Speech Recognition automatically by using Artificial Neural Network

Hamzah Asghar Farooqi

Faculty of Engineering Environment and Computing

Coventry University

farooqih@uni.coventry.ac.uk

Abstract

Paper evaluates the Automatic Speech Recognition (ASR) of neural network model built using a combination of a 2D Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Connectionist Temporal Classification (CTC) loss. ASR has become increasingly important in our daily lives. It involves converting spoken language into text, and it has numerous applications, ranging from virtual assistants to speech-to-text transcription services. One popular approach to ASR is using an artificial neural network, specifically the Connectionist Temporal Classification (CTC) architecture. CTC is a deep learning model that can learn to map variable-length input sequences to output labels without the need for an alignment between the input and output sequences. This approach has shown great promise in improving the accuracy of ASR systems, especially when dealing with noisy or accented speech. CTC-based ASR systems effectively applied to several tasks, such as phoneme recognition, word recognition, and end-to-end ASR systems. However, developing an effective CTC-based ASR system requires careful consideration of several factors, including the selection of the neural network architecture, the training data, the optimization strategy, and the language model used. Ongoing research is focused on addressing these challenges to improve the performance and robustness of CTC-based ASR systems, and it is expected that these efforts will lead to further advancements like the area of speech recognition.

***Key words**: neural network model, 2D Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), deep learning model, phoneme recognition*

1. Introduction

Automatic Speech Recognition is a critical tool of technology for converting spoken language into text. One popular approach to ASR is Connectionist Temporal Classification (CTC), which is a neural network architecture that can directly learn to map variable-length input sequences to output labels without the need for an alignment between input and output sequences. This approach has shown to be effective for various speech recognition tasks, including both phoneme and grapheme-level recognition, as well as end-to-end ASR systems. CTC-based ASR systems shows to reach competitive performance on standard benchmarks & have potential to simplify the ASR pipeline by eliminating the need for separate alignment and decoding steps. However, the performance of CTC-based ASR systems heavily varies the excellence and quantity of training data, the choice of network architecture and hyperparameters, and the optimization strategy. Ongoing research is focused on addressing these challenges to improve the accuracy and robustness of CTC-based ASR systems.

Many individuals across the globe have trouble in speaking fluently and accurately because of various conditions such as caresses, Parkinson's disease, amyotrophic side sclerosis (ALS), cerebral palsy or traumatic brain injury. On this problem, we propose applying an end-to-end NN- architecture, specifically the connectionist temporal classification to the convolutional NN (CTC-CNN), to assist this individual in communicating effectively. Speech recognition is an area of study that brings together several fields of knowledge, including CS, computational linguistics & computer engineering. Its objective is to create methodologies and technologies that allow computers to recognize and translate spoken language into written text. Alternative names for this domain include automatic speech recognition (ASR), computerized speech recognition, and speech-to-text (STT). To build systems that can effectively recognize speech, researchers draw upon knowledge and research from these different disciplines to create robust and accurate ASR systems. The code describes a demonstration where an automatic speech recognition (ASR) model is built using a combination of a 2D (CNN) (RNN), and Connectionist Temporal Classification (CTC) loss.

The CTC algorithm are used during train deep NNs in speech recognition & other sequence-based tasks, even when the alignment between input and output is unknown. In other words, it enables the model to recognize spoken words even when the timing of individual words in the audio is not explicitly known. The ASR model that is produced resembles DeepSpeech2, which is a well-known open-source ASR model, and it is trained on the LJSpeech dataset obtained from the LibriVox initiative. This dataset comprises brief audio segments of a solitary speaker reading excerpts from factual literature. Overall, demo showcases application of deep learning techniques in building an ASR system and illustrates the use of the CTC algorithm in training the model. The code outlines a process for evaluating the ASR model's quality, which involves utilizing a metric known as Word Error Rate (WER). This metric gauges the model's precision by assessing the number of replacements, insertions, and omissions that arise in a string of recognized words versus the authentic quantity of spoken words. For example, if the model recognized "the quick brown fox" as "the quick blue fox," there would be one substitution error. Similarly, if the model recognized "the quick fox" instead of "the quick brown fox," there would be one deletion error. The total number of errors (substitutions, insertions, and deletions) is then divided by the total number of words spoken to obtain the WER score. To calculate the WER score, the jiwer package must be installed, which provides a convenient way to implement the WER calculation. The WER score is an important metric for evaluating the accuracy of the ASR model and helps to identify areas for improvement in the system.

2. Related Work

a) *Speech Recognition via CTC-CNN Model*

In modern society, communication has become more frequent, and speech is essential for social interactions as it enables people to convey their emotions and ideas. Nevertheless, many people struggle to communicate because of language barriers. To cater to the needs of those who are speech-impaired or deaf-mute, this research suggests a methodology. By merging a voice database with a deep neural network, a rudimentary acoustic model for deep neural networks (DNN) was created. This model was then employed with a sound sensor to transcribe collected voice signals into text or their corresponding voice signals, thus enhancing communication. The proposed approach exhibits potential for diverse applications, including creating word-for-word transcriptions of meeting minutes, medical reports, automotive documents, sales records, and more. The findings of this investigation highlight the effectiveness of the suggested methodology and its importance in contemporary artificial intelligence technology.

b) *Voice Recognition using Artificial Neural Network and Gaussian Mixture Models*

The efficacy of recognition systems in precisely identifying speakers by their speech waveform distribution hinges on the degree of optimization achieved through model parameter training. The objective of this investigation is to identify specific speakers by analyzing their continuous speech waveform distribution through the application of a hybrid artificial neural network (ANN) and statistical Gaussian mixture model (GMM) approach. Discriminative classification and training were accomplished using a feed-forward multilayer ANN architecture containing 30 hidden neurons, while GMM scores were utilized to match speech features. To identify recognized speakers based on the goodness of speech feature frame matches from the ANN and GMM frameworks, the decision system employs correlation coefficient analysis. To evaluate the system's effectiveness, experiments were performed using speech utterances from 30 distinct speakers (comprising 20 males and 10 females). The outcomes demonstrated an average recognition rate of 77% for 5-word utterances and 43% for 20-word utterances for trained speech. When dealing with unknown utterances, the system achieved a recognition rate of 18% for 20-word utterances.

c) Fully Neural Network Based Speech Recognition on Mobile and Embedded Devices

ASR is on mobile & embedded devices has piqued interest for several years. This study proposes the implementation using RNN based acoustic models, RNN based language models, and beam-search decoding. The model is trained end-to-end by using the CTC loss. Nevertheless, on embedded devices of RNN can result in excessive DRAM entrances since neural network boundary size exceeds the store memory size & parameters is only used once. For addressing of this problem, the multi-time step technique of parallelization is employed, which computes multiple output samples concurrently using parameters obtained from DRAM, thereby minimizing DRAM accesses and speed of high attaining processing. Regrettably, conventional RNNs, for example (LSTM) or (GRU), are not support step of multi-time parallelization. To enable multi-time step parallelization, we create acoustic model using merging depth-wise 1-dimensional convolution layers with simple recurrent units (SRUs). We also create both role and models of word piece for acoustic modeling & employ parallel RNN-based linguistic styles for BSD. Despite an overall model size of approximately 15MB, our method achieves competitive Word Error Rates (WER) for the WSJ corpus, and real-time speed is achieved using only a single-core ARM without any GPU or specialized hardware.

