Long-range Dependency is all you need in sEMG Signal Processing

Jimmy Hou, Junying Li, Lionel Hu
University of Califronia, Los Angeles
{jimmyhou, lijunying02, lionelhu926}@g.ucla.edu

Abstract

Capturing long-range dependencies in sEMG signals is essential for robust decoding, as evidenced by CNN, RNN, and hybrid models augmented with Down-Up noise injection and temporal cropping. In particular, a well-regularized Bi-LSTM encoder trained on augmented data achieves an 11.80% test CER, confirming the pivotal role of robust temporal modeling in sEMG-based keystroke recognition.

1 Introduction

We concentrate on reducing Character Error Rate in a single-subject sEMG keystroke recognition setting, emphasizing the role of long-range temporal dependencies. Our approach systematically explores CNN, RNN, and hybrid architectures enhanced by Down-Up data augmentation and temporal random cropping, while also investigating channel efficiency and data-size constraints. By balancing model complexity, robust augmentation, and carefully selected electrode channels, we significantly improved performance in single-subject sEMG decoding.

2 Method

2.1 Data Augmentation and Data Preprocessing

We implemented **Downsampling-Upsampling Noise Injection** for **Data Augmentation**, by first downsampling the raw sEMG training dateset by applying decimation and then upsampled it back to the original length by using interpolation. Then, we combined the original training dateset with the Down-Up noisy data to obtain the augmented training dataset (Down-Up Boosted Dataset). Models were trained on the augmented training dataset but still validated and tested on original validation and testing datasets. In the downsampling step, the signal is first low-pass filtered to prevent aliasing, and then reduced in temporal resolution by keeping every n^{th} sample (where n is the downsampling factor). This operation decreases the sampling rate (e.g., from 2000 Hz to 1000 Hz for $n = 2$) while the spatial dimensions (e.g., electrode channels) remain unchanged. Next, the downsampled signal is then upsampled via interpolation to restore the original number of samples. The Downsampling-Upsampling mechanism is demonstrated in the Figure 1.

Observing no significant effect upon data preprocessing schemas on the frequency domain, we speculate that sole preprocessing on the temporal domain might render more robust results. Accordingly, we experimented with applying **Temporal Random Crop to the Log-spectrogram** for **Data Preprocessing**, which randomly selects and removes a continuous temporal segment along the time dimension. The function takes an argument specifying the number of frames to be removed and crops a maximum time segment of $\frac{1}{125} \times$ `crop_size` $\times 1000$ ms. The cropped segment is then resized back to the original time length using linear interpolation in PyTorch. Interpolation works by estimating the missing values between known points using a weighted average for each frequency bin, effectively smoothing transitions and preserving temporal structure. It ensures that the time dimension is restored without abrupt discontinuities, allowing the model to process a coherent spectrogram.

2.2 Architecture

2.2.1 An alternative CNN: ConvResNet

Taking inspiration from the general success of TDSConv as a parameter-efficient and Convolutional Neural Network architecture specialized for ASR-like tasks, we first attempted alternative CNN structures like **ConvResNet**. Wang et al. in their 2017 work identified the potential of such structure in Automatic Speech Recognition optimized by CTC loss [4]. Based on their architecture, we made slight modifications and attempted multiple parameter setups. The architecture features the critical **ResBlock** which contains two identical convolutional layers with 3×3 filter, with the skip connection being the identity mapping of input $\mathbf{x}$. In our design of the **ConvResNet Encoder**, we generally adopt the model pipeline of Wang et al., 2017 by pipelining ResBlock Groups each contains certain repetitions of ResBlocks with the first one taking a custom stride and the rest taking a fixed `stride = 1`.[4] To account for the downsampling effect brought by `stride > 1`, a learnable upsampling module to restore the original sampling rate was inserted before the layer norm. The ConvResNet Encoder architecture is given by Figure 2.

2.2.2 CNN + RNN: From ConvResNet+GRU to TDSConv+LSTM

The unsatisfying performance of a sole **ConvResNet Encoder** sparked new directions of architectural design. Upon the scaling of performance along less restraints on the model's learning, such as canceling bottleneck, lowering downsampling rate which on the other hand imposes the drawback of limiting the receptive field on the temporal dimension, we reckon that capturing long-range dependency information might aide the performance of the encoder. With that intention, we attached a GRU layer atop the ConvResNet taking the output of the ConvResNet module as its input and outputting features in the dimension of `hidden_dim`, to build a hybrid **ConvResNet+Bi-GRU Encoder**.

