Automatic Subtitle Generation from Movie Dialogues

Adrian Mincu
mincu.adrian13@gmail.com

Ciprian Huțanu
eusebiuciprian.hutanu03@gmail.com

Abstract

In this paper, we present a comprehensive pipeline for subtitle generation from movies, combining data preprocessing, custom dataset creation, and deep learning-based speech recognition models. We construct a custom dataset by developing a custom data pipeline that extracts and aligns audio-dialogue pairs from full-length movies.

Using this dataset, we infer a DeepSpeech2 [1] model from scratch. Additionally, we fine-tune the pre-trained Wav2vec2 (Baevski et al., 2020) model on our dataset. Our results highlight the importance of domain-specific training and custom data curation in enhancing automatic subtitle generation performance.

1 Introduction

Manual subtitle generation is time-consuming and prone to errors, negatively affecting the viewing experience, especially for those who rely on subtitles for comprehension.

As passionate movie enthusiasts, we've often faced the frustration of missing subtitles for our favorite films, which inspired us to automate subtitle generation.

We believe that automating subtitle generation improves accessibility, reliability, and the movie-watching experience.

2 Related Work

Recent advancements in Automatic Speech Recognition (ASR) have been driven by the development of large-scale, transformer-based models. Two notable contributions in this space are OpenAI's Whisper and Meta's Wav2Vec2.

OpenAI's Whisper model, introduced in 2022, represents the current state of the art on several benchmarks, including the LRS2 dataset. [2]

- OpenAI's Whisper (Radford et al., 2022)
- Meta's Wav2Vec2 (Baevski et al., 2020)

3 Approach

3.1 Dataset

Our dataset construction pipeline is designed to extract audio-text pairs from full-length movies, specifically those directed by Quentin Tarantino. The process involves the following stages:

1. **Scraping Movie Torrents:** We target the YTS.mx website, a popular source for movie torrents. Only movies directed by Quentin Tarantino are selected to maintain thematic consistency.

2. **Downloading Movies:** Torrents are programmatically added to the download queue using the `qBittorrent` client via its Web UI and the `qbittorrentapi` Python wrapper.

3. **Audio and Subtitle Extraction:**
   - Audio is extracted from `.mkv` movie files using `ffmpeg` and converted into standard format `.wav`.
   - Subtitles are also extracted using `ffmpeg`, with a focus on English *normal* subtitle tracks (excluding SDH and forced types).

4. **Subtitle Cleaning:** Extracted subtitle text undergoes preprocessing:
   - HTML tags and punctuation are removed (excluding apostrophes)
   - Dialogue hyphens are stripped
   - Whitespace is normalized
   - Text is lowercased and segmented into sentences
   - Non-textual symbols (e.g., music notes) are discarded

[1] Nvidia's DeepSpeech2 Model
[2] Papers With Code Automatic Speech Recognition on LRS2

5. **Audio Segmenting:** Using subtitle timestamps, we extract continuous speech segments of approximately 30 seconds. Silent or non-dialogue sections are discarded to focus on high-quality, spoken content aligned with the cleaned subtitles.

6. **Filtering:**
    - Text segments with fewer than 15 words are removed
    - Audio clips shorter than 10 seconds are discarded

This results in a dataset of synchronized audio and textual segments, well-suited for training models in speech recognition, alignment, or multimodal learning tasks.

3.2 Preprocessing

To prepare our raw data for training, we implemented distinct preprocessing pipelines for both the audio and text (subtitle) modalities. Each step was motivated by a specific need: to reduce noise, standardize inputs, and improve the model's ability to learn meaningful representations.

3.2.1 Audio Preprocessing

The audio pipeline was tailored for speech recognition tasks, with a focus on consistency, noise reduction, and efficient representation.

