Evaluating EEG Signal Classification across CNNs, RNNs, and Hybrid Architectures

Victoria Choi
Computer Science
vchoi04@g.ucla.edu

Albert Dong
Computer Engineering
albertd@g.ucla.edu

Ryan Yeo
Computer Science
ryeo@g.ucla.edu

Rachel Yu
Bioengineering
rachelyu25@g.ucla.edu

Abstract

In this project, we create and test the performance of CNNs, RNNs, and CNN/RNN hybrids on classifying EEG data. We discover that for the amount of available training data, our basic CNN performs the best, followed by hybrid architectures with recurrent layers before convolutional layers, followed by RNNs, and lastly hybrid architectures with convolutional layers before recurrent layers. In addition, we note that training our model on data generated from a single test subject leads to increased testing performance on test data from that subject but reduced test performance on combined data and that ensembling these models leads to somewhat improved accuracy.

1. Introduction

Convolutional Neural Networks (CNNs) are known for their effectiveness in tasks that involve spatial features, like image classification. On the other hand, Recurrent Neural Networks (RNNs) are well-suited for modeling time-series/sequential data, as they use internal memory to learn from past inputs [4]. CNNs have been well documented specifically for their usage in classifying EEG signals, while RNNs seem to inherently suit this problem due to the temporal nature of EEG signals.

This motivated the selection of the 5 distinct architectures we studied: regular CNN, pure LSTM-based RNN, pure GRU-based RNN, CNN and LSTM hybrid, and CNN and GRU hybrid neural networks. In particular, hybridizing CNN and RNN architectures allows us to evaluate the effectiveness of certain types of layers in comparison with each other.

In addition, as the provided dataset is split by subject, we also seek to explore how our training process affects classification accuracy. This motivated us to train 9 different models, each using data only from a specific subject. In addition, we investigated how the performance of these models compares to the model trained on all training data, as well as whether ensembling these smaller models could result in acceptable performance.

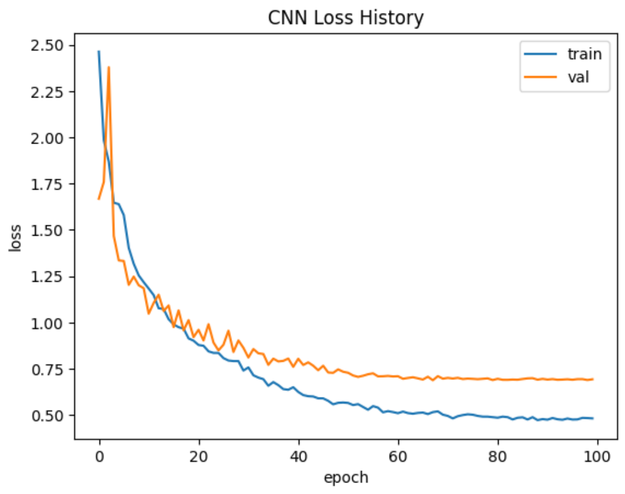

Figure 1. CNN training/validaton loss

2. Results

2.1. CNN

Our CNN achieved a a maximum test accuracy of 72.5%. During training, training accuracy plateaued at roughly 86% and validation accuracy plateaued at roughly 74%, indicating slight over-fitting. Training 5 different models (using 5 folds of our dataset) and averaging them during testing achieved a slightly improved maximum test accuracy of 73.1%.

2.2. Subject-Specific CNNs

We also trained multiple CNNs, each using data only from a single specific subject. This led to slightly reduced test accuracies when testing on data from the same subject the model was trained on and significantly reduced test accuracies when testing on the whole testing set. Ensembling these models together by averaging their results resulted in