The improvement in performance of the RNN supported CNN hybrid architecture guided us toward the idea of leveraging the parameter-efficient nature of TDSConv, as Hannun et al. stated in their original work, with an LSTM layer which requires more parameters and computational resources. We thus experimented with the **TDSConv+Bi-LSTM Encoder**.[] The architecture similarly features applying an RNN layer on top of the CNN architecture to form the hybrid decoder. We opt to keep the original parameter setting of the TDSConv module and install different configurations of an one-layer bidirection LSTM layer that takes the TDSConv output as its input and output features with a dimension equal to `hidden_dim`. The architectures of the CNN+RNN Encoders are given by Figure 3.

2.2.3 The power of RNN: a comparative approach towards LSTM and GRU

The general success with CNN + RNN hybrid architecture and the bitter-sweet of overfitting leads us to a bold hypothesis: could the RNN architecture solely achieves good performance as the encoder? From the perspective of the data set, the sEMG signals are sequential and often exhibit non-stationary, context-dependent patterns. The signal in each band may span hundreds of milliseconds before and after each keystroke, and these patterns vary widely across time and typing speed. It is natural to believe that RNN could capture the latent long-range dependencies which are prominent in sEMG signal.

We first attempted to install a Gated Recurrent Unit (GRU) encoder in the place of the TDSConv encoder. The new **GRU Encoder** essentially features a stack of bidirectional GRU layers. Each layer is equipped with hidden and cell states that update over each timestep. Parameters for the encoder are number of hidden dimensions, number of layers, and the dropout rate. By tuning the parameters `hidden_dim`, `num_layer`, and `dropout`, we were able to render GRU encoders of varying complexity.

The Long-Short Term Memory (LSTM) architecture is known for modeling temporal dependencies by maintaining a hidden state that evolves in response to incoming data.[1] We eventually experimented with replacing the TDSConv encoder with an **LSTM Encoder**, which consists of multiple stacked classic LSTM layers. We similarly tuned different configurations by tuning the parameters `hidden_dim`, `num_layer`, and `dropout`. We lastly made comparison between the performance of the two RNN architectures. The architectures of the RNN Encoders are given by Figure 4.

2.3 Number of Electrode Channel Exploration

We also explore how many channels of electrodes are enough to efficiently achieve strong decoding performance. A function was implemented at the preprocessing stage, controlled by three hyperparameters: the `start` index of the selected channel (inclusive), the `end` index of the selected channel, and the number of channels on each side to **pick out selected channels data among all the channels**. Regarding our limited understanding for the information provided by each channel, the selection always starts from index 0 to the specified channel index. Experiments were conducted over 20 epochs on the baseline model to compare variations in performance with the number of channels.

2.4 Amount of Training Data Exploration

To investigate how much training data is required to maintain strong decoding performance, we **randomly subset the dataset before training**. The proportion of data retained is controlled by an argument passed to the subsetting function. We ran experiments for both 20 and 50 epochs to illustrate the effects of different random subgroups on the baseline model's performance.

3 Results

3.1 Data Augmentation and Data Preprocessing

3.1.1 Data Augmentation: Downsampling-Upsampling Noise Injection Raw Data Boosting

We trained the baseline model on augmented training dataset using the **Downsampling-Upsampling** mechanism with `factor=2` ($2000\text{Hz} \rightarrow 1000\text{Hz} \rightarrow 2000\text{Hz}$)and `factor=4` ($2000\text{Hz} \rightarrow 500\text{Hz} \rightarrow 2000\text{Hz}$). Models are trained on 20 epochs to investigate the trend between values of factor and CER. Training and testing results are shown in the table 4.