3. Dataset

The LJ Speech dataset is a popular dataset used for Automatic Speech Recognition (ASR) tasks. It contains over 13,000 hours of speech spoken by a single speaker, extracted from audiobooks available on the LibriVox website. The speaker is a male with an American accent, and the recordings are in English.

The dataset is open for all free of cost and can be downloaded from the following link: https://www.kaggle.com/datasets/mathurinache/the-lj-speech-dataset

Also, from

https://keithito.com/LJ-Speech-Dataset/

The high-quality recordings and abundance of data in the LJ Speech dataset have made it a go-to benchmark for speech recognition tasks, particularly for end-to-end models that rely on Connectionist Temporal Classification or attention mechanisms. Its popularity can be attributed to its ability to support the training of deep neural networks. The dataset is typically preprocessed by converting the audio files into spectrograms, which are then used as input to the neural network models. Different models can be trained on this dataset, such as (CNNs), (RNNs), & their combinations, with the goal of achieving high accuracy in recognizing spoken words or phrases.

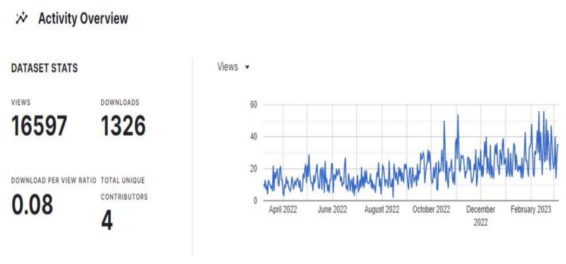


Figure 1 Activity Overview of Dataset taken from Kaggle.

From figure 1, here we can saw how many times this data set is used, their view how much time researchers contribute in from this dataset. We can Also use this dataset in Automatic speech recognition (ASR) using deep neural network for example (CNNs), (RNNs), (LSTM) networks, Transformer models, Speech synthesis using Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), WaveNet models and Speaker identification.

4. Methods

There is different method to solve but we explain those steps that we followed at first, we collect and prepare the dataset. We will need to obtain audio recordings of speech and their corresponding transcriptions in text format. We will also need to preprocess the audio data, such as by converting it to spectrograms or mel-spectrograms, which are commonly used in speech processing.

Then we design your neural network: There are several architectures we can use for ASR, such as convolutional neural networks, (RNNs), and transformers. We will also need to use the CTC loss function, which is a technique for training neural networks on unsegmented sequences, such as speech.

We used CTC neural network model CTC (Connectionist Temporal Classification) is a type of neural network architecture used in automatic speech recognition (ASR) tasks. The main purpose of CTC is to align the input sequence with the output sequence, which is particularly useful in tasks where the input and output sequences have different lengths. To address this, we have created an approach for speech emotion recognition (SER) that involves a (BLSTM) neural network with the attention mechanism, combined with a (CTC) objective function.

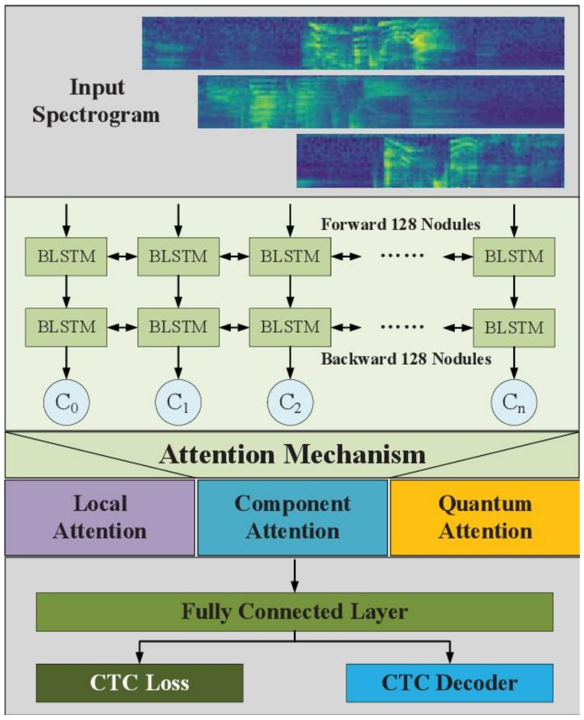


Figure 2: Framework of proposed model taken from semantic scholar.

The proposed model framework, as shown in Figure 1, involves feeding spectrograms into a BLSTM layer. After that, one of the three attention mechanisms is applied, and CTC is used to align labels to input frames that are emotionally significant.

The CTC solves the problem and how they solve we will show pictorially.

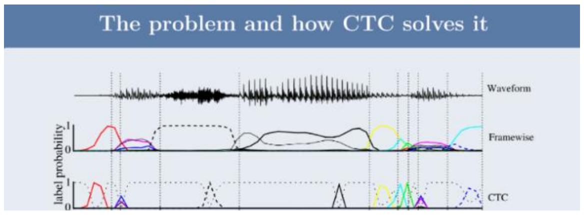
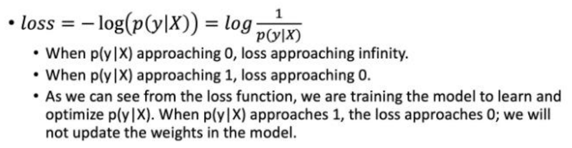


Figure 3: Output of classic framework, figure taken from semantic scholar.

Figure 3 shows the outputs of traditional frame-based phoneme classification and recurrent neural networks (RNN) trained with connectionist temporal classification (CTC) are compared. In CTC, the output sequence is produced by a softmax function used to output of network at each time step. The output sequence is then aligned with the input sequence using an algorithm that considers the probability of each output symbol at each time step. This alignment allows for the correct transcription of speech with variable-length input sequences. CTC is often applied to datasets where the input sequences are audio files, and the output sequences are transcriptions of the spoken words. LJ Speech dataset, for example, comprises the audio files of one loudspeaker reading sentences from the LJ Speech Corpus, along with the corresponding text transcriptions.

In CTC, the output sequence is produced by a softmax function used to output of network at each time step. The output sequence is then aligned with the input sequence using an algorithm that considers the probability of each output symbol at each time step. This alignment allows for the correct transcription of speech with variable-length input sequences. CTC is often applied to datasets where the input sequences are audio files, and the output sequences are transcriptions of the spoken words. LJ Speech dataset, for example, comprises the audio files of one loudspeaker reading sentences from the LJ Speech Corpus, along with the corresponding text transcriptions.

