Training Efficient Wav2Vec2.0 Automatic Speech Recognition Engine Transformer Model from Scratch with Different Heads

Final Project Research Report for the

Master of Science in

Data Science

The George Washington University, Washington, D.C.

Tristin Johnson

May 2022

Table of Contents

I. Abstract

Wav2Vec2.0 is a state-of-the-art model when it comes to Automatic Speech Recognition (ASR) due to its unique style of self-supervised training. The objective is to both pre-train and fine-tune the Wav2Vec2.0 model on unlabeled and labeled data to efficiently build an ASR engine from scratch. In doing so, opens endless possibilities to training a unique and individual speech recognition engine that allows for flexibility to a user's specific use case. Along the way, this project aims to apply every piece of a machine learning pipeline developed from scratch using well known software skills that adds to the flexibility in creating ASR engines. Furthermore, Wav2Vec2.0 is said to have the elasticity to be applied to other audio and speech datasets, in which we set out to give a finite answer to this hypothesis.

II. Introduction

Automatic Speech Recognition (ASR) is technology that allows people from all over the world to have the ability to speak into a computer and have that perfectly translated to text using the most efficient and accurate software. In general, the workflow of events when it comes to ASR is to speak into some computer, an audio file is generated, that audio file is fed into an already trained machine learning model, which then returns the text completely translated. In recent years, software developers across the world have been edging more and more towards highly accurate models, that leave very little to no error in the speech translation. Furthermore, these models can be trained using any language with enough data [3].

When it comes to ASR engines, these types of systems are extremely efficient, cheap, and convenient to multiple different environments including students in the classroom, business meeting, personal use, and more. Furthermore, ASR software can boost productivity by providing the ability to simply record any voice and have it written directly to a document compared to recording notes by hand, which can be time consuming [3]. A specific use-case for ASR software is when it comes to students, especially ones with learning disabilities such as ADD or ADHD. This allows for students to easily transfer their ideas onto a page with the ability to talk ideas through and go back to look at the thought process, rather than taking the time to handwrite notes and struggle to get any ideas down onto a piece of paper.

The research objective is to build a speech engine from scratch that has the ability to transcribe any given audio file to its pertained text. Specifically, the notoriously known speech recognition model, Wav2Vec2.0, will be used. Wav2Vec2.0 is a state-of-the-art model known for its ability to translate any audio file to the correlating text. Wav2Vec2.0 is a highly complex model that was originally trained on 960 hours of audio data. Having said that, the goal is to pre-train the Wav2Vec2.0 model from scratch, fine-tune the model, and then apply this pre-trained model to a speech classification dataset. The purpose of this research is to see if similar results can be obtained using only the Wav2Vec2.0 architecture and apply custom-built methods that are fed into the model. There is very little research out there when it comes to training ASR engines from scratch, which is what we set out to achieve.

III. Wav2Vec2.0 Architecture

The Wav2Vec2.0 is a highly complex model made up of a multi-layer convolutional feature encoder that takes raw audio as input and outputs latent speech representations for *t* time-steps [1]. Then, the output is fed into the contextualized representations that capture information regarding the entire sequence. Lastly, this output of the feature encoder is discretized with a define quantization module to represent the targets. This can be seen below:

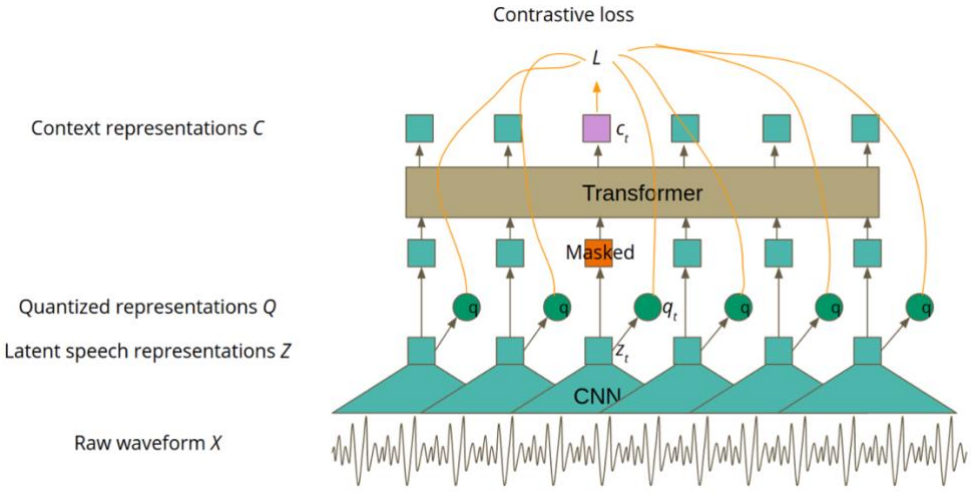

Figure 1: Wav2Vec2.0 architecture for self-supervised training (Image by Lukasz Sus [10])

This model also introduces a new concept called contrastive learning. Contrastive learning is where the input is transformed in two different ways: the first being transformer layers and the second one is made by quantization [13].

i. Feature Encoder

The feature encoder consists of several blocks including a convolutional layer followed directly by a normalization layer, and finally a Gaussian Error Linear Unit (GELU) activation function. The raw audio waveform is normalized as input before being fed into the feature encoder [1]. The purpose of the feature encoder is to get the raw audio into a latent representation.

ii. Context Representations

The output of the feature encoder is then fed into a contextualized network. The purpose of this is to use the convolutional layers from the feature encoder to further enhance the speech representations but with context. At the end of contextualizing the raw audio, a convolutional layer is added as a relative positional embedding layer to also enhance the context of the waveform [1].

iii. Quantized Representations

What makes Wav2Vec2.0 such a unique model is the concept of quantized representations. The process of quantization is converting values from a continuous space into a finite set of values [1]. For self-supervised training with Wav2Vec2.0, the output of the feature encoder is discretely compressed to a finite set of speech representations via quantization. This idea implemented into the model led to highly accurate results when combined with the learned discrete units from the contextualized representations [5]. Quantization comes down to choosing the right code word from a codebook, where there are G codebooks created, each consisting of V codewords, which are defined in training [13]. The quantized representation is created by selected the best word from each codebook in which the chose vectors are concatenated followed by a GELU activation function.

IV. Wav2Vec2.0 Training

The model is trained in two phases: the first phase being self-supervised mode and the second phase is supervised fine-tuning. During the first phase, self-supervised training is done using unlabeled data with the idea to achieve the best speech representations as possible. During the second phase of supervised fine-tuning, the model is trained on labeled data which in return is used to help teach the model to predict particular words. This can be seen below:

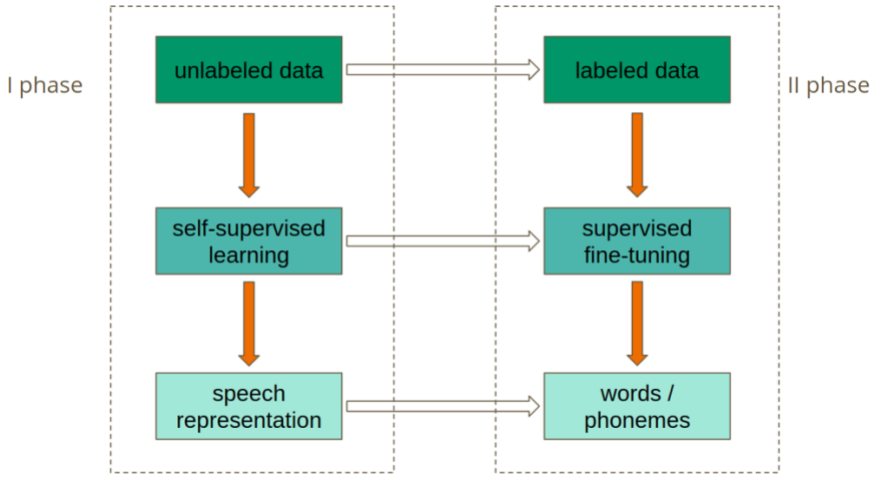

Figure 2: The two phases of training Wav2Vec2.0. (Image by: Lukasz Sus [11])

i. Masking

Before feeding the feature encoder outputs to the contextualized representations, a proportion of the feature encoder outputs are masked and then replaced with a trained feature vector shared between all the masked time steps [5]. This can be summarized as taking all the time steps from the space of the latent speech representations step, sample without replacement, and then for each index (randomly chosen time steps), a consecutive pre-defined number of time steps are masked [1, 13].