As expected, the augmentation introduces controlled temporal distortions in the sEMG signal, simulating variations in sensor resolution and forcing the model to learn robust and invariant features [4]. We believe that this augmentation technique injects more noise into the total training data set, when combined with the original data. This strategy has been previously shown to improve generalization performance in sequence modeling tasks [4]. This augmentation is expected to introduce **discrete temporal distortion** upon sampled data points from the signal by taking out certain portion of ground-truth information and injecting noise in a controlled manner. This strategy would effectively boost the size of the training data set when combined with the orginal data set. The table above compares the performance of baseline model trained on original training dataset and Down-Up boosted training dataset, clearly showing the increase in performance and decrease in overfitting of Down-Up boosted training dataset. Moreover, Down-Up Boosted Data-2 with smaller down-up factor preserves more temporal detail in the sEMG signal after downsampling and upsampling, which in turn yields lower CER values across training, validation, and testing.

3.1.2 Data Preprocessing: Temporal Random Cropping and Raw Data Downsampling

The Temporal Random Crop technique resembles the data augmentation strategies initially incorporated into the baseline model, but it specifically targets the time dimension. By randomly removing a segment and interpolating it back to the original length, we ensured that the frequency information remained unchanged while modifying the temporal structure. As anticipated, providing the model access to more diverse temporal patterns led to a significant improvement in performance. With the same 200ms time mask, the model achieved a validation CER of 61.41, compared to the baseline 65.88 after 20 epochs of training. We believe that this preprocessing schema simulates introducing **continuous temporal distortion** that is projected to help the model learn robust features on the long run, thus capturing the underlying pattern. The improved performance with sole manipulation on temporal domain instead of both temporal and frequency domains also highlights the significance of frequency continuity and completeness in speech recognition models as preservation of the information on the frequency domain helps maintain stable performance.

To investigate the relationship between sampling rate and CER, we experimented with `hop_length` parameter during the log spectrogram transformation. `hop_length` parameter determines the step size between consecutive STFT windows and serves as a downsampling factor for the raw signal.

We trained and tested the baseline model with various `hop_length` values, which correspond to different raw data sampling frequencies. By the results shown in the plot in Figure 5, we found out `hop_length=16` ($125Hz$) generated best performance, which is already used by the baseline model. When `hop_length<16`, the spectrogram frames are overly dense, capturing excessive micro-variations and risking overfitting [4]. Conversely, `hop_length>16` discards key transient features [4]. Hence, `hop_length=16` balances detail retention and overfitting control, yielding optimal CER.

3.2 Data Efficiency

The amount of data required to achieve optimal model learning performance is closely tied to the number of training epochs. When training has limited epochs, having a sufficiently large and complete dataset is crucial for effective learning. We observed that when less than 95% of the data was used, the model struggled to perform, yielding suboptimal results compared to the baseline. However, as the dataset size approached 95% of the original data, there was a sudden boost in performance, reaching levels comparable to the optimal configuration.

In contrast, during longer training sessions, such as 50 epochs, the model relies less on having the full dataset, as extended training allows it to extract meaningful patterns even with a reduced amount of data. In our experiments, we found that using only 60% of the original dataset was sufficient to achieve comparable performance to training on the full dataset. This indicates that while short training durations demand more data for rapid learning, longer training mitigates the impact of data reduction by allowing the model to refine its understanding over time. The statistics are demonstrated in the plots of Figure 6.

3.3 Channel Efficiency

The relationship between the number of channels and the baseline model's performance is non-linear. The first notable improvement in Test CER occurred with 20 channels, which required only 62.5% of the original channels while still achieving results comparable to the baseline. When training was extended to 50 epochs, this 20-channel configuration yielded validation CERs of 26.18 and 28.05, figures very close to the baseline model's performance. Additionally, a configuration with 28 channels produced performance nearly identical to using all channels, suggesting that a full set of channels is not strictly necessary to achieve optimal performance.

We also observed significant fluctuations in validation CER as the channel count increased, leading us to suspect potential collinearity among adjacent channels. This suggests that certain channels may share or depend on each other's information for effective learning. For example, when comparing the performance of using the first 10 channels (out of 20 total) to using channels $11-13$, we observed a higher CER in validation and testing with the latter. Specifically, including channels $11-13$ led to worse performance compare to only include 10 channels. This is likely because the model struggled without channel 14, which was crucial for supporting the learning information from channels $11-13$.

A similar pattern emerged with channel 15: while the performance of the first 14 channels was similar to using all channels, adding channel 15 in isolation caused confusion and increased CER. This issue was resolved when channel 16 was introduced, further supporting our hypothesis about collinearity among adjacent channels.

