520.638 Deep Learning
Final Project - Combining ASR and NMT as *speech2texto*

Katie Brandegee and Nick Hinke

May 10, 2022

***Abstract* - For this project, an end-to-end model was constructed as a two-stage pipeline with the goal of bridging the gap between automatic speech recognition and neural machine translation. By doing so, the model could serve as an automatic translator that can be spoken into by a user. When developing the model, each stage was designed and evaluated independently to maximize its performance. Upon the completion of both stages, the two were combined together as a single end-to-end model. This combined model was then packaged into a demo script which could record a user's English audio input and subsequently output both the English text transcription and the Spanish text translation. To learn more regarding the model implementation, see `https://github.com/nhinke/deeplearning-repo/tree/master/Project` for all of the code used to complete this work.**

1 Introduction

Within the natural language processing (NLP) community, two of the most historically significant problems are those of automatic speech recognition (ASR) and neural machine translation (NMT). Intuitively, each of these tasks can be thought of as a transformation from one representation of language to another. More explicitly, ASR involves transcribing a raw audio signal into comprehensible text, and NMT involves translating text from the input language to a second target language. These tasks are incredibly pervasive in today's modern world, and the average person may unknowingly interact with solutions to both of these problems on any given day–just think of Apple's Siri and Amazon's Alexa for ASR or Google Translate for NMT [16].

As these are not new problems within the NLP community, the first effective solutions to each of these problems were developed several decades ago. Historically, however, these solutions often involved complex pipelines comprised of sophisticated (and often hand-engineered) algorithms. In fact, it was not until much more recently that solutions leveraging end-to-end deep learning strategies were employed for these tasks [7, 9]. Inspired by many of the recent advances within these respective fields [4], the aim of this work was to produce a single "end-to-end" model that is capable of performing both of these tasks in sequence. In other words, given a raw audio signal in an input language, the resulting model should effectively complete the two language transformations required to output the transcribed text in a target language.

For the experiments conducted in this work, the input and target languages were chosen to be English and Spanish, respectively, but the model architecture and training techniques are generalizable to other languages (given appropriate datasets). Due to the choice of input and target languages used here, the combined model is referred to as *speech2texto*.

2 Proposed Approach

As the ultimate goal of this work was to produce a model capable of outputting translated transcriptions of audio inputs, the most obvious–and somewhat naïve–approach is to treat the model as a single "black-box"-style network. Unless a model of extreme complexity was designed and trained using an enormous dataset, pursuing this approach would likely yield poor results. Moreover, this approach would not be very generalizable to other languages, as each input-target language pair would require its own exceptionally complex model and specially tailored dataset.

Rather than treating it as a single black-box network, it would be highly advantageous to consider the ultimate model as a combined two-stage pipeline. Within this framework, it would then be possible to treat the two stages of the pipeline as an ASR stage followed by an NMT stage. Crucially, this approach makes it possible (and beneficial!) to individually design, train, and evaluate each stage of the model before combining them, thus resulting in the combined model architecture as shown in Figure 1. Not only does this allow for the application of already well-established knowl-

edge and techniques for each sub-problem, but this also makes the resulting combined model significantly more generalizable to other languages. This is primarily due to the availability of many open-source datasets–datasets which span a very wide variety of languages–that are designed for either ASR or NMT. Consequently, to change the input or target language of the combined model, one would need only to individually retrain the existing architecture for each stage on the appropriate dataset.

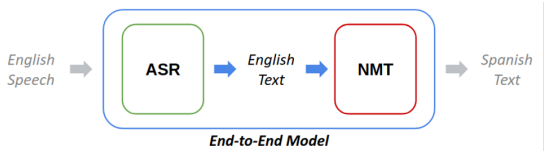

Figure 1: High level overview of combined model

2.1 Detailed Approach: ASR Stage

While today there are several well-designed architectures that leverage deep learning for ASR–arguably the most popular of which being Facebook's *wav2vec* self-supervised model [5]–one of the cornerstone advances in the field came with the initial proposal of *Deep Speech* as developed by Baidu Research in 2014 [7, 4]. The reason for the success of *Deep Speech* was simple: conceptually it was straightforward to follow the logic of the design choices, and each internal "block" within the model leveraged the strengths of a different deep learning technique. For this reason, the original *Deep Speech* model served as the primary inspiration for the ASR stage of *speech2texto*.

