Final Project: Introduction to Speech and Audio Processing

Ravid Levy, ID: 206576142

Shoval Messica, ID: 207377532

Final model:

1. **Model Architecture:** The core architecture of the model consists of an bidirectional LSTM recurrent neural network, followed by a linear layer and a softmax activation. We use the CTC loss function which handles the alignment issue between the variable-length audio inputs and output sequences. The model leverages a bidirectional architecture to capture context from both past and future frames of the audio.

2. **Components:**
    a. **LSTM Layer:** The LSTM layer is used to capture temporal dependencies and patterns in the input audio sequences. It processes sequential data, so it is suitable for capturing context.
    b. **Linear Layer:** The output from the LSTM layer is passed through a linear layer. This layer performs a linear transformation on the input features, which prepares them for the final classification step.
    c. **LogSoftmax Activation:** The linear output is passed through a log-softmax activation function. The function converts the raw scores into a probability distribution over different classes (vocabulary tokens), which is necessary for the CTC loss.

3. **Bidirectional LSTM:** We set the LSTM layer to be bidirectional, in order to process the input sequences both forward and backward. This allows the model to capture context from both past and future timesteps, which is important for accurately transcribing spoken language.

4. **Parameters and Hyperparameters:**
    a. **Architecture related parameters:**

i. **input_dim:** dimensionality of the input features (MFCCs).
ii. **hidden_dim:** number of hidden units in the LSTM layer.
iii. **output_dim:** number of classes in the output vocabulary (depends on the tokenizer we use).
iv. **num_layers:** number of LSTM layers stacked on top of each other.
v. **dropout:** dropout probability for regularization, applied to the LSTM layer.
vi. **ckpt_path:** path to a pre-trained model's weights, if available.

b. **Optimization related parameters:**
   i. **batch_size:** the number of training examples used in a single iteration of the optimization
   ii. **num_epochs:** the number of times the entire training dataset is passed through the model during training.
   iii. **print_interval:** indicates how often we print the loss or log the training progress.
   iv. **save_ckpt_interval:** how often we save the checkpoints of the model during training.
   v. **learning_rate:** controls the step size at which the optimization algorithm (ADAM optimizer) adjusts the model's weights.
      1. **lr_scheduler:** We reduce the learning rate by a factor of 0.1 every 80 epochs, starting with 0.001.
   vi. **regularization:** the form of controlling the model complexity, including L2 sum of the model's parameters for penalizining that results in optimization of the model generlization.

5. **Initialization and Loading Pre-trained Weights:** If a pre-trained model checkpoint path is provided, the model's weights are initialized using the saved parameters.
6. **Improvements:**
   a. **Comparing GMM-HMM to LSTM with CTC Loss:** We have discovered that LSTM with CTC Loss is outperforming GMM-HMM in both WER and CER.

b. **Sub-Word Tokenizer:** We integrated BPE sub-word tokenizer, which operates by decomposing words into smaller sub-word units. We tested its effectiveness across different hidden dimension sizes.
c. **Augmentations:** We have serially drawn probabilities for each of the following augmentation: white noise, time stretch, pitch scale, random gain and polar inverse. Adding regularization has enhanced the optimization process.
d. **Sub-Word Tokenizer with Augmentations:** Combining improvements B and C (with single augmentation out of the above due to the lack of computing resources), has brought significant improvements and reduced overfitting over the training data.

Performance (We both need to fill this)

Train: Average CER: 0.085670% Average WER: 0.0852%
Val: Average CER: 7.630787% Average WER: 10.2500%
Test: Average CER: 6.196351% Average WER: 9.5479%

Below we describe our research work in detail.