- **Segment Standardization:** Only clips with sufficient dialogue were retained. All audio samples were trimmed or zero-padded to a fixed length of **30 seconds**.
- **Downmixing:** Multi-channel (e.g., stereo) audio was converted to mono by averaging the channels, reducing input complexity.
- **Normalization:** Amplitude values were scaled to the range `[-1, 1]` to prevent numerical instability and promote training convergence.
- **Resampling:** Audio was downsampled to **16 kHz**, a rate sufficient for human speech per the Nyquist-Shannon Sampling Theorem [3], while also reducing data size.
- **Voice Frequency Filtering:** A band-pass filter was applied to isolate the 85–3000 Hz range, which captures most human speech content while discarding irrelevant frequency components.
- **Wiener Filtering:** We applied a Wiener filter to reduce background noise without severely distorting the speech signal, preserving the intelligibility of dialogue.
- **Log-Mel Spectrogram:** Final inputs were transformed into **log-Mel spectrograms**, which are perceptually meaningful representations of audio and well-suited for deep learning models. This transformation compresses dynamic range and mimics human auditory perception, especially in the low-frequency domain.

3.2.2 Subtitle Preprocessing

Subtitles were converted from raw text to numerical input suitable for transformer-based language models.

- **Cleaning:** Initial steps removed HTML tags, punctuation (except apostrophes), and special symbols. Dialogue hyphens were stripped, whitespace normalized, and the text was lowercased and segmented into sentences.
- **Tokenization:** Cleaned text was passed through the `BertProcessor`, which performed subword tokenization (WordPiece) and converted tokens into integer IDs. These IDs ranged from **0 to 30,522**, the full vocabulary size of BERT. While expressive, this large range requires the model's final classification layer to output probabilities over more than 30,000 possible tokens, significantly increasing computational cost.
- **Character-Based Tokenization (Alternative):** To reduce output layer size and improve efficiency, we experimented with **character-level tokenization** using `Wav2Vec2`. This approach uses a much smaller vocabulary of just **32 tokens**, including the 26 English letters, a space symbol (denoted by `|`), and several special symbols such as `<pad>` and `<unk>`. This compact vocabulary dramatically simplifies the model's final layer and reduces memory and computational requirements.

[3] The Nyquist-Shannon Sampling Theorem

3.2.3 Rationale and Impact

Each preprocessing choice was guided by a balance of empirical evidence and practical constraints:

- Standardizing segment length and amplitude facilitated batch processing and accelerated training convergence.
- Filtering and noise reduction improved speech clarity, which directly benefits both alignment and recognition accuracy.
- The log-Mel representation condensed and normalized feature distributions, reducing model sensitivity to volume variation.
- Text cleaning and tokenization ensured that textual inputs were consistent, clean, and compatible with language modeling architectures.
- Switching from BERT's 30,522-token vocabulary to a 32-token character set via Wav2Vec2 allowed for a significantly smaller and more efficient final output layer, enabling faster training and inference without severely impacting expressiveness for speech-to-text tasks.

Together, these preprocessing methods significantly improved model stability, training speed, and performance across both the speech and text domains.

3.3 Method

The goal of this project is to apply automatic speech recognition (ASR) to movie subtitles using modern pre-trained models and a model implemented from scratch. This section outlines the reasoning behind choosing the models, the architecture used, the experimental setup, and the evaluation of results against existing state-of-the-art (SOTA) benchmarks.

3.3.1 Wav2Vec2

Why Wav2Vec2? We selected **Wav2Vec2** due to its state-of-the-art performance in low-resource ASR tasks. It leverages unsupervised pretraining on large audio corpora, which allows it to capture rich speech representations. This makes it well-suited for the task of transcribing noisy, real-world movie dialogue, which often includes overlapping speech, background noise, and poor audio quality. The primary advantage of Wav2Vec2 is its ability to learn robust features from raw audio, which makes it effective for noisy environments, typical of movie sets.

How Does Wav2Vec2 Work? Wav2Vec2's architecture comprises three main components:

- **Feature Encoder:** A series of convolutional layers that map raw audio into continuous latent representations.
- **Contextualized Representations:** Transformer layers process the output of the feature encoder, adding temporal context and modeling long-range dependencies in speech.
- **Quantization Module:** For self-supervised pretraining, the model discretizes the encoder outputs into a finite vocabulary of speech units. During fine-tuning, the model uses these discretized representations for speech-to-text alignment, employing Connectionist Temporal Classification (CTC) loss to match speech units with transcription.

