Comparative Deep Learning Architectures and Post-Processing Methods for sEMG-to-Text Decoding

Justin Bou, Umair Khan, Aashman Rastogi
ECE 147C
UCLA
bou22716@gmail.com
aashman080303@gmail.com
umairk3052@g.ucla.edu

Abstract

This work presents a comparative study of different architectures for decoding surface electromyography (sEMG) signals into text based on the emg2qwerty baseline project. The first model employs a hybrid CNN+GRU structure that extracts local frequency features via convolution and models temporal dependencies with a bidirectional GRU, using CTC loss for training. The second model utilizes a Conformer-based architecture that combines a convolutional frontend with multiple Conformer blocks incorporating multi headed self-attention and convolution modules. The third model utilizes a Binary-CNN + RNN model for faster and more energy-efficient computation. Experimental results on validation and test sets are reported using Character Error Rate (CER) along with deletion, insertion, and substitution error rates. Additionally, we post-process the predicted text using Levenshtein-based spell-checking, to refine outputs and improve Word Error Rate (WER) to improve usability. Overall, the Conformer model achieved the lowest validation and test CER, indicating improved performance in decoding, though each model exhibits distinct error profiles. Our approaches can be found in the **prediction strategies** folder.

1 Introduction

Our motivation was twofold: to improve decoding accuracy and to evaluate the feasibility of efficient, low-computation models for real-world applications, where deployment on edge devices or resource-constrained environments is a key consideration. Additionally, we aimed to improve real-world typing performance by drawing inspiration from spellchecking systems typically employed in smartphone devices to mitigate typing errors and improve user experience. To this end, we implemented and evaluated three distinct models. The first model leverages a CNN+GRU architecture, where convolutional layers extract local frequency-based features and a bidirectional GRU captures temporal dependencies, trained using a Connectionist Temporal Classification (CTC) loss. The second model incorporates a Conformer-based architecture, combining convolutional frontends with Conformer blocks that integrate self-attention and convolutional modules, aiming to capture both global and local context more effectively. Finally, we designed a Binary-CNN + RNN model, motivated by the goal of reducing computational and memory overhead. By binarizing the convolutional layers (with weights constrained to ± 1), we enable efficient storage (1-bit representation) and computation (via XNOR and bitcount operations), while preserving sequence modeling capabilities through an RNN backend. Together, these models allow us to investigate the trade-offs between performance and efficiency. Similarly to offset character level errors, we also performed post-processing of our predicted text using Levenshtein distance between mispelled words and a corpus to improve word-error-rate (WER).

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

Benchmarking results—reported in terms of CER, as well as deletion, insertion, and substitution error rates—highlight the strengths and limitations of each approach (see Table 1).

Table 1: Base model benchmark

	Validation	Test
CER	19.9601	23.4709
DER	1.1218	1.3831
IER	7.1777	6.7863
SER	11.5640	15.3015
Loss	1.0219	1.2127

2 Methods

For the individuals models, no other preprocessing/alterations were made besides the model implemented within the TDS Conv Blocks.

2.1 Binary-CNN + RNN Model

Our model applied He et al.'s research on proxy CNN's where weights are determined by

$$sgn(Z) = sgn(W'R), W' = \phi(W)$$

where $W \in \mathbb{R}^{[h \times w \times n] \times c}$, $R \in \mathbb{R}^{c \times c}$ and $\phi()$ is a nonlinear mapping. W' is W decomposed into R, the basis of the latent parameter space (lower R meant higher compression of data) [He et al., 2016]. By doing so, we were able to solve BNN's main issues with 0 gradient back propagation and low feature expressivity. Therefore we applied it to the convolution layers of our model, and the code development for this form of convolution was aided by ChatGPT (class ProxyConv1D).

As for the RNN portion, we opted in for a GRU instead of an LSTM as it aligned with the goals of using such a model. Using a GRU made computation simpler and faster and the required memory was much less than a LSTM.