While *Deep Speech* provided inspiration for some of the design choices made within the ASR stage, it was hypothesized that the performance could be further improved by altering several key areas of the model. As consequence, the architecture used within the ASR stage was constructed and trained from scratch. A summary of the resulting sub-model architecture is provided in Figure 2.

```
==========================================================
Layer (type:depth-idx)                   Param #
==========================================================
ASR                                      --
├─Sequential: 1-1                        --
│    └─Conv2d: 2-1                       104
│    └─ReLU: 2-2                         --
│    └─Conv2d: 2-3                       808
│    └─ReLU: 2-4                         --
│    └─Conv2d: 2-5                       3,216
│    └─ReLU: 2-6                         --
├─Sequential: 1-2                        --
│    └─Linear: 2-7                       370,272
│    └─LayerNorm: 2-8                    1,216
│    └─ReLU: 2-9                         --
│    └─Linear: 2-10                      370,272
│    └─LayerNorm: 2-11                   1,216
│    └─ReLU: 2-12                        --
├─RNN: 1-3                               --
│    └─LSTM: 2-13                        63,750,144
├─Sequential: 1-4                        --
│    └─Linear: 2-14                      28,672
├─TrainingSoftmax: 1-5                   --
├─InferenceSoftmax: 1-6                  --
├─CTCLoss: 1-7                           --
==========================================================
Total params: 64,525,920
Trainable params: 64,525,920
Non-trainable params: 0
==========================================================
```

Figure 2: Summary of ASR stage architecture

In order to understand the reasoning behind the chosen architecture for the ASR stage, one must first consider how data is passed into the network. Since the input to the ASR stage within the combined model is designated as a raw audio signal, it is necessary to first transform that audio signal into a comprehensible format for the network. Starting from a raw .mp3 or .wav audio file, this pre-processing occurs in several steps, namely:

1. Load audio signal and re-sample it
2. Transform audio signal into Mel spectrogram
3. Perform data augmentation as desired

After completing the above transformation, each audio signal in the dataset–all of which have now been sampled at the same rate–will have been converted into its corresponding Mel spectrogram representation. This representation is especially useful as it can essentially be thought of as a visualization of the audio signal based on the frequency components that are most important to human hearing. As a result, they can be used to identify distinguishing characteristics between the phonemes comprise speech. Crucially, the Mel spectrograms are represented as 2D images, thus providing an easily understood input for the network. Two examples of these spectrograms are provided in Figure 3.

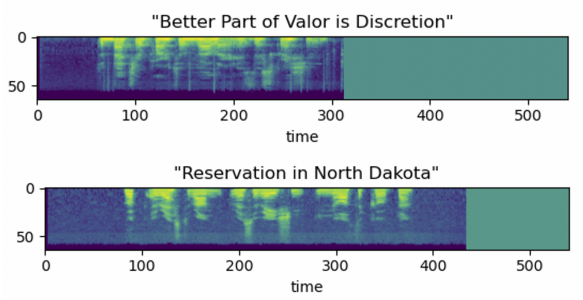

Figure 3: Examples of Mel spectrogram inputs

Since the Mel spectrograms are represented as images, a convolutional neural network (CNN) is the obvious choice for attempting to extract useful features from them. As such, the spectrogram inputs are first passed through a CNN block which is comprised of several 2D convolutional layers with rectified linear unit (ReLU) activations between them. In doing so, this block of the ASR stage transforms the spectrogram input from a single-channel 2D image into a set of sixteen identifying feature maps. Ideally, the feature maps that are output from the CNN will provide features that are both more informative and more robust than the raw pixel intensities from the spectrogram images themselves.