Given these dependencies, along with the near-baseline performance achieved with 28 channels, using 28 channels (87.5% of the total) represents a strong compromise between accuracy and computational efficiency. The statistics are demonstrated in the plots of Figure 7.

3.4 Architectural Experiment

3.4.1 The battle of CNN architecutre: TDSConv vs. ConvResNet

ConvResNet
We tested multiple parameter settings including different group numbers, repetition numbers, and stride size, yet achieving poor performance given by Table 5, compared with the Baseline model performance achieving a Test CER of mere 21.63 under the same 50-epoch training setting. The significantly higher test CER suggests that the ConvResNet model is comparatively hard to train

and is not projected to well fit the data after 150 epochs with the single-user data. Accordingly, for the purpose of our study, we may believe that TDSConv is a better CNN model regarding our experimental setting and objectives.

3.4.2 The redundancy of CNN + RNN architecture

ConvResNet + GRU
Compared with the previous ConvResNet Encoder, the result was shockingly good: a drastic decline in Test CER of more than 40% is observed with `droupout = 0.0` as shown in Table 6

With the ConvResNet+Bi-GRU Encoder, we could match the baseline performance after 50 epochs of training. Since we have already discovered that TDSConv outperforms ConvResNet in the experimental setting of this study in all regards, we transition to establish the TDSConv+Bi-LSTM Encoder to leverage TDSConv's performance and parameter-efficiency, meanwhile taking advantage of LSTM's finer long-range dependency details.

TDSConv + LSTM
We indeed rendered results of what we have expected: the TDSConv+BiLSTM is able to achieve slightly better testing CER at 50 epochs generally. But another issue arises: The model is already heavily overfitting with mere 50 epochs. As we aim to train the data with more generalizability while at the same time maintaining high performance on single-user case, we tried different regularizer strategy with results shown in Table 7.

Nonetheless, the results are not satisfying as simplifying and regularizing the LSTM did not seem to achieve the expected effect: Overfitting is still a major problem and stricter regularization would all the performance advantages gained over the baseline implementation. We reflect upon the problem and speculate that this could be the redundancy and the overwhlming complexity brought by the hybridization of CNN and RNN models.

3.4.3 Regulating RNNs to harness the power of long-range dependency

GRU
We attempted the following parameter initializations for the GRU Encoder and respectively got the result showin in Table 8

As we may see that the performance of the GRU encoder generally align with our expectations, we also suffered from heavy overfitting with mere 50 epochs of training when the `hidden_dim` is relatively large. Deploying the classic `droupout = 0.2`, we were able to contain the fitting of the encoder under the desired range of effective learning. We made

LSTM
We initialized the parameters with `hidden_dim=512`, `num_layer=3`, `bidirectional=True`, and `dropout_rate=0.0`. We trained it for 150 epochs and achieved an exceptional validation CER of 12.76 (Baseline model, 150 epochs, validation CER=18.91) and a testing CER of 14.71 (Baseline model, 150 epochs, testing CER=22.17). However, the model's training CER was 2.10, clearly indicating that the model was overfitting. We first tuned parameter `num_layer`. Based on the loss plot in Figure 8 of the previous training on 150 epochs, we found that around 50 epochs (6350 batches), the loss started to decrease slowly. Then, we decided to tune number of hidden dimension on 50 epochs for `hidden_dim=512` (LSTM-1) and `hidden_dim=256` (LSTM-2). Results are shown in the table below. LSTM-2 with `hidden_dim=256` was less overfitting. As expected, with a smaller hidden dimensionality (256 instead of 512), the LSTM-2 has fewer parameters, which reduces its capacity to memorize the training data and thus leads to less overfitting.

Table 1: Testing statistics of LSTM encoder for different `hidden_dim`, 50 epochs

	Hidden	Layers	Dropout	Train CER (%)	Val CER (%)	Test CER (%)
LSTM-1	512	3	0	7.46	15.57	15.93
LSTM-2	256	3	0	12.18	17.37	17.55

We trained the model for 100 epochs to magnify the overfitting phenomenon. We set `hidden_dim` $= 256$ and subsequently tuned `num_layers` and `dropout`. As the LSTM-2 with `hidden_dim` $= 256$ still exhibited signs of overfitting, we selected `dropout` $= 0.2$ to introduce a regularization effect. We then compared different values of `num_layers`; the results are presented in the table below.

