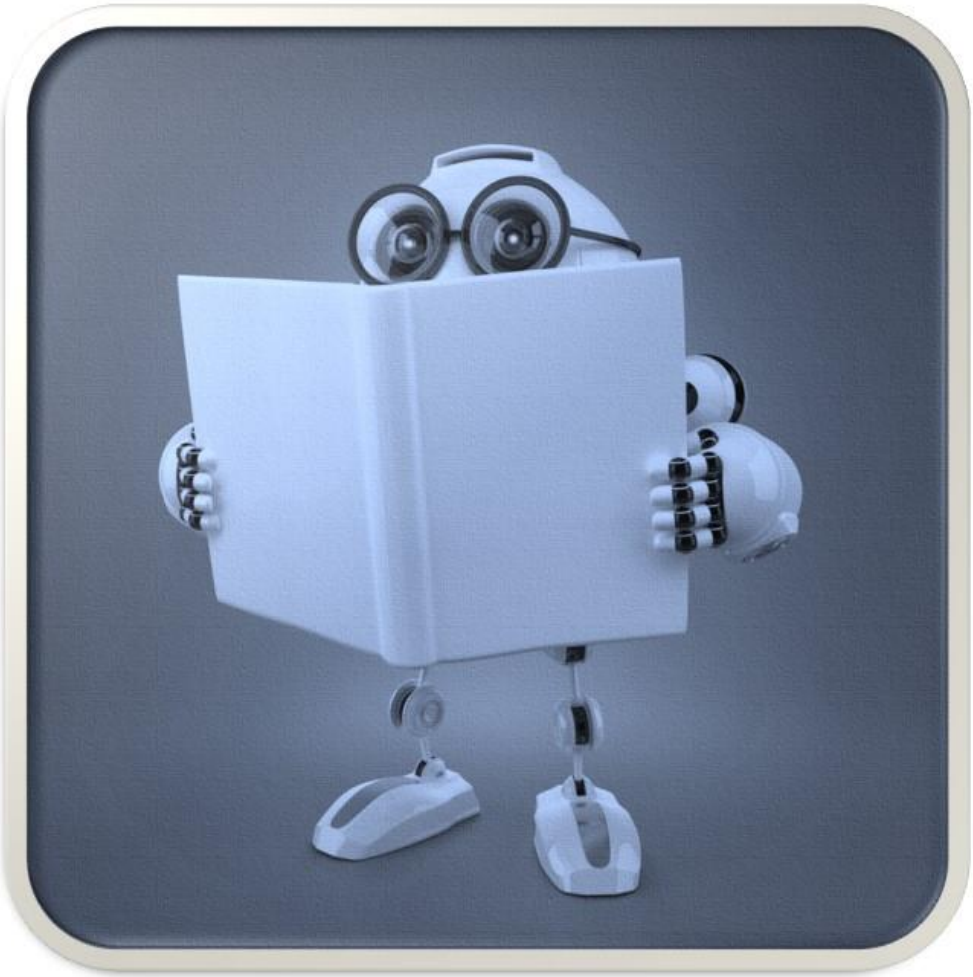

Deep Learning Abstractive Text Summarization

AIDI 2004: Final Project

Project Report

Group 17
Michael Molnar (100806823)
Vasundara Chandre Gowda (100800734)

Report Contents

1. Problem Definition

Condensing long passages of text into short and representative snippets is an important task in Natural Language Processing (NLP). Text is considered unstructured data, and there is an endless supply of unstructured data being continuously created. It has been estimated that up to 90 percent of the data being generated in the world daily is unstructured [1], and so being able to process this and derive insights from it is crucial for businesses in today's day and age.

From a consumer's point of view, there is a constant battle for clicks and views. Articles often make use of sensational, clickbait, headlines to attract readers, who can be left frustrated after spending the time reading something that was not what they had believed it would be.

With these problems in mind, automatic text summarization using artificial intelligence offers a way to condense text passages while retaining the key points, or essence, of the text. Businesses will be able to, for example, extract the key insights from their customers' reviews without having to manually read through them. Readers can spend less time searching for the content they want, and more time acquiring knowledge in short bursts.

2. Project Overview

In this project we have selected a dataset, the Amazon Fine Food Reviews Dataset, and performed a thorough exploratory data analysis on it. Next, we have built and trained a deep learning encoder-decoder model from scratch to produce novel summaries when fed review text.

Next, we have built a Flask application, using HTML and CSS, to use our trained model and other NLP techniques to process a user's input text and provide a summary of it. Finally, we have used Docker to containerize the application and Heroku to deploy it.

This report will detail all of these processes. We employed Git for version control and the full code of the project can be found at https://github.com/michael-molnar/text-summarizer.

3. Text Summarization Overview

The goal of automatic summarization is to produce a concise summary that captures the important content and context of a passage, while remaining coherent. There are two approaches to this task [2].

3.1 Extractive Summarization

Extractive summarization, as the name would suggest, is a way of pulling out key words or phrases and combining them to form the summary [3].

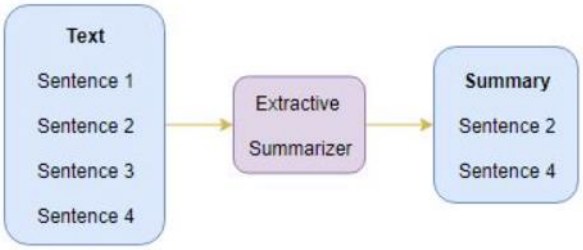

Figure 1: Extractive Summarization

The summary in this case is not created from scratch, but merely it is joined together from the original text. This can be accomplished by assigning scores to the words or phrases in the text, using metrics like Term Frequency Inverse Document Frequency (TFIDF). One downfall of this approach is that grammar can become very inconsistent as phrases are joined together.

3.2 Abstractive Summarization

On the other hand, the abstractive approach is similar to how a human being would summarize a document. The text is paraphrased in a way that captures the context of the passage, and the summary consists of entire sentences that may or may not have been originally present.