In the first step, feasible alignments for a label are generated based on the input spectrogram. Next, the conditional probabilities for each alignment are determined, followed by the elimination of alignment information to obtain only p(y | X) through marginalization. This involves summing up the probabilities of all possible alignments, i.e.,SUM_ALL_POSSIBLE_ALIGNMENT(p(y, alignment1|X), p(y, alignment2|X), p(y, alignment3|X)). The CTC process involves finding the p(y | X) for all possible alignments. Ultimately, the speech recognition model assigns the optimal label (optimized path) to all types of variations in the input spectrogram.



- $loss = -\log(p(y|X)) = log\frac{1}{p(y|X)}$
  - When p(y|X) approaching 0, loss approaching infinity.
  - When p(y|X) approaching 1, loss approaching 0.
  - As we can see from the loss function, we are training the model to learn and optimize p(y|X). When p(y|X) approaches 1, the loss approaches 0; we will not update the weights in the model.

Figure 4: CTC loss function, pic taken from medium.com

By using CTC, the neural network can accurately transcribe the spoken words in the audio files, even when there are variations in pronunciation, timing, or other factors that may affect the length and content of the input sequence. CTC is a neural network architecture commonly used for speech recognition tasks. CTC is particularly useful for transcribing speech data with variable-length inputs and outputs, in which the orientation among input & output sequence is unknown. CTC maps the input sequence directly to the output sequence without any intermediate steps.

The main reason for selecting CTC for speech recognition is that it can handle variable-length inputs and outputs. In speech recognition, the length of the input (audio signal) and output (transcription) can vary, making it challenging to use traditional neural network architectures. CTC tackles this problem by using an alignment-free approach and allows the model to predict the output sequence directly from the input sequence without requiring the exact alignment.

We Train our neural network once we have our dataset and neural network architecture, we can begin training our model. We will need to experiment with hyperparameters such as learning rate, batch size, and number of epochs to achieve the best results.

Then we Evaluate our model After training the model, we will need to evaluate its performance on a test set. We can use metrics for example word error rate & (CER) to measure accuracy of our model's transcriptions.

In terms of parameters, there are several hyperparameters that can be tuned to optimize execution of the CTC model. There are a few of the important hyperparameters include:

1. Number of layers: Total of layers in NN can impact model's capability to learn complex relationships between the input and output sequences.

2. Hidden units: Total hidden in each unit of layer determines capacity of model to capture patterns in the data.

3.Learning rate: Step by step learning rate determines on each iteration of the optimization algorithm and can affect the speed and accuracy of the model.

4.Dropout rate: During training, the regularization technique known as Dropout randomly drops out neurons to prevent overfitting. The proportion of neurons to drop out is controlled by the dropout rate.

5.Batch size: The optimization algorithm's iteration uses a certain number of samples, which is determined by the batch size, and can influence the stability and speed of the training process.

To Improve the model If our model isn't performing well, we can try several techniques to improve its accuracy. For example, can try the following techniques:

1. Increase length of training data: Collect more data to train the neural network, as more data will develop the performance and model’s accuracy.

2. Preprocess the data: Clean and preprocess the data to remove noise and irrelevant information, which can improve the quality of our data and the performance of our model.

3. Try different neural network architectures: Experiment with different neural network architectures, for example deep NNs or CNN, to see which one works best for our problem.

4. Tune hyperparameters adjust the hyperparameters of our neural network, such as the learning rate, batch size, or several no of hidden layers, then to improve performance of model.

5. We Use regularization techniques such as dropout or L2 regularization can prevent overfitting, that will increase the model’s accuracy.

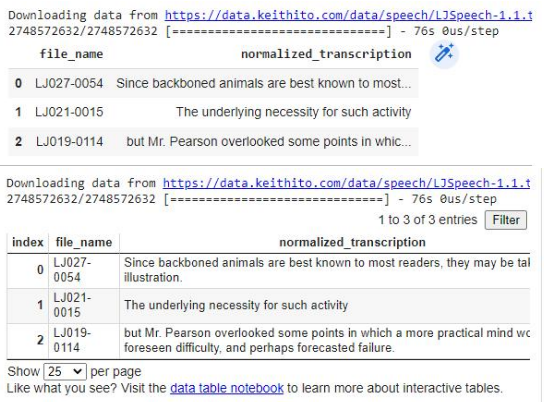
6. We use transfer learning, it is a technique where we will use a pre-trained neural network and fine-tune it on our specific problem, which can increase the performance the model’s accuracy.

7. Ensemble methods: We can use ensemble methods such as bagging or boosting to combine multiple models to improve the overall performance and accuracy of our neural network.

5. Experimental Result

Libraries: At first, we import several libraries and modules that are commonly used in machine learning and natural language processing applications. We used pip install jiwer is used to install the jiwer library, which is a package for calculating the Word Error Rate (WER) of two text strings. These libraries include pandas, numpy, tensorflow, keras, matplotlib, and jiwer. The code also imports specific functions from some of these libraries, such as wer from jiwer. The imported libraries and modules are used for data manipulation and analysis, building and training machine learning models, creating visualizations and plots, and calculating the Word Error Rate (WER) of two text strings.

Dataset: After importing the libraries code sets the variable data_url to a URL that contains a dataset of speech data called "LJSpeech-1.1" and then uses the keras.utils.get_file() function to download and extract the dataset to the local machine. The variable wavs_path is then set to the path where the audio files from the dataset are stored and metadata_path is set to the path where the metadata file is stored. The metadata file is read using pd.read_csv() function from the pandas library. The separator is set to |, which is used to separate the columns in the metadata file, and the header is set to None since the metadata file does not contain a header row. The quoting parameter is set to 3 to ignore any quotation marks in the file. The resulting DataFrame is assigned to the variable metadata_df, which contains the file name and normalized transcription of each audio file. The DataFrame is then shuffled using sample(frac=1) and reset the index. Finally, the first three rows of the DataFrame are displayed using head(3) as shown in figure 5.



```
Downloading data from https://data.keithito.com/data/speech/LJSpeech-1.1.t
2748572632/2748572632 [==============================] - 76s 0us/step
```

	file_name	normalized_transcription
0	LJ027-0054	Since backboned animals are best known to most...
1	LJ021-0015	The underlying necessity for such activity
2	LJ019-0114	but Mr. Pearson overlooked some points in whic...

```
Downloading data from https://data.keithito.com/data/speech/LJSpeech-1.1.t
2748572632/2748572632 [==============================] - 76s 0us/step
```

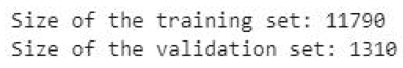
1 to 3 of 3 entries Filter

index	file_name	normalized_transcription
0	LJ027-0054	Since backboned animals are best known to most readers, they may be tal illustration.
1	LJ021-0015	The underlying necessity for such activity
2	LJ019-0114	but Mr. Pearson overlooked some points in which a more practical mind wc foreseen difficulty, and perhaps forecasted failure.

Show 25 per page
Like what you see? Visit the data table notebook to learn more about interactive tables.

Figure 5: Read file and parse.

