ECE C147/247 Final Project Report

Parker Gray, Ezekiel Kim, Tae Hwan Kim, and Brandon Li
UCLA Henry Samueli School of Engineering

***Abstract*—In this work, we use several neural network architectures to classify the electromyography (EMG) data into keystrokes on a QWERTY keyboard. The neural network architectures tested were mostly Recurrent Neural Network (RNN) architectures, basic RNNs along with Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). We also tested the perforamnce of a hybrid model of RNN and CNNs. The lowest test character error rate (CER) was achieved by the hybrid CNN+bidirectional LSTM model with a CER of 26.41%. The code for the project is located and accessible with GitHub at https://github.com/bradoa/EE247-Final-Project**

I. Introduction

The project seeks to classify keystrokes given the EMG data of subject #89335547 using CNN and RNN architectures. The goal of our group for this project was to find the type of model that would minimize the CER on this particular subject. We chose this as our goal as we did believe that with only data from a single subject, the model would not be able to find any general patterns for classification because the idiosyncrasies of the subject would not be identifiable and will result in the model overfitting regardless of what model is used. So, the goal of minimizing the CER for specifically this subject would allow us to learn what models are capable of accurate classification for transcriptions given EMG data.

II. Methods

A. Dataset

The project is based on the EMG2QWERTY data provided by the Meta research group available on GitHub[1]. The 9 EMG data files of subject #89335547 were split into training, validation, and test sets; 7 for training, 1 for validation, and 1 for testing. The data is stored in the form of HDF5 files and includes the left and right EMG signal data, prompted text, keylogger truth, and corresponding timestamps. The EMG data consist of 32 time series signals, with 16 corresponding to each hand. EMG data is inherently temporal, meaning the data is sequential and has dependencies over time. We expect temporal solutions and architectures to have a great advantage as compared to those that struggle with long-term dependencies.

B. Network Architectures

As the data consist of 32 correlated time series signals, we would want to use a network architecture capable of encoding/decoding and utilizing time-dependent data. RNNs are well suited for this, as in RNNs past signals can repeatedly influence network activations, allowing for higher performance in time-contextual tasks like transcriptions. All architectures used a consistent learning rate of 1e-3 to minimize variability across models. The Adam optimizer was employed along with a linear warm-up cosine annealing learning rate scheduler. Data augmentation techniques included random band rotation, temporal alignment jitter, and SpecAugment with 3 time masks and 2 frequency masks. The log-spectrogram transformation used an FFT size of 128 and a hop length of 16. If more time was available, different learning rates could also be tested, but this was not feasible given the computing resources available to our group. Each network architecture had a linear and log softmax activation layer to keep variables equal between models.

1) CNN: Convolutional Neural Networks (CNNs) are one of the most commonly used network architectures used for classification tasks, using convolutional, pool, and activation layers to classify higher-dimensional data into labels. The performance of the CNN model is to serve as a baseline to compare the RNN models to. Considering the highly time-contextual nature of the classification task, the accuracy of the CNN is unlikely to be high due to its lack of temporary memory and understanding of long-term dependencies. The CNN structure utilized for this project was developed by Hannun et al[2], this structure is also partially implemented for the other models.

2) LSTM: Long Short-Term Memory models are a form of RNN which can retain information over time to contribute to its classification of data. LSTM models are more capable at addressing the vanishing gradient that other RNN models might struggle with. This is done through the use of a memory cell and a set of three gates, input, output, and forget on the cell. The LSTM models trained for this project had a hidden size of 128, 256, or 512 and 4 layers. LSTMs in particular are often preferred in predicting temporal data like EMG data due to its gating mechanisms, recognizing important long-term information and forgetting irrelevant information.

3) GRU: Gated recurrent unit models are RNNs similar in structure LSTM, but generally have less parameters than the LSTM models, making them simpler and faster to train. A GRU has a memory cell but only has 2 gates, update and reset, compared to the 3 gates of LSTMs. In practice, many GRUs have shown performance similar to LSTM models [3] and are much lighter computationally lending to real-time applications. The GRU models trained for this project had hidden size of 128, 256, or 512 and 4 layers.

