Echo-Bridge: Bridging Voices and Signs Through AI for Enhanced Accessibility

Karim Habbal
kmh36@mail.aub.edu

Hamza Atout
hsa60@mail.aub.edu

Khaled Nasser
kwn02@mail.aub.edu

Khaled Ammoura
kaa74@mail.aub.edu

Github

Abstract

This project focuses on building an AI-driven system that bridges the communication gap between individuals with speech, hearing, or vision impairments and the broader community. It's a combination of 2 main models (each implemented in different ways for testing purposes due to our minimal resources). The first model is transforming from speech to text and the second is transforming from sign language captured using computer vision to text. The other ways around (text to speech and text to sign language) are to be implemented in our future work due to the limited time and resources we have. **dont forget to fill in the results**

1 Motivation and Background

We live in a region where tools that allow people with speech, hearing, and vision impairments to connect with the broader community are scarce or often difficult to access due to several reasons, one of which is high costs. Most of these tools are not even widely available in our mainstream communication platforms. This creates barriers for seamless interactions. Speech-to-text functionality is widely available and could be easily implemented with a simple python code integrated into our applications; however, the same cannot be said for sign language-to-text capabilities. Nonetheless, a fully-fledged system would not be complete without incorporating a speech-to-text model as well. Hence, for learning purposes, a section for transforming audio to text will be included as mentioned earlier. With a system that fully bridges the communication gaps we talked about earlier, a person with a vision impairment and another with a speech impairment can seamlessly communicate by translating sign language to speech. This is just an example of the use cases of our system. So why not make this applicable in most of our daily applications?

2 Speech to Text

2.1 Methodology

Our Speech-to-Text model utilizes the Connectionist Temporal Classification (CTC) algorithm, a commonly used approach for sequence-to-sequence models, which was introduced by Graves et al. (2006). This model is special because of its use case. It is used in cases where pre-segmented data is not available, and it is ideal for handling variable-length inputs. CTC dynamically learns the alignment by allowing the model to predict a blank token or repeat characters, enabling flexible mapping between inputs (e.g., mel spectrograms) and outputs (e.g., text or labels).

Two implementations were done, mainly because we faced lots of issues with the first, and we had to submit a result in due time. The first utilized Deeplake's lazy loading, integrated with Tensorflow's batch processing; however, we faced lots of issues with the .map() function which preprocesses each batch on the fly by not storing the entire dataset in memory in advance of training our model. For this reason, we opted for the approach of training our model manually (not on the fly) by processing the batch (the entire dataset) once and simply training the model afterwards. As a resource for our research, we mainly relied on the paper "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks" (2006) (1)

2.1.1 Preprocessing

Before feeding our data to the model, we first have to transform our waveform/mp3 audio into spectrograms, specifically the mel-spectrogram. What makes this spectrogram unique is its log-scale, more specifically the mel scale. On a higher level, it is specifically designed to approximate how humans perceive pitch. It spaces frequencies logarithmically, prioritizing lower frequences (where humans have better pitch resolution), resulting in what we call mel bins.

As well as presenting audio in the frequency domain, the mel spectrogram allows for the representation of the dynamics of an audio signal in time by segmenting the waveform into overlapping time slices. Each slice is assigned a time step that quantifies the distribution of energy over the spectrum at the instant of the time step. The relationship of these time steps and mel bins, can be represented in a two-dimensional space where the raw data point is, the x-axis which is a temporal axis, the y-axis which depicts the mel frequency bins, and the volume of each point indicates the intensity of value of the signal at the respective time.

Since in the earlier research paper (1) this configuration was used successfully, we chose to use 128 mel bins for the mel spectrogram transformation in our study. Furthermore, our 16,000 Hz audio sampling rate guarantees that the mel spectrogram records enough spectral detail while adhering to industry standards for speech-related tasks.

2.1.2 Dataset

We first tried using the TIMIT dataset using deeplake's API for 2 main reasons. The only way to access the dataset was through deeplake, and the second reason is that this dataset was the dataset used in the CTC paper (1). However, due to the difficulties we faced with deeplake, we went for the Common Voice dataset by Mozilla. Later in our training, we found out that this dataset doesn't work well with CTC because CTC loss is optimal with short sequences (4-5 seconds) of audio, and unfortunately Mozilla's Common Voice dataset is considered a dataset with long sequences relative to TIMITs dataset.