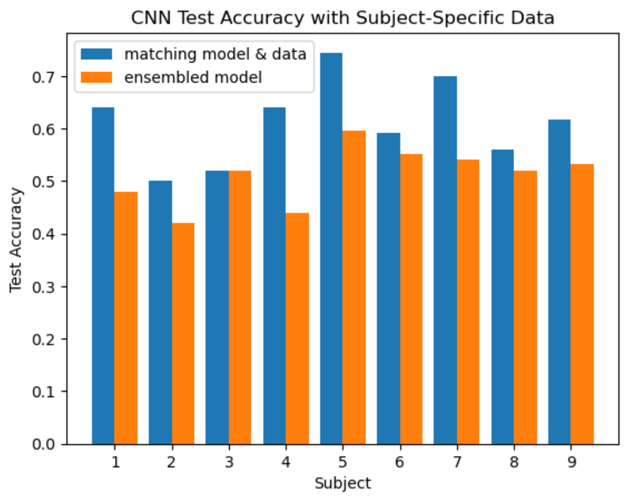

Figure 2. Test accuracy of models trained on data from specific subjects & ensembled model with data from specific subjects

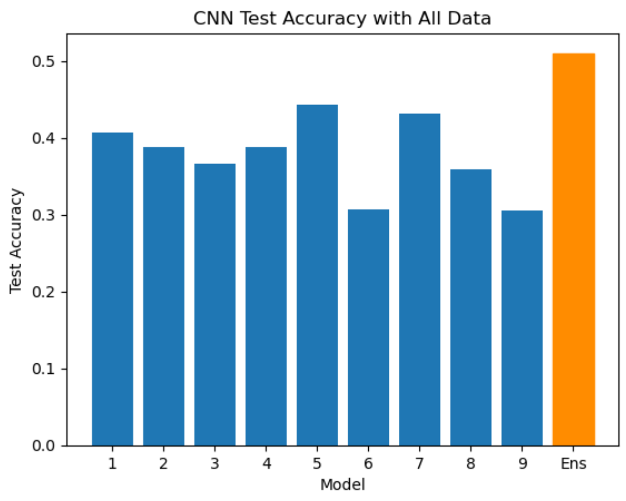

Figure 3. Test accuracy of models trained on specific subjects & ensembled model with all data

worse accuracy on subject-specific data but better performance overall (albeit still worse than the model trained on all data). All 9 models approached 100% training accuracy by the end of the training process.

2.3. RNN

The LSTM model achieved a maximum test accuracy of 65.7% with 3 LSTM blocks. Training accuracy plateaued at roughly 79% and validation accuracy plateaued at roughly 70% in 50 epochs, indicating slight over-fitting.

The GRU model achieved a maximum test accuracy of 68.4% with 4 GRU blocks. Training accuracy plateaued at roughly 86% and validation accuracy plateaued at roughly 67% in 50 epochs, indicating slight over-fitting.

