EEG Data Classification With CNN & RNN Architectures

Kalon Lucas Kelley
UID: 406039396
klkelley@ucla.edu

Steven Bash
UID: 805433693
stevenbash@ucla.edu

Talha Abadin
UID: 205614187
talhabadin@ucla.edu

Warren Kim
UID: 30600694
wjkim2311@ucla.edu

Abstract

We optimize the classification accuracy of different CNN and RNN architectures on EEG data of a single subject and on all subjects. We also evaluate the classification accuracy of one model as a function of time. We investigate the effects of data augmentation on classification accuracy. The EEG data, from BCI IV Dataset 2a [1], consists of 4 labels and 2115 trials across 9 subjects, with each trial having 22 channels of 1000 time bins. The highest test accuracy on augmented data for the optimized models were the CNN and CTransformer architectures, with 0.745 and 0.716 respectively. We found that classification accuracy of the CTransformer on augmented data improved by approximately 0.07 as it trained on more time bins. Data augmentation of the EEG data increased the classification accuracy of the CTransformer by 0.077.

1. Introduction

We chose five models to optimize the classification accuracy. There are two RNN architectures, two hybrid CNN architectures, and one CNN architecture.

1.1. Data Augmentation

We explored several data augmentation techniques aiming to increase model performance. All models were trained on data that was augmented with the following techniques: clipping, max pooling, averaging & gaussian noise, and subsampling[1]. We wanted to investigate whether augmentation techniques of channel dropout, smooth time mask, and time shifting would improve the classification accuracy[2]. When evaluating the classification accuracy as a function of time, we augmented the data using trivial augmentation. Based off previously conducted research, which specifically covered data augmentations for EEG data, we hypothesized that the non-trivial augmentation would significantly improve the test accuracy [3]. The intuition behind channel dropout is to make the model more robust by simulating instances where electrodes are malfunctioning or not collecting any data, and in the event that new data was presented where these instances occurred, the network could properly identify features despite the occurrences. The intuition behind smooth time mask is very similar; it would allow us to increase generalization by anticipating that tests would have gaps in the trial where all data is zero. The intuition behind time shifting was empirical, by inspecting some of the trials we would see that the major spikes signifying a certain class would sometimes be slightly before or after one another between trials. Using this, we concluded that the spike could be at many different locations, so adding data points with the spikes shifted over a random amount may help generalization.

[1]We will refer to these techniques as "trivial augmentations".
[2]We will refer to these techniques as "non-trivial augmentations".

1.2. Optimizer

All models were run with Adamax except for CTransformer which used AdamW.

1.3. Recurrent Neural Networks (RNNs)

Two Recurrent Neural Network architectures were implemented. The Long Short-Term Memory (LSTM), shown in Figure 7, served as a baseline to measure the relative performance with CRNN hybrid models. It stacked two LSTM layers. The Gated Recurrent Unit (GRU), shown in Figure 6, was implemented for its simpler structure, leading us to believe that it would reduce overfitting. There was one GRU layer applied to the input data. Both models were followed by a linear layer and log-softmax. Both models applied batch normalization and used the Rectified Linear Unit (ReLU) as the activation function.

1.4. Convolutional Neural Network (CNN)

The architecture of the Convolutional Neural Network is shown in Figure 2 and is based off of Figure 1 of Schirrmeister's paper [4]. It consists of four convolutional layers and one linear layer. The first layer aims to extract temporal features by applying a 1-dimensional convolution. The second layer aims to extract spatial features by applying a 2-dimensional convolution. The third and fourth convolu-

tional layers extract higher level spatial-temporal features. Finally, a dense linear layer followed by the log softmax loss function is applied to return a log-probability vector of the classes. All layers implement batch normalization and dropout for regularization. The activation function for the first and last layer is a Rectified Linear Unit (ReLU); the rest use an Exponential Linear Unit (ELU).

1.5. Convolutional Recurrent Neural Network (CRNN)

There are two Convolutional Recurrent Neural Network architectures.