After extracting a robust set of feature maps from the input spectrograms, it would then be extremely useful to somehow incorporate the temporal information present in the data. To accomplish this task, any flavor of recurrent neural network (RNN) would be particularly well-suited. Primarily since it would be desirable to capture more contextual information and thereby improve the network's performance on longer audio samples, a long short term memory RNN (LSTM) was chosen in place of standard RNN layers. More specifically, a series of three bidirectional LSTM layers was utilized in this block of the network in order to capture the temporal information within the CNN feature maps. However, before passing the CNN feature maps into the LSTM block, they are first propagated through a straightforward linear block whose job is to learn a mapping that makes the CNN block outputs more robust and more comprehensible as inputs to the LSTM block. This block is comprised of a few fully-connected layers with ReLU activations, and utilizes layer normalization to help prevent overfitting and improve the robustness of the learned parameters.

After incorporating the temporal information extracted by the LSTM block into the robust feature maps obtained by the CNN block, the network is finally capable of making well-informed predictions. In order to do so, the LSTM block outputs are passed through a single fully-connected layer with a softmax activation. Consequently, the outputs correspond to a set of character probabilities at each time step, which can then be interpreted as character predictions in a few different ways.

During training, the character predictions are determined simply by taking the character with the highest probability after the softmax activation–which is otherwise known as "greedy decoding". This approach is particularly useful during training since it attempts to force the network to produce the correct character at a given time step regardless of the probabilities of other neighboring characters. As an unfortunate consequence, however, this approach does not make use of all the available contextual information, and thus should not be used during inference.

Since there are many linguistic and grammatical rules associated with language, it is possible to effectively re-score the output character probabilities from the softmax layer according to which sets of characters best adhere to those rules. This can be done by utilizing a rules-based language model in combination with a connectionist temporal classification (CTC) beam search re-scoring algorithm–which is otherwise known as "CTC decoding". Consequently, using CTC decoding during inference allows the ASR stage to produce much more comprehensible words rather than simple greedy decoding. A very elementary depiction of CTC beams from [15] is provided in Figure 4.

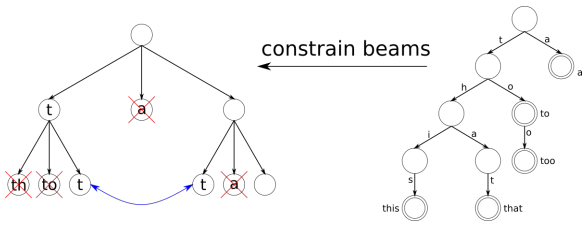

Figure 4: Simple illustrative example of CTC decoding

While the previously described network architecture may prove to be excellent at learning the parameters necessary to transcribe the training audio samples as text, very little has been done so far to prevent overfitting to that training data. As such, two key strategies were implemented to improve the robustness of the learned parameters. First and foremost, data augmentation was utilized to effectively increase the size of the training set by treating slightly corrupted versions of the original inputs with identical labels. More specifically, This was done using the strategies of time and frequency masking on the input Mel spectrograms as defined in the technique known as SpecAugment [13]. Additionally, a dropout probability of 20% was utilized within the LSTM block during training, thereby forcing it to learn more robust and generalizable parameters. Ideally, the combination of these techniques with the layer normalizations already in the network and the extremely large available training datasets should sufficiently mitigate the risks of overfitting.

Finally, it is also worth noting that the network architecture as described above was very carefully and deliberately implemented in order to ensure that it would remain agnostic to both the batch size and the length of the audio sequence (*i.e.* the network was capable of processing any size audio inputs without crashing). It is actually for this reason that the spectrogram examples in Figure 3 were padded accordingly–since they were from the same batch of data, they were each padded to the length of the longest audio sequence in that batch, while their true length information was preserved. This property of the network became especially advantageous when combing the two stages of the end-to-end model as it could process a single live audio sample during inference just as easily as a pre-recorded batch of training data.