ii. Contrastive Loss

The training objective is a sum of two different loss functions, with the first being contrastive loss. Contrastive loss is mainly responsible for helping predict the correct quantized representation among $K + 1$ quantized candidate representations [13]. The formula for the loss function can be seen below:

$$L_m = -log\frac{exp(sim(c_t, q_t)/\kappa)}{\sum_{\tilde{q}\epsilon Q_t} exp(sim(c_t, \tilde{q})/\kappa)}$$

Figure 3: Formula to calculate contrastive loss where sim is cosine similarity, L_m is like a softmax activation function, c_t are the context representations, k is constant, and q is the quantized representations.

iii. Diversity Loss

The second loss function is diversity loss. Diversity loss is a regularization technique that calculates the loss between choosing the right codeword from the right codebook, as mentioned earlier [13]. It is designed to increase the use of quantized codebook representations within training the model. The formula can be seen below:

$$H(X) = -\sum_{x} P(x)log(P(x))$$

Figure 4: Formula for diversity loss where x is a possible outcome of discrete random variable X, and P(X) is the probability of event x.

Since diversity loss is based on entropy, it can find the best codewords from each codebook, which significantly reduces the time and complexity of calculating loss. Entropy assumes the maximum values over a uniform distribution of the data, which means all the codewords are used with the same frequency [1, 13]. When maximizing this entropy, the model is encouraged to take advantage of the best codewords, which in return provides the minimization of negative entropy. Hence, diversity loss.

iv. Fine-Tuning

When fine-tuning the model, the main concept is that Wav2Vec2.0 doesn't apply quantization as that is strictly for pre-training. Instead, a randomly initialized linear projection layer is added to the contextualized representations to replace the concept of quantization. Furthermore, the model is fine-tuned using Connectionist Temporal Classification (CTC) loss and a modified version of masking is applied to help improve final error rates [1, 4].

V. Project Approach

Now that we have covered all the technical aspects of the Wav2Vec2.0 model, from its architecture to the training objectives, we can now dive into the approach taken to achieve similar results during research. The first step is to pre-train the Wav2Vec2.0 model, fine-tune the model, and finally apply the fine-tuned model to a speech classification dataset.

i. Pre-Training Wav2Vec2.0

The approach in pre-training the Wav2Vec2.0 is relatively simple, as all that is needed is the Wav2Vec2.0 configuration file. As we know, Wav2Vec2.0 was both pre-trained and fine-tuned, where its main and most accurate model was pre-trained using the LibriVox dataset and then fine-tuned on the LibriSpeech dataset. The downside to pre-training a model from scratch is that pre-training is both computationally expensive and financially expensive. Originally, the creators of Wav2Vec2.0 pre-trained the model on LibriVox using 128 GPU's which took just over 120 hours to complete [1]. Then, the model was fine-tuned on LibriSpeech with the same computation power, which took just over 55 hours to complete.

The objective for this research project was to build all the data preprocessing steps from scratch, train a custom Wav2Vec2.0 tokenizer (with a custom vocabulary file) and feature extractor, load in the configuration file with all the standard parameters, and start pre-training. The difference is rather than loading the already trained model with all its weights and biases loaded in, all of these are set to 0 and the training begins. The downside is that for this project, my computation is slightly more limited as only 1 GPU is available, and the cost of computation is restricted.

ii. Fine-Tuning Wav2Vec2.0

There are two ways to achieve fine-tuning the Wav2Vec2.0 model. First being to fine-tune the custom pre-trained model from scratch that is generated from the previous section. The second is to fine-tune the original Wav2Vec2.0 model by changing parts of the architecture along with hyperparameter tuning.