1.5.1 Convolutional Long Short-Term Memory Neural Network (CLSTM)

The Convolutional Long Short-Term Memory architecture, shown in Figure 4, applies the same convolutional layers as the CNN. A dense linear layer is applied before passing it to the LSTM. A final dense linear layer is applied followed by a log softmax loss function that returns the log-probability vector of the classes. All layers implement batch normalization and dropout for regularization. The activation function for the first and last layer is a Rectified Linear Unit (ReLU); the rest use an Exponential Linear Unit (ELU).

1.5.2 Convolutional Gated Recurrent Unit Neural Network (CGRU)

The Convolutional Gated Recurrent Unit architecture, shown in Figure 3, is identical to the CLSTM model, but with a GRU layer in the place of the LSTM layer. This architecture was motivated by its simple structure relative to the LSTM.

1.6. Convolutional Transformer Neural Network (CTransformer)

The Convolutional Transformer architecture is shown in Figure 5 and implements a shallow convolutional block followed by a transformer, inspired by [2]. The first two layers extract temporal and spatial features respectively. The features are then passed to a positional encoding layer which injects positional information into the sequence. The transformer's self attention mechanism extracts relationships between features. Finally, a dense linear layer and the log softmax function is applied to return a log-probability vector of the classes.

2. Results

2.1. Trivial vs. Non-Trivial Data Augmentation

Experimentally, trivial data augmentation increased the classification accuracy of most models, as shown in Table 3. The largest increase was 0.077 by the CTransformer, going from 0.639 to 0.716 while the CGRU was the only model to decrease in classification accuracy, going from 0.659 to 0.612, a decrease of 0.047. The rest of the models saw an increase in classification accuracy between 0.02 to 0.05.

2.2. Testing Accuracy on Subject 1

We found experimentally that there was minimal difference in classification accuracy on all subjects versus subject 1, regardless of trivial or non-trivial data augmentation. Most models' increase or decrease in classification accuracy was less than 0.07 but there were some outliers. When testing only on subject 1 and training on non-trivial augmented data, accuracy for the CTransformer decreased by 0.116 while the LSTM-LSTM decreased by 0.093. Accuracy for the LSTM-LSTM trained on trivial augmented data increased by 0.114. See Table 1 and Table 2 for data.

2.3. CNN vs. RNN Architectures

Regardless of data augmentation, CNN has the highest classification accuracy. The CRNN hybrids (CGRU and CLSTM) and CTransformer performed approximately the same in classifying the trivial augmented data, but the CTransformer performed much better than the CRNN hybrids on non-trivial augmented data. The RNN architectures (GRU and LSTM-LSTM) performed poorly on both types of augmented data and the LSTM-LSTM severely overfits the data. See Table 1 and Table 2 for data.

3. Discussion

3.1. Trivial Data Augmentation vs. Non-Trivial Data Augmentation

We chose to augment the original data and then concatenate the augmented data to the original data, thereby doubling the amount. We chose this in order to generate a diverse set of noisy data and believe that this choice improved the generalization of models, thus decreasing the overfitting. We validated our intuition behind these additional augmentations empirically through the models as we consistently observed higher testing accuracy in all models except the CGRU. See this in Table 2. The addition of non-trivial augmentations increased the models' testing accuracy while also exhibiting decreased overfitting with the non-trivial augmentations. We hypothesize that the testing accuracy of specifically the CGRU decreased because the model itself was incompatible with the augmentations, causing it to be unable to learn the features to the same extent as the trivially augmented dataset. It is possible that our hyperparameter search was not extensive enough, leading to poor hyperparameters for the model and worse performance.

