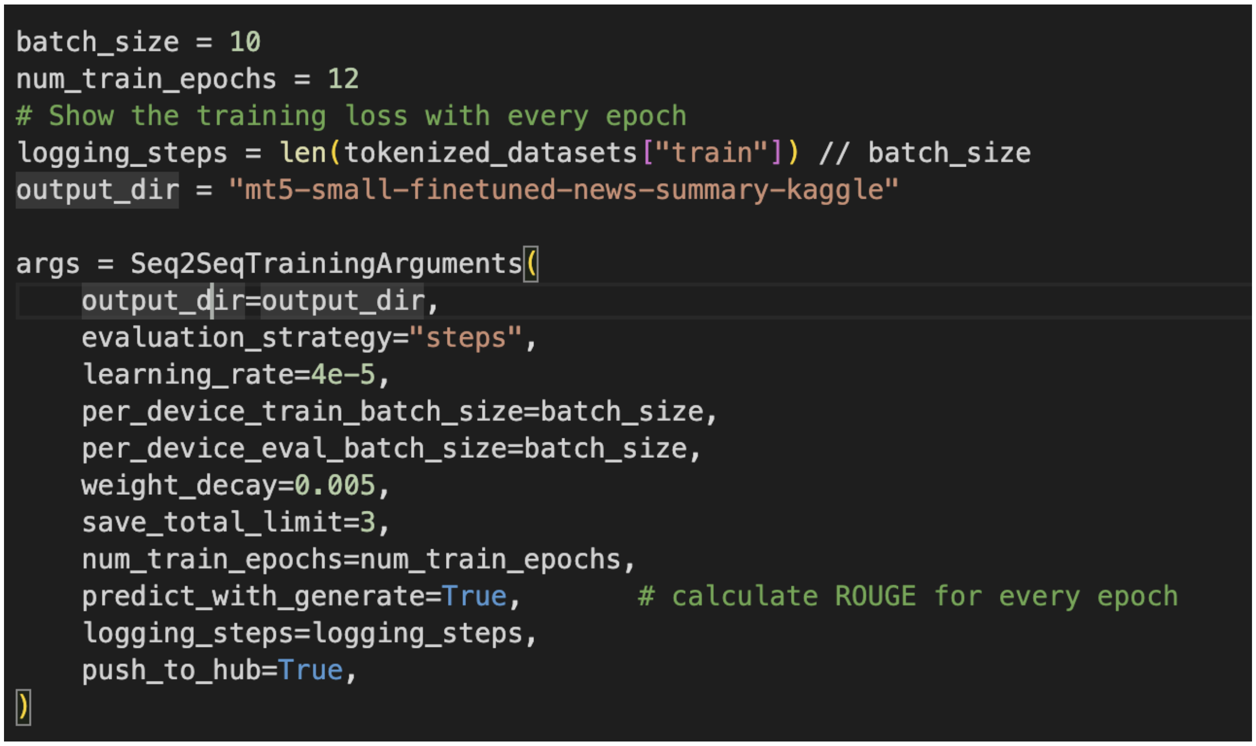
8. Evaluation:

The trained model is evaluated on the validation set.

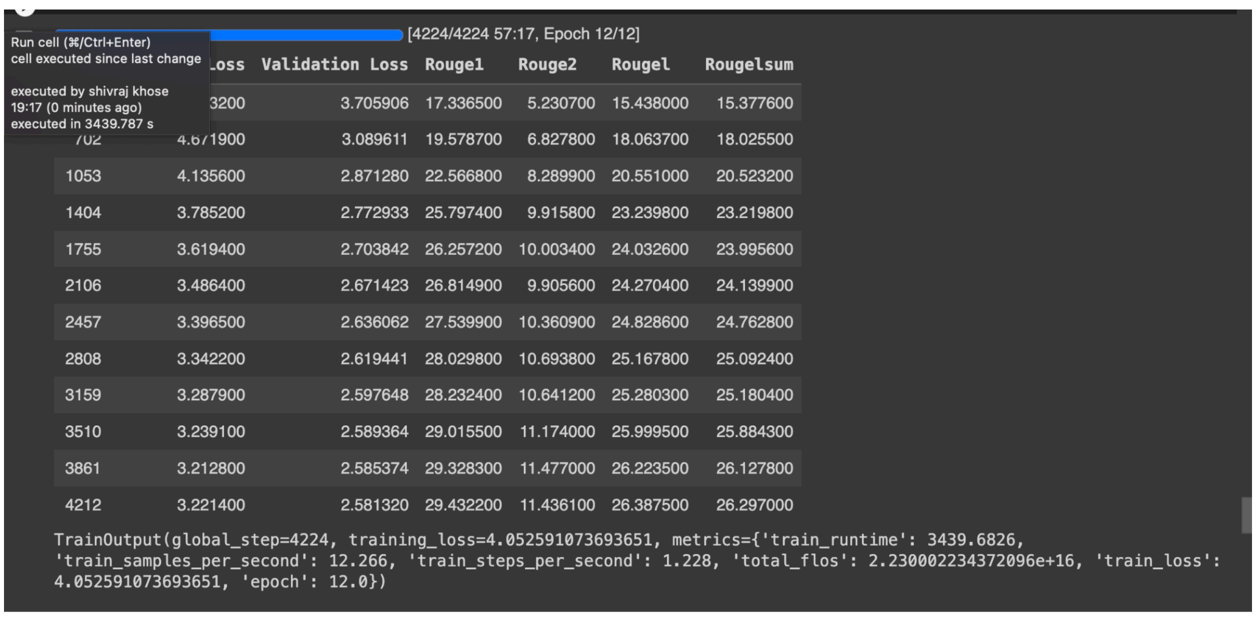
The code essentially performs the task of training a T5 model for news summarization using the provided dataset, evaluates its performance using ROUGE metrics, and pushes the trained model to the Hugging Face Hub for sharing and deployment. Finally, it sets up a pipeline for generating summaries from new articles using the trained model.