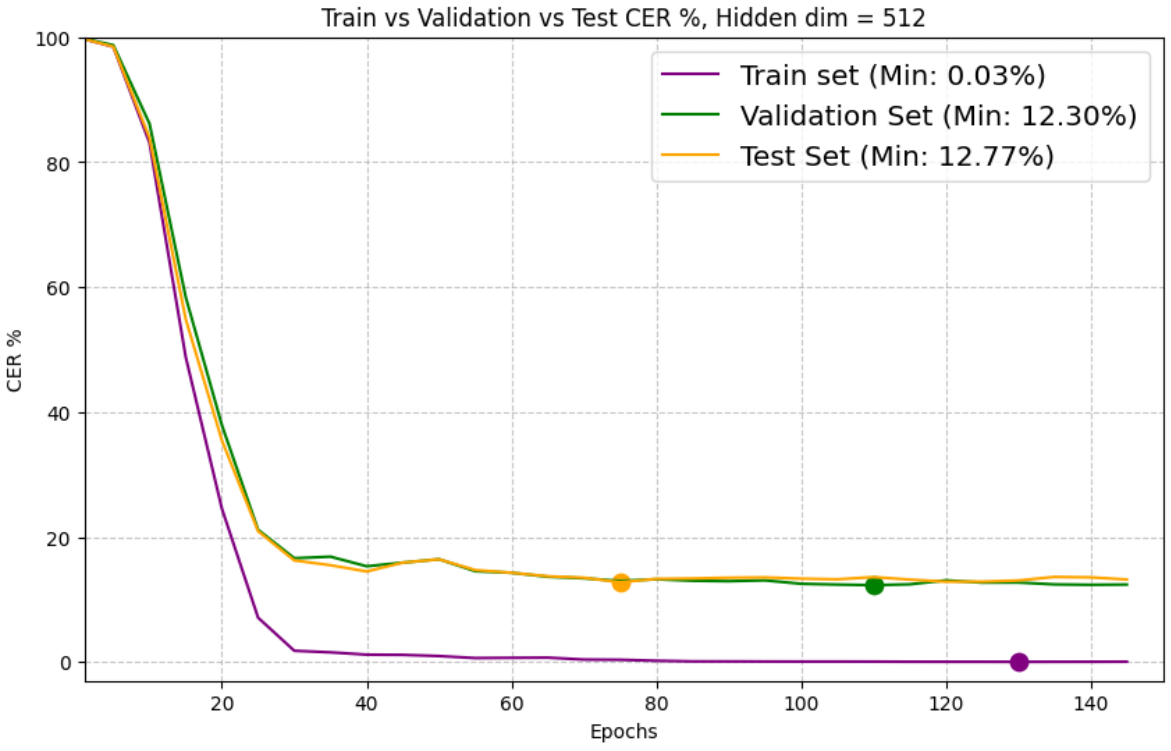
Phase 1: Naïve Baselines

We started with exploring two of the most naïve baselines:

The first model consists of 2 bidirectional LSTM layers, followed by a linear layer and a softmax activation.

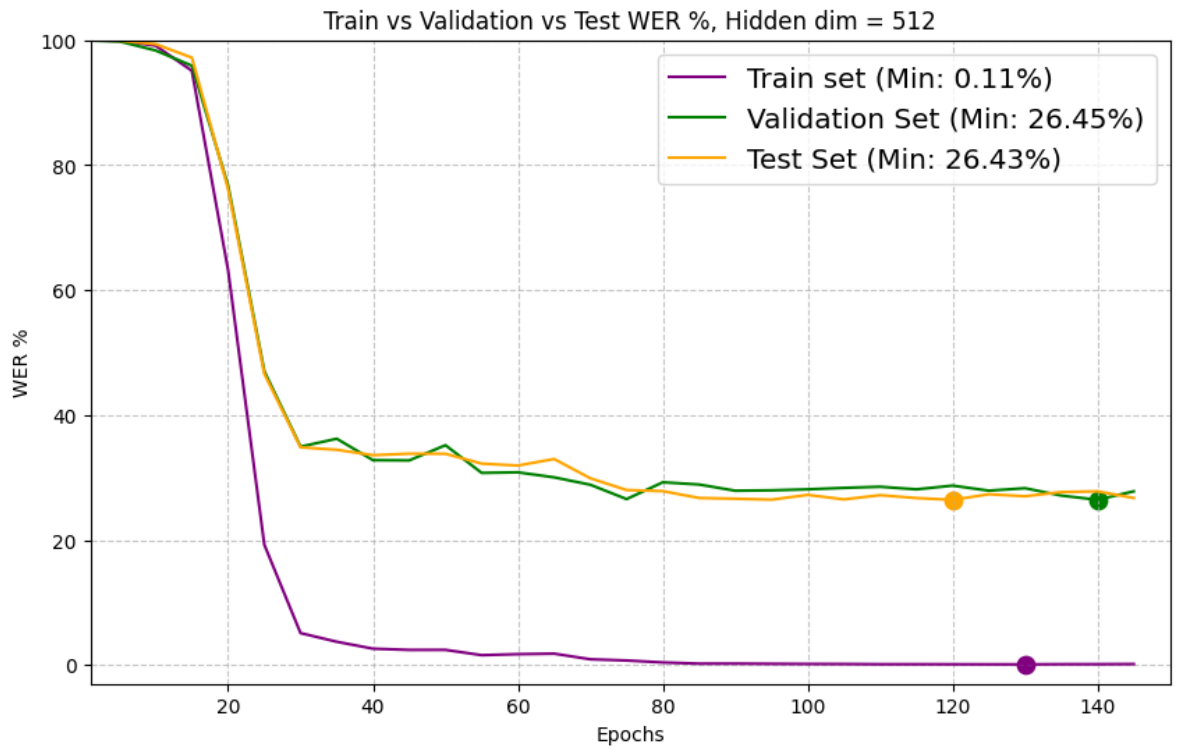
We wanted the model to be suited for the data it is trained on: the dataset is relatively small, so there's a high risk of overfitting, especially when using a complex model. A large model can capture noise in the training data, leading to poor performance on new data. Following the above, we came to the conclusion that 2 LSTM layers is optimal in this case. We also saw that adding more LSTM layers reduces the performance of the model, which is consistent with our assumptions.