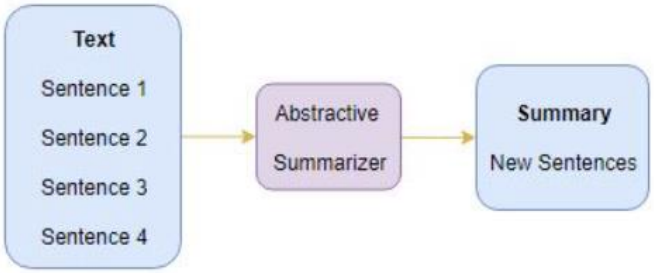

Figure 2: Abstractive Summarization

The grammar will better since full sentences are being generated. Ideally, the summary will flow naturally, be concise, and significantly shorten the original text while still maintaining all of its key information.

3.3 Pretrained Models

As mentioned at the outset of this report, text summarization has become a common task in NLP and there have been many models created that do it exceedingly well. One example we will highlight here is the T5 model from Google [4]. This is a state-of-the-art language model that performs many tasks, one being text summarization. This model is available for download from TensorFlow. As an experiment, we loaded the model and fed it the text of this news article https://www.cp24.com/news/ontario-reports-2-938-new-covid-19-cases-today-3-041-on-sunday-1.5374515. The summary it produced is displayed below.

```
Summarized text:
 2,938 new cases were reported today and 3,041 infections were recorded
on Sunday. the number of active infections confirmed over the past two
days is in line with daily case counts reported earlier this weekend.
```

Figure 3: T5's Summary of the CP24 Article

As can be clearly seen, this is how state-of-the-art summarizers are meant to function. The summary is perfectly coherent, it flows well and has proper grammar. It takes a long article and creates two novel sentences from it that capture the key points of the text.

3.4 Our Approach

For this project we have built an abstractive text summarizer using our own deep learning model that we build and train from scratch. Clearly, due to limitations on memory and computing power, our results are simple, but they are, however, promising and we are very pleased with them.

4. Dataset

As with any machine learning project, the first step is to find an appropriate dataset upon which to build and train a model. We found many great candidates for this project and elected to work with the Amazon Fine Food Reviews Dataset [5]. This is a famous dataset in the world of NLP projects due to its size and features that can be used for various tasks such as sentimental analysis, keyword extraction, summarization, and so on. The full code for our data analysis is available at https://github.com/michael-molnar/text-summarizer/blob/main/Notebook1_EDA_and_Data_PreProcessing.ipynb.

```
# Load the dataset
df = pd.read_csv('Reviews.csv')
df.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...

```
# Check the shape
df.shape
```

```
(568454, 10)
```

Figure 4: The Amazon Fine Foods Reviews Dataset

The dataset consists of over 568,000 rows, each consisting of 10 columns. For the purposes of this project, we will only be making use of two of these columns; "Text" (the actual review) and "Summary" (the user-written summary of their review).

5. Exploratory Data Analysis

We begin our analysis by removing duplicated rows and rows with missing text or summaries and are left with 394,967 unique text/summary combinations.

5.1 The "Text" Column

We split each of the text entries by words and then count how many are contained in each. We see that the average review text is about 80 words long and there are outliers on the high and low ends.

```
# Get the summary statistics
df['Text_Words'].describe()
```

```
count    394967.000000
mean         79.883347
std          77.449276
min           3.000000
25%          34.000000
50%          57.000000
75%          98.000000
max        3432.000000
Name: Text_Words, dtype: float64
```

Figure 5: Statistics for the Review Lengths

Filtering for reviews that are 50 words or less, we plot the distribution and see that the most common length is between 20 and 30 words.

```
# Plot the review length as a density plot
sns.displot(x = 'Text_Words', data = df[df['Text_Words'] < 50], kind = 'kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x20506851850>
```

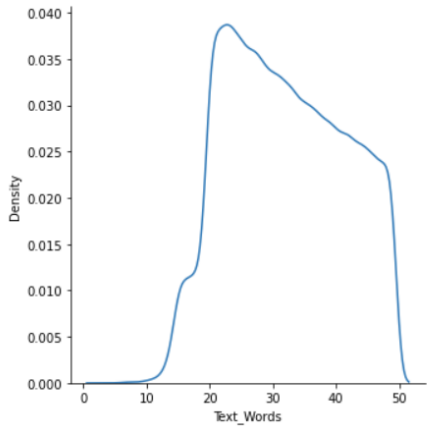

Figure 6: Distribution of Review Lengths

5.2 The "Summary" Column

We apply the same process to the summaries.

```
# Get the summary statistics
df['Summary_Words'].describe()
```

```
count    394967.000000
mean          4.090116
std           2.585658
min           1.000000
25%           2.000000
50%           3.000000
75%           5.000000
max          42.000000
Name: Summary_Words, dtype: float64
```

Figure 7: Statistics for the Summary Lengths

The average summary is only four words long and three quarters of them are five words or less. Since we are limited by computing power and memory, the shortness of these summaries will greatly assist our ability to build a functioning model. Filtering for summaries ten words or less, we see that the most common lengths are two to four words.

```
# Plot the summary length as a density plot
sns.displot(x = 'Summary_Words', data = df[df['Summary_Words'] <= 10], kind = 'kde', color = 'green')
```

```
<seaborn.axisgrid.FacetGrid at 0x2050dd54400>
```

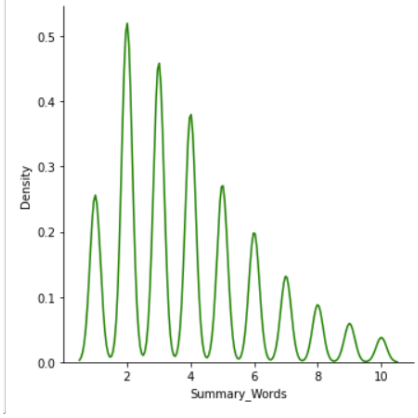

Figure 8: Distribution of Summary Lengths

5.3 The Relationship Between Review Length and Summary Length

We wanted to examine the relationship between the lengths of the reviews and the lengths of the summaries. Do people who write longer reviews also write longer summaries? We filter the data for reviews less than 100 words in length and plot a scatterplot of summary length versus review length.