2.2 Detailed Approach: NMT Stage

The neural machine translation portion of this project takes English inputs and learns their Spanish translation using an encoder-decoder structured LSTM RNN with attention. There is a significant amount of data preprocessing that must be done before the data can be passed through the network. Both the inputs and labels (English and Spanish respectively) must be encoded into a numbering scheme that can be passed to the network. This is done by first taking each sentence and splitting it into its word building blocks. Then two dictionaries are constructed, each containing every unique word that occurred in the input dataset for its respective language. Then each word is encoded to correspond to the correct dictionary entry. Start and end of sentence tokens are also added to this encoding scheme, and each sentence was padded with end of sentence tokens to force each sentence to be encoded to the same length.

There are two main structures in the RNN network, the encoder and the decoder. The encoder embeds the encoded English words into a fixed dimensional hidden space, which the decoder then uses to map to the Spanish encoded outputs. The encoder and decoder both have two hidden layers, are bidirectional, and use dropout. The encoder operates by first embedding the data and then applying dropout. These embeddings are fed into an LSTM, which also requires the hidden state information and the memory information that contributes to strengthening the encoder's long-term memory. These states are recomputed at each time step, where each time step analyzes a new word in the sentence. The LSTM operates by analyzing the current word input, the hidden state at the previous time step, and the memory state. These memory states work to increase the

RNN's long-term memory, by storing important information from previous time steps that exceed the short-term memory [10].

Global attention was implemented in the decoder stage. Attention uses the hidden states from different time steps to understand which parts of the sentence are more important and tend to drive the sentence structure and language. This works by computing attention scores which measure the similarity between the hidden state at the current time step and each hidden state from the previous time steps. Then the SoftMax of the attention scores is taken in order to weight the importance of each hidden state in determining the prediction at the current time step. Hyperbolic tangent nonlinearity is then applied to this information, and finally the logarithmic SoftMax function is used to determine the sentence prediction. This results in a prediction tensor that contains the likelihood that word is each dictionary entry in Spanish, for each word in the sentence. However, since the negative log likelihood loss function was used, the logSoftMax was applied to the predictions to ultimately be fed back into the loss in comparison to the targets. The encoder and decoder structure and parameters are shown in Figure 5.

```
EncoderRNN(
  (embedder): Embedding(16151, 1000)
  (dropout): Dropout(p=0.2, inplace=False)
  (lstm): LSTM(1000, 1000, num_layers=2, dropout=0.2, bidirectional=True)
  (fc): Linear(in_features=2000, out_features=1000, bias=True)
)
DecoderAttn(
  (embedder): Embedding(29794, 1000)
  (dropout): Dropout(p=0.2, inplace=False)
  (score_learner): Linear(in_features=2000, out_features=2000, bias=True)
  (lstm): LSTM(1000, 1000, num_layers=2, dropout=0.2, bidirectional=True)
  (context_combiner): Linear(in_features=4000, out_features=1000, bias=True)
  (tanh): Tanh()
  (output): Linear(in_features=1000, out_features=29794, bias=True)
  (soft): Softmax(dim=1)
  (log_soft): LogSoftmax(dim=1)
)
```

Figure 5: Encoder and decoder architectures

3 Model Performance

After developing initial designs for each stage of the combined model, each sub-model was implemented using the well-known PyTorch framework. Suitable model parameters could then be found for each stage by training the sub-models on popular publicly available datasets, namely the Mozilla Common Voice dataset [2] and the Tatoeba dataset [3]. Thanks to the usability of the PyTorch framework and carefully-organized sub-model implementations, it was much easier to iterate and improve upon the initially proposed designs for each stage. This process was repeated until finally settling on the architectures which were outlined in the previous section.