4) TCN: Temporal Convolutional Networks (TCN) gained popularity in recent years with a novel way to capture temporal data, utilizing concepts such as dilated convolutions and causual convolutions. TCNs utilize 1D convolutional layers to process sequences in parallel and employ dilated convolutions to increase the receptive field. [4] TCNs, however, are purely causal. One may infer that TCNs will perform similarly to unidirectional LSTMs.

5) CNN+LSTM: In addition to the above architectures, a hybrid architecture using both CNNs and LSTMs was constructed to observe the effects of stacking models. The hyperparameters for the models are the same as when they are ran by themselves. The output of the CNN layers serve as the inputs to the LSTM layers, which then goes through the same linear and log softmax activation layer as the other models. Previous implementations of CNN + LSTM show improvement in accuracy and success in temporal data such as EEG data [5].

6) Window Size Comparison: To compare the impact of changing the window length in the EMGData module, we explored window lengths of 4000, 6000, 8000, 10000, and 12000 above. The chosen architecture to test these window sizes is a CNN + Bidirectional LSTM model with hidden size of 256 and 4 layers, which has been found to perform best in our experiments comparing different architectures. The hyperparameters remain the same as the default settings to ensure a fair comparison of window length effects.

C. Additional Modifications

1) Batch Normalization: Batch normalization improves the regularization of models. Batch normalization was done through the pytorch module[6] specifically calling the BatchNorm2d function.

2) Dropout: Dropout improves the regularization of models. However, the performance of the models with only batch normalization was already quite regularized, so the group decided to not include dropout.

3) Bidirectionality: For LSTM we also utilized bidirectionality allowing for the LSTM to also preserve future data along with past data. Given the context of transcription, we thought that being able to utilize both future and past information would improve the LSTM model's performance in classifying keystrokes.

III. Results

A. CNN (Convolutional Neural Network)

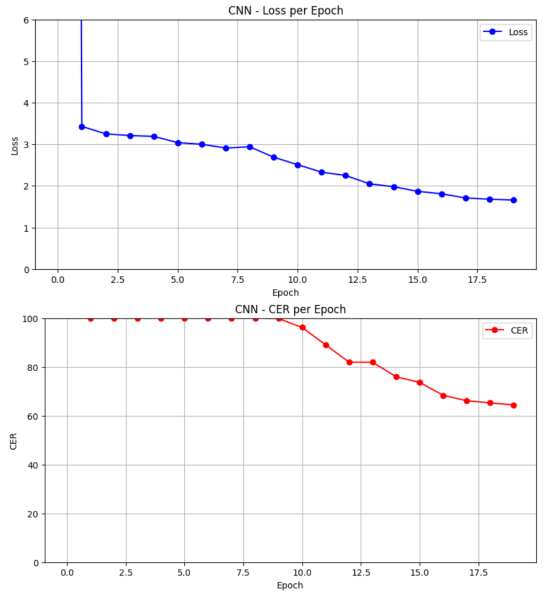

Fig. 1. CNN Loss and CER

epoch	loss	CER
1	3.43	100.00
5	3.04	100.00
10	2.51	96.16
15	1.87	73.68
19	1.66	64.44

TABLE I
CNN Training

CNNs had the worst CER among the models throughout training, having consistently higher CER than the other models for every epoch of training. It achieved a CER of %64.44 and a loss of 1.66. The loss function of CNN was initially lower than the other models, but does not decrease as much over the training epochs. The

CNN model was the first model to show improvement in its loss and CER.

B. *LSTM (Long Short-Term Memory)*

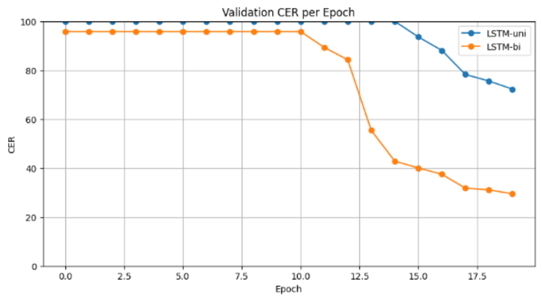

Fig. 2. Unidirectional LSTM vs. Bidirectional LSTM

Metric	**Unidirectional LSTM**	**Bidirectional LSTM**
Test CER (%)	69.81	29.24
Test DER (%)	0.00	2.14
Test IER (%)	58.40	5.60
Test SER (%)	11.41	21.50
Test Loss	1.82	0.91

TABLE II
TEST PERFORMANCE COMPARISON OF UNIDIRECTIONAL VS. BIDIRECTIONAL LSTM

1) Unidirectional LSTM: The unidirectional LSTM processes input data sequentially from past to future. It only relies on the previous time step to make predictions. The hidden layer is computed using the following equations:

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{c}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

This limit may make it less effective in capturing long-range dependencies for accurate sequence modeling.