```
# Plot a scatter plot comparing review length and summary length
sns.scatterplot(x = 'Text_Words', y = 'Summary_Words', data = df100, color = 'red')
```

```
<AxesSubplot:xlabel='Text_Words', ylabel='Summary_Words'>
```

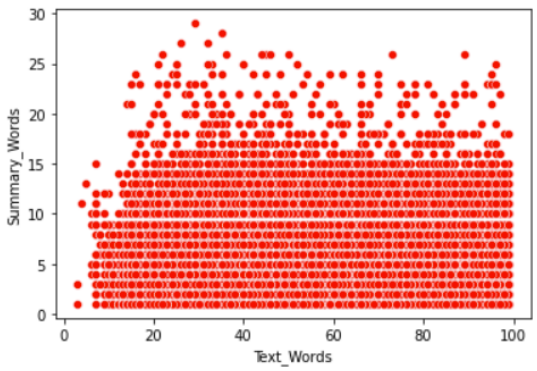

Figure 9: Correlation Between Review Length and Summary Length

We see no clear trend here, and a check of the correlation coefficient produces a value of 0.2846, indicating slightly positive correlation.

5.4 Examining Text/Summary Combinations

Since we are building an abstractive summarizer, where the summary text is novel and generated from scratch, we want to examine how well our dataset captures this process. To do this we first examine the first row of our dataset.

```
# Get the first review in the dataset
sample_review = df['Text'][0].lower()
print('Sample Review:', sample_review)
```

```
Sample Review: i have bought several of the vitality canned dog food products and have found them all to be of good quality. th
e product looks more like a stew than a processed meat and it smells better. my labrador is finicky and she appreciates this pr
oduct better than  most.
```

```
# Get the first summary in the dataset
sample_summary = df['Summary'][0].lower()
print('Sample Summary:', sample_summary)
```

```
Sample Summary: good quality dog food
```

Figure 10: The First Review and Summary in the Dataset

We see that in this example the summary does not actually appear in the review text, though the individual words do. This is the goal of what we are trying to build; a summary that captures the point of the review without merely pulling out a sentence or phrase. We perform this check on our entire dataset and find that of our 394,967 rows, there are only 26,931 in which the exact summary appears in the review.

```
# Plot this as a count plot
sns.countplot(x = 'Summary_in_Text?', data = df)
```

```
<AxesSubplot:xlabel='Summary_in_Text?', ylabel='count'>
```

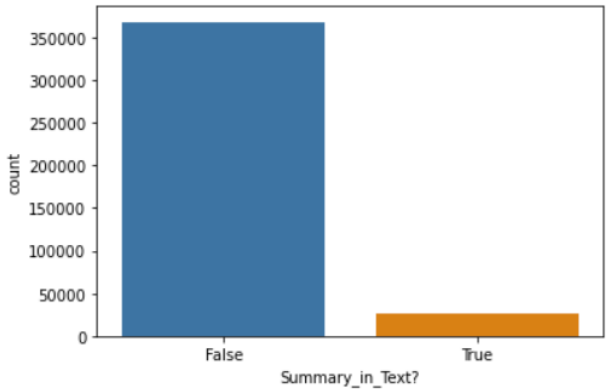

Figure 11: Is the Summary Text Contained in the Review?

5.5 Examining the Most Common Words

First, we look at the text column and created a word cloud consisting of the 50 most common words, where the size is representative of a word's frequency.

Figure 12: The 50 Most Common Words in the Text

We see that the majority of these are what are deemed to be stop words; very common English words that generally do not add much value to text. We will remove these stop words as part of the data pre-processing before we build our model. Since our summaries are so short, on average only four words, we will not remove stop words from them, allowing the model to produce a coherent sentence.

At this point we have reached the end of our exploratory data analysis and are now ready to detail the pre-processing of the data that we undertook before building and training our model.

6. Data Pre-Processing

In NLP projects there are a number of cleaning operations that must be performed on the text data before feeding it into a model. We did the following:

- Lowercased all text
- Removed special characters
- Expanded contractions
- Removed non-alphabetic characters
- Removed stop words (from review text only)
- Removed single character words
- Removed strings that had been made empty by the previous steps

At the end of this process, we were left with 394,618 rows. We found that our reviews contained 108,025 unique words, and our summaries contained 32,647. To assist our model, we add starting and ending tokens to our reviews. The first three rows of our cleaned and processed data appear below:

```
Review Text: bought several vitality canned dog food products found good quality product looks like stew processed meat smells
better labrador finicky appreciates product better
Summary Text: _START_ good quality dog food _END_


Review Text: product arrived labeled jumbo salted peanuts peanuts actually small sized unsalted sure error vendor intended repr
esent product jumbo
Summary Text: _START_ not as advertised _END_


