Video-to-Text Summarization Model

Wilfred Okajevo, Oleksandr Pometun, Benedikt Jung

31.03.2023

1 Context and Motivation

As students of the course "ANNs with Tensorflow," we are excited to introduce our project: a self-built video-to-text summarization model. This model uses a pre-trained speech-to-text model and a self-built text summarization model to automatically generate summaries of videos.

The motivation for this project comes from the growing need for efficient and accurate video summarization. With the increasing amount of video content available online, it has become more and more difficult and time-consuming for us to find and watch videos that are relevant to our interests. By automatically generating summaries of videos, our model can help users quickly understand the content of a video and decide whether it is worth watching. In addition, our model has the potential to be useful in a variety of other applications. For example, it could be used to generate summaries of news videos or educational lectures, making it easier for users to stay informed and learn new information.

To build our model, we decided to use a pre-trained speech-to-text model to transcribe the audio from the videos. This allows us to leverage the power of existing speech-to-text models and focus our efforts on building an effective text summarizer. Building the text summarization model from scratch was a challenging but rewarding process. We had to carefully select and preprocess our training data, design an appropriate architecture for the model, and fine-tune its parameters to achieve good performance.

Overall, while our video-to-text summarization model did not turn out as brilliant as we had hoped, the experience of working on this project was incredibly valuable. We learned a lot about the challenges and complexities of building a machine learning model from scratch and gained valuable skills and knowledge that we can apply to future projects.

2 Related Literature

To reach our goal of building a video-to-text summarization model, we looked for inspiring and helpful papers and scientific works. At the beginning of our project, we first intended to build the speech-to-text model on our own while relying on pre-trained models for the summarization part. Therefore, we focused on this topic when we picked the papers for the project. The idea behind this was that it is easier to build a speech-to-text model rather than a summarization model.

From the first paper, "Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques" by Lindasalwa Muda, Mumtaj Begam, and Irraivan Elamvazuthi [6], we expected that it could be helpful in building a model that can recognize and transcribe speech from the video. Specifically because MFCC is a common feature extraction technique used in speech recognition systems and DTW is a technique used to align audio signals.
Our second picked paper, "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations" by Alexei Baevski, Henry Zhou, Abdelrahman Mohamed and Michael Auli [3], presents a framework for learning speech representations from raw audio data in a self-supervised manner. This seemed to be useful in building a model that can extract meaningful information from the audio track of the video.
To address the summarization aspect of the problem, where we wanted to fine-tune a pre-trained model, we picked as third paper "Fine-tune BERT for Extractive Summarization" by Yang Liu [5], as we expected it to be useful in building a model that can generate summaries of the transcribed text.

After a consolidation with our tutor we switched this approach, which in hindsight was definitely the better option. So now that we use the pre-trained model for the speech-to-text part our task was to build a text summarization model on our own.

Since we gained sufficiently helpful knowledge from the 'wav2vec 2.0' paper, we decided to keep it and in addition find two more papers that will help us build a text summarization model.
We decided to work with the paper "Glove: Global vectors for word representation" by Jeffrey Pennington, Richard Socher, and Christopher D. Manning[7], which presents a method for learning word representations using a co-occurrence matrix to capture the meaning of words. This, we expected, could be useful in building a model that can understand the content of the transcribed text and generate meaningful summaries.
The second newly picked paper is the famous "Attention is all you need" by Ashish Vaswani et al[8]. It presents the Transformer architecture that generally improved the field of natural language processing. We expected that attention within our text summarization model will support our project well.

3 The Methods

To solve the problem of creating a summary of the video we first converted the video into audio format, then with the help of Speech recognition model create transcript of the audio. Finally, using self-made Summarization model we made a summary of the transcript. Speech recognition can be done with limited labeled data.

Our first model has two parts: a Feature encoder and a Transformer network. The feature encoder consists of seven blocks, each with convolutional layers of different kernel widths and strides. Input waveform is processed by series of convolutional layers to capture local temporal patterns in the waveform. The output of the convolutional layers is then passed through a non-linear activation function and a layer normalization step. This results in a sequence of fixed-dimensional feature vectors that encode the high-level representation of the input waveform. In general, the feature encoder takes raw audio as input and outputs latent compressed speech representations (without irrelevant information). These representations are then forwarded to the Transformer network to build contextualized representations. The output of the feature encoder is discretized and used as targets in the self-supervised learning objective [3].

For summarization part Bidirectional LSTMs with Attention was used. Bidirectional LSTM is an architecture that processes input sequence in both directions(forward and backward) simultaneously. This was achieved by two separate LSTMs, one of them processed the input sequence from the beginning to the end and the other in the opposite direction. Finally, the output was created by concatenation of both of outputs. This gave us an opportunity to capture both past and future contexts. As a result, model could capture long range dependencies. Attention was used to prevent the lost of valuable information during the encoding process with the help of bidirectional LSTM and generating summary with the help of decoder(by creating context vector).