In order to iterate and improve upon each sub-model, it was necessary to use some measure of error on a consistent test dataset to fairly compare the results. Since the ultimate goal of the model was to convert spoken words into translated text words, the primary error metric utilized within both stages was the word error rate (WER). This metric is related to the Levenshtein distance used for comparing sequences, and is defined as:

$$WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C} \qquad (1)$$

where C is the number of correct words in the predicted sequence, S is the number of substitutions or erroneous words, I is the number of insertions or extra words, D is the number of deleted or missing words, and N is the number of words in the target sequence [11]. In addition to the WER, the character error rate (CER) was also used within the ASR stage due to the nature of its outputs as character probabilities. The CER is defined identically to the WER but references characters rather than words (*e.g.* I is the number of extra characters in the predicted sequence) [11].

After evaluating and improving the performance of each sub-model stage, the two were finally combined as a single end-to-end model. Unfortunately, since it is exceedingly rare that the target outputs of the ASR test dataset match the inputs of the NMT test dataset, it is very non-trivial to quantitatively evaluate the performance of the combined model using the aforementioned metrics. As such, an interactive demo script was

written to qualitatively evaluate the end-to-end model performance. This demo will be described in greater detail in Section 4.

3.1 Detailed Results: ASR Stage

After implementing the previously described model architecture, the model was trained from scratch on a NVIDIA GeForce RTX™ 3060GPU. As the specified input language for *speech2texto* is English, the English corpus of the publicly available Mozilla Common Voice dataset was used for training [2]. While training the sub-model on this dataset, the CTCloss criterion as implemented by PyTorch was utilized in combination with the AdamW optimizer for updating the parameters during backpropagation. It is also worth reiterating that each training sample was given a 75% chance of being augmented using the previously described techniques–in random ways and to varying degrees–when it was loaded for training.

As is often the case for many models targeting NLP tasks, training this stage of the end-to-end model took a tremendous amount of time and computation–despite its relatively low complexity with only about 64M parameters as compared to the 175B parameters of GPT-3 [6]. Indeed, it was postulated by Baidu researchers that an ASR model of this nature requires over 11,000 hours of training audio and an enormous amount of computational resources to produce good results [7]. As such, it was necessary to train the ASR sub-model in stages due to time and resource constraints.

In order to effectively train the model in stages, different portions of training data from several different versions of the Mozilla Common Voice dataset were used. Using this approach, the strategy was to train the model on one version of the training dataset (*e.g.* version 1.0) until the validation error plateaued, and then train the model on a different version of the training dataset (*e.g.* version 2.0) while monitoring the validation error on the original validation dataset. This process could then be repeated to ensure that the validation error continued to go down (using a constant validation dataset) while incorporating more and more training data spanning several different versions of the Common Voice dataset. An example of the resulting loss plots is provided in Figure 6.

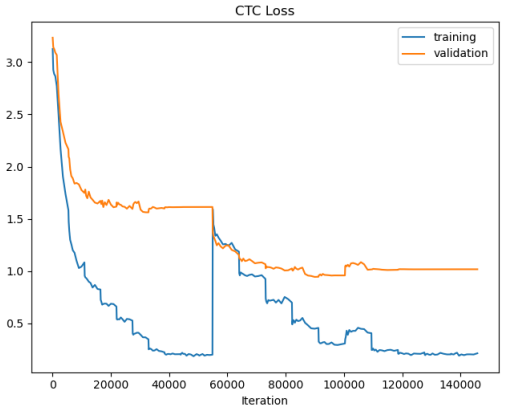

Figure 6: CTC loss plots after switching datasets

Notice in Figure 6 that the validation error continued to decrease after switching training datasets, thereby implying that the utilization of the new training data improved the overall model performance. Additionally, the previously described error metrics were continuously monitored during training on both the training and validation sets. Plots of the average WER and average CER over every 100 batches of data during the same period of training as Figure 6 are provided in Figure 7 and Figure 8, respectively.