With the goal to reduce parameters our model followed. Three 1D binary convolution layers, using ReLU activations, max-pooling, stride set to 1, padding set to 2, and batch normalization. For all convolution layers, R was set to 50 to allow more expressivity with the data. Trained on 40 epochs for time. Following the convolutions, the data is reshaped from (N, C, T) and sent through a 1-layer GRU layer. The data is then reshaped once more to the dimensions (N, T, C) and sent through two FC layers, with the first layer having the ReLU activation. See Table 4 for model performance

2.2 Hybrid CNN+GRU Model

he model begins by reshaping the input—merging the two frequency bands and 16 electrodes into a unified 32-channel representation. It then processes this representation through a convolutional neural network (CNN) designed to progressively extract deeper and more abstract features. The CNN consists of three convolutional layers that expand the channel depth from 32 to 64, then to 96, and finally to 128 channels. Each convolutional layer is followed by batch normalization and a ReLU activation to stabilize and enhance the learning process. The use of 1D convolutions enables the network to capture local frequency patterns across the electrode-time dimension. A max-pooling layer is then applied to reduce the feature map size and introduce translational invariance, followed by dropout to prevent overfitting. These extracted frequency-domain features are then pooled and reshaped into a sequential format suitable for temporal modeling. This sequence is passed through a three-layer bidirectional GRU with a hidden size of 256, which captures temporal dependencies in both forward and backward directions. The final GRU outputs are projected into class logits using a fully connected layer, and a LogSoftmax function converts these logits into log probabilities, which are used for Connectionist Temporal Classification (CTC) loss. See Table 2 for model performance

2.3 Conformer Model

The Conformer model integrates convolutional and self-attention mechanisms to capture both local and global dependencies in the input data. The initial stage consists of a 1D convolutional layer applied independently at each time step. This layer maps the 32 EMG channels to a higher-dimensional representation space while capturing local frequency patterns in the input spectrogram. Batch normalization, ReLU activation, max-pooling, and dropout are used to stabilize and regularize this feature extraction stage. The use of global average pooling across the frequency dimension reduces the spatial dimensionality, resulting in a per-time-step vector representation that feeds into the subsequent conformer encoder.

The main temporal modeling is handled by a stack of Conformer blocks. Each block integrates four key components [Anmol, et al 2020]:

- A feed-forward module with residual connections for non-linear feature transformation.
- A multi-head self-attention mechanism to model long-range dependencies across time steps.
- A convolutional module that captures local dependencies in the temporal domain, enriched with gating via Gated Linear Units (GLUs).
- A second feed-forward block and final layer normalization to stabilize learning.

After passing through the Conformer layers, the sequence is projected to the vocabulary size via a fully connected layer. Afterwards, the same softmax layers and CTC loss function are applied. See Table 3 for model performance

2.4 Post-Processing

After generating the model's outputs, we refine the predicted text by first performing all the backspace operations and removing the enter keys. After obtaining the cleaned text, we spell-check using Levenshtein distance. The Levenshtein distance between 2 strings A and B of lengths m and n is measured by the number of minimum character edits required to transform one string into another. We perform spellchecking by comparing each word to the corpus and greedily assigning each misspelled word to its closest match.

After spell checking we then perform the WER between the spellchecked outputs and the prompt:

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

The variables in the Word Error Rate (WER) are defined as follows: S is the number of **substitutions**, D is the number of **deletions**, I is the number of **insertions**, and N is the **total number of words** in the reference prompt.

3 Results

The Conformer and Hybrid CNN-GRU model outperformed the benchmark, achieving a lower validation CER of 13.91 and 15.99 respectively.

Table 2: Hybrid CNN + GRU model performance

	Validation	Test
CER	15.994	22.649
DER	2.1045	0.345
IER	3.256	11.6057
SER	10.6335	10.6980
Loss	0.541	0.783

Table 3: Conformer model performance

	Validation	Test
CER	13.9122	17.605
DER	2.0159	2.4902
IER	2.3482	2.8800
SER	9.5480	12.2347
Loss	0.5157	0.6650

Table 4: Binary-CNN + GRU model performance

	Validation	Test
CER	24.5680	22.7577
DER	3.8103	1.9451
IER	5.0952	4.6466
SER	15.6624	16.1660
Loss	0.8549	0.7824

Table 5: Levenshtein Correction Performance

	Word Error Rate (WER) (%)	
Dataset (Baseline)	Cleaned Text vs. Prompt	Spell-checked Text vs. Prompt
Validation Set	71.5	62.1
Test Set	76.6	64.8

Throughout the process, we encountered occasional computing limitations while working on Colab. After utilizing the free GPU units, we collaboratively decided to pool resources and purchase shared GPU access. While this setup required us to coordinate training schedules rather than running experiments in parallel, it encouraged efficient team collaboration and resource management. With additional time and computing resources, we believe there is strong potential to further improve our results by testing more models with different hyperparameters.

4 Discussion

4.1 Hybrid CNN + GRU

We use 1D convolution instead of 2D as it best suited for time series data (One-dimensional CNN is mainly used to process time series data) [Zhang et al., 2023].