S The initial step of the process involves dividing the dataset into training and validation sets. To achieve this, the metadata dataframe's total number of samples is utilized to calculate the variable split, which is 90%. Then, the first split number of rows from the metadata_df are assigned to the df_train dataframe. The len() function is then utilized to compute the number of samples in each of the newly created dataframes. Ultimately, this stage separates the dataset into two portions: a training set comprising 90% of the samples and a validation set consisting of the remaining 10%. To confirm that the split was executed accurately, the sizes of these sets are printed.



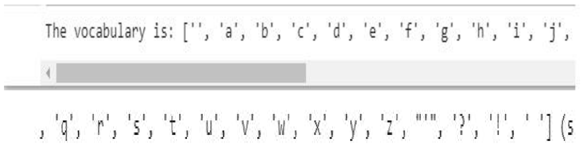
```
Size of the training set: 11790
Size of the validation set: 1310
```

Figure 6: The result of Training and Validation set.

As you shown in figure 6, the result of training and validation set that explain 90 % of training and 10% of validation set.

Accepted character in transcription: In this sets up a character set of accepted characters in the transcription, which is a list of lowercase letters from 'a' to 'z', and some special characters like apostrophe, question mark, and exclamation mark. The keras.layers.StringLookup function is then used to map the characters to integers. The vocabulary parameter takes the characters list as input, and the oov_token parameter is set to an empty string. This function returns a lookup table that maps characters to unique integers.

The num_to_char variable is also created using the StringLookup function. The vocabulary parameter takes the vocabulary of char_to_num and invert=True is used to indicate that we want to invert the mapping from integers to characters.



```
The vocabulary is: ['', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
```

```
, 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', "'", '?', '!', ' '] (s
```

Figure 7: The Vocabulary

Defines function (encode_single_sample): Then we define a function called encode_single_sample that takes in a file name and its transcription label. The function performs the following steps:

Reads and decodes the wave file from the specified path.

Squeezes and casts the audio tensor to float32 data type.

Computes the spectrogram of the audio using STFT.

Computes the magnitude of the spectrogram using tf.abs and applies a power transformation to the spectrogram.

To normalize the spectrogram, the mean of each frequency bin is subtracted, and the result is divided by the standard deviation.

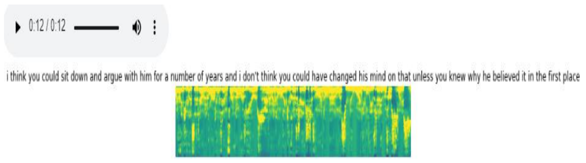
Converts the transcription label to lowercase and maps each character to its corresponding integer.

Returns a dictionary containing the spectrogram and the label as integers.

T Next, we establish the batch size and generate two datasets, one for training and another for validation, through the utilization of the tf.data.Dataset API. The from_tensor_slices method is employed to produce a dataset comprising the file names and their corresponding transcription labels. To execute the encode_single_sample function in parallel on each element in the dataset, the map method is applied, with the num_parallel_calls parameter set to tf.data.AUTOTUNE. To ensure that the dataset's spectrograms and labels are all of the same length, the resulting dataset is padded to the maximum length using the padded_batch method. Finally, the prefetch method is utilized to prefetch dataset elements in the background while the model is training, improving its performance.

Spectrogram and waveform: As shown in figure 8 and 9 we create a figure with two subplots, one for displaying a spectrogram and the other for displaying the waveform of an audio file. Then the loop iterates over the first batch of data in the train_dataset and extracts the spectrogram and label. The spectrogram tensor is converted to a numpy array and transposed to swap the time and frequency dimensions, and the label is converted to a string. Two subplots are created for displaying the spectrogram and waveform, respectively. The figure is displayed using the plt.show() function, which also plays the audio signal using the display.Audio function from IPython.display.

DeepSpeech2 model: We defines a DeepSpeech2 model for speech recognition. It contains two functions, CTCLoss and build_model. CTCLoss computes the CTC loss between predicted and true labels, while build_model constructs the DeepSpeech2 model. Incorporated in the model are two convolutional layers, a sequence of bidirectional GRU layers, and two dense layers. The final dense layer is activated by a softmax function. The Adam optimizer with a learning rate of 1e-4 and CTCLoss are utilized as the loss function for model compilation. Figure 10 displays the summary of the model, which is printed using the model.summary method.



i think you could sit down and argue with him for a number of years and i don't think you could have changed his mind on that unless you knew why he believed it in the first place

Figure 8: Spectrogram

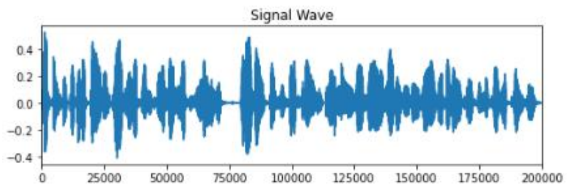
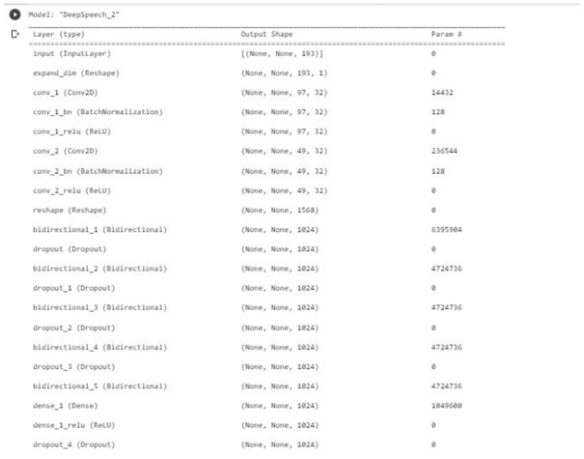


Figure 9: Single wave



```
Model: "DeepSpeech_2"
_________________________________________________________________________________
 Layer (type)                        Output Shape                  Param #
=================================================================================
 input (InputLayer)                  [(None, None, 193)]           0

 expand_dim (Reshape)                (None, None, 193, 1)          0

 conv_1 (Conv2D)                     (None, None, 97, 32)          14432

 conv_1_bn (BatchNormalization)      (None, None, 97, 32)          128

 conv_1_relu (ReLU)                  (None, None, 97, 32)          0

 conv_2 (Conv2D)                     (None, None, 49, 32)          236544

 conv_2_bn (BatchNormalization)      (None, None, 49, 32)          128

 conv_2_relu (ReLU)                  (None, None, 49, 32)          0

 reshape (Reshape)                   (None, None, 1568)            0

 bidirectional_1 (Bidirectional)     (None, None, 1024)            6395904

 dropout (Dropout)                   (None, None, 1024)            0

 bidirectional_2 (Bidirectional)     (None, None, 1024)            4724736

 dropout_1 (Dropout)                 (None, None, 1024)            0

 bidirectional_3 (Bidirectional)     (None, None, 1024)            4724736

 dropout_2 (Dropout)                 (None, None, 1024)            0

 bidirectional_4 (Bidirectional)     (None, None, 1024)            4724736

 dropout_3 (Dropout)                 (None, None, 1024)            0

 bidirectional_5 (Bidirectional)     (None, None, 1024)            4724736

 dense_1 (Dense)                     (None, None, 1024)            1049600

 dense_1_relu (ReLU)                 (None, None, 1024)            0

 dropout_4 (Dropout)                 (None, None, 1024)            0
```

Figure 10: DeepSearch2 Model

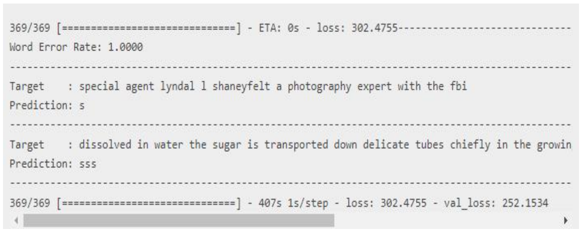
Table 1: Trainable and non-Trainable params

Total params	26,628,480
Trainable params	26,628,352
Non-trainable params	128

Here the table 1 shows the total number of params their trainable and non-trainable result on deepsearch2 model.