During training, the unidirectional LSTM struggled to achieve a low CER. It improved to around 72 by the end of training. This suggests that the model has insufficient contextual understanding. Without access to future information, the model may fail to recognize patterns that require broader context and lack complex sequence learning tasks.

2) Bidirectional LSTM: The bidirectional LSTM significantly improved performance compared to the unidirectional one. As the model processes input sequences in both forward and backward directions, richer dependencies can be captured within the sequence.

Training results show that the bidirectional LSTM converges faster and achieves a significantly lower CER. It improves to 29.46 by the final epoch. The final test CER of 29.24, compared to 69.81 for the unidirectional model, it has superior generalization capabilities. Since the model effectively learns relationships throughout the sequence, it is better suited for tasks requiring long-range dependencies.

The bidirectional LSTM shows a higher Substitution Error Rate though. This increase is likely due to its greater sensitivity since the model has access to both past and future information. The model may attempt to adjust predictions based on both contexts, occasionally leading to incorrect substitutions. Although this trade-off results in a higher SER, it ultimately contributes to a lower overall CER. The bidirectional LSTM could be a more effective choice for tasks requiring sequence modeling with contextual awareness.

C. *GRU (Gated Recurrent Unit)*

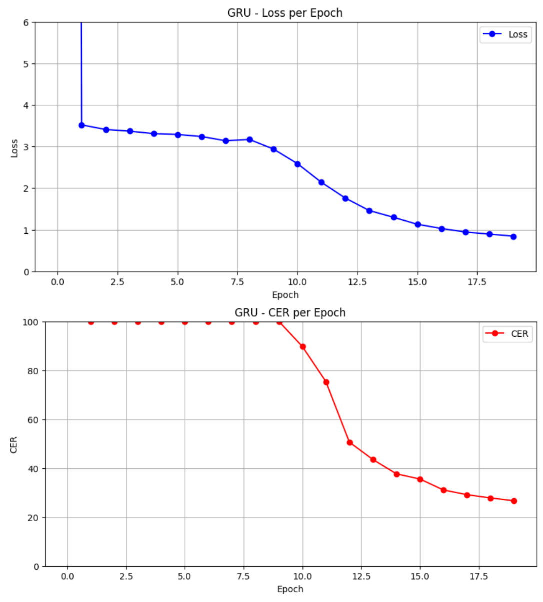

Fig. 3. GRU Loss and CER

The results show that the GRU with the hidden size of 512 and 4 layers achieved good performance comparable to the Bidirectional LSTM, even slightly outperforming it in terms of test CER (27.60 vs. 29.24) while maintaining a simpler architecture. Despite lacking the

Metric	Validation	Test
CER (%)	26.65	27.60
DER (%)	2.17	1.30
IER (%)	7.84	8.80
SER (%)	16.64	17.51
Loss	0.82	0.82

TABLE III
GRU VALIDATION AND TEST PERFORMANCE METRICS

explicit bidirectional processing of future context, the GRU model efficiently captures temporal dependencies and has a more stable and optimized training process. This suggests that GRU can surpass the performance of more complex LSTM architectures while being computationally more efficient. Given its lower parameter count and faster convergence, GRU proves to be an effective alternative for EMG-based text decoding and a strong candidate for real-time applications for accuracy and efficiency.