Table 2: Testing statistics of LSTM encoder for different `num_layer`, 100 epochs

	Hidden	Layers	Dropout	Train CER (%)	Val CER (%)	Test CER (%)
LSTM-3	256	3	0.2	4.51	13.76	14.46
LSTM-4	256	2	0.2	6.84	15.00	15.97

As expected, LSTM-4 with `num_layers=2` is slightly less overfitting than LSTM-3 with `num_layers=3`, but is still overfitting. Since RNN architectures have been shown to better capture the complex and hierarchical temporal dependencies in sequential data, shallow models with very few layers lack sufficient depth to model intricate patterns effectively [5]. Thus, we did not decrease the number of layers to prevent overfitting with the cost of model performance. To further improve model performance while capping the overfitting effect brought by deep RNN models, we introduced data augmentation to address the issue. Down-up augmented training dataset (down-up factor = 4)and temporal random cropping on log-spectrogram (`crop_size=25`) were used to train LSTM-4, resulting our best performance model shown in section 3.6 below. `crop_size=25` is a sweet spot between preventing overfitting and preserving temporal dependencies. We also tested with higher `crop_size=40`, but it resulted in slightly worse testing CER because too much temporal dependencies in sequential data were removed, as shown in the Table 9.

3.5 Best Performance

The architecture of our Best Performance Model Pipeline is demonstrated in Figure 9.

Key modules and parameters of our architectural design:

1. Downsampling-Upsampling augmented training dataset, factor = 2;
2. Temporal Random Cropping of Log-Spectrogram, maximum cropped time = 320ms;
3. Bidirectional LSTM Encoder, `hidden_dim=256`, `num_layers=2`, `dropout` $= 0.2$.

Table 3: Comparison of Best Testing statistics of Baseline Model Architecture and Best-performance

	Epochs	Train CER (%)	Val CER (%)	Test CER (%)
Baseline Model	150	2.41	18.91	22.17
Best-performance Model	150	9.87	12.27	11.80

4 Discussion

4.1 Methods not applicable to the task

4.1.1 Data Preprocessing

Contrary to expectations, neither bandpass filtering [6] nor the mel-spectrogram transformation yielded performance improvements. Similarly, applying the Hilbert envelope [7] with z-score normalization resulted in a training loss plateau, indicating that the model failed to converge effectively. Interestingly, when we omitted the z-score normalization step, we observed a slight improvement in validation performance, suggesting that preserving raw amplitude differences might retain more relevant signal characteristics. However, even with this minor improvement, the overall performance remained suboptimal. Besides, solely downsampling the training data will not yield ideal performance because temporal resolution and critical transient features are necessary for accurate signal interpretation [4].

4.2 Architectural Design

LSTM captures more nuanced long-term dependencies—essential for sEMG typing, where certain keystrokes may recur after long intervals. Compared to GRU, LSTM's more sophisticated memory mechanism better models these distant patterns. However, both GRU and LSTM risk overfitting on single-user data, highlighting the need for data augmentation and regularization to boost generalization and performance.

4.3 Insights for Performance Enhancement

4.3.1 Temporal distortion aids more robust learning

Several key insights emerged from our experiments. First, temporal distortion, whether discrete (via Down-Up augmentation) or continuous (temporal random cropping), significantly improves model robustness. Second, maintaining frequency-domain detail stabilizes training and performance, highlighting the importance of frequency-domain continuity. Additionally, capturing long-range dependencies proved essential for achieving optimal model performance. Finally, data augmentation was confirmed as a crucial step to improve generalization even when the model is intended for a single-subject use case.