Utility and class back function: To defines a utility function and a callback class used to evaluate the performance of a speech recognition model during training. The decode_batch_predictions function takes the model's output tensor and returns the predicted transcription of the speech using a greedy search algorithm. The CallbackEval class is a subclass of the keras.callbacks.Callback class and outputs the transcriptions predicted by the model during training. During each epoch, the class predicts the transcriptions for each batch in the dataset using the model, calculates and outputs the word error rate (WER) between the predicted transcriptions and the true transcriptions in the batch, and finally outputs the predicted transcription and the true transcription for two randomly selected samples from the batch.

Training: The Keras Sequential model's fit method is employed to train a speech recognition model. The number of epochs is restricted to 1, signifying that the model will be trained for only one epoch. The CallbackEval class generates a validation_callback object that is then passed as a parameter to the fit method's callbacks argument. This callback object is executed at the end of each epoch to display the predicted transcriptions and the word error rate (WER) for the validation dataset.



```
369/369 [==============================] - ETA: 0s - loss: 302.4755------------------------------
Word Error Rate: 1.0000
----------------------------------------------------------------------------------------------
Target    : special agent lyndal l shaneyfelt a photography expert with the fbi
Prediction: s
----------------------------------------------------------------------------------------------
Target    : dissolved in water the sugar is transported down delicate tubes chiefly in the growin
Prediction: sss
----------------------------------------------------------------------------------------------
369/369 [==============================] - 407s 1s/step - loss: 302.4755 - val_loss: 252.1534
```

Figure 11: Training Speech Recognition model

The fit method is implemented with the training dataset as its primary argument and the validation dataset as its validation_data parameter. The number of epochs is assigned to 1, and the validation_callback object is specified as the callbacks parameter, which executes at the end of each epoch. The fit method returns a history object containing details about the training procedure, including the loss and accuracy values for each epoch, as seen in Figure 11 and Table 2.

Table 2: Training Process

Word Error Rate	Prediction 1	Prediction 2
1.0000	s	sss

And finally, evaluates the performance of the speech recognition model on more validation dataset. The model outputs the WER score and prints the true transcription and predicted transcription for 5 randomly selected samples from the validation dataset as shown in table 3

Table 3: Performance of SRM on 5 samples

Word Error Rate	1.0000
Prediction 1	ssnssss
Prediction 1	sr
Prediction 1	ssssss
Prediction 1	s
Prediction 1	sssrs

6. Discussions and future work

There are several avenues for future work on this speech recognition model using the CTC loss function. Some potential areas for improvement include:

1.Data augmentation: The model can be improved by using data point growth skills to increase the amount of training data. This can include techniques such as adding noise to the audio, changing the pitch or speed of the audio, and altering the background noise.

2.Transfer learning: Transfer learning can be improving the performing of_model. A pre-trained model can be used as a starting point and fine-tuned on the specific speech recognition task.

3. Language modeling: Incorporating language modeling techniques can help the model understand the context and improve its transcription accuracy. This may involve utilizing n-gram language models, recurrent neural networks for language modeling, or language models based on transformers.

4. Attention mechanism: To enhance the model's accuracy, attention mechanisms can be incorporated, enabling the model to focus on crucial elements of the audio. This can be achieved by integrating a self-attention mechanism or a multi-head attention mechanism into the model.

5. Model architecture: The model architecture can be modified to improve its performance. This can include adding additional layers, changing the size of the layers, or changing the type of layers used.

6. Hardware optimization: The model can be optimized for hardware such as GPUs or TPUs to improve training and inference speed. This can include techniques such as model parallelism or data parallelism.

7.Multilingual speech recognition: The model can be extended to support multilingual speech recognition. This can include training the model on multiple languages or building separate models for each language.

Overall, there is still a lot of room for improvement in speech recognition using the CTC loss function. By exploring these areas, the accuracy and performance of the model can be improved, making it more useful for a wider range of applications. The CTC algorithm has several advantages over traditional ASR models. First, it can handle variable-length input sequences, which is important for speech recognition tasks where the duration of the spoken words can vary. Second, it can produce output sequences of variable lengths, which is important for tasks where the number of spoken words can vary. Finally, it is an end-to-end model, which means that it can be trained directly on the acoustic features without the need for intermediate representations such as phonemes or words.

7. Reference

[1] Speech Recognition via CTC-CNN Model. Available: https://assets.researchsquare.com/files/rs-2226611/v1_covered.pdf?c=1668390627, [Accessed November 14th, 2022].

[2] Voice Recognition using Artificial Neural Network and Gaussian Mixture Models [Accessed May 2013].

[3] Automatic Speech Recognition using CT. Available: https://keras.io/examples/audio/ctc_asr/ [Accessed 29 11, 2021].

[4] Wen-Tsai Sung, Hao-Wei Kang, Sung-Jung Hsiao "Speech Recognition via CTC-CNN Model" *Research Article*, November 14th, 2022.

[5] Aaron Nichie, Godfrey a. Mills "Voice Recognition using Artificial Neural Network and Gaussian Mixture Models" *Research Article* [Accessed May 2013].

[6] Yoonho Boo, Wonyong Sung, Sungho Shin, Iksoo Choi, Jinhwan Park "Fully Neural Network Based Speech Recognition on Mobile and Embedded Devices" [Accessed, 2018].

[7] Graves, Alex & Fernández, Santiago & Gomez, Faustino & Schmidhuber, Jürgen. (2006). CTC: "Labelling unsegmented sequence data with recurrent neural 'networks [Accessed, 2006].

[8] Ziping Zhao, Zhongtian Bao, Zixing Zhang, Nicholas Cummins, Haishuai Wang, Björn W. Schuller "Attention-Enhanced Connectionist Temporal Classification for Discrete Speech Emotion Recognition" [Accessed, 2019].

[9] H. Suominen, T. Lehtikunnas, B. Back, H. Karsten, T. Salakoski, and S. Salanterä, "Theoretical considerations of ethics in text mining of nursing documents," *Stud. Health Technol. Inform.*, vol. 122, p. 359—364, 2006.