D. TCN

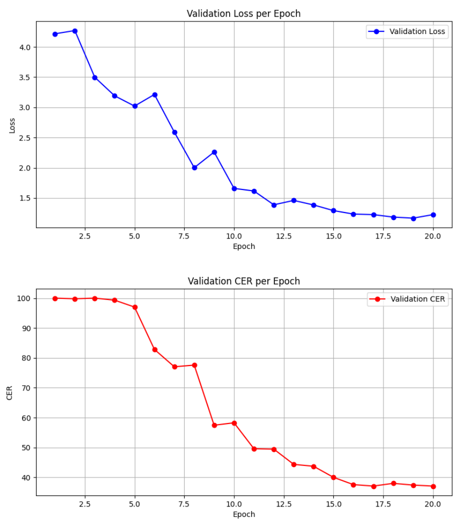

Fig. 4. TCN Loss and CER

Metric	Validation	Test
CER (%)	37.11	36.70
DER (%)	1.86	1.40
IER (%)	20.25	17.68
SER (%)	14.99	17.61
Loss	1.22	1.23

TABLE IV
TCN VALIDATION AND TEST PERFORMANCE METRICS

The results for TCN show that with an architecture of a hidden size of 512 and 6 layers, TCN achieved good performance which was much better than the unidirectional LSTM but not as healthy as the bidirectional LSTM/GRU. Similar to GRU, TCN is a lightweight model and may perform well in realtime systems. In terms of training, TCN seemed to converge the fastest as well towards an optimum. [4] While novel in technology, TCN seems to lack behind GRU as an ”efficiency” candidate, inferentially due to its lack in temporal understanding of future samples and perhaps due to its peculiar lack of dimensionality.

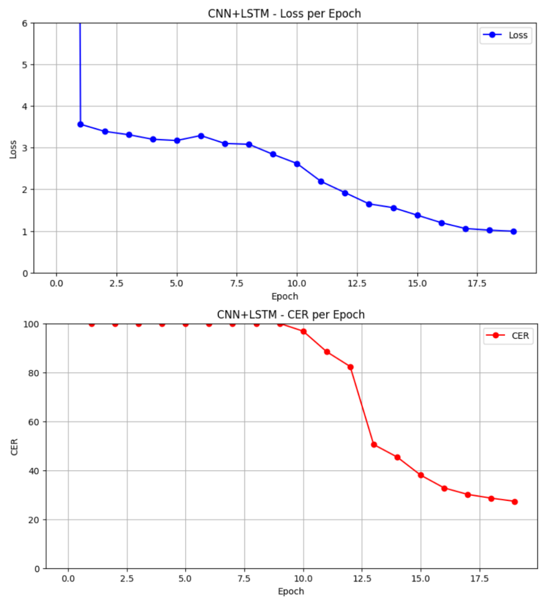

Fig. 5. CNN+LSTM

E. CNN+Bidirectional LSTM

epoch	loss	CER
1	3.56	100.00
5	3.17	100.00
10	2.62	96.83
15	1.38	37.99
19	0.99	27.32

TABLE V
CNN+BIDIRECTIONAL LSTM TRAINING

Metric	Validation	Testing
CER	27.3150	26.4102
DER	1.5729	2.0099
IER	7.6207	4.9060
SER	18.1214	19.4943
Loss	0.8747	0.8397

TABLE VI
VALIDATION AND TESTING METRICS

The CNN+Bidirectional LSTM model with 256 hidden units and 4 layers for LSTM demonstrated the best performance among the tested architectures, achieving a test CER of 26.41%. Compared with the previous testing, the GRU model with 512 hidden units and 4 layers, this model performed slightly better, but the additional complexity did not yield significant accuracy improvements. Given the results, of this project, a GRU model could be sufficient for basic EMG-to-text conversion tasks because GRUs are computationally more efficient and require fewer parameters and less memory so it is a practical choice. However, for more complex tasks beyond this project, the CNN+LSTM model would be a better choice to potentially outperform because its convolutional layers and bidirectional LSTM both work to capture richer spatial patterns from EMG signals and improve temporal dependencies.