Hyperparameter Choices

- **Learning Rate (1e-7):** We chose a very low learning rate to avoid overshooting the optimal weights during fine-tuning, ensuring stability throughout the training process.
- **Batch Size (2):** This was limited due to GPU memory constraints. Larger batch sizes would result in out-of-memory errors, so this compromise was necessary.
- **Gradient Clipping (max_grad_norm=0.05):** To prevent exploding gradients, we clipped the gradients during backpropagation.
- **Epochs (80):** We selected 80 epochs to give the model sufficient time to converge while monitoring for signs of overfitting.

The model used for training is the pretrained `Wav2Vec2` base model from Hugging Face, fine-tuned on our specific task using the movie subtitle data.

3.3.2 DeepSpeech2 (Implemented from Scratch)

Why DeepSpeech2? We also implemented **DeepSpeech2** from scratch to explore the impact of a completely different architecture for ASR.

How Does DeepSpeech2 Work? DeepSpeech2's architecture consists of:

- **Convolutional Layers:** These layers are used for feature extraction, learning high-level speech representations from raw audio.
- **Recurrent Layers:** Long Short-Term Memory (LSTM) layers capture temporal dependencies in speech sequences.
- **Fully Connected Layers:** These layers are responsible for final speech-to-text conversion using softmax output, often coupled with CTC loss.

In our implementation, we reduced the model size by **halving the number of parameters** in each layer compared to the standard DeepSpeech2 architecture. This was necessary to accommodate computational constraints while still retaining core architectural principles.

Pretraining and Dataset Before infering on our target domain, the model was pretrained on the **LibriSpeech 100-hour clean subset**. This provided a strong foundation for learning general speech patterns, despite the smaller model size.

Hyperparameter Choices Similar to Wav2Vec2, we chose the following hyperparameters for DeepSpeech2:

- **Learning Rate (1e-4):** The learning rate is set higher than in Wav2Vec2 to promote learning, not fine-tuning.
- **Batch Size (4):** A slightly higher batch size was used, given the model's lower memory usage compared to Wav2Vec2.
- **Epochs (20):** DeepSpeech2 was trained for 20 epochs due to its high computation cost.
- **LSTM Layers (3):** The network is composed of three LSTM layers, as used in typical implementations of DeepSpeech2.

We will be comparing the performance of this model with that of Wav2Vec2 using similar datasets.

3.3.3 Experiments

Comparing with State-of-the-Art (SOTA) Our approach was evaluated against the **SOTA** for ASR in noisy and diverse acoustic conditions. Wav2Vec2, which has been shown to work well in a variety of environments, serves as our baseline. The goal of our experiment was to assess how well both models handle real-world challenges such as:

- Background noise (e.g., music, crowd sounds),
- Overlapping speech from multiple speakers,
- Variability in audio quality.

The average **Word Error Rate (WER)** and **Character Error Rate (CER)** across test segments were computed to quantify model performance for both models.

Results The model's results on our test set are compared to a baseline in the following table: 1

Training Loss Training loss and validation loss are shown below, demonstrating that both losses decreased together, indicating effective learning for both models:

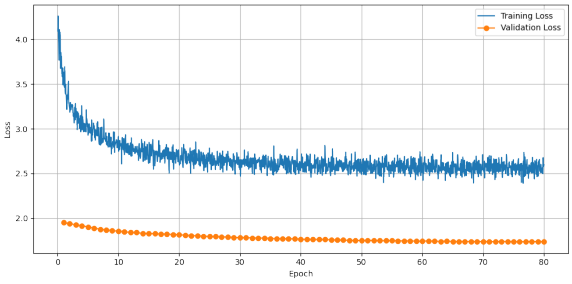

Figure 1: Training vs Validation Loss for Wav2Vec2.