2.1.3 LSTM + Results

The architecture of LSTM (Long Short-Term Memory) model fits well for our application since it can model temporal dependencies present in sequential data like mel spectrograms. The fact that it can remember long-term dependencies and accept inputs of varying sizes is in absolute congruence with the CTC loss function that is employed for unsegmented sequence learning. Having been applied in tasks such as speech and sign language recognition, LSTMs are able to handle sequences by using information from the beginning and the end of the sequence thanks to the use of bi-directional layers. Together with our 128 mel bins sampled at 16 000 Hz, the LSTMs make the predictions on the sign language to text conversion accurate and efficient. For this reason, we built an LSTM with 4 main layers: two bidirectional LSTM layers (each with 512 units), a TimeDistributed dense layer (128 units), and a final dense output layer with a softmax activation. It has approximately 9 million parameters and resulted in an approximate loss of 1062. This loss is considered good at first but as the number of epochs increases, the loss should rapidly decrease. Unfortunately, this was not the case, and this is living proof of why CTC's loss function is not good with long sequences. We also tried to add a 3rd bidirectional layer which resulted in the same average.

2.1.4 Vanilla RNN + Results

For testing purposes, we also tried working with vanilla RNN's which proved to be a failure as expected. It produced even higher losses with an **average loss** of 1200.

2.1.5 Final Results

Our first assumption, being that CTC won't work with Mozilla's Common Voice dataset proved to be true. This was proven true from the constant unchanging loss resulting from the previous models. For our future work, we are planning on fixing this loss by utilizing a CNN + LSTM model or transformers while using the suitable dataset for each model. In case you want to give our 2 approaches a check (deeplake vs manual), make sure to check out our posted github repo.

3 Sign to Text

3.1 Methodology

To achieve our goal, several ways emerged. The most intuitive way was to use a 2D-Convolutional Neural Network (CNN) with a Recurrent Neural Network (RNN), which will be discussed later in Section 3.1.3. An alternative way was to merge these Neural Networks (NN) into a one 3D-CNN, which will be discussed in Section 3.1.4. Moreover, advanced methods were implemented that bound the frame of interest in an image. These two methods are Pose-RNN and Pose-TGCN which are an altered version of 2D-CNN + RNN and 3D-CNN respectively(2).

3.1.1 Dataset

Sign recognition is considered a pillar of the communication bridge we are constructing in this project. To solve this communication barrier, we started with alphabetical-level sign recognition using the Modified National Institute of Standards and Technology (MNIST) dataset, which follows the American Sign Language(ASL), in which our approach was to feed the letters to a Large Language Model (LLM) to construct a meaningful word. Therefore, we consulted this paper (3) on Real-Time Sign Language Recognition with Convolutional Neural Networks to gain insight about the models that serve the purpose of this problem. After taking a deep look into this issue, we came across a new word-level dataset, Word Level American Sign Language (WLASL), which is the largest ASL dataset consisting of the top 2000 words with over 12,000 videos performed by different signers. As a result, the WLASL paper (2) was a perfect fit to address this problem.

3.1.2 Preprocessing

For data preprocessing we had various options with pros and cons. First, we standardized the number of frames extracted from videos to 16 uniformly distributed frames. Although, extracting all the frames from each video is much better in preserving temporal details, we decided to disregard this option, as it is computationally expensive. Afterwards, we applied set of functions on each frame to maximize our models' generalization, such as, resizing the frames to (128,128) or (224,224) depending on the model, applying horizontal flipping with a probability of 0.5 for 3D-CNN, randomly cropping based on the bounding box (which is provided in the dataset) and transforming each frame's scale from RGB to Gray, which reduces the dimensionality of the input, speeding-up the training process. Finally, we chose to work on the WLASL-100 (contains only 100 unique words instead of 2000) due to the lack of computational resources.