Since the original Wav2Vec2.0 model was already trained on two of the notoriously well-known speech recognition datasets, to properly fine-tune the model we well use a different

dataset. Secondly, for research purposes, the idea is to generate a medium and small sized Wav2Vec2.0 model and compare the results to the original model. The reasoning is that Wav2Vec2.0 has just over 98 million trainable parameters that make up its highly complicated architecture. The downside is that training on this model is very expensive and time-consuming.

Therefore, we can first decrease the number of trainable parameters to around 61 million trainable parameters to generate a medium-sized Wav2Vec2.0 model. This is done by decreasing the number of convolution layers in the network, along with the feature encoder layers, the number of attention heads, and a handful of hyperparameters such as the learning rate, the optimizer, and learning rate scheduler. Then, we can make an even smaller model of around 36 million trainable parameters to create a small-sized Wav2Vec2.0 model, by following the same methods as the medium-sized model in terms of decreasing the complexity of the architecture. The purpose of this experiment is to test the training time versus accuracy and see if decreasing the size of the Wav2Vec2.0 model can significantly decrease training time while keeping similar levels of accuracy.

iii. Wav2Vec2.0 on Speech Classification

After pre-training a Wav2Vec2.0 from scratch and fine-tuning the model to have different sizes, the last step in this research project is to apply a classification head to the model and see if the speech recognition model can accurately perform speech classification. To add a classification head to the end of the model, a linear layer is added to create a single feed-forward network that accepts the output of Wav2Vec2.0 and compresses that output to the number of classes in the dataset.

The original creators of the model have stated that Wav2Vec2.0 can be applied to any speech-related dataset, and to test that theory, we can use speech classification to see if we can obtain high levels of accuracy. Furthermore, testing the pre-trained model, the fine-tuned model, and the two smaller versions of the model, we set out to see if the size of the model has any effect on the performance the speech classification with regards to training time.

VI. Experimental Setup

In order to start the experiment, we first need to define the aspects that come with training a model. Specifically, the datasets we plan on using, the metric for speech recognition, and all the custom-built methods to make this model pipeline from scratch rather than using pre-made software packages.

i. Datasets

Since there are 3 different aspects to training the models, we need 3 different datasets. For pre-training, LibriSpeech will be used [10]. LibriSpeech is an audio corpus of around 1,000 hours of audio books that are a part of the LibriVox project. We will be using the clean training, validation, and test set, where the training set consists of 360 hours of audio and both the test and validation sets include around 5 hours of audio.

Since the original Wav2Vec2.0 model was also trained on LibriVox, along with the custom build pre-trained model, we need a different dataset to fine-tune the model. For fine-tuning, the TI-MIT (Texas Instruments – Massachusetts Institute of Technology) dataset will be used [6]. TI-MIT is an acoustic-phonetic speech corpus to further help with the development of ASR. This dataset includes 630 speakers of 8 major dialects of American English, each reading 10 phonetically rich sentences. The unique part of TI-MIT is the fact that the dataset also comes with the phonemes of each speech recording, meaning a model can be trained using just the phonemes of speech. However, for our purposes, we will be sticking with the textual data to enhance speech recognition.

Lastly, we need a dataset to test the Wav2Vec2.0 model on speech classification. The dataset we decided to use is the Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) corpus [9], where the goal is to correctly classify the level of emotion among each speech and song file. This corpus contains 24 different speakers, 12 male and 12 female, vocalizing lexically matched statements in a similar and neutral North American accent. For the speech files, the types of emotion include calm, happy, sad, angry, fearful, surprise and disgust, where the song files include the emotions of calm, happy, sad, angry, and fearful. Since we are focusing on speech classification, we only use the speech and song audio files from this dataset, and none of the video data.

ii. Word Error Rate (WER)

When training on any speech recognition datasets where the goal is to translate speech to text, the metric used is Word Error Rate (WER). When training a speech engine, we can't just use accuracy as the metric to measure how close a string is to another, as there are key differences when it comes to predicting text. Therefore, we can use WER instead which considers several aspects of a predicted text versus the true text. WER can be calculated using the formula seen below:

$$WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}$$

Figure 5: Formula to calculate Word Error Rate, where S is the number of substitutions, D is the number of deletions, I is the number of insertions, and C is the number of correct words.