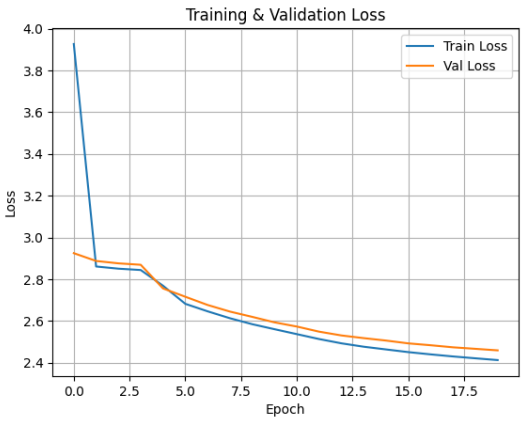

Figure 2: Training vs Validation Loss for DeepSpeech2.

3.3.4 Conclusion

In conclusion, our experiments with **Wav2Vec2** and **DeepSpeech2** demonstrate their strengths in transcribing noisy movie dialogues.

Segment	Wav2Vec2 WER	DeepSpeech2 WER	Wav2Vec2 CER	DeepSpeech2 CER
000200.wav	0.4285	**1.3846**	0.2867	**0.8602**
000184.wav	0.4000	**1.0000**	0.2134	**0.8314**
000047.wav	0.6626	**1.2394**	0.4644	**0.8341**
000096.wav	0.7215	**1.0000**	0.6066	**0.8319**
000001.wav	0.5588	**1.1129**	0.2849	**0.8337**
Average	**0.5543**	**1.1474**	**0.3713**	**0.8383**

Table 1: WER and CER results for Wav2Vec2 and DeepSpeech2 on test segments.

While **Wav2Vec2** demonstrated strong preliminary performance, **DeepSpeech2** underperformed, primarily due to the use of a smaller model trained on limited data, given its high computational requirements. This highlights the need for further optimization, particularly in challenging scenarios involving background noise or overlapping speech.

Our results reflect the challenges faced by ASR systems when dealing with real-world audio. Additionally, our findings suggest that fine-tuning the models for more specific acoustic environments could potentially improve performance.

4 Future Work

While our results show promising transcription accuracy, future work could focus on aligning transcriptions with audio to produce time-stamped subtitles (e.g., `.srt` files). This would enable automatic subtitle generation, real-time captioning, and searchable movie dialogue databases.

To achieve this, we could integrate tools like `aeneas`[4] for forced alignment or modify our models to predict timestamps directly.

5 Conclusion

This project taught us that building an end-to-end AI pipeline from scratch is no small feat, it's a bit like trying to edit a Tarantino movie: intense, full of challenges, and requiring a lot of patience.

But in the end, we learned that while AI pipelines can be tedious, it's all worth it, especially when you can binge-watch movies while working on them!

Limitations

Our system currently supports only English, as both the training data and pre-trained models (e.g., Wav2Vec2) are tailored to English. Extending to other languages would require multilingual datasets, language-specific tokenizers, and model fine-tuning.

While scalable, the system demands high-performance GPUs for training and fine-tuning, limiting its accessibility in low-resource environments. Future work could focus on optimizing for efficiency or exploring lightweight ASR models for better scalability.

Ethical Statement

Potential Misuse: The technology could be used to generate misleading subtitles for harmful or inappropriate content, spreading misinformation.

Bias: The models are biased towards English, as they were trained only on English data and films by Quentin Tarantino, limiting their performance on other accents, dialects, or languages.

Recommendations: Future work should diversify datasets to include a wider range of movies which contain a broader range of languages, accents, and cultural contexts to mitigate bias.

Acknowledgements

A special thanks to Quentin Tarantino for making such iconic films that not only redefine cinema but also provide us with endless hours of dialogue to transcribe. Without his remarkable ability to craft memorable lines, this project would have been much quieter, and much less entertaining.

Keep making movies, Quentin, we'll keep transcribing!

References

Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. *arXiv.org*.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2022. Robust speech recognition via large-scale weak supervision. *openai.com*.

[4] Aeneas GitHub Repository