F. Performance Comparison Across Different Window Sizes

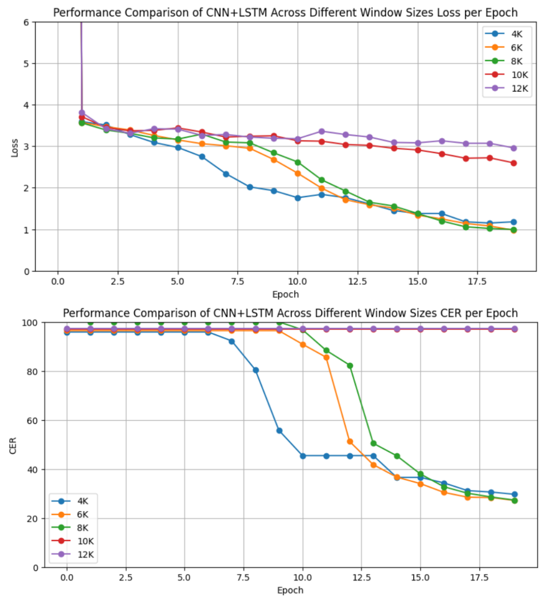

Fig. 6. Performance Comparison of CNN+LSTM Across Different Window Sizes

Comparing different window sizes, the CNN + Bidirectional LSTM model performed best with an 8000-window size, achieving a test CER of 26.41%. Among the models with 4000, 6000, and 8000 window sizes, performance was relatively similar, but the 8000-window model showed faster convergence, stabilizing the loss at earlier epochs. However, increasing the window size beyond 8000 led to a decline in performance, likely due to excessive input length unnecessary noise, or making optimization more challenging.

G. Performance Comparison Across Different Window Infeatures

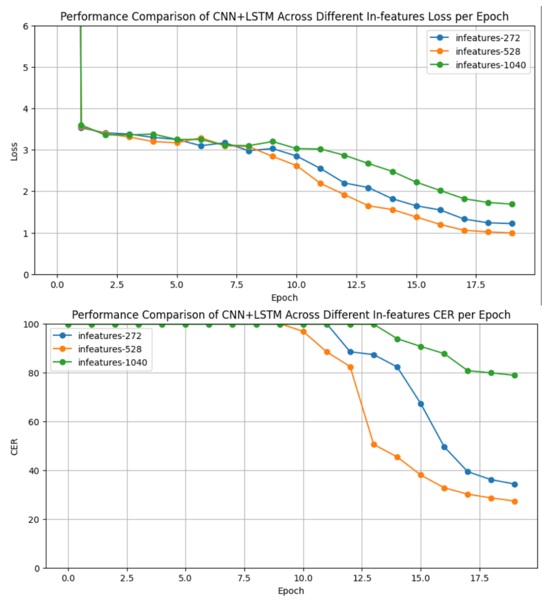

Fig. 7. Performance Comparison of CNN+LSTM Across Different Infeatures

Based on the comparison of three different input features configurations (272, 528, and 1040), the 528 input features model performed best with the lowest test CER of 26.41% and the lowest validation and test loss with 0.875% and 0.84 respectively. In terms of training speed, the 528 in-features model maintains a reasonable training speed of 4.7 iterations per second. The 272 in-features model trained the fastest with 7.1 iterations per second but had slightly worse accuracy with a test CER of 33.56% and a higher loss than the 528 in-features model. On the other hand, the 1040 input features model performed the worst with a test CER of 78.58% and the highest loss, while also taking the longest to train with 2.7 iterations per second. Given these results, the 528 input features model is the best choice for balancing accuracy and training efficiency.

H. Validation Performance

The validation performance of the models followed the same trend as their training performances for both loss and CER. The model with the best validation performance at the end of training is the GRU model at 26.65%, which is closely followed by the hybrid CNN-bidirectional LSTM model at 27.31%. The bidirectional