The WER metric takes into account not only the total number of words in the correct spot, but also the number of words that are not in the correct position, along with any words that are spelled incorrectly. Therefore, with WER, we aim to align the recognized word sequence with the reference word sequence using dynamic string alignment and can apply WER to the LibriSpeech and TI-MIT datasets. With regards to the RAVDESS emotional dataset, we can simply use accuracy. This is because this dataset is a classification problem, and the prediction is either correct or not.

iii. Methods from Scratch

One of the main aspects of this research project is to build every part of the data pipeline from scratch. Instead of using pre-built packages, we want to create every method using simple software tools to truly build a complete model from scratch.

Firstly, we want to download the raw datasets into a data directory by downloading the files directly from the source. Once the data has been downloaded, we can create a function to generate a .CSV file that includes all the metadata about each audio file. The purpose of this function is to provide the path to the audio files, and then create this metadata file where we include the audio name, the path to the audio file, a unique identifier, and either the translated target text (for LibriSpeech and TI-MIT) or the emotion class (for RAVDESS).

Now that we have the metadata file, we can create a function to clean up the text column by removing special characters such as periods, hyphens, commas, etc. This will help our model generalize especially for pretraining along with fine-tuning. Once the text is clean, we can implement our own method to create a custom tokenizer. Rather than using the already trained Wav2Vec2.0 tokenizer, we create our own vocabulary file and add the tokens for beginning/end of a sentence, a padding token, and a word delimiter token. Once we have this, we can also create a Wav2Vec2.0 feature extractor, and feed both the tokenizer and feature extractor into the Wav2Vec2.0 processor.

Next, we can use the metadata file to point to the full path of each audio file where we can apply the custom made Wav2Vec2.0 processor in order to get the input IDs to feed into the model, along with the target IDs to help the model learn. Since Wav2Vec2.0 trained using 16 kHz as the sampling rate, we want to mimic this and transform every audio file to a sampling rate of 16 kHz.

Thus far, we have downloaded the data, created a metadata file, a custom Wav2Vec2.0 processor, and the input and target IDs, which can now be fed into the model. However, the model will only accept a batch of inputs that are the same length, which means we need to create a data collator function to add padding to both the input and target IDs to make sure each batch of data have the same size. The processor has a built-in padding attribute that will add padding to each input to make them the same size.

Now that we have all the inputs and targets the same size, all that's left is to call these functions in a custom PyTorch Dataset, which returns each batch as a dictionary including the input IDs and target IDs. This dataset can then be fed into a PyTorch DataLoader, that allows us to iterate over the batches of data. Additionally, we define the Adam algorithm as the model optimizer and use the same learning rate as the original creators of Wav2Vec2.0 use, which is 1e-4.

Looking back, we have created several functions to download the data, create a metadata file, create a Wav2Vec2.0 processor from scratch, got the input and target IDs all padded to the same length, and load the final dataset into a PyTorch DataLoader. All that's left is to define the model and create the PyTorch training loop where we can get the output from the model, the predictions, update the gradients using the optimizer and perform backward propagation. While PyTorch is the preferred way of defining a training loop, we also implemented HuggingFace's

Trainer and TrainingArgs, which also use PyTorch but in a much more sophisticated way. We still follow all the above steps in the pipeline, but HuggingFace's Trainer is slightly more efficient considering the computation limitations.

VII. Results

Now that we know everything about Wav2Vec2.0 along with the methodology for training the model, we can dive into the results of each model along with how well each model did on each task. All the results were achieved using the Google Cloud Platform (GCP) with one NVIDIA Tesla T4 GPU, 32 CPU's and 64 GB of RAM.

i. Training Results on TI-MIT Speech Recognition Dataset

Firstly, are the results from training the model on TI-MIT. For training, a batch size of 16 was used along with 50 epochs. The results, on the test set, can be seen below:

	WER	Loss	Total Time
Original Wav2Vec2.0 (~95 million trainable parameters)	**16.298%**	**0.1254**	**17:14:36**
Medium Wav2Vec2.0 (~61 million trainable parameters)	31.578%	69.999	13:48:26
Small Wav2Vec2.0 (~36 million trainable parameters)	53.434%	103.774	12:40:58

Table 1: Results on the test set of TI-MIT speech recognition dataset including the original Wav2Vec2.0 model along with a medium and small-sized version of Wav2Vec2.0.