3.1.3 CNN+LSTM

Once spatio-temporal dependencies are needed, CNN + RNN model is the first solution that comes to mind. First, after preprocessing the data and extracting the required frames from each video, we input these frames to the CNN model. The CNN model is mainly used for feature extraction, which is done by passing the frame through various layers in order determine the spatial dependencies of the frame. After that, we feed the output of the CNN, which is the feature map obtained, into the RNN model that determines the temporal dependencies of the frames of each video. Instead of RNN, we used LSTM as it (4). To give an overview of the architecture of this model, 2D-kernels (5), pooling (6) and LSTM layers were used.

First, we feed each frame of the video independently to the CNN that applies a (3x3) kernel to it with varying number of neurons on each layer (32, 64, 128). The kernel moves over the whole frame that has 2 dimensions (height and width) to extract spatial features needed. We used ReLU, a commonly used activation function, as an activation function. Then it passes through a batch normalization layer that normalizes the data by scaling and shifting them to accelerate training process and improve convergence. After that, we pass the data to pooling layer that passes through the 2D matrix obtained and reduces its dimensionality. This process repeats several times till reaching the features best determining the spatial dependencies.

Afterwards, the features are fed into the LSTM model which processes the sequential features by capturing long-term dependencies using a memory cell and gating mechanisms to regulate the flow of information over time (4). The LSTM layers contain 128 neurons. Dropout layer is used to avoid overfitting by dropping random neurons from the network. Finally, we use softmax regression that maps the output layer to the word with the highest probability.

3.1.4 3D-CNN

In the realm of accounting for both spatio-temporal dependencies at once, 3D-CNN comes in-handy. This model could be thought of as the higher dimensional analog of 2D-CNN. As mentioned in Section 3.1.3, the usual flow of frames was to extract the feature map of each feature, then feed it to the RNN, which determines the temporal details between the frames. However, 3D-CNN suggests a cleverer approach, in which it stacks the frames of each video on each other. To describe the architecture of this model, kernels, pooling, and activation functions were used.

First, we stack the frames and then apply a 3D-kernel/filter(**?**) which moves through three dimensions: height, width, and depth (time or frames). This allows the network to figure patterns such as motion dynamics of the video, making it particularly useful in tasks like video analysis and action recognition. As the kernel moves it typically reduces the dimensions of the input, depending on the stride and padding. Large strides and valid padding (no padding) down samples the input, improving the computational efficiency of the process. Moreover, the output of this step results in a set of feature maps which are then applied to an activation function (e.g., ReLu or sigmoid) which introduces non-linearity to the model, allowing it to capture more complex features. Afterwards, a 3D-MaxPooling is applied to the new resulted set of feature maps which, usually, reduces the dimensions significantly, resulting in an another set of feature maps. Optionally, we can repeat this process to extract complex features, by feeding the output of the last cycle as the input of the second.

Finally, we flatten our set of feature maps and then feed it to out deeply dense fully connected NN, in which we designed it as follows: 2 hidden layers, the first one consists of 512 neurons, while the second consists of 128 neurons. Moreover, the output layer contains number of neurons equals to the number of unique words, which are in our case 25 words extracted from WLASL-100, where we use softmax classification to output the word of the highest probability.

3.2 Results

3.2.1 CNN+LSTM

We faced some problems in training the CNN+LSTM model on a large protion of the dataset due to lack of computational resources. So, we trained the model on a very small portion of the dataset where we got an accuracy of 37.5% on train set, 14.2857% on validation set, and a test accuracy of 25%.

3.2.2 3D-CNN

We've splitted out the input into three set: training, validation, and testing. This approach helps us evaluate our model and fine-tune it when inconsistencies occur. We trained our model, and evaluate it on each of the training, validation and testing phases.

As showed in Fig. 1, we got a training accuracy = 92.48%, validation accuracy = 14.29% and test accuracy = 12%. We can see that there is some kind of over-fitting on the training set compared to that of the validation and testing. So, we applied L2 regularization, which minimized the over-fitting but not much.

As seen in Fig. 2, we got a training accuracy = 78.23%, validation accuracy = 17.14% and test accuracy = 16%.The core issue is that the number of training examples is much more larger than that of the validation and testing, where len(training.data)=177, len(val.data)=35 and len(test.data)=25.