To train the model WikiHow dataset was used as it contains wide range of every day topic that might fit needs of our vocabulary. The model leverages pre-trained GloVe embeddings, which provide an advantage in capturing semantic information from the text. The embedding matrix is created to map word indices to their corresponding GloVe vectors, and this matrix is used to initialize the Keras Embedding layers for both the encoder and decoder.

4 The implementation

4.1 Dataset Selection

The conception that the video summarisation project should aim at being able to summarise videos or audio of everyday people talking led us to search for

datasets that would be representative of this level of demographic diversity. The WikiHow dataset at the time was as a matter of fact the best fit as other forms of datasets found were strongly inclined to specific domains like News, documents, and datasets from scientific papers.

WikiHow was a more converging text database of everyday people and this sparked our interest as it would be a great fit for the summarisation project at hand. However, during the process of building and training our model on the dataset, it was somewhat not sufficient. More on this later.

4.2 Implementation Overview

I. Video to Audio
II. Audio to Text
III. Text to Summarization

4.3 Phase 1

As discussed in the Methods of this report, the project is strategically in two facets; video to text (via the audio channel), essentially a Speech to Text problem, and a summarization model built from scratch with TensorFlow.

The pre trained wav2vec2 was used by us, which is TensorFlow equivalent of PyTorch[facebook/wav2vec2-base-960h] [1] [4], which was fine-tuned on 960h of LibriSpeech dataset for Automatic Speech Recognition. The pre-trained and finetuned model we used accepts only sequences with a length of 246000 as TF SavedModelexpects a fixed shape input during inference. Due to that, we had to split our sequence on smaller sequences and feed them separately.

We built a custom script to handle almost all common forms of continuous modalities like Flac, MP4, WAV, WEBrip e.t.c and pass it through the pre-trained model. The python script [2] on the GitHub handles everything under the hood and installs all dependencies when it is called automatically. Importing the script and running the below command prompts you to upload your media for speech-to-text conversion.

```
# your_script.py

from vid_speech_to_text import SpeechToText

stt = SpeechToText()
transcript = stt.process()
print(transcript)
```

4.4 Phase 2: Text Summarisation using Sequence to Sequence LSTM Architecture with Attention and GloVe Embedding

4.4.1 Cleaning and Tokenization

Clean data is the true strength of any advanced and useful neural network pipeline. The best-designed architecture would be useless if fed an inconsistent and dirty dataset. Since basically, the seq2seq model would ingest vector representations of words, it is by far most essential to have the data in its cleanest form possible.

This phase comprises a more sophisticated cleaning regime. As a matter of fact, we had 2 cleaning stages for our WikiHow Dataset. The first stage was a more holistic approach to getting a pseudo-usable CSV file of "Articles" and "Summaries" pairs. There is a script dedicated to this on our GitHub repo (WikiHow Data Processing Script.py).

Our next cleaning and processing stage handled more cleaning criteria and tokenization. Below is a pseudocode for the cleaning, tokenization, padding, and sequencing stage up to getting inputs and targets that are feedable to our neural network. Refer to the notebook in the Repo to get a more in-depth look.

```
function preprocess_data(articles, summaries):
    # Text cleaning
    for each article in articles:
        cleaned_article = clean_text(article)

    for each summary in summaries:
        cleaned_summary = clean_text(summary)
        add_start_end_tokens(cleaned_summary)

    # Tokenization
    article_tokenizer = create_tokenizer(articles)
    summary_tokenizer = create_tokenizer(summaries)

    # Convert text to sequences
    article_sequences = text_to_sequences(articles, article_tokenizer)
    summary_sequences = text_to_sequences(summaries, summary_tokenizer)

    # Padding and truncation
    padded_article_sequences = pad_sequences(article_sequences, max_len_article)
    padded_summary_sequences = pad_sequences(summary_sequences, max_len_summary)

    # Create datasets
    train_dataset = create_dataset(padded_article_sequences, padded_summary_sequences, batch_size)

    return train_dataset, article_tokenizer, summary_tokenizer
```

Here is an explanation of the cleaning and tokenization phase employed in this project more stretchly, feel free to jump to the Model Architecture stage if the pseudocode above was sufficient.

Text cleaning: a. The text is first cleaned to remove unwanted characters, such as HTML tags, special characters, and extra spaces. This is done using a built clean-text function, which employs regular expressions to remove these characters. b. The summaries are preprocessed to add 'oostartoo' and 'ooendoo' tokens to the beginning and end, respectively. These tokens help the model learn the start and end of summaries during training.