Review Text: confection around centuries light pillowy citrus gelatin nuts case filberts cut tiny squares liberally coated powd
ered sugar tiny mouthful heaven chewy flavorful highly recommend yummy treat familiar story lewis lion witch wardrobe treat sed
uces edmund selling brother sisters witch
Summary Text: _START_ delight says it all _END_
```

Figure 13: First Three Reviews and Summaries After Pre-Processing

7. Model Building and Training

Now that our dataset has been cleaned and processed, we are almost ready to begin building and training our model. First, we must set some limitations on the data we will use and the summaries we are expecting our model to generate. We set the maximum review text length to be 50 words and the maximum summary length to be 10. Finally, we limit our dataset to be 100,000 randomly selected rows. Next, we split into training and testing sets (90% for training, 10% for testing). The full code for this section of the project is available at https://github.com/michael-molnar/text-summarizer/blob/main/Notebook2_Model_Building.ipynb.

For each of the review texts and summaries we fit tokenizers, transform the raw text into integer sequences using those tokenizers, and then pad the sequences so that all rows have the same length (50 for reviews, 10 for summaries). The model will require our vocabulary sizes from the tokenizers, and we find that the review vocabulary contains 40,080 words and the summary vocabulary 14,303 words.

Our task is a many-to-many sequence-to-sequence one (sequences of text in, sequences of text out) and consists of two stages: training and inference. We have used TensorFlow to build our models.

7.1 Training Phase

The architecture we are using is a LSTM (Long Short Term Memory) encoder-decoder [3].

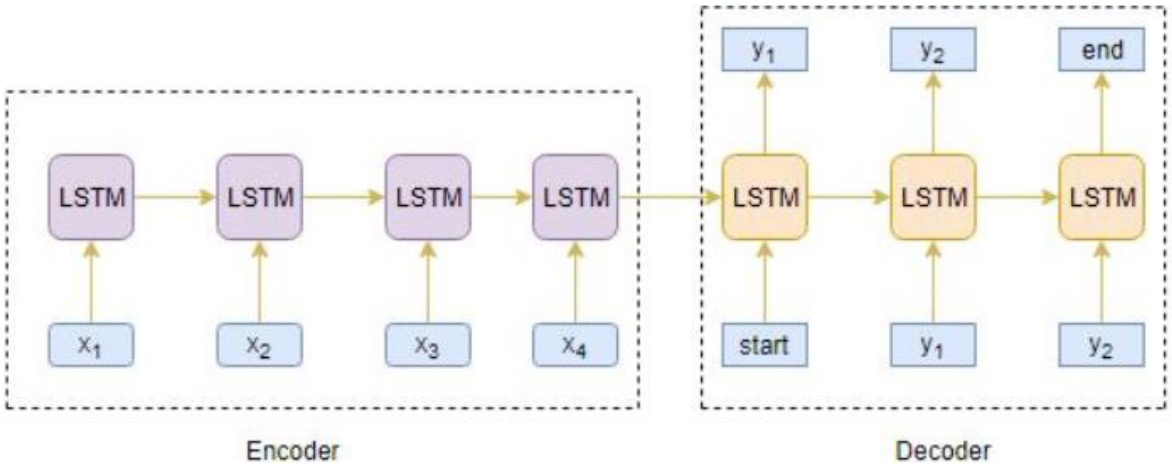

Figure 14: Encoder-Decoder Architecture

This type of model is commonly used in summarization or other tasks where the input and output sequences are of different lengths [6]. We build and compile our model.

```
model.summary()
```

```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 50)]         0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 50, 500)      20040000    input_1[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     [(None, 50, 500), (N 2002000     embedding[0][0]
                                                                 lstm[0][0]
                                                                 lstm[1][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 500)    7151500     input_2[0][0]
__________________________________________________________________________________________________
lstm_3 (LSTM)                   [(None, None, 500),  2002000     embedding_1[0][0]
                                                                 lstm[2][1]
                                                                 lstm[2][2]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 14303)  7165803     lstm_3[0][0]
==================================================================================================
Total params: 38,361,303
Trainable params: 38,361,303
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 15: Summary of Our Training Model

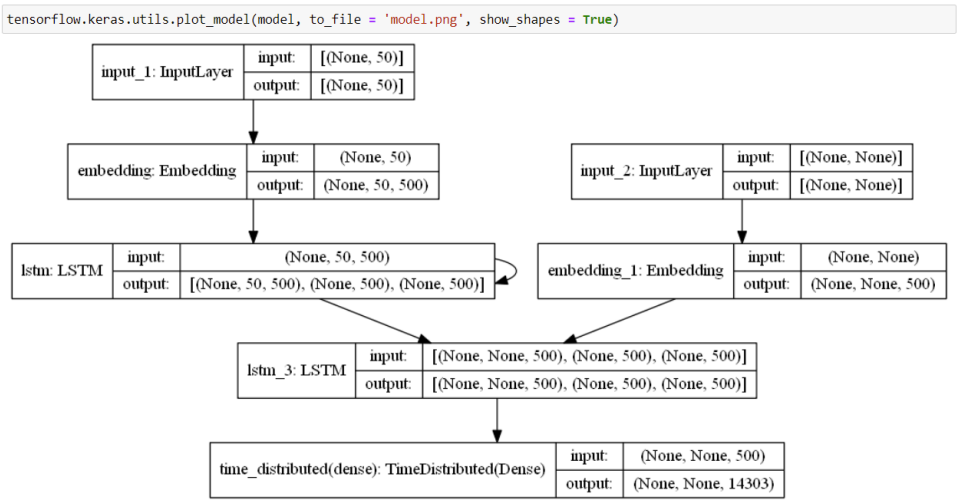

Figure 16: Plot of Our Training Model

We then train our model. It takes about four hours' time before hitting early stopping after thirteen epochs. The losses on the training and testing datasets are reported in the following plot.

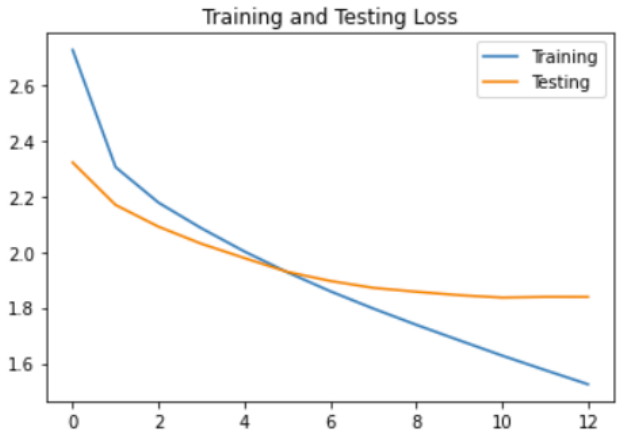

Figure 17: Model Loss (RMSprop)