3.2. Accuracy on Subject 1

Initially, we trained the models on only subject 1 and then tested on all the subjects. The models severely overfit, most likely due to such little data being available to train on (221 trials for training, 16 trials for validation, and 50 trials for testing). As a result, the models generalized poorly. We switched to training the models on all the subjects and then testing them on subject 1. The models ended up performing similarly when tested on all subjects versus testing on subject 1 (see Table 1 & Table 2, column: Testing Accuracy Difference). This might be the case because, regardless of augmentation, the models are trained on a diverse set of data since it comes from all nine subjects. As a result, the models generalize much better to subject 1 data compared to models trained on subject 1 and tested on all the subjects. However, it performed similarly to the models tested on all subjects because the main difference is that we are testing on a subset of the test data. The models were trained on the same data, so its generalization performance would be approximately the same regardless of the testing data amount.

3.3. CNN vs. RNN Architectures

The CNN had the highest test accuracy on both trivial and non-trivial augmented data. EEG data is sequential in nature and has both spatial and temporal features. We believe that the CNN model might have performed the best because it was able to encapsulate all the spatial features of the data, which CNNs are normally good at doing, while also learning the temporal features of the data. It is possible that where the signal occurs in the brain (the electrode the signal originated from) may be more important than when the signal occurred. Although the EEG data is temporal in nature, it is possible that the temporal features were not as complex as the spatial features or perhaps the EEG data relied more on spatial features than temporal, so it was easier for the model to learn both features.

We found that performance decreased in any CNN model with more than four convolutional layers. The model became too complex and overfit the data because there were so many learnable parameters that allowed it to memorize the training data, resulting in poor generalization.

We found that applying a dense linear layer before the recurrent unit increased accuracy by about 0.35. We already had a dense linear layer after the recurrent unit, but this addition before the unit is what improved the accuracy. In hybrid models, we pass the data through convolutional layers to get high dimensional features and then pass it through a linear layer to reduce dimensionality, so the most prominent features are compact. We believe this allows for the CRNN models to learn the features better, resulting in better generalization. As a result, the CGRU, CTransformer, and CLSTM have better accuracy results than the RNN models.

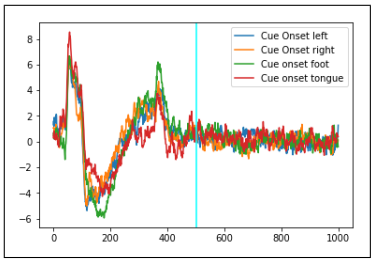
The RNN models both performed poorly on all tests. Vanilla RNNs have to learn the features themselves because it takes in the raw input, so they learn only sequentially. The lack of convolutional layers meant the models were unable to extract any spatial features. We think the poor performance may be due to a lack of learning as a result of vanishing gradients because we experimentally tried three layers of the same RNN and observed no learning, which implies vanishing gradients. We observed that the LSTM-LSTM model overfitted the data while still having poor test accuracy, but the GRU had similar training and testing accuracy, although it was also poor. The overfitting occurred because the LSTM-LSTM model was quite complex with 352,694 learnable parameters Figure 7, allowing it to memorize the data, while the GRU only had 192,438 learnable parameters Figure 6, which kept it simpler.

3.4. Function of Time

We chose to use the CTransformer model for evaluating classification accuracy as a function of time because the transformer architecture takes in sequential data and tries to find relationships between data coming in and the context that it has. We trained a new model every time, so as more time bins were allocated to training, the transformer would find connections between more time bins, which we believed would result in better performance. Initially, this did happen as the training, validation, and testing accuracy all improved significantly from 200 time bins to 400 time bins.

However, this increase could not be replicated. As time bins increased, the training accuracy remained approximately the same, validation decreased, and test accuracy remained around 0.6. The highest validation accuracy came when training on time bins 0-400 and the highest test accuracy came when training on time bins 0-800. This might be the case because, as shown in Figure 1 below, the noisiest data is between time bins 0 to 400 while the rest hovers around zero, so there was not much meaningful data for the model to learn patterns from. The accuracy for training, validation, and test decreased when time bins were increased to 1000, so we decided to clip the time bins to 800 for our trivial data augmentation. See this data in Table 4.