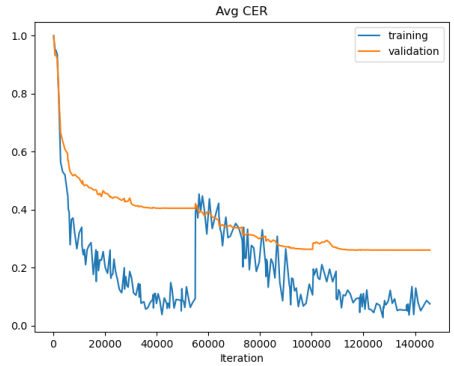

Figure 7: Average CER after switching datasets

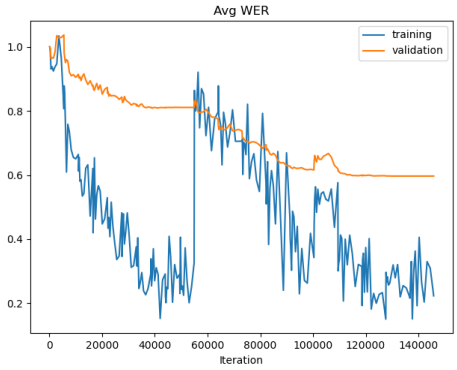

Figure 8: Average WER after switching datasets

Once again, it is worth pointing out how the network's performance on the validation dataset continued to improve where it otherwise would have plateaued. As consequence, this strategy was used to continually improve the model performance on the validation dataset. That being said, however, it is hard to miss the glaringly high WER metrics in Figure 8 in comparison to the much more impressive CER metrics in Figure 7. As previously discussed, this is partially due to the use greedy decoding during training.

Although the error metrics using CTC decoding were not monitored during training, the advantage could easily be seen just by printing a few examples such as that in Figure 9. As clearly shown in the example, the use of CTC decoding provides for much more comprehensible outputs from the model. Since this behavior matched the expected behavior as previously described, it confirmed the decision to use CTC decoding over greedy decoding when performing inference. Additionally, it should be noted that the CTC beam search algorithm and rules-based language model were obtained via open-source repositories at [14] and [8], respectively.

```
Actual label:       'was introduced as early as seventeen ninety by mr blackburn'
Greedy prediction:  'was intrudused as erly as sevente nine dy by mistr blackrngs'
CTCbeam prediction: 'was introduced as early as seventeen nine dy by mister black rns'
```

Figure 9: CTC decoding improving performance

After using the previously described training strategy, the model performance was then further evaluated using the test dataset from version 1.0 of the Mozilla Common Voice dataset. To demonstrate the effectiveness of the utilized training strategy, the resulting error metrics on the test dataset were directly compared to those of another instance of the same model which had only been trained until the first validation loss plateau. After computing the CER and WER for each batch of the training dataset, the results as shown in Figure 10 were found for each of the two models.

```
Partially-trained model CER:  ( mean = 0.48 , max = 0.61 , min = 0.36 )
Partially-trained model WER:  ( mean = 0.89 , max = 1.13 , min = 0.73 )

Totally-trained model CER:    ( mean = 0.14 , max = 0.29 , min = 0.03 )
Totally-trained model WER:    ( mean = 0.28 , max = 0.53 , min = 0.14 )
```

Figure 10: Model performance on test dataset

After completing all of the aforementioned training and testing on the publicly available datasets from Mozilla [2], it was then desired to experiment with fine-tuning the model on a personally developed dataset containing only the user's own voice. The motivation behind this idea was to determine how much user-generated data would be required to bias the model for better performance on just his or her voice. To that end, several bash scripts were written to utilize the open-source mimic recording studio [12] to first record the training data and subsequently transform it to be stored in the same manner as the Mozilla datasets. Unfortunately, there has not yet been enough data gathered to make any assertions regarding this question.