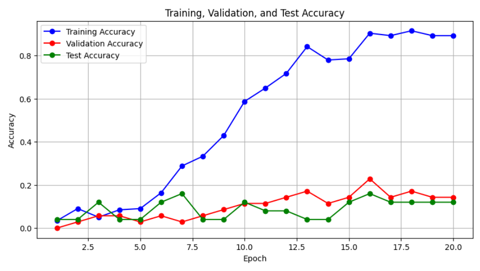

Figure 1: Overfitting

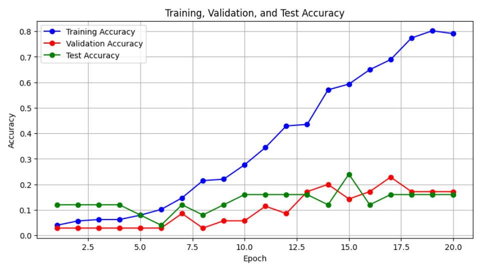

Figure 2: Regularized-Overfitting

To solve this issue we should use some type of a method that generates data from existing data to increase the number of examples in each of the validation and testing sets.

3.2.3 Comparison Failure

As mentioned in section 3.2.1, CNN+LSTM model required more computational resources to train. So, the models were training based on different portions of the chosen sub-dataset. Accordingly, it was a failed attempt to compare both models, due to the inconsistency in training.

4 Ethical Considerations

When implementing an application for communication between deaf and normal people, it is essential to take into consideration the ethical implications that might occur. One important thing, is it prohibit the users from saying curse words as the goal of our application is to fill the gap between two parties in a good way, such as educational purposes. By filtering out profanity, the system mitigates the risk of unintentional or harmful usage, reinforcing its role as a constructive tool for bridging communication gaps. However, this approach must also be balanced with transparency, ensuring that users are informed about such limitations to maintain trust and set clear expectations regarding the application's functionality. Furthermore, the model must be designed to minimize the risk of catastrophic errors in recognition, such as misinterpreting a benign phrase like "Hello, how is your day?" as something harmful or offensive like "Kill yourself." Such errors could have severe consequences, including emotional distress or harm to users, and therefore require rigorous testing, ethical oversight, and continuous refinement of the model to ensure accuracy and safety. By addressing these concerns, the system can reinforce trust, promote effective communication, and contribute positively to the communities it serves.

5 Conclusion and Future Work

This project demonstrated the development of an AI-driven system which attempts to eliminate communication barriers for persons who are mute, hard of hearing and/or are vision impaired. Understandably, the text and sign languages had to form two main components around which our models attempted to find solutions. The speech-to-text model was based on CTC loss utilizing 128 mel bins and 16000 Hz sampling rate which enabled the model to extract audio features. Albeit coping with long range high dimensional data may be problematic, the LSTM architecture does look feasible for sequential data processing. The sign-to-text model made use of CNN+LSTM and 3D-CNN approach during the research, where the last one produced better results for the training data set, but the model suffered from overfitting and poor results on validation data due to the lack of appropriate hardware resources. These results underline the necessity of proper datasets and sufficient computational power for viable pipelines. Subsequently, we will consider the refinement of these models applying transformers and CNN-LSTM hybrids taking into account hardware limitations to incorporate text-to-speech and text-to-sign language tools. This project has been a step in building a more accessible world for people with disabilities.

References

[1] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd International Conference on Machine Learning*, pp. 369–376, ACM, 2006.

[2] D. Li, C. Rodriguez, X. Yu, and H. Li, "Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison," in *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pp. 1459–1469, 2020.

[3] D. Rakhmatov and A. Breccia, "Real-time sign language recognition with convolutional neural networks," 02 2024.

[4] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural Computation*, vol. 31, pp. 1235–1270, 07 2019.

[5] N. Babu, J. Kumari, J. Mathew, U. Satija, and A. Mondal, "Multiclass categorisation of respiratory sound signals using neural network," pp. 228–232, 10 2022.

[6] H. Gholamalinezhad and H. Khosravi, "Pooling methods in deep neural networks, a review," *arXiv preprint arXiv:2009.07485*, 2020.