4.3.2 Long-range dependency dominates performance

4.3.3 The Necessity of Data augmentation in Generalizability

1. Temporal distortion helps make learning more robust
2. Validated that keeping details in frequency domain helps stabilize learning in sEMG tasks, at least in single-user case
3. Long-range dependency is the dominant factor in achieving good overall performance
4. Data augmentation is a necessary measure to train a more generalizable model, even in single-user case

4.4 Future Directions

Several avenues remain unexplored and hold potential for further improvement:

- **Principal Component Analysis (PCA)**: PCA on raw data was not explored in our preprocessing steps. Applying PCA could efficiently identify the minimal essential data needed and potentially enhance training efficiency.
- **Randomized Channel Selection**: Our current method selects electrode channels sequentially, which may neglect critical signal information. A systematic, randomized channel selection approach combined with a grid search could identify more informative channels and better characterize channel collinearity.
- **Fine-tuning Hybrid CNN-RNN Models**: Our exploration of CNN-RNN hybrid architectures, while promising, was limited. Additional fine-tuning and exploration of parameter spaces on the CNN module could further leverage the strengths of these combined models.
- **Transformer-based Architectures**: Transformer-based architectures, either alone or in combination with CNN/RNN modules, have yet to be explored. Given their proficiency at capturing long-range dependencies through self-attention mechanisms, transformer-based encoders could further enhance model performance.

References

[1] Hochreiter, S. & Schmidhuber, J. (1997) Long short-term memory. In *Neural Computation*, **9**(8), pp. 1735–1780.

[2] Wang, Y., Deng, X., Pu, S. & Huang, Z. (2017) Residual Convolutional CTC Networks for Automatic Speech Recognition. *arXiv preprint*, **arXiv**:1702.07793.

[3] Yu, D., Xiong, W., Droppo, J., Stolcke, A., Ye, G., Li, J. & Zweig, G. (2016) Deep convolutional neural networks with layer-wise context expansion and attention. In *Proceedings of the 17th Annual Conference of the International Speech Communication Association (Interspeech 2016)*, pp. 17–21.

[4] Park, D. S., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, B., Cubuk, E. D. & Le, Q. V. (2019) SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. In *Proceedings of Interspeech 2019*, pp. 2613–2617.

[5] Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006) Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. Pittsburgh, PA: MIT Press.

[6] Chen, Z., Zhang, N., Wang, Z., Zhou, Z. & Hu, D. (2017) Hand gestures recognition from multi-channel forearm EMG signals. In F. Sun, H. Liu, & D. Hu (eds.), *Cognitive Systems and Signal Processing*. ICCSIP 2016. *Communications in Computer and Information Science* (Vol. 710). https://doi.org/10.1007/978-981-10-5230-9_13.

[7] Raez, M. B. I., Hussain, M. S., & Mohd-Yasin, F. (2006) Techniques of EMG signal analysis: Detection, processing, classification, and applications. *Biological Procedures Online*, **8**, pp. 11–35. https://doi.org/10.1251/bpo115.

[8] Hannun, A., Lee, A., Xu, Q., & Collobert, R. (2019) Sequence-to-Sequence Speech Recognition with Time-Depth Separable Convolutions. *arXiv preprint*, **arXiv**:1904.02619.

[9] Synnaeve, G., Xu, Q., Kahn, J., Likhomanenko, T., Grave, E., Pratap, V., Sriram, A., Liptchinsky, V., & Collobert, R. (2020) End-to-end ASR: from Supervised to Semi-Supervised Learning with Modern Architectures. *arXiv preprint*, **arXiv**:1911.08460.

A Appendix

Figure 1: Downsampling-upsampling architecture

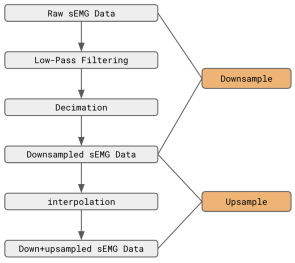
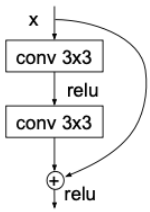
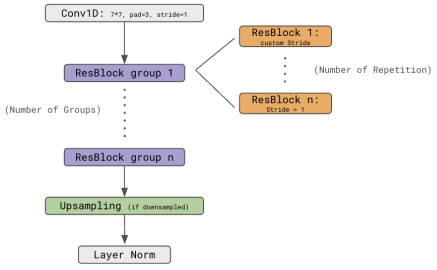


Figure 2: Architectural design of the ConvResNet Encoder

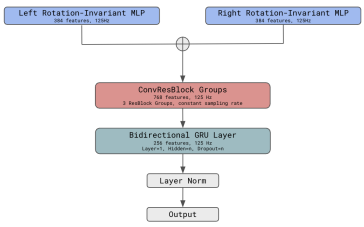


(a) The structure of a residual block (ResBlock) (Yu et al., 2016)