Figure 1. Channel 8 Time Bins Averaged Across 1000 Trials



References

[1] G. R. Müller-Putz A. Schlögl C. Brunner, R. Leeb and G. Pfurtscheller. Bci competition 2008 – graz data set a. Jan. 2008. 1

[2] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. Conformer: Convolution-augmented transformer for speech recognition, 2020. 2

[3] Cédric Rommel, Joseph Paillard, Thomas Moreau, and Alexandre Gramfort. Data augmentation for learning predictive models on eeg: a systematic comparison. *Journal of Neural Engineering*, 19(6):066020, Nov. 2022. 1

[4] Robin Tibor Schirrmeister, Jost Tobias Springenberg, Lukas Dominique Josef Fiederer, Martin Glasstetter, Katharina Eggensperger, Michael Tangermann, Frank Hutter, Wolfram Burgard, and Tonio Ball. Deep learning with convolutional neural networks for eeg decoding and visualization. *Human Brain Mapping*, 38(11):5391–5420, Aug. 2017. 1

Model Performance

Model	Training Accuracy	Validation Accuracy	Testing Accuracy on All Subjects	Testing Accuracy on Subject 1	Testing Accuracy Difference
CNN	0.936	0.684	0.716	0.780	-0.064
CGRU	0.948	0.732	0.659	0.600	0.059
CLSTM	0.914	0.565	0.619	0.660	-0.041
CTrans	0.869	0.642	0.639	0.660	-0.021
GRU	0.323	0.229	0.269	-	-
LSTM-LSTM	0.922	0.243	0.246	0.360	-0.114

Table 1. Training Across all Subjects with Trivial Data Augmentation

Model	Training Accuracy	Validation Accuracy	Testing Accuracy on All Subjects	Testing Accuracy on Subject 1	Testing Accuracy Difference
CNN	0.904	0.678	0.745	0.740	0.005
CGRU	0.893	0.610	0.612	0.660	-0.048
CLSTM	0.876	0.645	0.648	0.680	-0.032
CTrans	0.901	0.677	0.716	0.600	0.116
GRU	0.384	0.211	0.284	-	-
LSTM-LSTM	0.988	0.257	0.293	0.200	0.093

Table 2. Training Across all Subjects with Non-Trivial Data Augmentation

Model	Training Accuracy	Validation Accuracy	Testing Accuracy on All Subjects	Testing Accuracy on Subject 1
CNN	-0.032	-0.006	0.029	-0.040
CGRU	-0.055	-0.122	-0.047	0.060
CLSTM	-0.038	0.080	0.029	0.020
CTrans	0.032	0.035	0.077	-0.060
GRU	0.061	-0.018	0.015	-
LSTM-LSTM	0.066	0.014	0.047	-0.160

Table 3. Difference in Accuracy between Non-Trivial and Trivial Data Augmentation

Time Bins	Training Accuracy	Validation Accuracy	Testing Accuracy on All Subjects
200	0.832	0.586	0.582
400	0.908	0.705	0.625
600	0.919	0.638	0.582
800	0.886	0.568	0.655
1000	0.917	0.612	0.594