Looking at the results in Table 1, we can see that the original Wav2Vec2.0 model, in bold, was by far the best and most accurate when training on the TI-MIT dataset with a WER of 16.298%. The medium-sized model, which had a smaller number of feature encoder layers, had a fairly low WER of 31.578% when compared to benchmark scores. The small-sized model did not achieve the levels of WER we were hoping for at 53.434%, where we shortened the number of feature encoder layers, the number of attention heads, and the number of adaptive layers.

Furthermore, the contrastive loss was only minimized in the original model, whereas the loss was still quite large across the other two models. Lastly, looking at the total training time, we were able to cut training time by 4 hours with the medium-sized model and still get some noticeable results. The smaller model had the quickest training time but was not efficient enough considering its WER score.

ii. Training Results on RAVDESS Speech Classification Dataset

Next, we can look at the results from applying the models generated in the previous section to a speech classification problem, specifically the RAVDESS emotional dataset. Here, we aim to correctly classify an actor's emotion based off the audio file. For training, a batch size of 32 was used along with 70 epochs, and the results can be seen below:

	Accuracy	Loss	Total Time
Original Wav2Vec2.0 (~95 million trainable parameters)	**89.121%**	**0.205**	**31:21:56**
Medium Wav2Vec2.0 (~61 million trainable parameters)	88.897%	0.223	26:20:33
Small Wav2Vec2.0 (~36 million trainable parameters)	85.776%	0.309	23:47:32

Table 2: Results on the test set of the RAVDESS Emotion Classification dataset including the original Wav2Vec2.0 model along with a medium and small-sized version of Wav2Vec2.0, all trained on the TI-MIT dataset.

Looking at the results from Table 2, we can see that once again the original Wav2Vec2.0 model was able to achieve the highest level of accuracy at 89.121%, and a total training time of just over 17 hours. However, the results on the medium-sized Wav2Vec2.0 model was quite impressive with an accuracy of 88.897%. This model was able to achieve similar results as the original Wav2Vec2.0 being just over 0.2 percent less accurate, but with 5 hours of training time removed. This proves that this model was able can produce similar results while being much more efficient in terms of time. Furthermore, the small-sized Wav2Vec2.0 model was also able to achieve great results, with an accuracy of 85.776%. This is impressive considering the results

this model produced when training on TI-MIT, which were not as good as the other models. The total training time for this model was 7 hours quicker than the original Wav2Vec2.0 model and 3 hours quicker that the medium-sized model.

VIII. Limitations & Future Work

With regards to the limitations in pursing this research project, the main one was computation and financial resources, specifically towards pre-training the Wav2Vec2.0 model. Unfortunately, we were unable to achieve any results when it came to pre-training, as the computation resources provided (1 GPU, 32 CPU, 64 GB RAM) were not enough to get valid results as every iterance of an epoch would take upwards of 48 hours on the LibriSpeech 360-hour training set. As mentioned earlier, the original creators of Wav2Vec2.0 had access to 128 GPU's and even then, the model still took over 120 hours of training to get such accurate and fine results. Comparing our resources to the ones the original creators had access to doesn't even come close.

Furthermore, since we were using cloud resources (GCP), we were only able to receive cloud credits in a set number of increments. Once those credits ran out, training would end as the virtual machine would shut down until another credit is applied, meaning it became difficult to let a model train any longer than 72 hours. Furthermore, there is the possibility that we could achieve even better results on fine-tuning Wav2Vec2.0 for speech recognition along with the speech classification datasets given access to more computation resources to let the model train longer and more efficiently. Other than that, if given the necessary computation and financial resources, we would have been able to successfully pre-train the Wav2Vec2.0 model from scratch.

With regards to future work, the main concept is to complete the pre-training of the Wav2Vec2.0 model. This is most definitely achievable as the data preprocessing and the pipeline has been set up correctly, we just need the necessary computation power to compete. In the future, once we can pre-train the model, the next steps would be to fine-tune the pre-trained model on the TI-MIT dataset, and then apply that fine-tuned model towards the RAVDESS emotional classification dataset. It would also be interesting to compare the results in the previous section to the results obtained by pre-training our own model from scratch, as I believe we could compete with the original creators' results and outdo the scores we achieved.