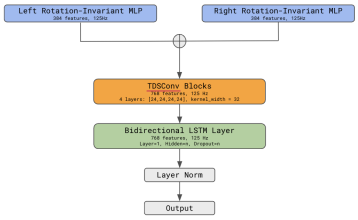


(b) The overall structure of a ConvResNet Encoder

Figure 3: Architectural design of the CNN+RNN Hybrid Encoders: ConvResNet+GRU and TDSConv+LSTM

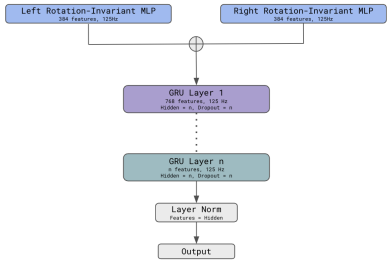


(a) ConvResNet+Bi-GRU Encoder

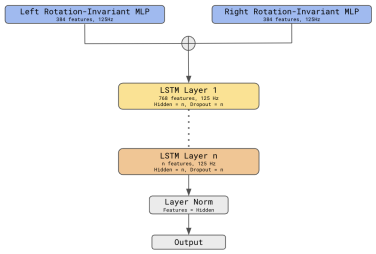


(b) TDSConv+Bi-LSTM Encoder

Figure 4: Architectural design of RNN Encoders: GRU and LSTM



(a) GRU Encoder architecture



(b) LSTM Encoder architectural design

Figure 5: `hop_length` vs. CER(Validation and Test)

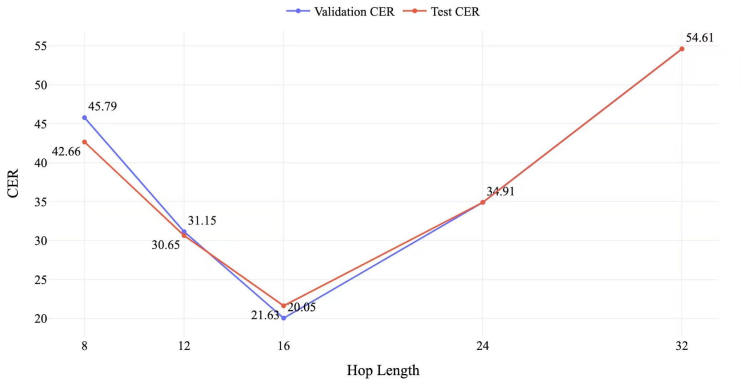


Figure 6: Validation CER for Different Amount of Training Data

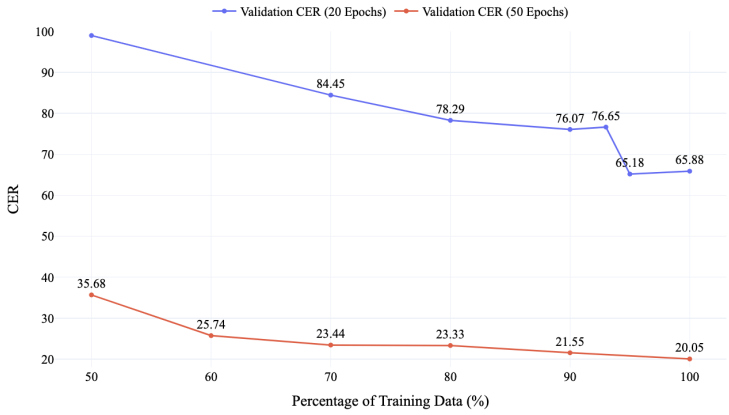


Figure 7: Validation and Test CER for Different Number of Channels

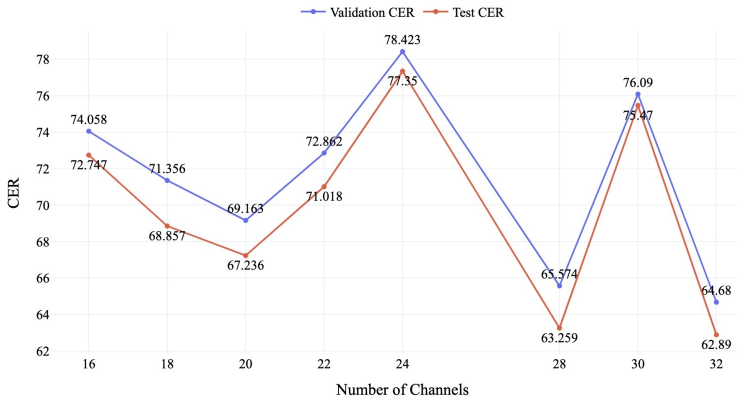


Figure 8: Loss of LSTM when `hidden_dim=512`, 150 epochs: plateaued after 50 epochs