Table 4. Training CNN-Transformer on Data as a Function of Time

Model Architectures

Figure 2. CNN

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNN                                      [64, 4]                   --
├─Sequential: 1-1                        [64, 25, 396]             --
│    └─Conv1d: 2-1                       [64, 25, 396]             2,775
│    └─ReLU: 2-2                         [64, 25, 396]             --
│    └─BatchNorm1d: 2-3                  [64, 25, 396]             50
│    └─Dropout: 2-4                      [64, 25, 396]             --
├─Sequential: 1-2                        [64, 50, 1, 124]          --
│    └─Conv2d: 2-5                       [64, 50, 1, 372]          31,300
│    └─ELU: 2-6                          [64, 50, 1, 372]          --
│    └─BatchNorm2d: 2-7                  [64, 50, 1, 372]          100
│    └─MaxPool2d: 2-8                    [64, 50, 1, 124]          --
│    └─Dropout: 2-9                      [64, 50, 1, 124]          --
├─Sequential: 1-3                        [64, 100, 39]             --
│    └─Conv1d: 2-10                      [64, 100, 119]            30,100
│    └─ELU: 2-11                         [64, 100, 119]            --
│    └─BatchNorm1d: 2-12                 [64, 100, 119]            200
│    └─MaxPool1d: 2-13                   [64, 100, 39]             --
│    └─Dropout: 2-14                     [64, 100, 39]             --
├─Sequential: 1-4                        [64, 405, 11]             --
│    └─Conv1d: 2-15                      [64, 405, 35]             202,905
│    └─ELU: 2-16                         [64, 405, 35]             --
│    └─BatchNorm1d: 2-17                 [64, 405, 35]             810
│    └─MaxPool1d: 2-18                   [64, 405, 11]             --
│    └─Dropout: 2-19                     [64, 405, 11]             --
├─Linear: 1-5                            [64, 4]                   17,824
==========================================================================================
Total params: 286,064
Trainable params: 286,064
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 1.50
==========================================================================================
```

Figure 3. CGRU

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CGRU                                     [64, 4]                   --
├─Sequential: 1-1                        [64, 25, 396]             --
│    └─Conv1d: 2-1                       [64, 25, 396]             2,775
│    └─ReLU: 2-2                         [64, 25, 396]             --
│    └─BatchNorm1d: 2-3                  [64, 25, 396]             50
│    └─Dropout: 2-4                      [64, 25, 396]             --
├─Sequential: 1-2                        [64, 50, 1, 124]          --
│    └─Conv2d: 2-5                       [64, 50, 1, 372]          31,300
│    └─ELU: 2-6                          [64, 50, 1, 372]          --
│    └─BatchNorm2d: 2-7                  [64, 50, 1, 372]          100
│    └─MaxPool2d: 2-8                    [64, 50, 1, 124]          --
│    └─Dropout: 2-9                      [64, 50, 1, 124]          --
├─Sequential: 1-3                        [64, 100, 39]             --
│    └─Conv1d: 2-10                      [64, 100, 119]            30,100
│    └─ELU: 2-11                         [64, 100, 119]            --
│    └─BatchNorm1d: 2-12                 [64, 100, 119]            200
│    └─MaxPool1d: 2-13                   [64, 100, 39]             --
│    └─Dropout: 2-14                     [64, 100, 39]             --
├─Sequential: 1-4                        [64, 405, 11]             --
│    └─Conv1d: 2-15                      [64, 405, 35]             202,905
│    └─ELU: 2-16                         [64, 405, 35]             --
│    └─BatchNorm1d: 2-17                 [64, 405, 35]             810
│    └─MaxPool1d: 2-18                   [64, 405, 11]             --
│    └─Dropout: 2-19                     [64, 405, 11]             --
├─Sequential: 1-5                        [64, 200]                 --
│    └─Linear: 2-20                      [64, 200]                 891,200
│    └─ReLU: 2-21                        [64, 200]                 --
│    └─BatchNorm1d: 2-22                 [64, 200]                 400
│    └─Dropout: 2-23                     [64, 200]                 --
├─GRU: 1-6                               [64, 200]                 362,400
├─Linear: 1-7                            [64, 4]                   804
==========================================================================================
Total params: 1,523,044
Trainable params: 1,523,044
Non-trainable params: 0
```