[10] Lobov, S., Krilova, N., Kastalskiy, I., Kazantsev, V. and Makarov, V., (2018). 'Latent factors limiting the performance of sEMG-interfaces'. Sensors [online], 18(4), p.1122. available from <https://www.mdpi.com/1424-8220/18/4/1122> [26 March 2019]

[11] Hanan Aldarmaki a, Asad Ullah b, Sreepratha Ram a, Nazar Zaki "Unsupervised Automatic Speech Recognition" [Accessed, 2022].

[12] Alex Graves, Santiago Fernández, Faustino Gomez, Jürgen Schmidhuber "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural 'networks"[Accessed, 2006]

[13] K. Khysru, J. Wei and J. Dang," Research on Tibetan speech recognition based on the Am-do Dialect," CMC-Computers, Materials & Continua, vol.73, no.3, pp. 4897-4907, 2022

[14] L. H. Juang and Y. H. Zhao, "Intelligent speech communication using double humanoid robots,"Intelligent Automation & Soft Computing, vol.26, no.2, pp. 291-301, 2020.

8. Appendix 1 SS

This code is not mine; I follow and implement some changes and follow the step and content more thoroughly and past research on this topic help me a lot to complete and learn the code**.**

Dataset

https://www.kaggle.com/datasets/mathurinache/the-lj-speech-dataset
Also, from
https://keithito.com/LJ-Speech-Dataset/

Code on GitHub

https://github.com/HamzahAsghar/Speech_Recognition_automatically_by_using_ANN_-.git

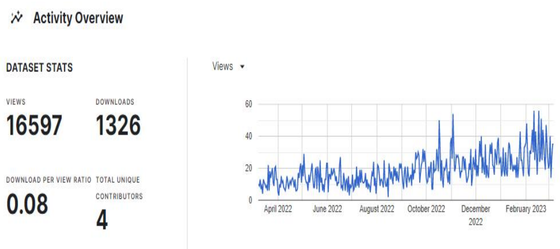
We Already use this photo in above experimental results.

Before Starting, we should take some necessary steps and then on that steps we implement the model.

A. Figure out ANN problem:

On step one we should find the problem that are related to ANN

B Selection of Dataset:



Activity Overview

DATASET STATS

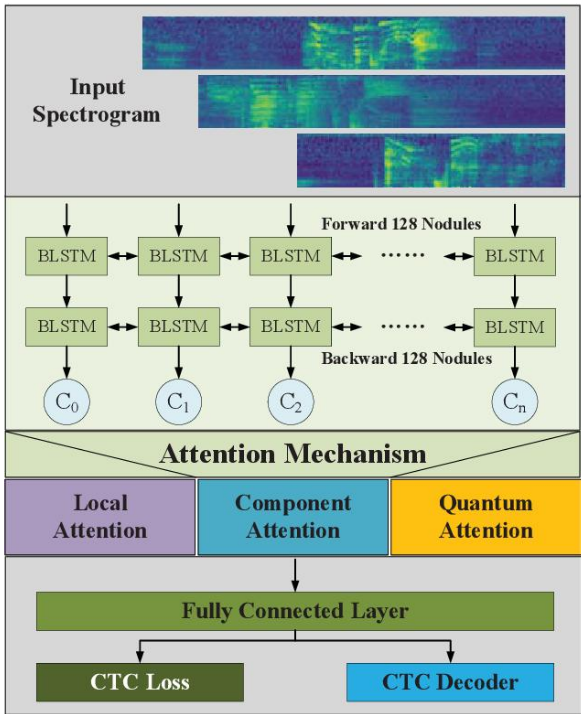
VIEWS	DOWNLOADS
16597	1326

DOWNLOAD PER VIEW RATIO	TOTAL UNIQUE CONTRIBUTORS
0.08	4

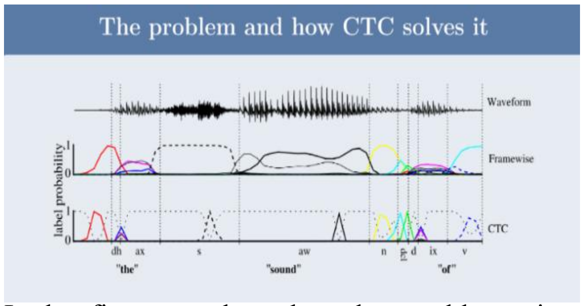
On the above figure we select the dataset

C Proposed Framework CTC:

In that step we proposed the CTC framework to obtain the result as shown in figure



D CTC Problem Solver:



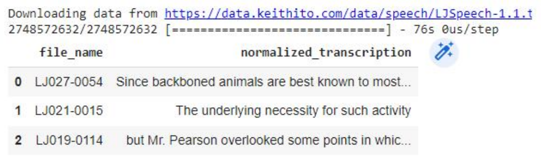
In that figure we show that when problem arises how CTC solve it

E Mathematically CTC Loss function:

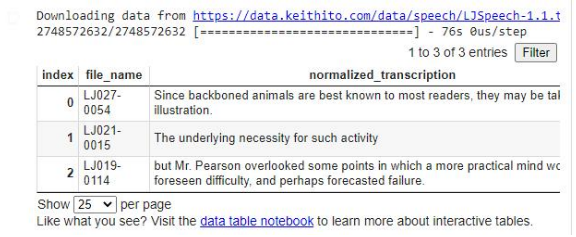
- $loss = -\log(p(y|X)) = log\frac{1}{p(y|X)}$
  - When p(y|X) approaching 0, loss approaching infinity.
  - When p(y|X) approaching 1, loss approaching 0.
  - As we can see from the loss function, we are training the model to learn and optimize p(y|X). When p(y|X) approaches 1, the loss approaches 0; we will not update the weights in the model.

Here we define how mathematical equation works in specified problem.

F. Read File and parse:



```
Downloading data from https://data.keithito.com/data/speech/LJSpeech-1.1.t
2748572632/2748572632 [==============================] - 76s 0us/step
   file_name                     normalized_transcription
0  LJ027-0054  Since backboned animals are best known to most...
1  LJ021-0015         The underlying necessity for such activity
2  LJ019-0114    but Mr. Pearson overlooked some points in whic...
```



```
Downloading data from https://data.keithito.com/data/speech/LJSpeech-1.1.t
2748572632/2748572632 [==============================] - 76s 0us/step
```

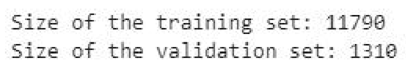
1 to 3 of 3 entries Filter

index	file_name	normalized_transcription
0	LJ027-0054	Since backboned animals are best known to most readers, they may be tal illustration.
1	LJ021-0015	The underlying necessity for such activity
2	LJ019-0114	but Mr. Pearson overlooked some points in which a more practical mind wc foreseen difficulty, and perhaps forecasted failure.

Show 25 per page
Like what you see? Visit the data table notebook to learn more about interactive tables.