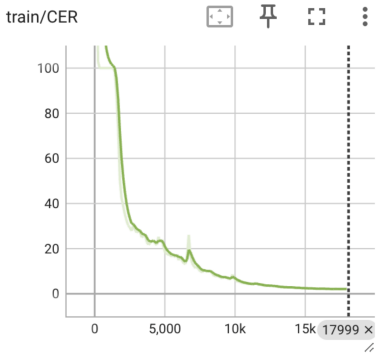


Figure 9: Overal Architectural Pipeline of the best-performance Model Configuration

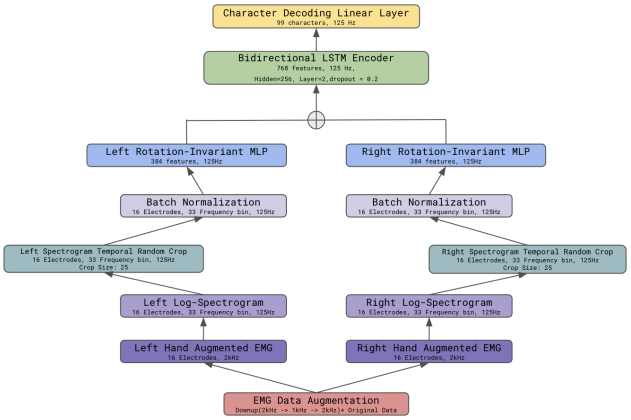


Table 4: Testing statistics for using boosted datasets of different down-up factor, 20 epochs

	Down-up Factor	Train CER (%)	Val CER (%)	Test CER (%)
Original Data	None	70.34	65.88	64.30
Down up Boosted Data-1	4	77.69	64.73	63.13
Down up Boosted Data-2	2	49.09	36.82	34.93

Table 5: Testing statistics of the ConvResNet Encoder under different `num_group` configurations, 50 epochs

	Groups	Dimensions	Repetitions	Strides	Bottleneck	Test CER (%)
ConvRes-1	4	[64, 96, 128, 256]	[3, 2, 2, 2]	[1, 2, 2, 2]	[false, false, true, true]	92.69
ConvRes-2	3	[64, 128, 256]	[3, 2, 2]	[1, 1, 2]	[false, false, false]	68.66

Table 6: Testing statistics of the ConvResNet + GRU Encoder of different GRU `hidden_dim` configurations, 50 epochs

	Groups	Dimensions	Repetitions	Strides	GRU Hidden	Test CER (%)
ConvRes+GRU-1	3	[64, 128, 256]	[3, 2, 2]	[1, 1, 1]	768	20.27
ConvRes+GRU-2	3	[64, 128, 256]	[3, 2, 2]	[1, 1, 1]	512	19.51

Table 7: Testing statistics of the TDSConv + LSTM Encoder of different LSTM configurations, 50 epochs

	LSTM Hidden	LSTM Dropout	Train CER (%)	Val CER (%)	Test CER (%)
TDSConv+LSTM-1	512	0.1	9.49	17.89	18.60
TDSConv+LSTM-2	256	0.2	14.16	18.20	20.40

Table 8: Testing statistics of various Bi-GRU Encoders, 50 epochs

	Hidden	Layers	Dropout	Train CER (%)	Val CER (%)	Test CER (%)
GRU-1	768	3	0.0	7.69	24.52	17.19
GRU-2	512	3	0.0	9.19	23.21	21.53
GRU-3	256	3	0.2	18.71	19.52	19.92

Table 9: Testing statistics for random cropping of different crop size, 150 epochs

	Crop Size	Train CER (%)	Val CER (%)	Test CER (%)
Crop-1	25	9.87	12.27	11.80
Crop-2	40	11.05	12.65	13.25