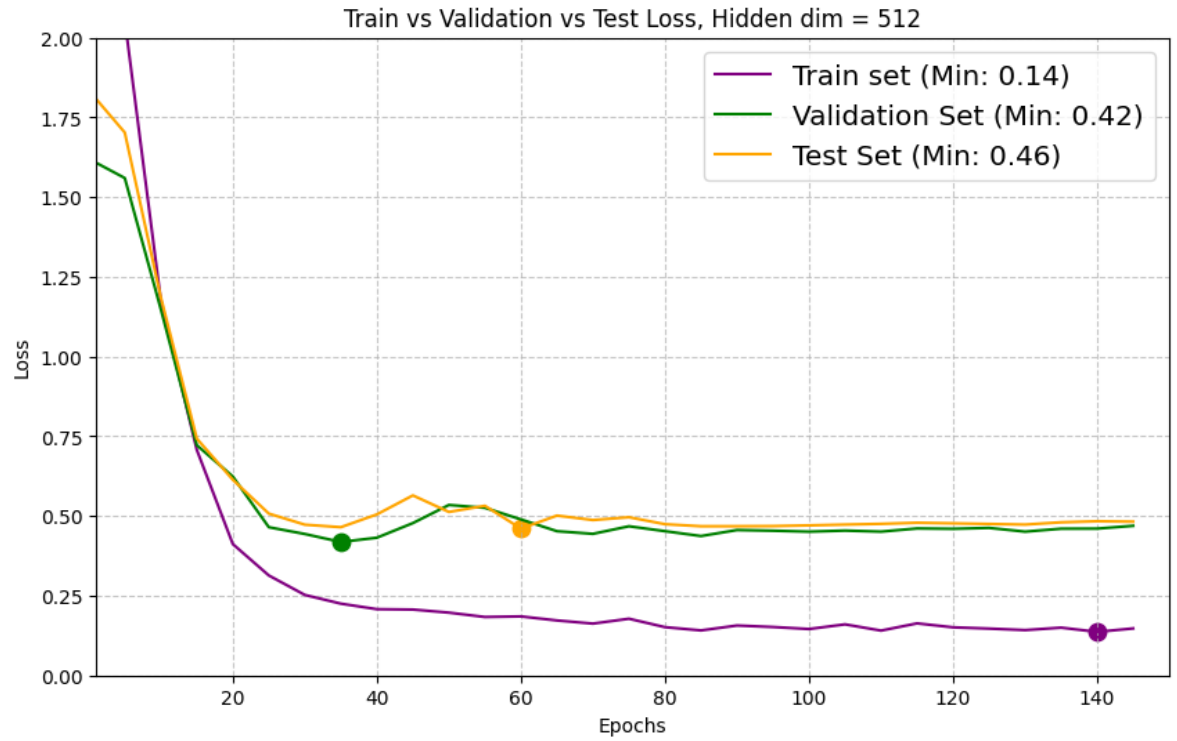
We optimize the model using the following hyperparameters and configuration: input_dim = 39, output_dim = 199 (198 vocabulary size + 1 blank symbol), hidden_dim = 512, batch_size = 16, num_epochs = 150, learning_rate = 0.001, print_interval = 15, save_ckpt_interval = 45.



Results for train/val/test sets are as follows:

(Of course that the model was trained only on the training set, but through the epochs we calculated its **average** performance on the validation and test sets).