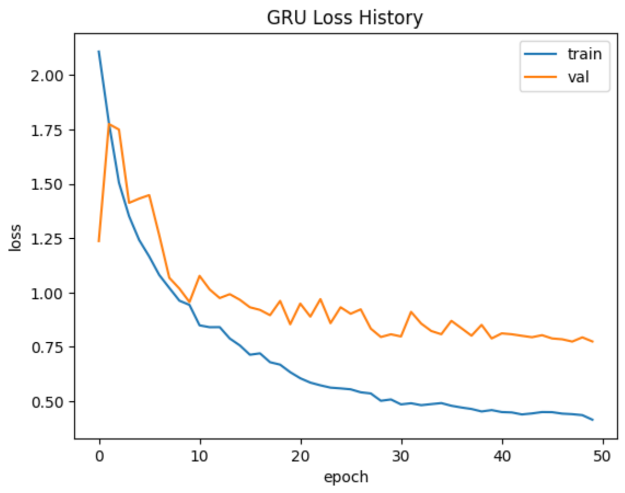

Figure 4. GRU training/validation loss

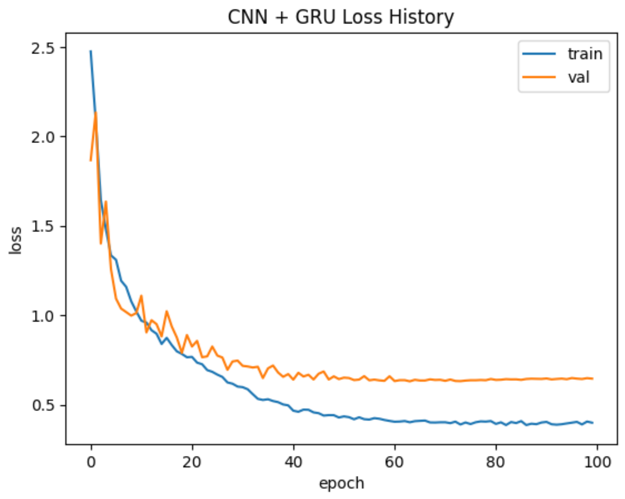

Figure 5. GRU + CNN3 training/validation loss

2.4. CNN + RNN Hybrid

We implemented three variations of a hybrid CNN & RNN model using LSTM recurrent layers. The model with 2 convolutional blocks followed by 1 LSTM block (denoted CNN2 + LSTM1) achieved a 52.6% accuracy, CNN3 + LSTM1 achieved a 67.9% accuracy, and LSTM1 + CNN3 achieved a 69.1% accuracy.

Likewise, we implemented three variations of a hybrid CNN & RNN model using GRU layers. CNN2 + GRU1 achieved a 63% accuracy, CNN3 + GRU1 achieved a 67.7% accuracy, and GRU1 + CNN3 achieved a 72.0% accuracy. Figure 5 displays the training/validation loss of our best performing model, GRU + CNN3.

3. Discussion

3.1. CNN

With the limited amount of data we were given to fit our models, over-fitting was a significant concern. As we attempted to create models with more layers/filters, loss would quickly reach values close to 0 even with large amounts of regularization. Experimentally, models with more than 4 convolutional layers would generally overfit and models with less than 4 convolutional layers could not achieve good performance, indicating that they perhaps did not have enough parameters.

Ensembling 5 models (trained using 5 different test/validation splits) improved performance by a small margin as well, perhaps somewhat ameliorating the limited amount of training data.

3.2. Subject-Specific CNNs

The performance of CNNs trained on data from specific subject agrees with expectations. Since there were 9 total subjects, the amount of training data for each specific subject was significantly limited, meaning testing performance for every model is significantly worse than the performance of the CNN trained on the whole dataset.

Ensembling these models by averaging their outputs results in worse performance when testing on subject-specific testing data but improves performance when tested on all testing data. This is likely because the EEG data of each subject contain uniquely differing features; the individually trained networks were able to learn these features and perform worse on data from subjects they were not trained on. By averaging the predictions of individual models, the ensemble can smooth out errors and capture broader patterns present across subjects. However, this improvement may not extend to subject-specific testing data since the ensemble may dilute the specific features learned by each individual model.

3.3. RNN

The overall architecture of our RNN is similar to that of our CNN but with LSTM/GRU layers in place of our convolutional layers.

Experimentally, we determined that reducing the number of layers from 4 to 3 was effective in improving testing accuracy, potentially by reducing over-fitting. We also found that increasing the batch size from 32 to 64 allowed for better testing accuracy, likely due to accelerated convergence and more generalization. As the RNN models converged faster, training over 50 epochs yielded better accuracy than training over 100 epochs.

Our GRU-based RNN performed slightly better than our LSTM-based RNN. This aligns with research by Chung et al. [2], which showed improved performance from GRU units when compared to LSTM units, likely because GRU units are more efficient than LSTM units.

3.4. CNN + RNN Hybrid

The CNN + GRU models demonstrated superior or comparable performance across their respective CNN + LSTM counterparts, which reinforces the findings by Abbaspour et al. [1]. Among the models examined, CNN2 + RNN1 exhibited the lowest accuracy, followed by CNN3 + RNN1, and finally RNN1 + CNN3.

Our results suggest that with a sufficient number of layers, the performance of the hybrid models were comparable to those of the CNN and RNN-only models. By increasing dimensionality of our CNN and RNN models, the hybrid models have potential to be significantly superior to the CNN and RNN-only models, similar to the results found by Shi et al. [3]. This is due to the fact that the hybrid model employs a vertical learning method, combining the strong modeling power of CNN in temporal feature extraction and the advantage of RNN in processing sequential information.