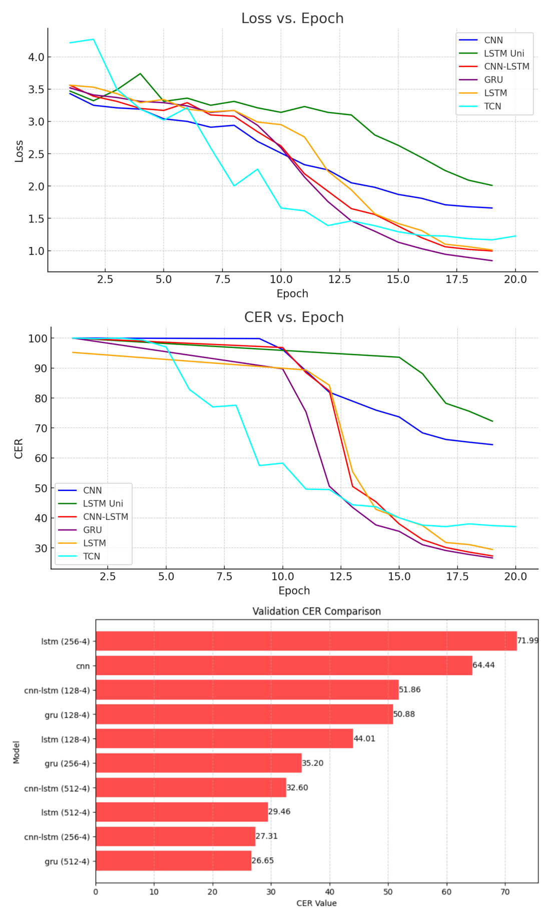

Fig. 8. Validation CER Metric

LSTM model by itself had a final validation CER of 29.46%. The CNN model had a significantly higher CER of 64.44%.

I. *Test Performance*

For the test data set the best performing model was the hybrid model with a CER of 26.41%. The GRU was the second best at 27.59%, followed by the bidirectional LSTM model of 29.24%. Again the worst performing model with a CER over double the other model's is the CNN model with a CER of 63.19%.

IV. Discussion

The chosen neural network architectures proved to have different levels of performance on the EMG2QWERTY single-user data set. This section will comment on the different models, architectures, key findings, limitations, and potential improvements.

A. *Comparison of Architectures*

The four neural network architectures that were chosen for this project were CNN, LSTM, GRU, and

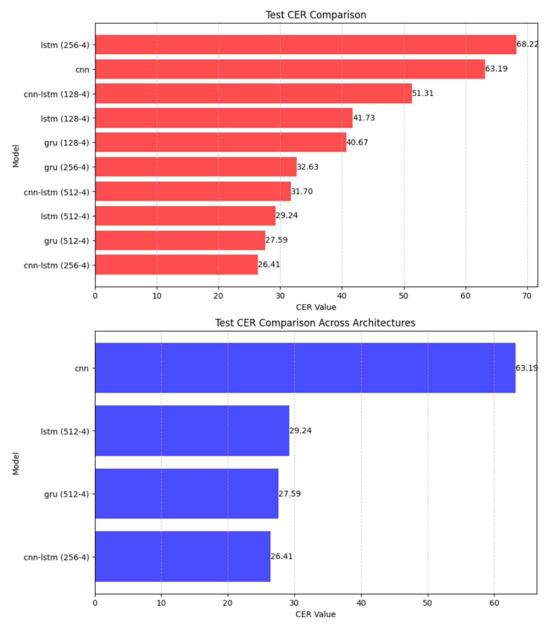

Fig. 9. Test CER Bar Charts

CNN+LTSM. All of these architectures were trained and tested using a learning rate of 1e-3, and on 20 epochs. The evaluation methods for a model's effectiveness was based on the loss and CER results. Convolution Neural Network (CNN) models are designed to capture spatial dependencies, seen in image processing applications. For this emgqwerty data where it is time-dependent, the CNN architecture was not expected to perform as well as the other models. This was confirmed in the training and testing results. Long Short-Term Memory (LSTM) models are a Recurrent Neural Network (RNN) that are designed to retain memory over long time sequences. This feature allowed the model to perform well compared to the limitations of the CNN model. The EMG data, being timestamped sequentially, was processed well by a model with the ability to store memory of past inputs. This storage of inputs though comes with a computational expense that can be limited by using a different model such as the Gated Recurrent Unit (GRU) model. GRU models are simplified LSTMs, but they have fewer parameters, and update their hidden states more efficiently. This simplification allows the model to converge faster which was needed in our project as computational resources were limited. CNN+LSTM model combines the strength of the CNN's model of capturing spatial dependencies with the time dependent memory capabilities of the LSTM. This hybrid model had the best performance, but was complicated to implement and expensive to run computationally wise.

In evaluating our efficiency models, GRU and TCN, we see that GRU performs markedly better. GRU and TCN have explanations mainly in empirical results, TCN proving effective in sequential data specifically such as polyphonic music and word level language modeling [4] whereas GRU has been specifically linked to reading EMG/EEG data[7].