First Cycle of Training T5 Model

For first when we were training the model using the T5 transformer, we used the following hyperparameters



```
batch_size = 10
num_train_epochs = 12
# Show the training loss with every epoch
logging_steps = len(tokenized_datasets["train"]) // batch_size
output_dir = "mt5-small-finetuned-news-summary-kaggle"

args = Seq2SeqTrainingArguments(
    output_dir=output_dir,
    evaluation_strategy="steps",
    learning_rate=4e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    weight_decay=0.005,
    save_total_limit=3,
    num_train_epochs=num_train_epochs,
    predict_with_generate=True,       # calculate ROUGE for every epoch
    logging_steps=logging_steps,
    push_to_hub=True,
)
```



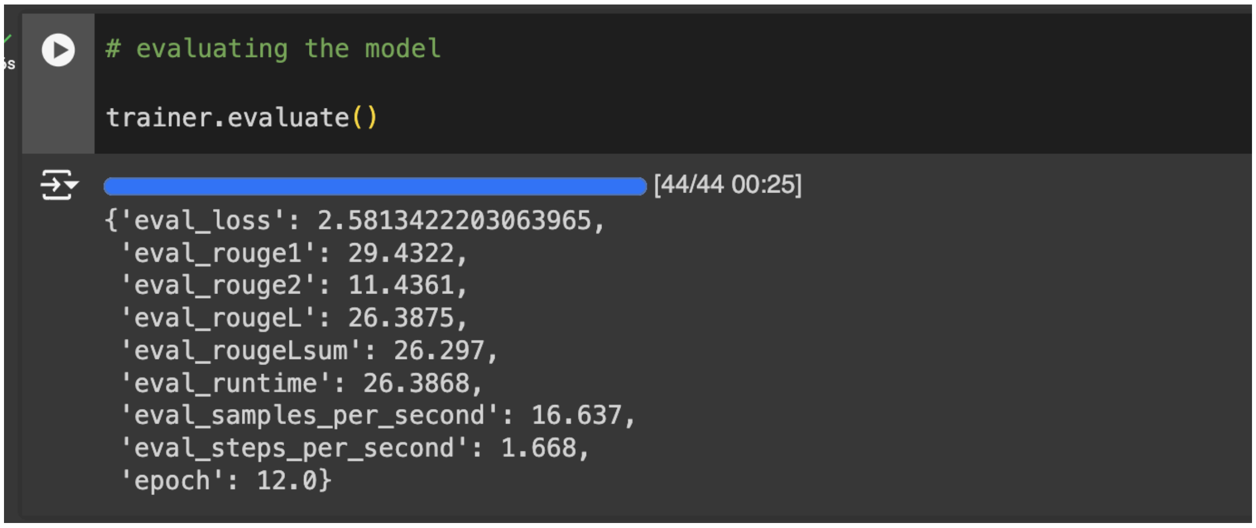
[4224/4224 57:17, Epoch 12/12]