Using the strategies outlined in this section, the model architecture as previously described was successfully trained from scratch. Although there is certainly still room for further improvement–especially given more time and computational resources–this will be a subject for future work. For now, however, the ASR stage within the combined model is reasonably capable of transforming spoken English language into transcribed English text that can be fed as an input to the NMT stage.

3.2 Detailed Results: NMT Stage

The dataset used to train the NMP portion of the network was English to Spanish translated sentences from the Tatoeba project [3]. The whole dataset consisted of 130,000 sentence pairs. The NMT model was trained on 80,000 sentence pairs over 16 epochs with a batch size of 20. The encoder and decoder parameters were optimized using stochastic gradient descent with a learning rate of .01. The negative log likelihood was used as the loss function. The loss decreased consistently with epoch iteration, which is shown in figure 11.

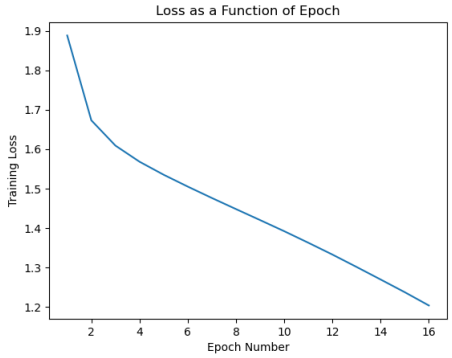

Figure 11: Loss in terms of epochs for NMT training

It is also worth presenting qualitative results, in order to understand how the model might be mispredicting inputs. From figure 12, it appears that the model favors words that appear in conversation more often like pronouns and question words. It also seems that the model understands how capital letters and punctuation are used. Ultimately, it seems that more time is required to both train the model over the entire dataset with data augmentation for many more epochs in order to achieve the desired results.

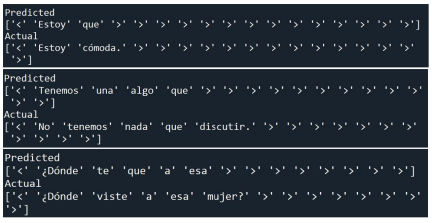

Figure 12: Target and predicted Results

4 End-to-End Demo

As previously mentioned when discussing model performance, an interactive demonstration was developed using Python–and the PyAudio library in particular–in order to qualitatively evaluate the performance of the completed end-to-end model. In essence, this demo worked by first recording audio from the user's device for a user-specified amount of time, and then sequentially propagating that audio signal through both sub-models (with pre-processing before each stage as required) in order to exactly simulate the execution of the combined model. Rather than first constructing a true combined model that would perform both sub-tasks all in one forward pass, this demo was intentionally done in steps in order to retrieve the output from the ASR stage. Otherwise, the transcribed English text representation between the two stages could only be considered as a hidden encoding of the input English audio.

As previously mentioned, this interactive demo provided a qualitative and enjoyable way to measure the performance of the resulting end-to-end model. A brief video which displays the usage of this demo for a small variety of basic English audio inputs can be viewed online at `https://youtu.be/H9oafUxmLao`. Additionally, a screenshot of several demo outputs is provided in Figure 13.

Figure 13: Demo results for a variety of spoken inputs

Unfortunately, as can be observed in Figure 13, the NMT portion of the demo typically "failed" to produce any translation. That being said, these failures were not due to any inability of the NMT stage to process valid inputs; rather, the failures came as a result of the NMT stage's current inability to process any input word that is not already in its vocabulary. As such, it will be the subject of future work to remove this constraint in order to more effectively demonstrate the performance of the combined two-stage model.

Currently, the demo must be run from a Python environment with all of the required dependencies installed and all of the model parameters saved in the appropriate locations. An interesting subject of future work would be to package all of these executable scripts, parameter files, and dependencies in a Docker container and push it to a publicly available server such as the Elastic Container Registry (ECR) hosted by Amazon Web Services (AWS). Doing so would make the completed *speech2texto* model publicly and easily accessible for use as an automated speech translator.