Figure 4. CLSTM

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CLSTM                                    [64, 4]                   --
├─Sequential: 1-1                        [64, 25, 396]             --
│    └─Conv1d: 2-1                       [64, 25, 396]             2,775
│    └─ReLU: 2-2                         [64, 25, 396]             --
│    └─BatchNorm1d: 2-3                  [64, 25, 396]             50
│    └─Dropout: 2-4                      [64, 25, 396]             --
├─Sequential: 1-2                        [64, 50, 1, 124]          --
│    └─Conv2d: 2-5                       [64, 50, 1, 372]          31,300
│    └─ELU: 2-6                          [64, 50, 1, 372]          --
│    └─BatchNorm2d: 2-7                  [64, 50, 1, 372]          100
│    └─MaxPool2d: 2-8                    [64, 50, 1, 124]          --
│    └─Dropout: 2-9                      [64, 50, 1, 124]          --
├─Sequential: 1-3                        [64, 100, 39]             --
│    └─Conv1d: 2-10                      [64, 100, 119]            30,100
│    └─ELU: 2-11                         [64, 100, 119]            --
│    └─BatchNorm1d: 2-12                 [64, 100, 119]            200
│    └─MaxPool1d: 2-13                   [64, 100, 39]             --
│    └─Dropout: 2-14                     [64, 100, 39]             --
├─Sequential: 1-4                        [64, 405, 11]             --
│    └─Conv1d: 2-15                      [64, 405, 35]             202,905
│    └─ELU: 2-16                         [64, 405, 35]             --
│    └─BatchNorm1d: 2-17                 [64, 405, 35]             810
│    └─MaxPool1d: 2-18                   [64, 405, 11]             --
│    └─Dropout: 2-19                     [64, 405, 11]             --
├─Sequential: 1-5                        [64, 200]                 --
│    └─Linear: 2-20                      [64, 200]                 891,200
│    └─ReLU: 2-21                        [64, 200]                 --
│    └─BatchNorm1d: 2-22                 [64, 200]                 400
│    └─Dropout: 2-23                     [64, 200]                 --
├─LSTM: 1-6                              [64, 200]                 483,200
├─Linear: 1-7                            [64, 4]                   804
==========================================================================================
Total params: 1,643,844
Trainable params: 1,643,844
Non-trainable params: 0
```

Figure 5. CTrans

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CTransformer                             [64, 4]                   --
├─Sequential: 1-1                        [64, 25, 396]             --
│    └─Conv1d: 2-1                       [64, 25, 396]             2,775
│    └─ReLU: 2-2                         [64, 25, 396]             --
│    └─BatchNorm1d: 2-3                  [64, 25, 396]             50
│    └─Dropout: 2-4                      [64, 25, 396]             --
├─Sequential: 1-2                        [64, 200, 1, 124]         --
│    └─Conv2d: 2-5                       [64, 200, 1, 372]         125,200
│    └─ELU: 2-6                          [64, 200, 1, 372]         --
│    └─BatchNorm2d: 2-7                  [64, 200, 1, 372]         400
│    └─MaxPool2d: 2-8                    [64, 200, 1, 124]         --
│    └─Dropout: 2-9                      [64, 200, 1, 124]         --
├─Positional: 1-3                        [64, 124, 200]            --
│    └─Dropout: 2-10                     [64, 124, 200]            --
├─TransformerEncoder: 1-4                [64, 124, 200]            --
│    └─ModuleList: 2-11                  --                        --
│    │    └─TransformerEncoderLayer: 3-1 [64, 124, 200]            264,456
│    │    └─TransformerEncoderLayer: 3-2 [64, 124, 200]            264,456
│    │    └─TransformerEncoderLayer: 3-3 [64, 124, 200]            264,456
├─Linear: 1-5                            [64, 4]                   804
==========================================================================================
Total params: 922,597
Trainable params: 922,597
```

Figure 6. GRU

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
GRU                                      [64, 54]                  2,688
├─GRU: 1-1                               [64, 400, 128]            182,784
├─Linear: 1-2                            [64, 54]                  6,966
==========================================================================================
Total params: 192,438
Trainable params: 192,438
```

Figure 7. LSTM-LSTM

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
LSTM_LSTM                                [64, 54]                  2,688
├─LSTM: 1-1                              [64, 400, 128]            144,384
├─LSTM: 1-2                              [64, 400, 128]            198,656
├─Linear: 1-3                            [64, 54]                  6,966
==========================================================================================
Total params: 352,694
Trainable params: 352,694
```