The second model has been composed of two sub models: an acoustic model which was a dictionary of word-to-GMMHMM of the hmmlearn package. The language model was an NGram, which its N was chosen according to the complete model's performance over the validation set. The N-Gram with the inverse log likelihood, multiplied by the language model N-Gram, producing the highest score as for the next N-Gram to be produced, was chosen. Unfortunately, the results included high error rates, even when considering the 4-Gram which was with the highest performances (WER: 0.96, CER: 0.97). Nonetheless, we wanted to mention this model during the research process.

The results suggest that the first naïve baseline works pretty well, and that the architecture choice was suited for the ASR task as well as for the training data.

Phase 2a: Sub-Word Tokenization with Custom Vocabulary Enhancement

In this section of the project, we investigate the impact of sub-word tokenization on the performance of an ASR model. We explore how different hidden dimension sizes interact with the sub-word tokenization strategy (traditional character-based vs sub-word tokenization like BPE) and discuss the implications of these findings.

Tokenization Methods:

1. Character Tokenization: In this approach, text is split into individual characters, treating each character as a token.
2. Sub-Word Tokenization (BPE): BPE divides the text into meaningful sub-word units, based on the frequency and co-occurrence patterns of characters and character sequences in the training data. We specified additional common tokens that we wanted to include in the vocabulary: number names (e.g., "ZERO", "ONE") and month names (e.g., "JANUARY", "FEBRUARY"). This strategy allows the model to learn from a larger vocabulary.

Hidden Dimension Sizes:

1. Small (256): The smallest hidden dimension size.
2. Medium (512): A moderately larger hidden dimension size.

3. Large (1024): A significantly larger hidden dimension size.

Results and Analysis:

- Small Hidden Dimension: For the smallest hidden dimension size, both Word Error Rate (WER) and Character Error Rate (CER) show improvements when using sub-word tokenization (BPE) compared to character tokenization. This suggests that the sub-word tokenization captures more meaningful linguistic patterns within the limited hidden space, enabling the model to perform better in recognizing words and characters accurately.

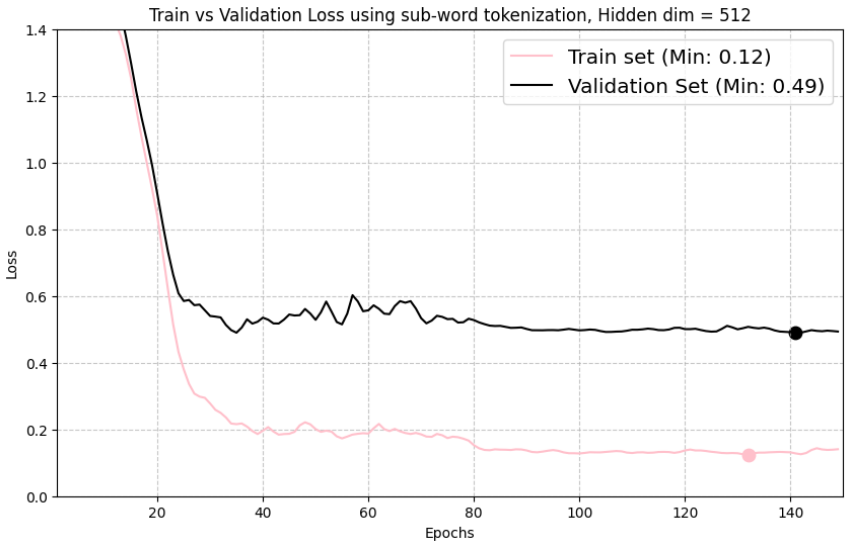
- Medium Hidden Dimension: In the case of a medium hidden dimension size, an interesting trend emerges. While CER experiences a slight deterioration when using BPE tokenization, WER demonstrates a significant improvement (around 6% as can be seen in the graphs). The improvement in WER indicates that the sub-word tokenization enhances the model's ability to identify and align words more effectively. However, the degradation in CER suggests that the model might struggle with maintaining precise character-level alignments and predictions.

- Large Hidden Dimension: For the large hidden dimension size, the results take a different turn. CER worsens when using BPE tokenization, implying that the sub-word units might not align optimally with the chosen hidden space's capacity. However, WER remains relatively unchanged. This suggests that with a larger hidden dimension, the model's capacity might be sufficient to accommodate both character-based and sub-word-based representations without significant differences in transcript accuracy.

The results underscore the importance of considering tokenization methods and model capacity when training ASR models. Sub-word tokenization (BPE) proves advantageous for smaller hidden dimensions, where it effectively captures linguistic nuances, resulting in improved CER and WER. However, for larger hidden dimensions, the impact of tokenization diminishes, with WER remaining relatively consistent but CER showing some degradation.