After importing the libraries code sets the variable data_url to a URL that contains a dataset of speech data called "LJSpeech-1.1" Finally, read the file and parse it, the first three rows of the DataFrame are displayed using head(3) as shown in above figure

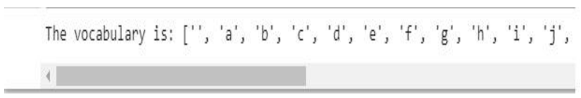
G Evaluation of Training and validation set:



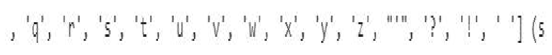
```
Size of the training set: 11790
Size of the validation set: 1310
```

In that stage we obtain the result of training and validation set that explain 90 % of training and 10% of validation set.

H. Define Vocabulary:



```
The vocabulary is: ['', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
```

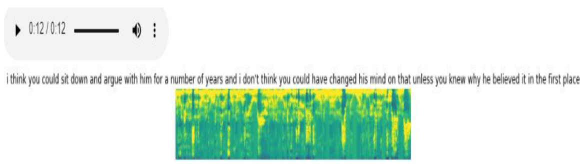


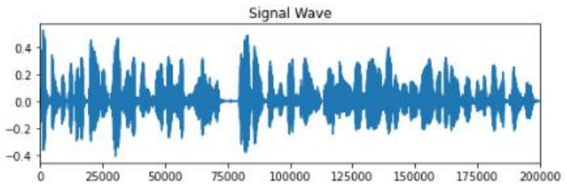
```
, 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', "'", '?', '!', ' '] (s
```

In that stage we define the vocabulary and print it on console.

I plot spectrogram and Wave

Here we evaluate and output as plot the spectrogram and wave signal as shown in figure



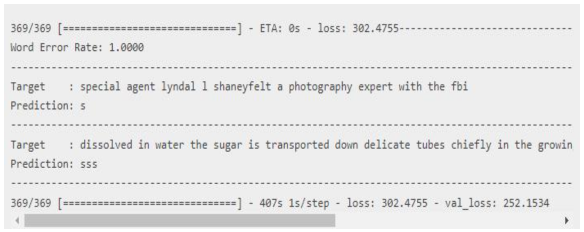


J Using Deep Search model to find Trainable and Non trainable values

Total params	26,628,480
Trainable params	26,628,352
Non-trainable params	128

Here the above table we show the total number of params their trainable and non-trainable result on deepsearch2 model.

K Training process and Performance of different samples:



```
369/369 [==============================] - ETA: 0s - loss: 302.4755-----------------------------
Word Error Rate: 1.0000
-----------------------------------------------------------------------------------------------
Target    : special agent lyndal l shaneyfelt a photography expert with the fbi
Prediction: s
-----------------------------------------------------------------------------------------------
Target    : dissolved in water the sugar is transported down delicate tubes chiefly in the growin
Prediction: sss
-----------------------------------------------------------------------------------------------
369/369 [==============================] - 407s 1s/step - loss: 302.4755 - val_loss: 252.1534
```

In this final stage we show To trains a speech recognition model using the fit method of the Keras Sequential model.The number of epochs is set to 1, which means that the model will only be trained for one epoch. A validation_callback object of the CallbackEval class is created and passed as an argument to the fit method's callbacks parameter. This callback object will be called at the end of each epoch to output the predicted transcriptions and the word error rate (WER) for the validation dataset.

9. Appendix 2

This code is not mine; I follow and implement some changes and follow the step and content more thoroughly and past research on this topic help me a lot to complete and learn the code**.**