IX. Conclusion

In the beginning, we came across the powerful and notorious Wav2Vec2.0 model that is state-of-the-art when it comes to anything withing the ASR realm and wanted to create our own version by applying methods from scratch. Looking back, what we achieved from implementing all our own methods to train a speech recognition engine, creating two of our own versions of Wav2Vec2.0, to fine-tuning these models on a speech recognition dataset, to then applying this model to speech classification, we were able to build a machine learning pipeline from scratch and apply our own methodology in the creation of an accurate speech engine.

Even though there were a few bumps along the way, we still came out on top with some great findings and learned a lot about working with speech related systems. We will continue to work on this research project with hopes to have our own Automatic Speech Recognition engine that can compete with not only Wav2Vec2.0, but other impressive ASR models such as Speech2Text2.0, XLSR-Wav2Vec2.0, and others. Although we are far from where we want to be, we will continue making forward progress in creating an accurate and helpful ASR engine to support anyone out there who benefits from such advances in technology.

X. References

[1] Baevski, Alexei, et al. “wav2vec2.0: A framework for self-supervised learning of speech representations.” *Advances in Neural Information Processing Systems* 33 (2020): 12449-12460.

[2] Baevski, Alexei. “High-performance speech recognition with no supervision at all.” *Meta AI*, ai.facebook.com/blog/wav2vec-unsupervised-speech-recognition-without-supervision. Accessed 28 Apr. 2022.

[3] Bakanay, Hilal. “The Advantages of Speech Recognition.” *SESTEK*, 13 Sept. 2020, www.sestek.com/2020/08/the-advantages-of-speech-recognition-technology.

[4] Fabien, Maël. “Self-Training and Pre-Training, Understanding the Wav2vec Series.” *Mael Fabien*, 27 Oct. 2020, maelfabien.github.io/machinelearning/wav2vec/#.

[5] Gondi, Santosh. “Wav2Vec2.0 on the Edge: Performance Evaluation.” *arXiv preprint arXiv:2202.05993 (2022).*

[6] Garofolo, J. & Lamel, Lori & Fisher, W. & Fiscus, Jonathan & Pallett, D. & Dahlgren, N. & Zue, V.. (1992). TIMIT Acoustic-phonetic Continuous Speech Corpus. Linguistic Data Consortium.

[7] Guatam, Tanishq. “Wav2Vec2: Automatic Speech Recognition Model | Transformers v4.3.0.” *Analytics Vidhya*, 15 Feb. 2021, www.analyticsvidhya.com/blog/2021/02/hugging-face-introduces-the-first-automatic-speech-recognition-model-wav2vec2.

[8] Kolb, Thomas, David van Leeuwen, and Louis ten Bosch Doctor. "Fine-tuning Wav2vec2. 0 on caption data." (2022).

[9] Livingstone SR, Russo FA (2018) The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English. PLoS ONE 13(5): e0196391. https://doi.org/10.1371/journal.pone.0196391.

[10] Panayotov, Vassil, et al. "Librispeech: an asr corpus based on public domain audio books." *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2015.

[11] Platen, Patrick. “Fairseq - Pytorch/Fairseq.” *GitHub*, github.com/pytorch/fairseq/blob/main/examples/wav2vec/README.md.

[12] Platen, Patrick. “Fine-Tune Wav2Vec2 for English ASR in Hugging Face with Transformers.” *HuggingFace*, 12 Mar. 2021, huggingface.co/blog/fine-tune-wav2vec2-english.

[13] Sus, ŁUkasz. “Wav2Vec 2.0: Self-Supervised Learning for ASR | Towards Data Science.” *Medium*, 6 Jan. 2022, towardsdatascience.com/wav2vec-2-0-a-framework-for-self-supervised-learning-of-speech-representations-7d3728688cae.

[14] “Wav2Vec2” *HuggingFace*, huggingface.co/docs/transformers/model_doc/wav2vec2.