These findings highlight the intricate interplay between tokenization granularity, model capacity, and ASR performance. The choice between character and sub-word tokenization should be made based on the specific context and performance goals. We note that these observations are particularly relevant in the context of a small dataset like AN4, where vocabulary coverage and generalization are key challenges.

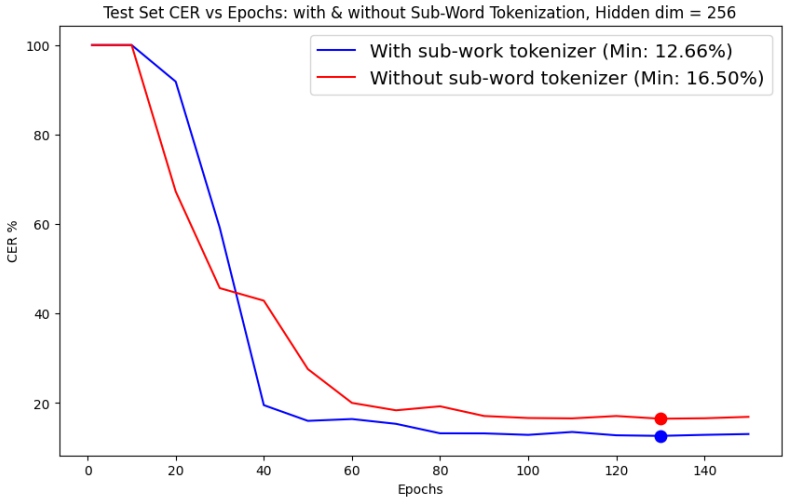
Phase 2b: Experimental Graphs

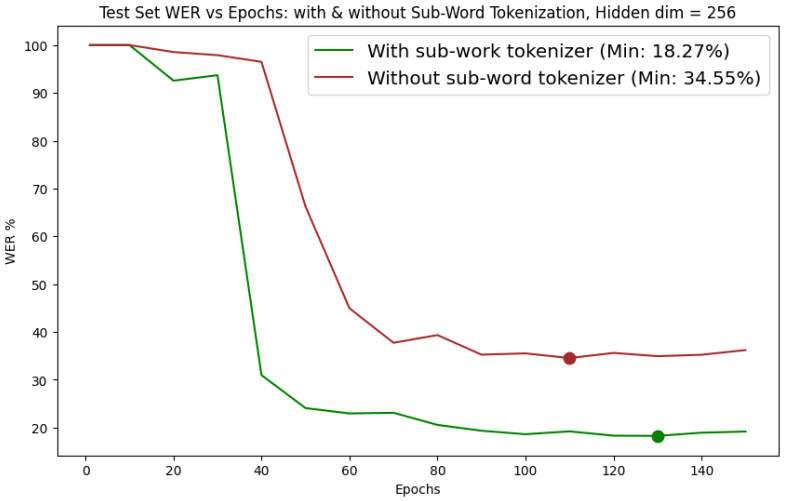


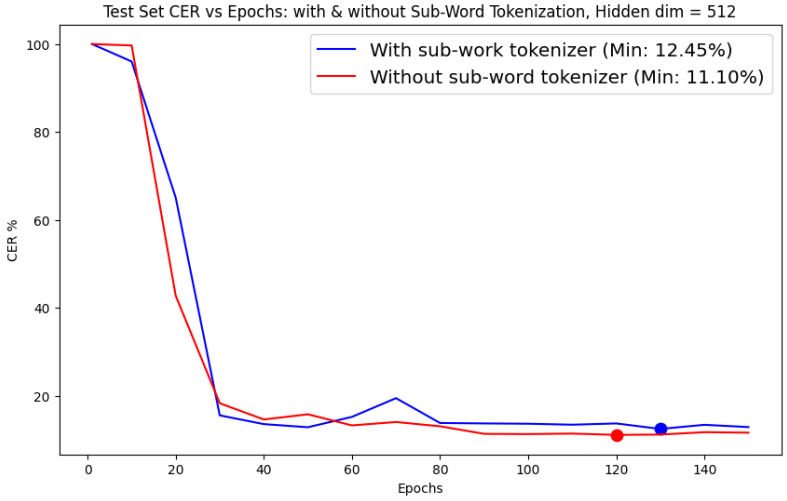
1. *Training vs. Validation set loss through the epochs, for hidden_dim=512:*