7.2 Inference Phase

The inference phase, or where we actually produce the summaries, consists of two models: the encoder and the decoder.

```
tensorflow.keras.utils.plot_model(encoder_model, to_file = 'encoder_model.png', show_shapes = True)
```

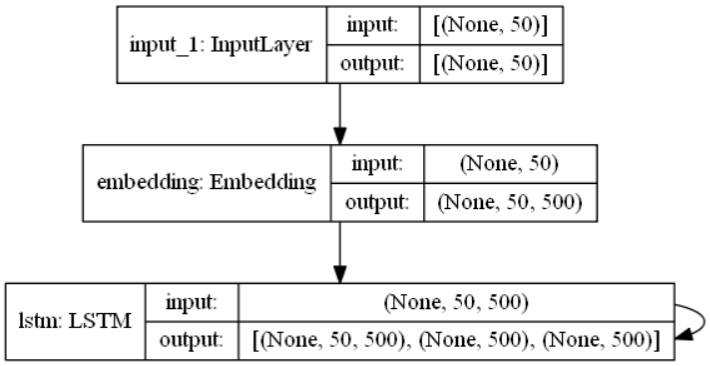

Figure 18: Encoder Model for Inference

```
tensorflow.keras.utils.plot_model(decoder_model, to_file = 'decoder_model.png', show_shapes = True)
```

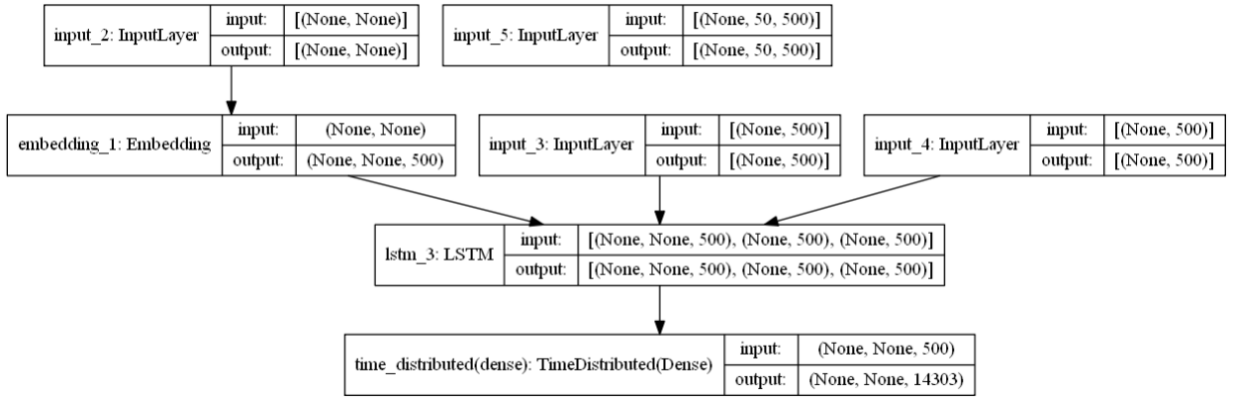

Figure 19: Decoder Model for Inference

We use these two models and our two tokenizers to compare our generated summaries to those given in the testing dataset. The first three results are shown here.

```
Review: glad find amazon looking store long time absolutely makes best cheeseball alternatives better nothing makes delish
Original summary: simply the best
Predicted summary:  best ever


Review: bought product review read said dissolved easily dissolve cold broth clumps heating mess cooked egg powder hard lumps
one nauseous chemo trying get nutrition satisfactory secret method make work packaging
Original summary: does not dissolve easily
Predicted summary:  not what expected


Review: two boxes tea bags really great tea white tea best tea
Original summary: best tea
Predicted summary:  great tea
```

Figure 20: Comparing Generated Summaries to Actual Summaries

With our models built, trained, and saved, we can use these to generate summaries of new, user-inputted text.

8. Producing Summaries

When a user enters text, we must first clean and process it the same way we did the training data. We then transform it into a sequence using the previously fit and saved tokenizer, and then pad it to our pre-set length. Next, we use our encoder and decoder models to generate an output sequence, for which we retrieve the words from our summary tokenizer. Please see https://github.com/michael-molnar/text-summarizer/blob/main/Notebook4_Text_Summarizer_Module.ipynb for the full code of this process. As an example, when given the text "I really like this pizza. It has so much cheese and it is really tasty!", the model produces the summary, "great taste". Due to the limitations on our training data and sequence lengths, we find that our model produces only short summaries, though they are novel and do well to capture the essence of the input text.

9. Flask Application

We have built a Flask application to perform the process outlined in the previous section. We first load our models and our tokenizers.

```
app = Flask(__name__)
# Load the encoder and decoder models
encoder_model = load_model('encoder_model.h5')
decoder_model = load_model('decoder_model.h5')
# Import the tokenizers
text_tokenizer = pickle.load(open('text_tokenizer.pkl', 'rb'))
summary_tokenizer = pickle.load(open('summary_tokenizer.pkl', 'rb'))
```

Figure 21: Loading Models and Tokenizers in Flask Application

We then define our text cleaning function.

```
def clean_text(text):
    """
    Performs all necessary cleaning operations on text input.
    """
    # Lowercase the text
    new_text = text.lower()
    # Remove special characters
    new_text = re.sub(r'\([^)]*\)', '', new_text)
    new_text = re.sub('"', '', new_text)
    # Expand contractions
    new_text = ' '.join([contraction_mappings[x] if x in contraction_mappings else x for x in new_text.split(' ')])
    # Remove 's
    new_text = re.sub(r"'s\b", '', new_text)
    # Replace non-alphabetic characters with a space
    new_text = re.sub('[^a-zA-Z]', ' ', new_text)
    # Split the text into tokens and remove stopwords
    tokens = [word for word in new_text.split() if word not in stopwords]
    # Keep only tokens that are longer than one letter long
    words = []
    for t in tokens:
        if len(t) > 1:
            words.append(t)
    # Return a rejoined string
    return (' '.join(words).strip())
```