Tokenization: a. The cleaned text is tokenized using Keras' Tokenizer class. This process involves converting the text into a list of words (tokens), which can be further converted into numerical indices. b. Two separate tokenizers are created for the articles and summaries. They are fit on the training data to build the vocabulary and word-to-index mapping. We also tried the SentencePiece implementation capable of representing words to prevent Out-Of-Vocabulary issues. However, we didn't keep this as it was longer to train and computationally expensive.

Padding and truncation: a. To ensure that all input and target sequences have the same length, padding, and truncation are performed. This is done using Keras' pad$_sequences function. b. The input articles are padded or truncated to a fixed length ($max_len_article$), and th

$This effect also means that no matter how long a text input is, it would always be truncated to the Max_Len, which is str$

$Creating datasets : a. The preprocessed inputs and targets are then converted into TensorFlow datasets, which are$

4.5 Model Architecture: 3 BiDirectional LSTMS With Attention Mechanism

If we were to use a pre-trained model and finetune, like BERTSUM which we initially considered, this project wouldn't have been so much fun and we would be throwing away the "implementation" in 'IANNwT', however, knowing the difficulty, we jump heads on to using a bidirectional LSTM with Attention.

Bidirectional LSTMs with Attention were chosen for this implementation due to their ability to effectively capture context and dependencies in the input data while addressing some limitations of traditional LSTMs. They enhance the model's understanding of the input data by processing it in both forward and backward directions, capturing complex patterns and long-range dependencies, and this was basically why we chose them. The attention mechanism addresses information loss in the encoder-decoder architecture by creating a dynamic context vector and allowing the model to focus on the most relevant information. This results in more 'relatively' accurate and coherent summaries of when no Attention was used. Furthermore, the attention mechanism adds interpretability to the model, providing insights into its decision-making process. Overall, this combination leads to 'relatively' improved performance in our text summarization tasks using the wikiHow dataset.

Below is the pseudocode of our architecture, inference, and helper functions for decoding and summarising new texts.

4.5.1 Function glove-embeddings:

Load GloVe embeddings from file
Create an embedding matrix from GloVe embeddings and word-to-index dictionary
Return Keras Embedding layer initialized with the embedding matrix

4.5.2 Function seq2summary:

Convert target sequence to a summary string using reverse$_target_word_index$

4.5.3 Function seq2text:

Convert input sequence to a text string using reverse$_source_word_index$

4.5.4 Creating Datasets

Convert preprocessed inputs and targets into TensorFlow datasets
Batch datasets with a specified batch size

4.5.5 Model Architecture

Define encoder with GloVe embeddings and three bidirectional LSTM layers
Define decoder with GloVe embeddings, an LSTM layer, an attention layer, and a time-distributed dense layer
Compile Model with RMSPROP (Tried Adam too)

4.5.6 Training and Validation

Set up TensorBoard for monitoring and visualization
Use early stopping to prevent overfitting
Train the model using the model.fit function with training and validation datasets

4.5.7 Inference

Define separate encoder and decoder models for inference
Define the decode$_sequencefunctiontogeneratesummariesfornewtextinputs$

4.5.8 Summarize new text inputs using the trained model

For each new input text:
Call the decode sequence function to generate

4.5.9 Performed Rogue Evaluation

Aside from these, more helper functions for specific ideas were implemented alongside for improvement.

4.6 Choices Of HyperParameters and Other Decisions

We tried the most reasonable architectural design. Single LSTMS wasn't as good as the bidirectional and we also tried Adam and RMSPROP. RMSPROP performed better for the sparse categorical task we were implementing. To save time after multiple runs and trials, the Bayesian Optimisation method was used to pick the most suitable hyperparameters, which were close to our initial trials.

The Bayesian Optimisation process involves building a probabilistic model of the objective function and using it to find the optimal hyperparameters with a limited number of function evaluations.