Instead of using CNN layers with a constant number of filters, we implemented a sequential increase in the number of channels across the convolutional layers. This design decision is based on the following rationale:

- **Multi-Scale Feature Extraction:** Early CNN layers, with fewer filters, are tuned to capture fine-grained, local features from the spectrogram. As we progress deeper into the network, the layers with more filters capture increasingly abstract and high-level features.
- **Hierarchical Representation:** This progressive increase in channel capacity allows the model to build a robust, hierarchical representation of the input data, accommodating the variability and complexity inherent in EMG signals. [He et al., 2016]

For the temporal modeling component, we upgraded the GRU architecture by:

- **Increasing the Number of Layers:** The number of GRU layers has increased from 2 to 3. This deeper architecture provides a more expressive model for capturing long-range dependencies in the temporal domain.
- **Expanding the Hidden State Size:** The hidden size was increased from 128 to 256. This enhancement allows the GRU to maintain richer representations over time, further improving sequence-modeling performance.

The architectural improvements—particularly the sequentially increasing CNN channels and the deeper GRU layers—result in a more effective extraction of multiscale features and a stronger modeling of temporal dependencies. These enhancements contributed to a notable performance boost over the baseline model. These architectural updates were implemented after initial experiments with simpler CNN-based feature extraction and fewer GRU layers, which were optimized primarily for computational efficiency. However, increasing the complexity of the model allowed for more effective pattern recognition in sEMG data, leading to a significant reduction in CER.

4.2 Conformer Model

For the conformer model, I increased the number of layers from 4 to 5 just to increase the complexity in order to capture better representations of data.

We encountered `the Torch.cuda.OutOfMemoryError`, an error due to processing an extremely long sequence at the test time `trying to allocate 295.23 GiB. GPU`. The multihead attention in the conformer block tries to compute an attention matrix with a size that scales quadratically with the sequence length, which quickly exhausts GPU memory.

To overcome this issue, we updated the `test_dataset` in the WindowedEMGDataModule class in `lightning.py` module to use window length and padding. This helps break up the test session into manageable chunks, avoiding the huge memory allocation in the attention layers.

4.3 Binary-CNN + RNN

Because of the lack of time and computing resources, there were a couple of parameters with which we wanted to experiment. For one, the He et al.'s research covers using the tanh activation function to discard gradients for when W is large. Additionally, there could have been greater experimentation with different values of R. We only tested values of 1, 5, and 50, with 50 giving the best results since it allowed for more features to be represented.

In terms of performance, we found that using max pool, even though outdated, between the convolution layers resulted in a better performance and average pooling having no difference. This also allowed us to reduce parameters and computations for the following layers. As can be seen with 4, comparable results to the base model were reached despite the limitations of binary convolution layers (not having full floating point to represent weights) all while having faster computation times. While the responsiveness of the model was difficult to test, using the time taken to process the test set and the time indicated by running the Jupyter cell, the binary-CNN + RNN model ran on average 5 seconds faster.

4.4 Post-processsing

Initially, after cleaning the text by processing backspaces and removing enter keys, we planned to create a dataset of word pairs, where each word from the predicted text was paired with the corresponding word from the prompt. Then, using a sequence-to-sequence transformer model, we aimed to correct recurring spelling errors in the sEMG model by learning character-level patterns of each word.

We successfully created a dataset of corresponding predicted and prompt words using dynamic programming techniques to handle misalignment between the predicted output and the prompt. By comparing each predicted word with its relative nearby neighbors in the prompt, we identified the

closest match for pairing. However, our limited NLP experience led to challenges in tokenizing and formatting our input data, leading to difficulties in training our model.

Due to time constraints, we opted for Levenshtein correction for spell-checking. While it improved WER, it has significant limitations. Most notably, it ignores context, greedily matching misspelled words to their nearest counterparts in the corpus. For example, it might incorrectly correct "The brwn fox" to "The bran fox". Additionally, its $O(mn)$ runtime complexity—where m is the length of the predicted output and n is the corpus size, makes it computationally slow and impractical for edge computing.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. doi:10.1109/CVPR.2016.90.

[2] R. Zhang, Y. Wang, and X. Liu, "CNN-GRU Model Based on Attention Mechanism for Large-Scale Energy Storage Optimization," *Frontiers in Energy Research*, vol. 11, 2023. doi:10.3389/fenrg.2023.1228256.

[3] Gulati, Anmol, et al. Conformer: Convolution-augmented Transformer for Speech Recognition. Submitted to Interspeech 2020, 16 May 2020. arXiv, https://doi.org/10.48550/arXiv.2005.08100.