Figure 22: Text Cleaning Function to Pre-Process Input Text

And our function to decode the generated summaries.

```
def decode_sequence(input_seq):
    """
    Take the processed input sequence and generate the summary using the
    econder and decoder models.
    """
    # Encode the input as state vectors.
    e_out, e_h, e_c = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))

    # Chose the 'start' word as the first word of the target sequence
    target_seq[0, 0] = target_word_index['start']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + [e_out, e_h, e_c])

        # Sample a token

        sampled_token_index = np.argmax(output_tokens[0, -1, :])

        sampled_token = reverse_target_word_index[sampled_token_index]

        if(sampled_token!='end'):
            decoded_sentence += ' '+sampled_token

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == 'end' or len(decoded_sentence.split()) >= (MAX_LEN_SUMMARY-1)):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        # Update internal states
        e_h, e_c = h, c

    return decoded_sentence
```

Figure 23: Function to Generate Summaries

We set our limits on the text and summary lengths and create mappings to use the tokenizers to retrieve the English words by their integer indexes.

```
# Set the maximum summary and text length
MAX_LEN_SUMMARY = 10
MAX_LEN_TEXT = 50

# Create mappings with the imported tokenizers
reverse_target_word_index = summary_tokenizer.index_word
reverse_source_word_index = text_tokenizer.index_word
target_word_index = summary_tokenizer.word_index
```

Figure 24: Loading Models and Tokenizers in Flask Application

Finally, we define our application functions.

```python
@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    # Take user input text
    input_text = request.form['your_text']
    # Clean the text
    review = clean_text(input_text)
    # Condense to the maximum length of 50 tokens
    condensed_review = review.split()[:MAX_LEN_TEXT]
    # Join into a string
    condensed_review = ' '.join(condensed_review)
    # Tokenize the review
    tokenized_review = text_tokenizer.texts_to_sequences([condensed_review])
    # Pad the sequence
    pad_tokenized_review = pad_sequences(tokenized_review, maxlen = MAX_LEN_TEXT, padding = 'post')
    # Generate the summary
    summary = decode_sequence(pad_tokenized_review.reshape(1, MAX_LEN_TEXT))

    # Show the output
    pred_text0 = 'Your Text:'
    pred_text1 = 'Summary:'

    return render_template('index.html', your_text_header = pred_text0, your_text = input_text,
      summary_text = pred_text1, prediction_text = summary)

if __name__ == '__main__':
    app.run(debug=True)
```

Figure 25: Flask Application

We use HTML and CSS to style and format our application.

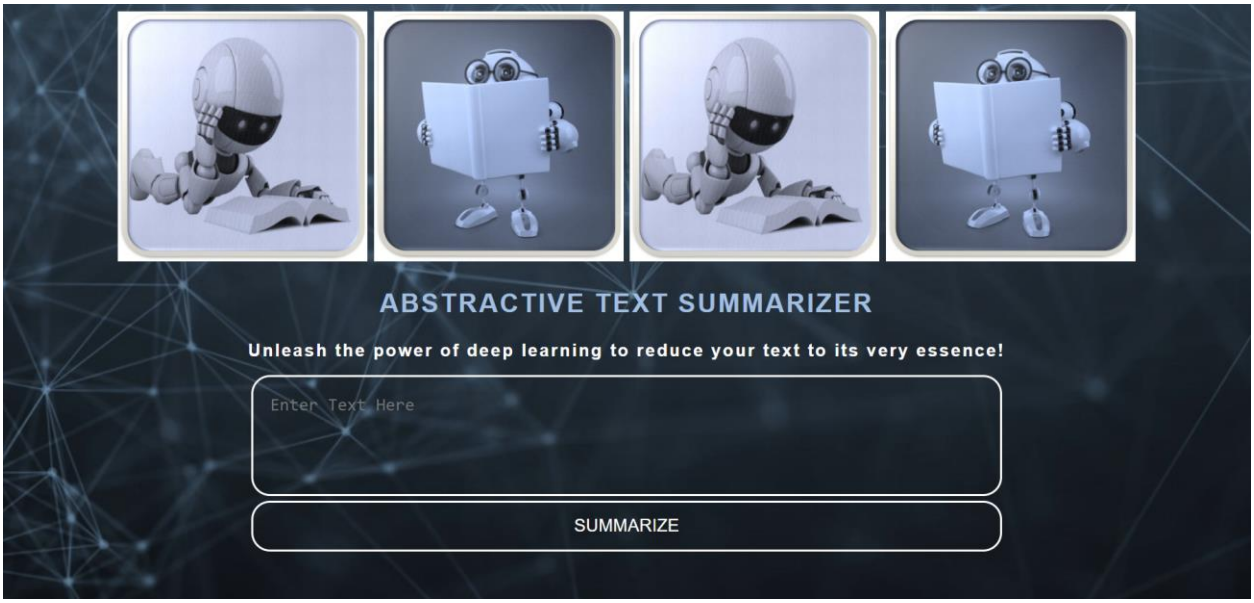

Figure 25: Our Styled Flask Application

We can provide text into the box and the application will use our models to generate a summary for it.

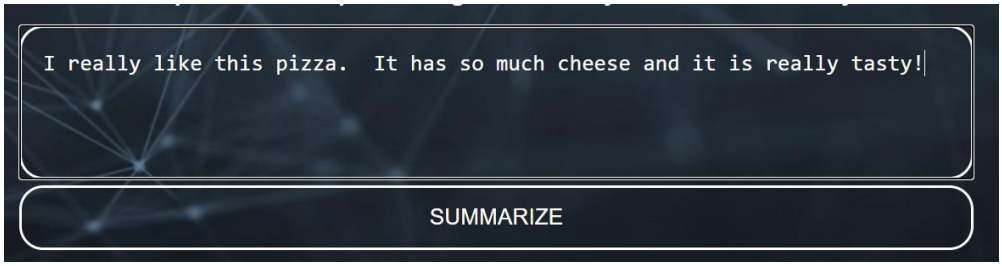

Figure 26: Inputting Text into Our Application

Figure 27: Summary Generated by Our Model and Application

10. Dockerization

Before deployment we use Docker to containerize our application.