The CNN2 + RNN1 models performed worse than the pure RNN models. It is likely that there simply are not enough parameters in this network to adequately train a model for good testing accuracy. This is further supported with the increased accuracy of the CNN3 + RNN1 models (12.3% for the LSTM and 4.7% for the GRU), indicating that the extra convolutional layer aids learning.

Furthermore, experimentation with different arrangements of layers revealed that models with RNN layers preceding convolutional layers demonstrated higher accuracy. Notably, the GRU1 + CNN3 achieved an accuracy of 72.0%, higher than any other hybrid or RNN model and similar to the regular CNN model. We theorize that convolutional layers remove some temporal relationships inherent to the unprocessed EEG data, meaning that placing the RNN block first allows the recurrent layer to process unfiltered temporal information, improving model accuracy.

References

[1] Saedeh Abbaspour, Faranak Fotouhi, Ali Sedaghatbaf, Hossein Fotouhi, Maryam Vahabi, and Maria Linden. A comparative analysis of hybrid deep learning models for human activity recognition. *Sensors (Basel, Switzerland)*, 20(19):5707, 2020. 3

[2] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. 3

[3] X. Shi, T. Wang, L. Wang, H. Liu, and N. Yan. Hybrid convolutional recurrent neural networks outperform cnn and rnn in task-state eeg detection for parkinson's disease. *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. 3

[4] Jiyeon Yu, Angelica de Antonio, and Elena Villalba-Mora. Deep learning (cnn, rnn) applications for smart homes: A systematic review. *Computers*, 11(2):26, 2022. 1

4. Performance

Note the notation for hybrid models: LAYER<X> indicates X layers of LAYER type. The following accuracies come from testing on all given testing data.

Architecture	Test Acc
CNN	72.5%
5 Ensembled CNNs	73.1%
LSTM	65.7%
GRU	68.4%
CNN2 + LSTM1	52.6%
CNN2 + GRU1	63.0%
CNN3 + LSTM1	67.9%
CNN3 + GRU1	67.7%
LSTM1 + CNN3	69.1%
GRU1 + CNN3	72.0%

The following table lists test accuracy of models trained on data from specific subjects. Model 1 used data from subject 1 to train, etc. Matching subject test accuracy indicates the testing accuracy when model 1 was tested on test data only from subject 1, etc.

Subject-Specific CNN	Test Acc (Matching Subject)	Test Acc (All)
1	64.0%	40.6%
2	50.0%	38.8%
3	52.0%	36.6%
4	64.0%	38.8%
5	74.5%	44.2%
6	59.2%	30.7%
7	70.0%	43.1%
8	56.0%	35.9%
9	61.7%	30.5%

In addition, the following table presents the test accuracy of the ensembled model (which uses the average of the output layer of all 9 individually trained models as its output) on the 9 separated test datasets as well as on the complete test dataset.

Test Data Subject	Ensembled Subject-Specific CNN Test Acc
1	48.0%
2	42.0%
3	52.0%
4	44.0%
5	59.6%
6	55.1%
7	54.0%
8	52.0%
9	53.0%
All	51.0%

5. Methods

5.1. Data Augmentation

Our model was only given the first 500 timestamps out of the 1000 provided. This was implemented after a visual inspection (which indicated the latter half of the 1000 timestamps seemed like noise) and experimentally supported, as giving the model more or significantly fewer timestamps caused testing/validation accuracy to decrease.

5.2. Training

With some exceptions, all models used the same hyperparameters for training. Generally, we used a cross entropy loss function and the Adam optimizer, with the learning rate set to 10^{-3} and L2 regularization at 0.1 strength. We annealed the learning rate over time by multiplying it by a factor of 0.95 every epoch. We used a batch size of 32, which experimentally gave the best results.

The only exceptions are that L2 regularization was set to 0.15 for models trained on specific subject data and LSTM-only and GRU-only models were trained with a batch size of 64.

LSTM and GRU-only models trained for 50 epochs; all other models trained for 100 epochs.