Run cell (⌘/Ctrl+Enter)
cell executed since last change

executed by shivraj khose
19:17 (0 minutes ago)
executed in 3439.787 s

[illegible]	Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum
[illegible]	3200	3.705906	17.336500	5.230700	15.438000	15.377600
702	4.671900	3.089611	19.578700	6.827800	18.063700	18.025500
1053	4.135600	2.871280	22.566800	8.289900	20.551000	20.523200
1404	3.785200	2.772933	25.797400	9.915800	23.239800	23.219800
1755	3.619400	2.703842	26.257200	10.003400	24.032600	23.995600
2106	3.486400	2.671423	26.814900	9.905600	24.270400	24.139900
2457	3.396500	2.636062	27.539900	10.360900	24.828600	24.762800
2808	3.342200	2.619441	28.029800	10.693800	25.167800	25.092400
3159	3.287900	2.597648	28.232400	10.641200	25.280300	25.180400
3510	3.239100	2.589364	29.015500	11.174000	25.999500	25.884300
3861	3.212800	2.585374	29.328300	11.477000	26.223500	26.127800
4212	3.221400	2.581320	29.432200	11.436100	26.387500	26.297000

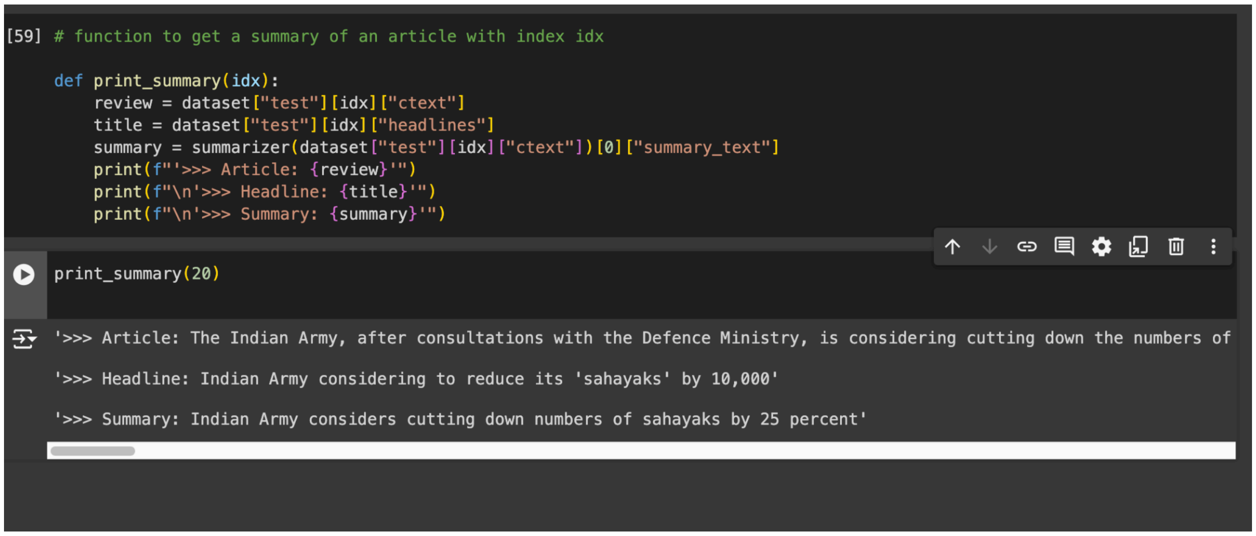
```
TrainOutput(global_step=4224, training_loss=4.052591073693651, metrics={'train_runtime': 3439.6826,
'train_samples_per_second': 12.266, 'train_steps_per_second': 1.228, 'total_flos': 2.230002234372096e+16, 'train_loss':
4.052591073693651, 'epoch': 12.0})
```



```
# evaluating the model

trainer.evaluate()
```

```
[44/44 00:25]
{'eval_loss': 2.5813422203063965,
 'eval_rouge1': 29.4322,
 'eval_rouge2': 11.4361,
 'eval_rougeL': 26.3875,
 'eval_rougeLsum': 26.297,
 'eval_runtime': 26.3868,
 'eval_samples_per_second': 16.637,
 'eval_steps_per_second': 1.668,
 'epoch': 12.0}
```

Example after training:



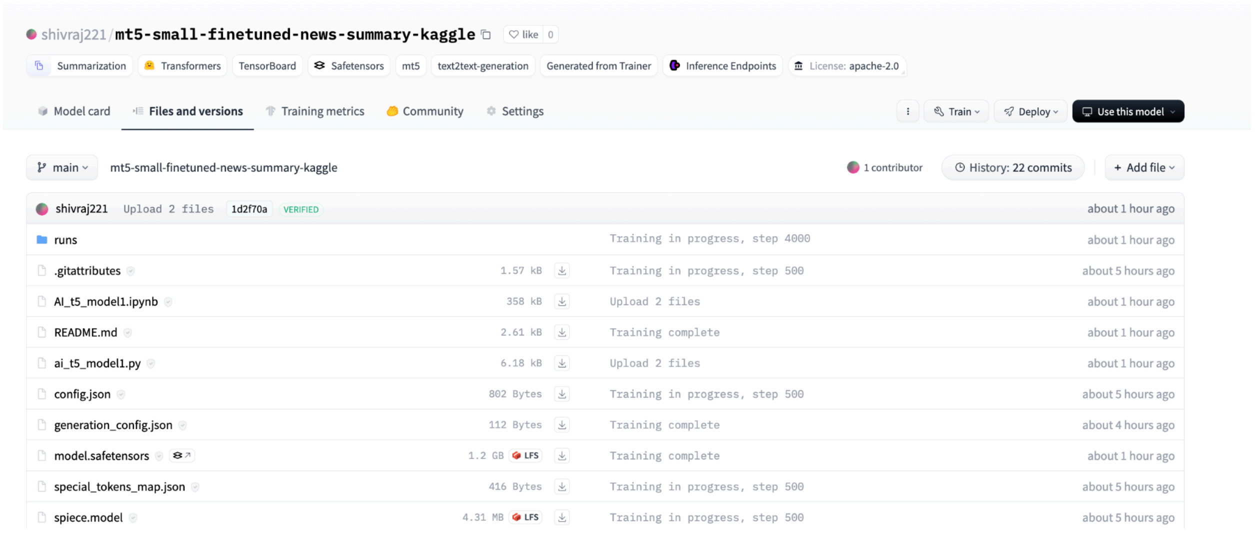
```
[59] # function to get a summary of an article with index idx

     def print_summary(idx):
         review = dataset["test"][idx]["ctext"]
         title = dataset["test"][idx]["headlines"]
         summary = summarizer(dataset["test"][idx]["ctext"])[0]["summary_text"]
         print(f"'>>> Article: {review}'")
         print(f"\n'>>> Headline: {title}'")
         print(f"\n'>>> Summary: {summary}'")
```

```
print_summary(20)
```

```
'>>> Article: The Indian Army, after consultations with the Defence Ministry, is considering cutting down the numbers of

'>>> Headline: Indian Army considering to reduce its 'sahayaks' by 10,000'

'>>> Summary: Indian Army considers cutting down numbers of sahayaks by 25 percent'
```