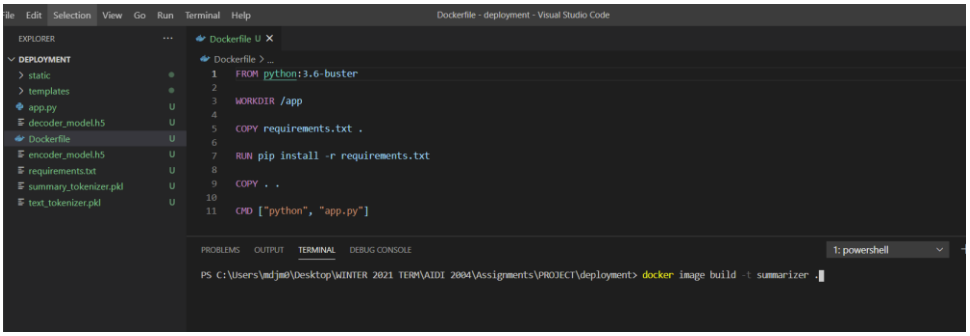

Figure 28: Creating the Dockerfile and Building the Image

We forward the port and run our container.

```
PS C:\Users\mdjm0\Desktop\WINTER 2021 TERM\AIDI 2004\Assignments\PROJECT\deployment> docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
summarizer   latest    e61c5b0267f7   2 minutes ago   1.92GB
PS C:\Users\mdjm0\Desktop\WINTER 2021 TERM\AIDI 2004\Assignments\PROJECT\deployment> docker run -p 5000:5000 -d summarizer
dbe92cffb808c4552ec364512c3fbc9df717041e352ccb6d21f4c1a2d4ad1fae
PS C:\Users\mdjm0\Desktop\WINTER 2021 TERM\AIDI 2004\Assignments\PROJECT\deployment>
```

Figure 29: Running the Container

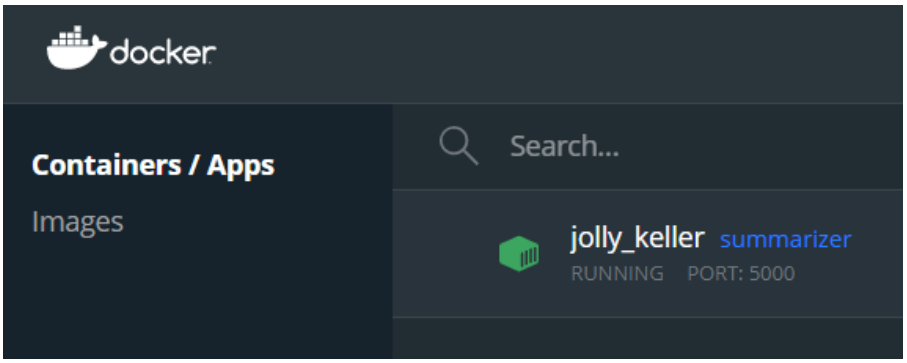

Figure 30: The Running Docker Container

And then we make user of Docker-Compose.

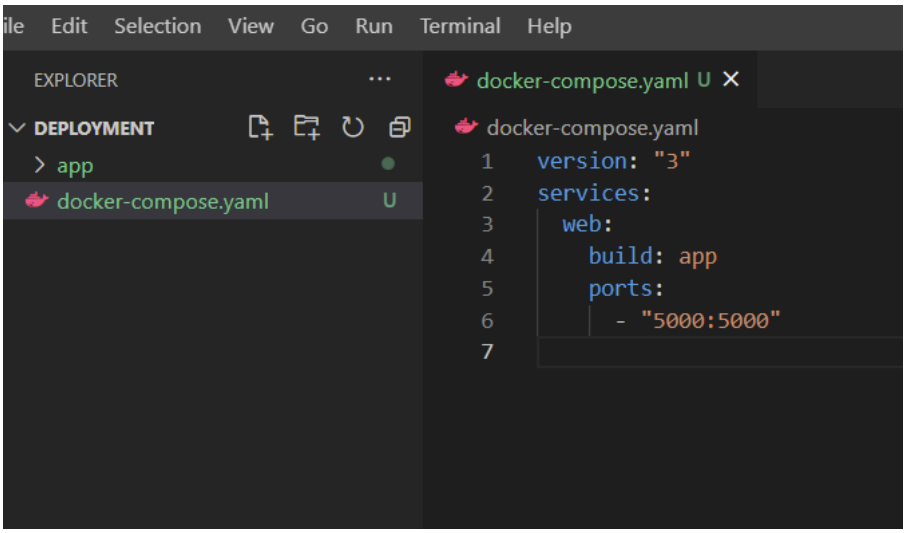

Figure 31: Docker-compose.yaml

And we can see our running containers after using Docker-Compose.

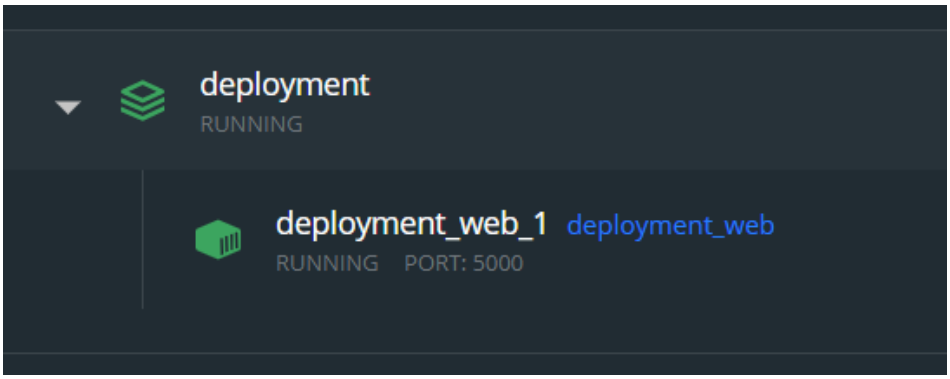

Figure 32: The Running Docker Containers

11. Deployment

To deploy our application, we used Heroku. Unfortunately, our model makes use of TensorFlow, which is a very large library. For this reason, upon deployment our application would not load before the request would timeout.