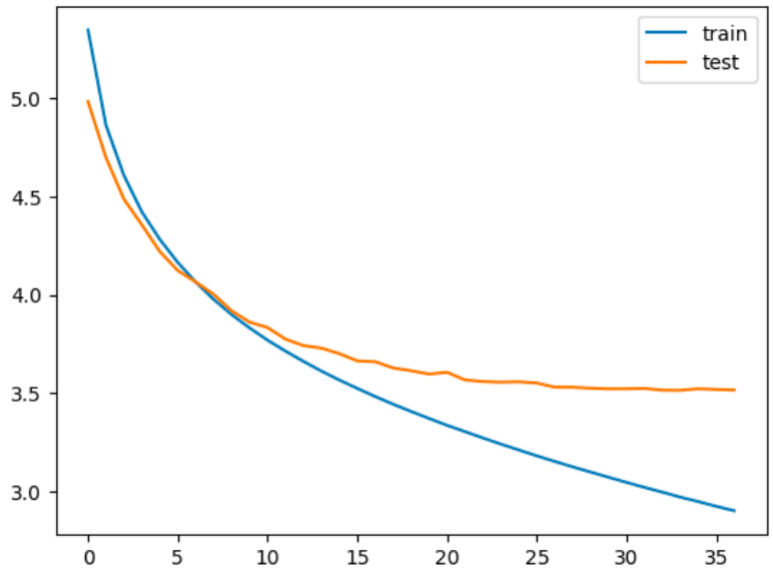
latent dim = 226
dropout rate = 0.31
batch size = 457

To stop the model from because it has learned the noise in the training data instead of the underlying patterns, Earlystopping regularization that tracked the Validation loss was implemented. It had a patience value of 3.

5 Results and Discussion

With the level of the wikiHow dataset, and its quite short summary of a paragraph, it is difficult to have a great true sequence pair due to its holistic abstractive summaries(Since most summaries were quite super short for what they were summarising, we even had to drop a handful of rows), the model didn't perform gloriously as expected, however, it is a good testament to how important datasets and tasks to be accomplished go hand in hand. However, for this level of the dataset, and the pair abstractiveness, the model performs quite well and the optimization tactics employed to attain this were also the silver lining to having something usable since the model generates 'relatively' good summaries most of the time.

Below is a graph showing the validation loss across epochs. More qualitative and interactive graphing and histograms are available on the notebook Tensorboard.



```
Rouge1 {'r': 0.16834986402486338, 'p': 0.2374098713786218, 'f': 0.1827395936593267}
Rouge2 {'r': 0.03902860722610713, 'p': 0.0564747427572427, 'f': 0.04249051920756799}
Rouge3 {'r': 0.16255651098901047, 'p': 0.22796884351759353, 'f': 0.17607513019804338}
```

Figure 1: The ROUGE evaluation score of the model performance

ROUGE scores typically comprise three components: precision, recall, and F1 score. Precision represents the fraction of N-grams (or LCS) present in both the generated and reference summaries, while recall signifies the fraction of N-grams (or LCS) in the reference summary that also appears in the generated summary. The F1 score, which is the harmonic mean of precision and recall, offers a balanced single-number evaluation of these two metrics.

The Rouge results are ok considering the level of abstraction the wikiHow dataset is and that the summarisation style isn't as detailed as we would have hoped to create a much higher level of sequence mapping. The problem of OOV is one thing too, but like we've said, solvable with a different approach, one of which is doing the tokenization with sentence pieces and specifying a much larger vocabulary size. Relatively, this took very long to train and got terminated.

There is still the option of running the generated summaries of input text through a neural network that gives it more contextual meaning and helps with coreference. This is outside the scope of this present project and we visibly see this is worth doing or building something similar from the ground up to further cement the neural network learning journey.

The last limitation of our model is the maximum length of text you can pass to it before it is truncated. Since the 95 percentile of the 230,000 summaries and articles were around 16 and 150 words respectively and so the input to the model would always be truncated outside this length. One way to combat this was to feed the model inputs in "chunks". The code implementation is shown in the notebook.

This is an implementation to run the generated summary chunk through an external neural network. NeuralCoref uses a neural network to learn representations of text that can be used to identify coreferences. It is built on top of the spaCy library, which provides the underlying natural language processing capabilities. NeuralCoref can be used to identify coreferences in a wide variety of text genres, including news articles, scientific papers, and social media posts and also some function to further process the outputs if needed

6 References

References

[1] URL: https://huggingface.co/facebook/wav2vec2-base-960h.

[2] URL: https://github.com/prokajevo/IANNwT-Final-Project-Video-Summarization/blob/main/vid_speech_to_text.py.

[3] Alexei Baevski et al. "wav2vec 2.0: A framework for self-supervised learning of speech representations". In: *Advances in neural information processing systems* 33 (2020), pp. 12449–12460.

[4] Vasudev Gupta. URL: https://tfhub.dev/vasudevgupta7/wav2vec2-960h/1.

[5] Yang Liu. "Fine-tune BERT for extractive summarization". In: *arXiv preprint arXiv:1903.10318* (2019).

[6] Lindasalwa Muda, Mumtaj Begam, and Irraivan Elamvazuthi. "Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques". In: *arXiv preprint arXiv:1003.4083* (2010).

[7] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[8] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

We are fine with the project and report being used as example for further iterations of the course.