5.3. Architectures

CNN Architecture (includes CNNs, i.e. those trained for specific subjects)

```
(cblock1): SmallConvBlock(
  (conv1): Conv1d(22, 128, kernel_size=(5,), stride=(1,), padding=same)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (bn1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
  (drop1): Dropout(p=0.5, inplace=False)
)
(cblock2): SmallConvBlock(
  (conv1): Conv1d(128, 256, kernel_size=(5,), stride=(1,), padding=same)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (drop1): Dropout(p=0.5, inplace=False)
)
(cblock3): SmallConvBlock(
  (conv1): Conv1d(256, 512, kernel_size=(5,), stride=(1,), padding=same)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (drop1): Dropout(p=0.5, inplace=False)
)
(cblock4): SmallConvBlock(
  (conv1): Conv1d(512, 1024, kernel_size=(5,), stride=(1,), padding=same
        )
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (bn1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
  (drop1): Dropout(p=0.5, inplace=False)
)
(output): Linear(in_features=6144, out_features=4, bias=True)
```

LSTM Architecture

```
(block1): SmallBlock(
  (lstm1): LSTM(22, 128, batch_first=True)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (drop1): Dropout(p=0.5, inplace=False)
)
(block2): SmallBlock(
  (lstm1): LSTM(128, 256, batch_first=True)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (drop1): Dropout(p=0.5, inplace=False)
)
(block3): SmallBlock(
  (lstm1): LSTM(256, 512, batch_first=True)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (bn1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
  (drop1): Dropout(p=0.5, inplace=False)
)
(output): Linear(in_features=9216, out_features=4, bias=True)
```

CNN2 + LSTM1 Architecture

```
(cblock1): SmallConvBlock(
  (conv1): Conv1d(22, 128, kernel_size=(5,), stride=(1,), padding=same)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (drop1): Dropout(p=0.5, inplace=False)
)
(cblock2): SmallConvBlock(
  (conv1): Conv1d(128, 256, kernel_size=(5,), stride=(1,), padding=same)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (bn1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
  (drop1): Dropout(p=0.5, inplace=False)
)
(lstm1): SmallBlock(
  (lstm1): LSTM(256, 512, batch_first=True)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (drop1): Dropout(p=0.5, inplace=False)
)
(output): Linear(in_features=9216, out_features=4, bias=True)
```

CNN3 + LSTM1 Architecture

```
(cblock1): SmallConvBlock(
  (conv1): Conv1d(22, 128, kernel_size=(5,), stride=(1,), padding=same)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (drop1): Dropout(p=0.5, inplace=False)
)
(cblock2): SmallConvBlock(
  (conv1): Conv1d(128, 256, kernel_size=(5,), stride=(1,), padding=same)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (bn1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
  (drop1): Dropout(p=0.5, inplace=False)
)
(cblock3): SmallConvBlock(
  (conv1): Conv1d(256, 512, kernel_size=(3,), stride=(1,), padding=same)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (drop1): Dropout(p=0.5, inplace=False)
)
(lstm1): SmallBlock(
  (lstm1): LSTM(512, 1024, batch_first=True)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (bn1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
  (drop1): Dropout(p=0.5, inplace=False)
)
(output): Linear(in_features=6144, out_features=4, bias=True)
```

LSTM1 + CNN3 Architecture

```
(lstm1): SmallBlock(
  (lstm1): LSTM(22, 128, batch_first=True)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (bn1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
  (drop1): Dropout(p=0.5, inplace=False)
)
(cblock1): SmallConvBlock(
  (conv1): Conv1d(128, 256, kernel_size=(5,), stride=(1,), padding=same)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (drop1): Dropout(p=0.5, inplace=False)
)
(cblock2): SmallConvBlock(
  (conv1): Conv1d(256, 512, kernel_size=(5,), stride=(1,), padding=same)
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (drop1): Dropout(p=0.5, inplace=False)
)
(cblock3): SmallConvBlock(
  (conv1): Conv1d(512, 1024, kernel_size=(5,), stride=(1,), padding=same
        )
  (relu1): ReLU()
  (maxpool1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1,
        ceil_mode=False)
  (bn1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
  (drop1): Dropout(p=0.5, inplace=False)
)
(output): Linear(in_features=6144, out_features=4, bias=True)
```

GRU, CNN2 + GRU1, CNN3 + GRU1, and GRU1 + CNN3 are the same as their corresponding LSTM architectures with LSTM layers swapped out for GRU layers.