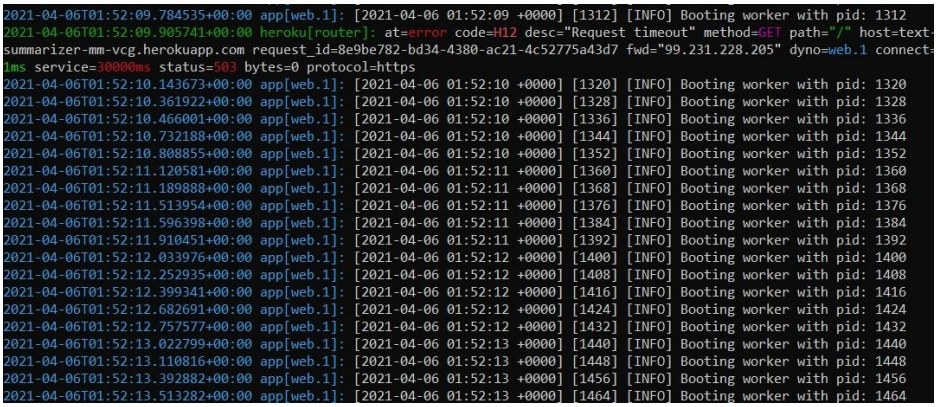

Figure 33: Heroku App Logs

We looked into alternative solutions – paying for additional resources on Heroku and Elastic Beanstalk on Amazon Web Services – but we unable to rectify this. As such, for the purposes of this project we chose to remove our model and deploy our application without it.

Our application is deployed now at https://summarizer-mm-vcg.herokuapp.com/ , though it does not function there as a machine learning solution.

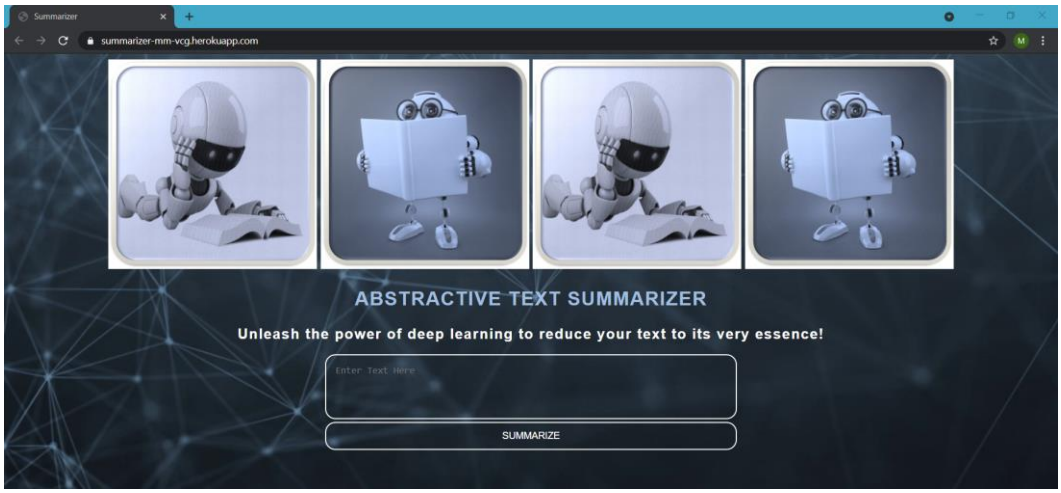

Figure 34: App Deployed to Heroku

For the demonstration and presentation portion of this project we will show our functioning application on our local machine.

12. Conclusions

Although our deployed application does not function as we would have hoped – it would if we invested in additional resources – we have still produced an end-to-end machine learning project that accomplished the goals we had set out to achieve.

We have made use of many technologies and techniques taught to us in this course: Git, GitHub, Flask, Docker, Docker-Compose, and Heroku. We have built a deep learning model that will produce summaries of user text and have a stylish application that makes use of it.

13. References

[1] Marr B. What Is Unstructured Data And Why Is It So Important To Businesses? An Easy Explanation For Anyone. Forbes. https://www.forbes.com/sites/bernardmarr/2019/10/16/what-is-unstructured-data-and-why-is-it-so-important-to-businesses-an-easy-explanation-for-anyone/?sh=1c81b41715f6. Published October 16, 2019. Accessed April 6, 2021.

[2] Garbade MJ. A Quick Introduction to Text Summarization in Machine Learning. Towards Data Science. https://towardsdatascience.com/a-quick-introduction-to-text-summarization-in-machine-learning-3d27ccf18a9f. Published September 18, 2018. Accessed April 6, 2021.

[3] Pai A. Comprehensive Guide to Text Summarization using Deep Learning in Python. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/. Published June 10, 2019. Accessed April 6, 2021.

[4] Exploring Transfer Learning with T5: the Text-To-Text Transfer Transformer. Google AI Blog. https://ai.googleblog.com/2020/02/exploring-transfer-learning-with-t5.html. Published February 24, 2020. Accessed April 6, 2021.

[5] Leskovec J. Web data: Amazon Fine Foods reviews. Stanford University. https://snap.stanford.edu/data/web-FineFoods.html. Accessed April 6, 2021.

[6] Brownlee J. How to Develop an Encoder-Decoder Model for Sequence-to-Sequence Prediction in Keras. Machine Learning Mastery. https://machinelearningmastery.com/develop-encoder-decoder-model-sequence-sequence-prediction-keras/. Published November 2, 2017. Accessed April 6, 2021.