<u>Hugging Face Repository:</u>

https://huggingface.co/shivraj221/mt5-small-finetuned-news-summary-kaggle

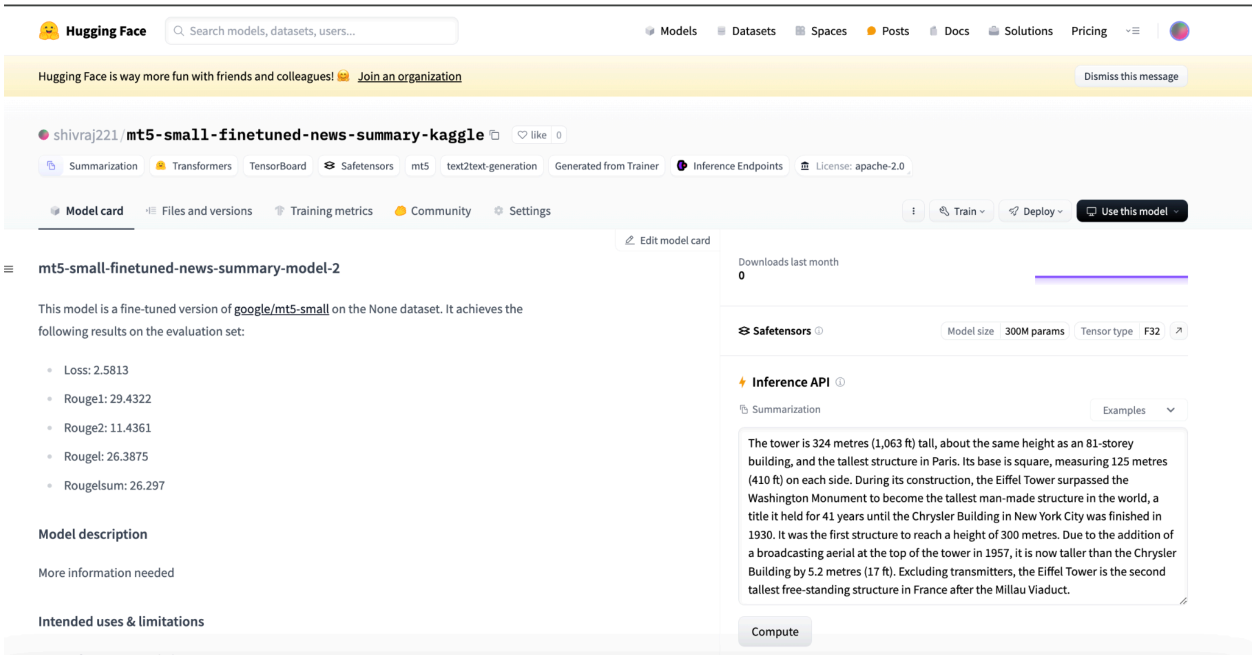


shivraj221/**mt5-small-finetuned-news-summary-kaggle**

mt5-small-finetuned-news-summary-kaggle

shivraj221 Upload 2 files 1d2f70a VERIFIED about 1 hour ago

runs		Training in progress, step 4000	about 1 hour ago
.gitattributes	1.57 kB	Training in progress, step 500	about 5 hours ago
AI_t5_model1.ipynb	358 kB	Upload 2 files	about 1 hour ago
README.md	2.61 kB	Training complete	about 1 hour ago
ai_t5_model1.py	6.18 kB	Upload 2 files	about 1 hour ago
config.json	802 Bytes	Training in progress, step 500	about 5 hours ago
generation_config.json	112 Bytes	Training complete	about 4 hours ago
model.safetensors	1.2 GB LFS	Training complete	about 1 hour ago
special_tokens_map.json	416 Bytes	Training in progress, step 500	about 5 hours ago
spiece.model	4.31 MB LFS	Training in progress, step 500	about 5 hours ago



mt5-small-finetuned-news-summary-model-2

This model is a fine-tuned version of google/mt5-small on the None dataset. It achieves the following results on the evaluation set:

- Loss: 2.5813
- Rouge1: 29.4322
- Rouge2: 11.4361
- Rougel: 26.3875
- Rougelsum: 26.297

Model description

More information needed

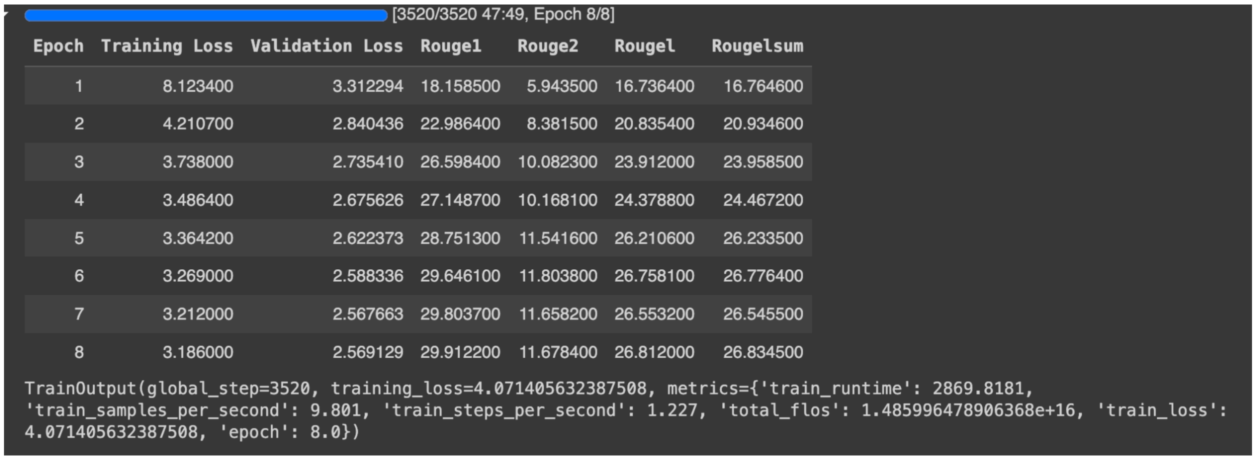
Intended uses & limitations

The tower is 324 metres (1,063 ft) tall, about the same height as an 81-storey building, and the tallest structure in Paris. Its base is square, measuring 125 metres (410 ft) on each side. During its construction, the Eiffel Tower surpassed the Washington Monument to become the tallest man-made structure in the world, a title it held for 41 years until the Chrysler Building in New York City was finished in 1930. It was the first structure to reach a height of 300 metres. Due to the addition of a broadcasting aerial at the top of the tower in 1957, it is now taller than the Chrysler Building by 5.2 metres (17 ft). Excluding transmitters, the Eiffel Tower is the second tallest free-standing structure in France after the Millau Viaduct.

Second Cycle of Training T5 Model

For second, when we were training the model using the T5 transformer, we used the following hyperparameters.
We tweaked the parameters used to train this model compared to the first 1st cycle of training and the new parameters are:

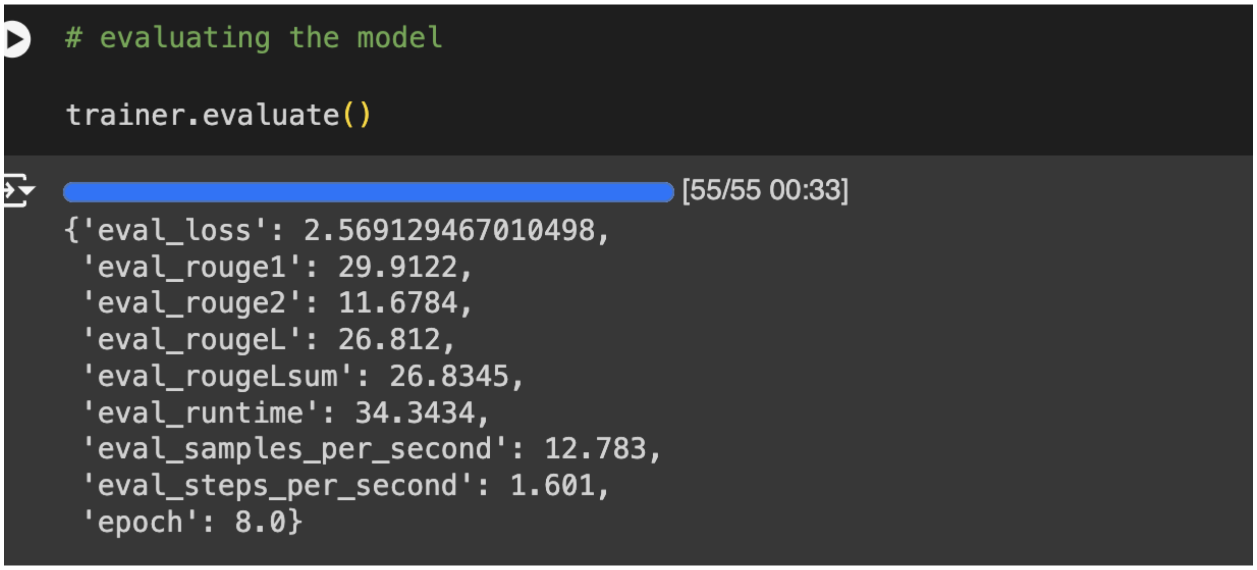
```
# setting arguments

batch_size = 8
num_train_epochs = 8
# Show the training loss with every epoch
logging_steps = len(tokenized_datasets["train"]) // batch_size
output_dir = "news-summary-t5-model-2"

args = Seq2SeqTrainingArguments(
    output_dir=output_dir,
    evaluation_strategy="epoch",
    learning_rate=5.6e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    weight_decay=0.01,
    save_total_limit=3,
    num_train_epochs=num_train_epochs,
    predict_with_generate=True,       # calculate ROUGE for every epoch
    logging_steps=logging_steps,
    push_to_hub=True,
)
```



[3520/3520 47:49, Epoch 8/8]

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum
1	8.123400	3.312294	18.158500	5.943500	16.736400	16.764600
2	4.210700	2.840436	22.986400	8.381500	20.835400	20.934600
3	3.738000	2.735410	26.598400	10.082300	23.912000	23.958500
4	3.486400	2.675626	27.148700	10.168100	24.378800	24.467200
5	3.364200	2.622373	28.751300	11.541600	26.210600	26.233500
6	3.269000	2.588336	29.646100	11.803800	26.758100	26.776400
7	3.212000	2.567663	29.803700	11.658200	26.553200	26.545500
8	3.186000	2.569129	29.912200	11.678400	26.812000	26.834500

```
TrainOutput(global_step=3520, training_loss=4.071405632387508, metrics={'train_runtime': 2869.8181,
'train_samples_per_second': 9.801, 'train_steps_per_second': 1.227, 'total_flos': 1.485996478906368e+16, 'train_loss':
4.071405632387508, 'epoch': 8.0})
```

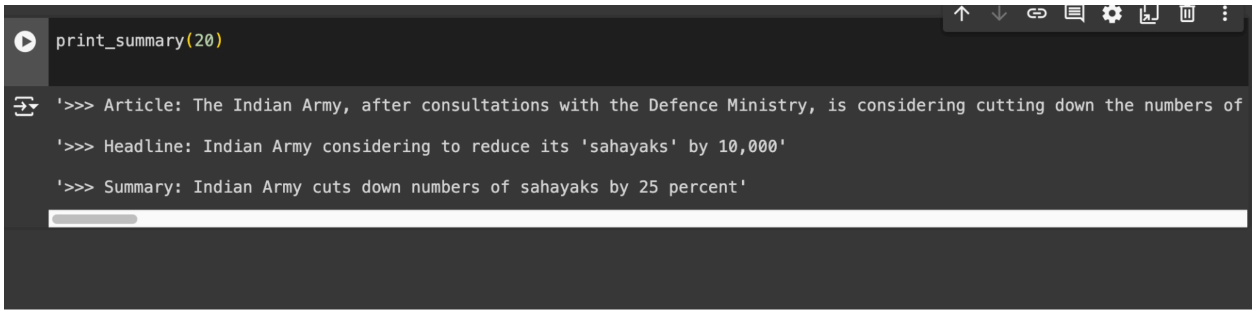


```
# evaluating the model

trainer.evaluate()
```