In the spirit of making *speech2texto* into a usable (and helpful!) service, the internal models could first be JIT traced and compiled before containerizing the demo in order to improve their runtime efficiency on resource-constrained devices (*e.g.* smart phones). This would allow for them to be ported to CoreML models which are officially supported by the iOS App Store [1]. Upon doing so, it would be relatively straightforward to develop an iOS app that utilizes *speech2texto* and is publicly available from one of the most widely used application distributors in the world.

5 Conclusion

One of the major downfalls of this model is that the English inputs to the NMT portion were often either not spelled correctly, or spelled correctly, but not in the dictionary that the RNN was trained to recognize words, despite the implemented language model. Therefore, it would be beneficial to train the LSTM RNN in the NMT half more heavily on a broader dataset, so that it would be able to recognize most input words. Though it was originally emphasized that a black box end-to-end model would not perform well, it might help to create an automated method that takes in ASR test data and NMT test data to produce an end-to-end dataset to train or fine-tune the combined model for better overall performance. This would also enable the WER to be computed for the combined model, which is an important and necessary metric for evaluating end-to-end performance. Though the model presented here was trained on English to Spanish pairs, the structure of the model would remain the same for different language pairs, making this model easy to generalize. Lastly, given more time, this model could be extended to output Mel spectrogram in Spanish for audio output.

References

[1] Machine learning: Coreml. https://developer.apple.com/machine-learning/core-ml/. Accessed: 2022-05-05.

[2] Common voice. https://commonvoice.mozilla.org/en/datasets, 2022. Accessed: 2022-03-14.

[3] Tatoeba project: Spanish-english. http://www.manythings.org/anki/, 2022. Accessed: 2022-03-14.

[4] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. F. Diamos, E. Elsen, J. Engel, L. J. Fan, C. Fougner, A. Y. Hannun, B. Jun, T. Han, P. LeGresley, X. Li, L. Lin, S. Narang, A. Ng, S. Ozair, R. J. Prenger, S. Qian, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, A. Sriram, C.-J. Wang, Y. Wang, Z. Wang, B. Xiao, Y. Xie, D. Yogatama, J. Zhan, and Z. Zhu. Deep speech 2 : End-to-end speech recognition in english and mandarin. *ArXiv*, abs/1512.02595, 2016.

[5] A. Baevski, H. Zhou, A. rahman Mohamed, and M. Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *ArXiv*, abs/2006.11477, 2020.

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. J. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020.

[7] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. F. Diamos, E. Elsen, R. J. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Ng. Deep speech: Scaling up end-to-end speech recognition. *ArXiv*, abs/1412.5567, 2014.

[8] K. Heafield. Kenlm: Faster and smaller language model queries. https://github.com/kpu/kenlm. Accessed: 2022-05-04.

[9] N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *EMNLP*, 2013.

[10] Q. Lanners. Neural machine translation. https://towardsdatascience.com/neural-machine-translation-15ecf6b0b. Accessed: 2022-03-14.

[11] K. Leung. Evaluate ocr output quality with character error rate (cer) and word error rate (wer). https://towardsdatascience.com/evaluating-ocr-output-quality-with-character-error-rate-cer-and-word-error-rate-wer-853175297510. Accessed: 2022-03-14.

[12] MycroftAI. Mimic recording studio. https://github.com/MycroftAI/mimic-recording-studio. Accessed: 2022-05-04.

[13] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le. Specaugment: A simple data augmentation method for automatic speech recognition. *ArXiv*, abs/1904.08779, 2019.

[14] parlance. Pytorch ctc decoder bindings. https://github.com/parlance/ctcdecode. Accessed: 2022-05-04.

[15] H. Scheidl. Word beam search: A ctc decoding algorithm. https://towardsdatascience.com/word-beam-search-a-ctc-decoding-algorithm-b051d28f3d2e. Accessed: 2022-03-14.

[16] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.