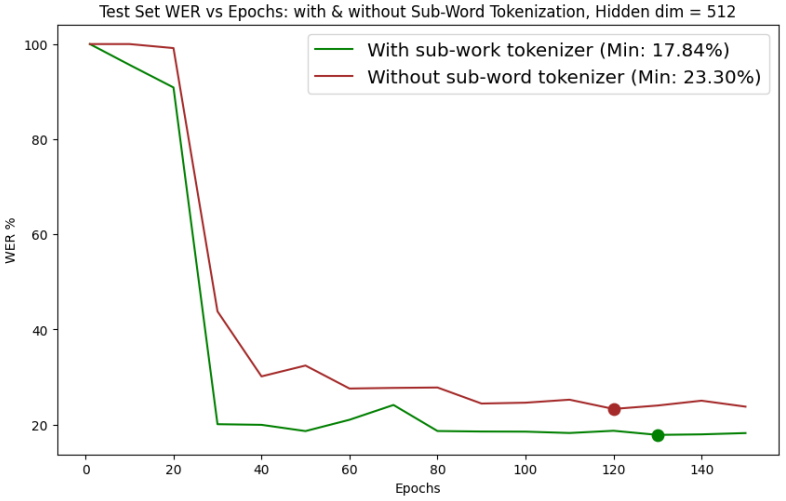
2. *Test set average WER, CER for different hidden_dim values:*

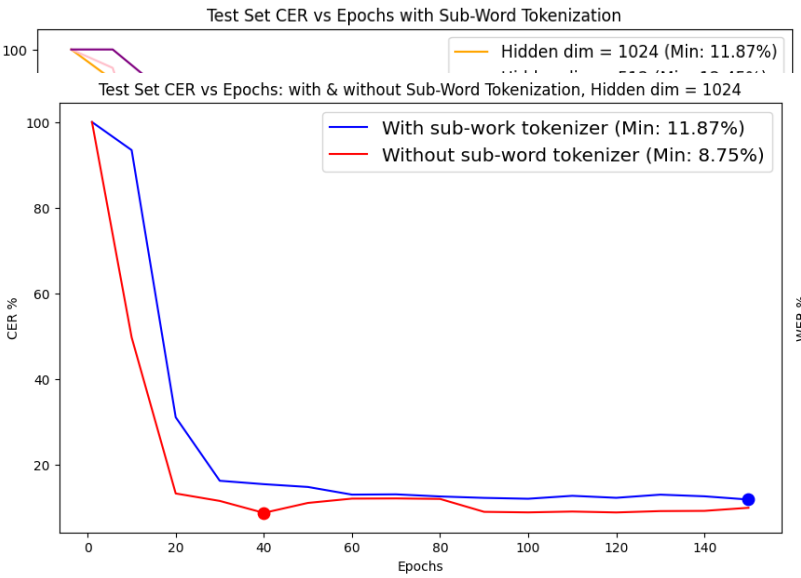
3. *WER, CER average with & without sub-tokenizer, different hidden_dim values:*

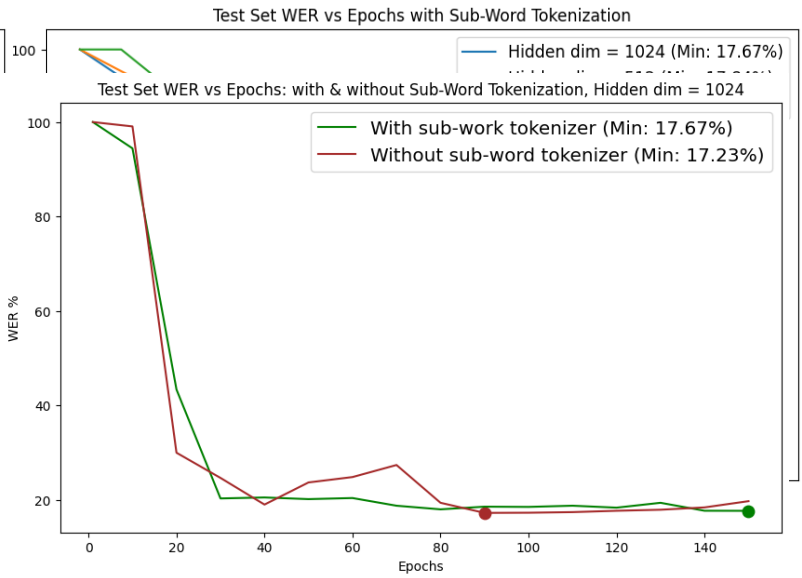












Phase 2c: Generalization using Augmentations

We have added the following augmentation for the training data processing. On each epoch, we took a sample and randomly drew our selection of each augmentation serially. The augmentations are the following:

1. **White Noise:** add random noise that would not top the order of 1e-2 of the original signal.
2. **Time Stretch:** Stretching/shrinking the wavelength to a fixed size at the same order of the original signal.
3. **Pitch Scale:** Shifting the pitch of the waveform by a few number of semitones, in order to increase the pitch.
4. **Random Gain:** Increase the value of the wave function by a factor of slightly higher than 1.
5. **Invert Polarity:** Multiply the wave function by -1.