[55/55 00:33]

```
{'eval_loss': 2.569129467010498,
 'eval_rouge1': 29.9122,
 'eval_rouge2': 11.6784,
 'eval_rougeL': 26.812,
 'eval_rougeLsum': 26.8345,
 'eval_runtime': 34.3434,
 'eval_samples_per_second': 12.783,
 'eval_steps_per_second': 1.601,
 'epoch': 8.0}
```

Example after training:



```
print_summary(20)
```

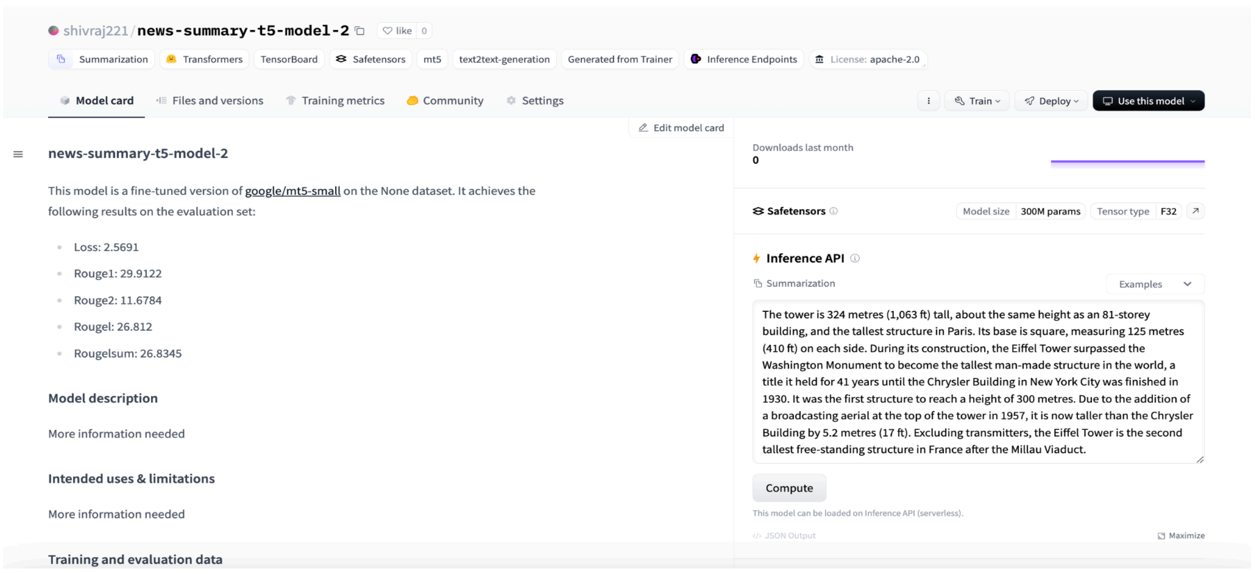
```
'>>> Article: The Indian Army, after consultations with the Defence Ministry, is considering cutting down the numbers of
'>>> Headline: Indian Army considering to reduce its 'sahayaks' by 10,000'
'>>> Summary: Indian Army cuts down numbers of sahayaks by 25 percent'
```

__Overall Interpretation of both T5 models :__

We have evaluated two fine-tuned T5 models on a news summary dataset. Model 1 (mt5-small-finetuned-news-summary-model-2) achieved a loss of 2.5813, with ROUGE scores of 29.4322 for Rouge1, 11.4361 for Rouge2, 26.3875 for Rougel, and 26.297 for Rougelsum. Model 2 (news-summary-t5-model-2) performed slightly better, with a lower loss of 2.5691 and higher ROUGE scores: 29.9122 for Rouge1, 11.6784 for Rouge2, 26.812 for Rougel, and 26.8345 for Rougelsum. Thus, Model 2 demonstrated marginally better performance in both loss and ROUGE metrics.

Hugging Face Repository:

https://huggingface.co/shivraj221/news-summary-t5-model-2



news-summary-t5-model-2

This model is a fine-tuned version of google/mt5-small on the None dataset. It achieves the following results on the evaluation set:

- Loss: 2.5691
- Rouge1: 29.9122
- Rouge2: 11.6784
- Rougel: 26.812
- Rougelsum: 26.8345