Code on GitHub

https://github.com/HamzahAsghar/Speech_Recognition_automatically_by_using_ANN_-.git

```
pip install jiwer
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
from IPython import display
from jiwer import wer
import matplotlib.pyplot as plt
data_url = "https://data.keithito.com/data/speech/LJSpeech-1.1.tar.bz2"
data_path = keras.utils.get_file("LJSpeech-1.1", data_url, untar=True)
wavs_path = data_path + "/wavs/"
metadata_path = data_path + "/metadata.csv"



# Read metadata file and parse it
metadata_df = pd.read_csv(metadata_path, sep="|", header=None, quoting=3)
metadata_df.columns = ["file_name", "transcription", "normalized_transcrip
tion"]
metadata_df = metadata_df[["file_name", "normalized_transcription"]]
metadata_df = metadata_df.sample(frac=1).reset_index(drop=True)
metadata_df.head(3)



split = int(len(metadata_df) * 0.90)
df_train = metadata_df[:split]
df_val = metadata_df[split:]

print(f"Size of the training set: {len(df_train)}")
print(f"Size of the validation set: {len(df_val)}")
# The set of characters accepted in the transcription.
characters = [x for x in "abcdefghijklmnopqrstuvwxyz'?! "]
```

```
# Mapping characters to integers
char_to_num = keras.layers.StringLookup(vocabulary=characters, oov_token="
")
# Mapping integers back to original characters
num_to_char = keras.layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
)

print(
    f"The vocabulary is: {char_to_num.get_vocabulary()} "
    f"(size ={char_to_num.vocabulary_size()})"
)
# An integer scalar Tensor. The window length in samples.
frame_length = 256
# An integer scalar Tensor. The number of samples to step.
frame_step = 160
# An integer scalar Tensor. The size of the FFT to apply.
# If not provided, uses the smallest power of 2 enclosing frame_length.
fft_length = 384


def encode_single_sample(wav_file, label):
    ###########################################
    ##  Process the Audio
    ##########################################
    # 1. Read wav file
    file = tf.io.read_file(wavs_path + wav_file + ".wav")
    # 2. Decode the wav file
    audio, _ = tf.audio.decode_wav(file)
    audio = tf.squeeze(audio, axis=-1)
    # 3. Change type to float
    audio = tf.cast(audio, tf.float32)
    # 4. Get the spectrogram
    spectrogram = tf.signal.stft(
        audio, frame_length=frame_length, frame_step=frame_step, fft_lengt
h=fft_length
    )
    # 5. We only need the magnitude, which can be derived by applying tf.a
bs
    spectrogram = tf.abs(spectrogram)
    spectrogram = tf.math.pow(spectrogram, 0.5)
    # 6. normalisation
    means = tf.math.reduce_mean(spectrogram, 1, keepdims=True)
    stddevs = tf.math.reduce_std(spectrogram, 1, keepdims=True)
    spectrogram = (spectrogram - means) / (stddevs + 1e-10)
    ###########################################
    ##  Process the label
    ##########################################
    # 7. Convert label to Lower case
    label = tf.strings.lower(label)
    # 8. Split the label
    label = tf.strings.unicode_split(label, input_encoding="UTF-8")
    # 9. Map the characters in label to numbers
    label = char_to_num(label)
    # 10. Return a dict as our model is expecting two inputs
    return spectrogram, label

batch_size = 32
# Define the trainig dataset
train_dataset = tf.data.Dataset.from_tensor_slices(
    (list(df_train["file_name"]), list(df_train["normalized_transcription"]
))
)
train_dataset = (
    train_dataset.map(encode_single_sample, num_parallel_calls=tf.data.AUT
OTUNE)
    .padded_batch(batch_size)
    .prefetch(buffer_size=tf.data.AUTOTUNE)
)

# Define the validation dataset
validation_dataset = tf.data.Dataset.from_tensor_slices(
    (list(df_val["file_name"]), list(df_val["normalized_transcription"]))
)
validation_dataset = (
    validation_dataset.map(encode_single_sample, num_parallel_calls=tf.dat
a.AUTOTUNE)
    .padded_batch(batch_size)
    .prefetch(buffer_size=tf.data.AUTOTUNE)
)
fig = plt.figure(figsize=(8, 5))
for batch in train_dataset.take(1):
    spectrogram = batch[0][0].numpy()
    spectrogram = np.array([np.trim_zeros(x) for x in np.transpose(spectro
gram)])
    label = batch[1][0]
    # Spectrogram
    label = tf.strings.reduce_join(num_to_char(label)).numpy().decode("utf
-8")
    ax = plt.subplot(2, 1, 1)
    ax.imshow(spectrogram, vmax=1)
    ax.set_title(label)
    ax.axis("off")
    # Wav
    file = tf.io.read_file(wavs_path + list(df_train["file_name"])[0] + ".
wav")
    audio, _ = tf.audio.decode_wav(file)
    audio = audio.numpy()
    ax = plt.subplot(2, 1, 2)
    plt.plot(audio)
    ax.set_title("Signal Wave")
    ax.set_xlim(0, len(audio))
    display.display(display.Audio(np.transpose(audio), rate=16000))
plt.show()


def CTCLoss(y_true, y_pred):
    # Compute the training-time loss value
    batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")
    input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64")
    label_length = tf.cast(tf.shape(y_true)[1], dtype="int64")

    input_length = input_length * tf.ones(shape=(batch_len, 1), dtype="int
64")
    label_length = label_length * tf.ones(shape=(batch_len, 1), dtype="int
64")

    loss = keras.backend.ctc_batch_cost(y_true, y_pred, input_length, labe
l_length)
    return loss


def build_model(input_dim, output_dim, rnn_layers=5, rnn_units=128):
    """Model similar to DeepSpeech2."""
    # Model's input
    input_spectrogram = layers.Input((None, input_dim), name="input")
    # Expand the dimension to use 2D CNN.
                                    x          =          layers.Reshape((-
1, input_dim, 1), name="expand_dim")(input_spectrogram)
    # Convolution layer 1
    x = layers.Conv2D(
        filters=32,
        kernel_size=[11, 41],
        strides=[2, 2],
        padding="same",
        use_bias=False,
        name="conv_1",
    )(x)
    x = layers.BatchNormalization(name="conv_1_bn")(x)
    x = layers.ReLU(name="conv_1_relu")(x)
    # Convolution layer 2
    x = layers.Conv2D(
        filters=32,
        kernel_size=[11, 21],
        strides=[1, 2],
        padding="same",
        use_bias=False,
        name="conv_2",
    )(x)
    x = layers.BatchNormalization(name="conv_2_bn")(x)
    x = layers.ReLU(name="conv_2_relu")(x)
    # Reshape the resulted volume to feed the RNNs layers
    x = layers.Reshape((-1, x.shape[-2] * x.shape[-1]))(x)
    # RNN layers
    for i in range(1, rnn_layers + 1):
        recurrent = layers.GRU(
            units=rnn_units,
            activation="tanh",
            recurrent_activation="sigmoid",
            use_bias=True,
            return_sequences=True,
            reset_after=True,
            name=f"gru_{i}",
        )
        x = layers.Bidirectional(
            recurrent, name=f"bidirectional_{i}", merge_mode="concat"
        )(x)
        if i < rnn_layers:
            x = layers.Dropout(rate=0.5)(x)
    # Dense layer
    x = layers.Dense(units=rnn_units * 2, name="dense_1")(x)
    x = layers.ReLU(name="dense_1_relu")(x)
    x = layers.Dropout(rate=0.5)(x)
    # Classification layer
    output = layers.Dense(units=output_dim + 1, activation="softmax")(x)
    # Model
    model = keras.Model(input_spectrogram, output, name="DeepSpeech_2")
    # Optimizer
    opt = keras.optimizers.Adam(learning_rate=1e-4)
    # Compile the model and return
    model.compile(optimizer=opt, loss=CTCLoss)
```

```
    return model


# Get the model
model = build_model(
    input_dim=fft_length // 2 + 1,
    output_dim=char_to_num.vocabulary_size(),
    rnn_units=512,
)
model.summary(line_length=110)


# A utility function to decode the output of the network
def decode_batch_predictions(pred):
    input_len = np.ones(pred.shape[0]) * pred.shape[1]
    # Use greedy search. For complex tasks, you can use beam search
    results = keras.backend.ctc_decode(pred, input_length=input_len, greed
y=True)[0][0]
    # Iterate over the results and get back the text
    output_text = []
    for result in results:
        result = tf.strings.reduce_join(num_to_char(result)).numpy().decod
e("utf-8")
        output_text.append(result)
    return output_text


# A callback class to output a few transcriptions during training
class CallbackEval(keras.callbacks.Callback):
    """Displays a batch of outputs after every epoch."""

    def __init__(self, dataset):
        super().__init__()
        self.dataset = dataset

    def on_epoch_end(self, epoch: int, logs=None):
        predictions = []
        targets = []
        for batch in self.dataset:
            X, y = batch
            batch_predictions = model.predict(X)
            batch_predictions = decode_batch_predictions(batch_predictions)
            predictions.extend(batch_predictions)
            for label in y:
                label = (
                    tf.strings.reduce_join(num_to_char(label)).numpy().dec
ode("utf-8")
                )
                targets.append(label)
        wer_score = wer(targets, predictions)
        print("-" * 100)
        print(f"Word Error Rate: {wer_score:.4f}")
        print("-" * 100)
        for i in np.random.randint(0, len(predictions), 2):
            print(f"Target    : {targets[i]}")
            print(f"Prediction: {predictions[i]}")
            print("-" * 100)
# Define the number of epochs.
epochs = 1
# Callback function to check transcription on the val set.
validation_callback = CallbackEval(validation_dataset)
# Train the model
history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=epochs,
    callbacks=[validation_callback],
)
# Let's check results on more validation samples
predictions = []
targets = []
for batch in validation_dataset:
    X, y = batch
    batch_predictions = model.predict(X)
    batch_predictions = decode_batch_predictions(batch_predictions)
    predictions.extend(batch_predictions)
    for label in y:
        label = tf.strings.reduce_join(num_to_char(label)).numpy().decode(
"utf-8")
        targets.append(label)
wer_score = wer(targets, predictions)
print("-" * 100)
print(f"Word Error Rate: {wer_score:.4f}")
print("-" * 100)
for i in np.random.randint(0, len(predictions), 5):
    print(f"Target    : {targets[i]}")
    print(f"Prediction: {predictions[i]}")
    print("-" * 100)
```