B. Key Findings

As stated above the metrics used to measure a models performance was the loss and CER. All of the models had a loss value that approached or eclipsed the value of 1. This indicates that each model was "learning" and changing their parameters, but for models such as the CNN with a very high test CER, 63.19, it indicates that the model is assigning a false sense of confidence when it comes to emg signals and character matching. This then leads to the CER results of the different models. While the CER is still high for the other three models, hovering around 27, indicating that still about a third of the time the models are predicting the wrong character. This is a large improvement from initial testing of models where the CER was going no lower than 97. All 5 models CNN, LSTM, GRU, TCN, and CNN+LSTM started to see improvement around the 10th epoch. The rate at which the models started to converge though varies. The CNN has the slowest convergence rate, never seeing an improvement from epoch to epoch more than 6 after the first epoch. The GRU architecture has the fastest rate of convergence seen in FIG 10, by the 12th epoch reaching a CER of about 50 while the other architectures are still around 83. This is expected as the model is very efficient in updating its parameters. The other models LSTM and CNN+LSTM perform and converge similarly epoch to epoch on the validation set, but as they reach their convergence value, the CNN+LSTM CER value settles at a lower rate. The final test CER performance seen in FIG 12 shows that the CNN+LSTM has the lowest CER value. When considering which model to use though it is important to think about the compute power that is available. The GRU model has a validation CER of only one point higher, but is computationally cheaper, so depending on the hardware capabilities the GRU model has similar performance for a cheaper architecture.

C. Limitation and improvements

This project proved to have significant hurdles in the developmental stages – mainly surrounding the data set and the computational power required to train and run the models. The full data set was much too large to train on for the scope of this project. The decision was made to use one single users data which is about 1% of the entire data available. This choice to use a limited data set exponentially improved the base training time.

Another problem with the project that we struggled with was the computing power needed, even with the single user dataset. The training time when using the local machine took around 15 minutes per epoch to train. This training time was significantly improved to about 25 seconds per epoch to train when using the A100 GPU processor on Google Colab. This processor though has limited processing tokens, and was only able to be run with Colab pro. This caused initial confusion as the GPU timed out before the entire data set was trained. In the future it would be wise to understand the limitations of your IDE if there is training being done outside of a local machine.

Tuning of the hyper parameters could also lead to a better performance. Changing the learning rate to a higher value, could possibly increase performance. If there was more time exploring this could lead to a better CER. A possible downside of this though would be an overfitting to the training data and lead to a much higher CER when it comes to the test data.

For further experimentation on this project, we could employ a transformer encoder to replace LSTM and GRU for better long-range dependencies from an architectural perspective. Better optimization strategies by adjusting weight decay regularization, increasing the batch size, and extending the training duration could lead to more stable training and further improve performance. Furthermore, we could expand our experiments by increasing the spectrogram resolution, time mask parameter, and max jitter offset to analyze how data processing and augmentation impact performance.

As the applications of this project extend to realtime processing, more efficient and quick models in the form of distillation, lightweight transformers, etc. may also prove to be highly beneficial. In addition, data streaming techniques like sliding-window would be useful in creating these realtime applications.

References

[1] V. Sivakumar, J. Seely, A. Du, S. R. Bittner, A. Berenzweig, A. Bolarinwa, A. Gramfort, and M. I. Mandel, "emg2qwerty: A large dataset with baselines for touch typing using surface electromyography," 2024. [Online]. Available: https://arxiv.org/abs/2410.20081

[2] A. Y. Hannun, A. Lee, Q. Xu, and R. Collobert, "Sequence-to-sequence speech recognition with time-depth separable convolutions," *CoRR*, vol. abs/1904.02619, 2019. [Online]. Available: http://arxiv.org/abs/1904.02619

[3] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[4] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018. [Online]. Available: https://arxiv.org/abs/1803.01271

[5] P. Bashivan, I. Rish, M. Yeasin, and N. Codella, "Learning representations from eeg with deep recurrent-convolutional neural networks," *CoRR*, vol. abs/1511.06448, 2015. [Online]. Available: http://arxiv.org/abs/1511.06448

[6] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019. [Online]. Available: https://arxiv.org/abs/1912.01703

[7] C. Lin, C. Chen, Z. Cui, and X. Zhu, "A bi-gru-attention neural network to identify motor units from high-density surface electromyographic signals in real time," *Frontiers in Neuroscience*, vol. 18, Mar 2024.