Using serial draws for each of the augmentation above, including overlapping cases of using multiple augmentations, has outperformed the case of using a single random augmentation out of the following, including the probability of 30% to not use any augmentation at all in both cases.

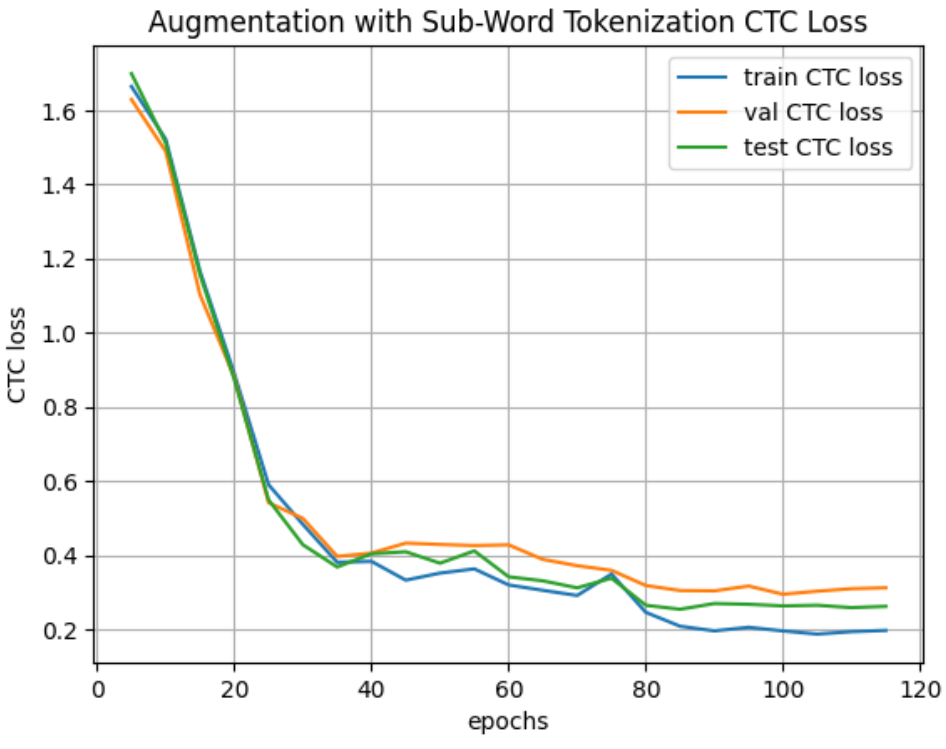
The results suggest that augmentations under a limited amount of training data is a major performance boost (16% WER, 8% CER) for the inference stage, leading us to the conclusion that it is an important part of the training process. The augmentations were introduced in the last phase of the model development.

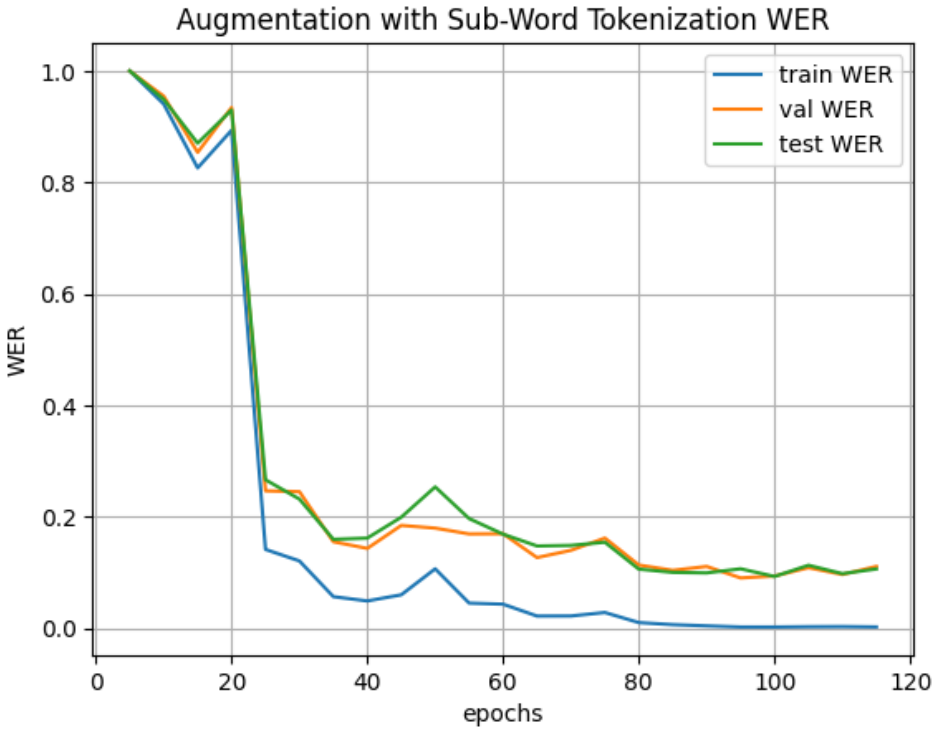
Phase 3: Combining all the Improvements:

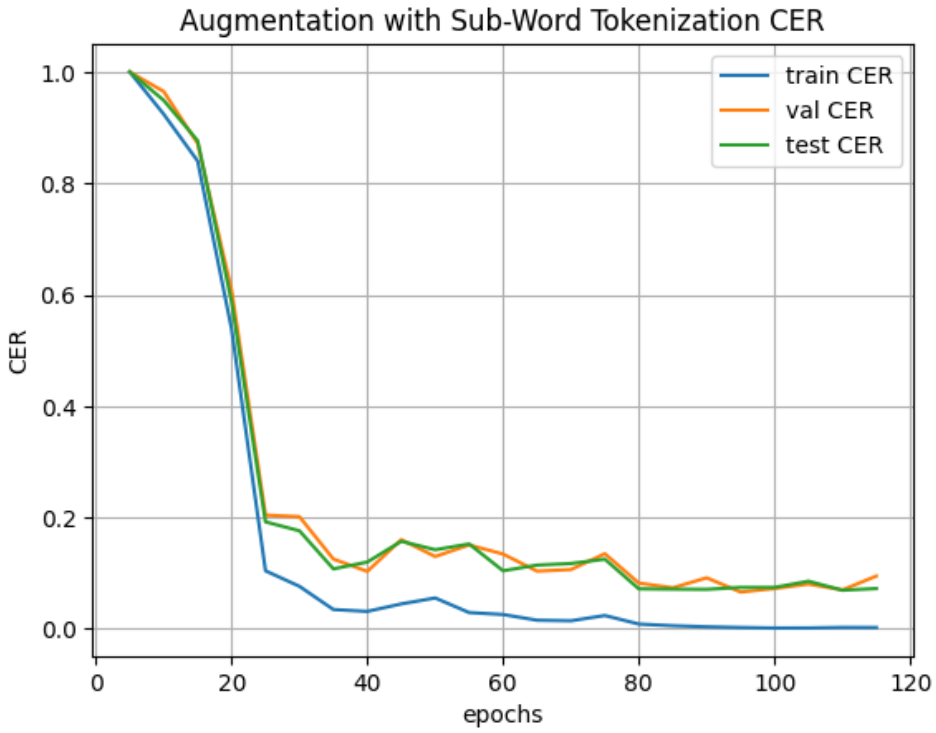
At this stage we have decided to use each of the improvements in one final process that would enhance the model performance at most. Combining the sub-word tokenizer with the augmentation manipulations, which both enhanced the performance of the original LSTM model.

One key observation in terms of augmentations, is that the lack of computing resources has led us to use single augmentation per sample at each epoch, preventing us from utlizing the potential generalization ability achieved by multiple number of augmentations on a single sample, according to the previous phase. This might be an interesting direction for future development, since the serial augmentation draws have outperformed the single random augmentation in the previous phase, although it would significantly increase the duration of the training process (inference time is not affected). Another key note is the fact that we have reached the following enhanced results with a bit more than half of the epochs compared to previous phases, making the current model training process very effective, as we can see in the following graphs.

Results for train/val/test sets are as follows:
Train: Average CER: 0.127038% Average WER: 0.1086%
Val: Average CER: 8.996259% Average WER: 11.4974%
Test: Average CER: 7.974979% Average WER: 10.8077%




Augmentation with Sub-Word Tokenization WER
train WER
val WER
test WER
WER
epochs
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
120


Augmentation with Sub-Word Tokenization CER
train CER
val CER
test CER
CER
epochs
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
120