Model description

More information needed

Intended uses & limitations

More information needed

Training and evaluation data

Downloads last month
0

Safetensors — Model size 300M params — Tensor type F32

Inference API

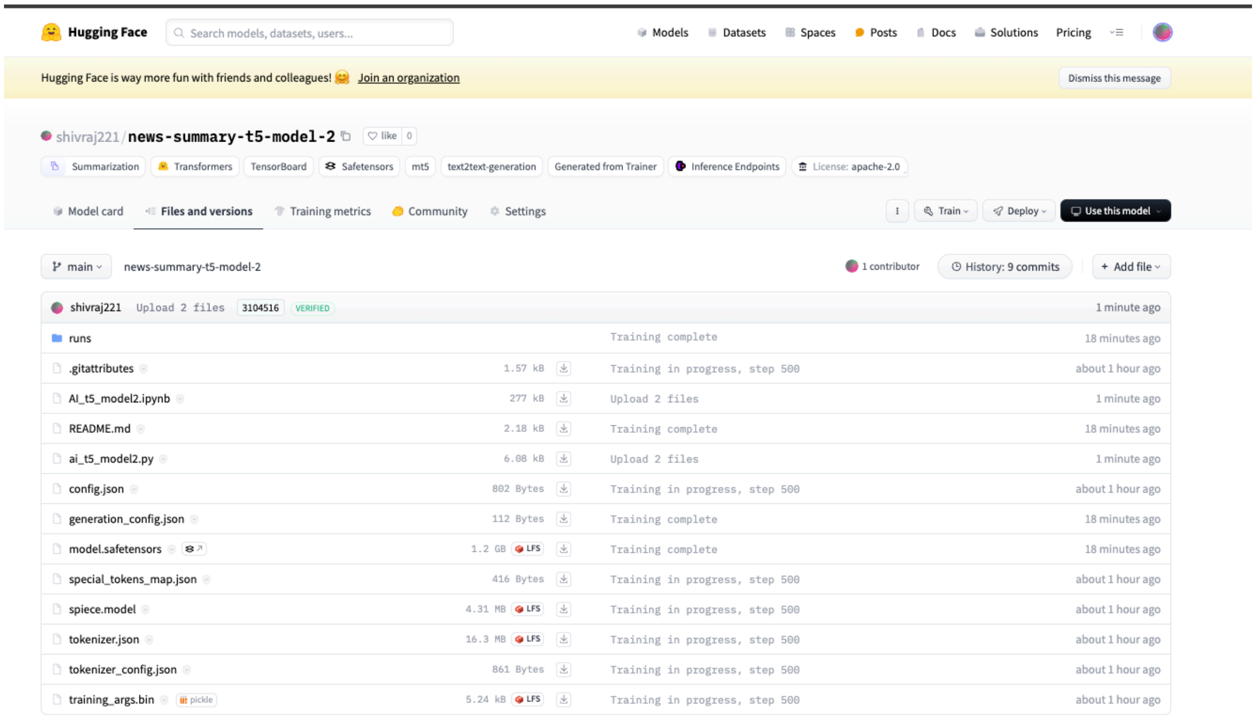
Summarization — Examples

The tower is 324 metres (1,063 ft) tall, about the same height as an 81-storey building, and the tallest structure in Paris. Its base is square, measuring 125 metres (410 ft) on each side. During its construction, the Eiffel Tower surpassed the Washington Monument to become the tallest man-made structure in the world, a title it held for 41 years until the Chrysler Building in New York City was finished in 1930. It was the first structure to reach a height of 300 metres. Due to the addition of a broadcasting aerial at the top of the tower in 1957, it is now taller than the Chrysler Building by 5.2 metres (17 ft). Excluding transmitters, the Eiffel Tower is the second tallest free-standing structure in France after the Millau Viaduct.

Compute

This model can be loaded on Inference API (serverless).

JSON Output — Maximize



Hugging Face is way more fun with friends and colleagues! Join an organization

news-summary-t5-model-2

shivraj221 Upload 2 files 3104516 VERIFIED 1 minute ago

runs		Training complete	18 minutes ago
.gitattributes	1.57 kB	Training in progress, step 500	about 1 hour ago
AI_t5_model2.ipynb	277 kB	Upload 2 files	1 minute ago
README.md	2.18 kB	Training complete	18 minutes ago
ai_t5_model2.py	6.08 kB	Upload 2 files	1 minute ago
config.json	802 Bytes	Training in progress, step 500	about 1 hour ago
generation_config.json	112 Bytes	Training complete	18 minutes ago
model.safetensors	1.2 GB LFS	Training complete	18 minutes ago
special_tokens_map.json	416 Bytes	Training in progress, step 500	about 1 hour ago
spiece.model	4.31 MB LFS	Training in progress, step 500	about 1 hour ago
tokenizer.json	16.3 MB LFS	Training in progress, step 500	about 1 hour ago
tokenizer_config.json	861 Bytes	Training in progress, step 500	about 1 hour ago
training_args.bin pickle	5.24 kB LFS	Training in progress, step 500	about 1 hour ago