# Deep Learning Methods for EEG Action Classification

William Smith
University of California, Los Angeles
williamsmith@ucla.edu

Alexander Chien
alexchien22@ucla.edu

Gabriel Castro
gcastro7@ucla.edu

Joe Lin
joelintech@ucla.edu

## Abstract

*This paper investigates the efficacy of deep learning methods for classifying actions based on electroencephalogram (EEG) signals. We explore various architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and hybrid models combining CNNs with Long Short-Term Memory (LSTM), Gated Recurrent Units (GRUs), and multihead attention mechanisms. Our results demonstrate that hybrid CNN-LSTM architectures are better at capturing the patterns of time-series data associated with EEG signals, leading to high classification accuracy. We also find that using longer time sequences and larger kernel sizes in convolutional layers leads to better performance. We discuss the insights gained from our experiments and highlight the importance of weight penalization and shallow network models for effective EEG action classification.*

## 1. Introduction

Convolutional Neural Networks have been at the pinnacle of recent deep learning advances and have shown proficiency in working with time-series data. For this reason, we heavily utilize this deep learning algorithm in the array of architectures we experiment with. It is also well known that the class of recurrent neural networks is greatly effective for sequential data. Hence, it was natural for us to investigate hybrid architectures, which utilized a CNN as an encoder and an RNN as a decoder.

## 2. Methods

We first establish the procedures used to preprocess the EEG dataset. This is followed by a summary of the neural network architectures considered and evaluated on the dataset. We then present our final model architecture and outline the ensembling process.

### 2.1. Preprocessing

The training data consists of EEG signals, which are high-frequency signals with a non-negligible amount of noise. To effectively process this data, we conduct a maxpooling-based subsampling procedure. This allows us to reduce the input sequence length and reduce the impact of signal noise. In our experiments, we stuck with a subsampling rate of 2. We perform other data augmentation procedures to make our model more robust to fluctuations, given the smaller size of the EEG dataset. The techniques implemented include: time reversal, sign flip, frequency shift, fourier transform surrogate, and channels dropout, each occurring with a probability of $p = 0.5$. These augmentations have been empirically shown to improve the accuracy of EEG classification tasks in [1].

### 2.2. Model Architectures

#### 2.2.1 Convolutional Neural Network

As a base, we train a vanilla convolutional neural network that consists of 1D convolutions operating on the sequence length dimension. These convolutions detect signal patterns of varying complexity and empirically produces features that can effectively classify subject actions.

#### 2.2.2 Recurrent Neural Network

RNNs are specifically designed to process sequential data by incorporating a hidden state that allows the retention of previous states. In terms of EEG data, RNNs can capture the temporal dependencies in the signal data. However, vanilla RNNs have the issue of exploding and vanishing gradients, especially with longer sequences, due to backpropagation through time. This leads to limited learning and poor network performance.

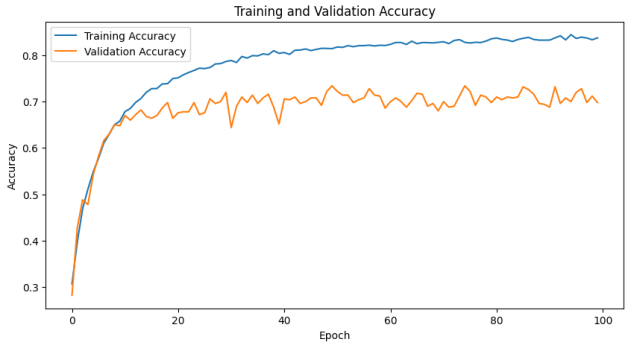


Figure 1. Vanilla CNN Accuracy

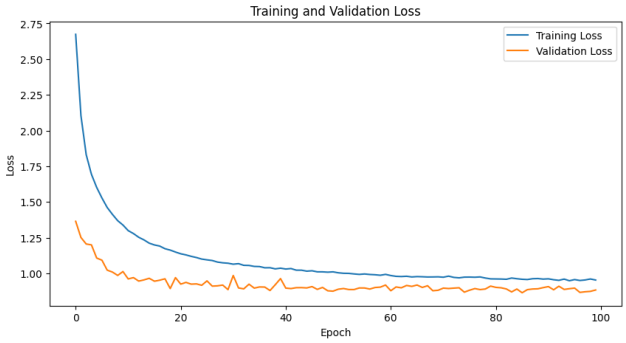


Figure 2. Vanilla CNN Loss

#### 2.2.3 CNN + Long Short-Term Memory

To solve the exploding/vanishing gradients issue in vanilla RNNs, Long-Short Term Memory networks (LSTMs) introduce gates that control the information flow within the cell. The addition of the cell state also allows us to backpropagate through fewer operations, acting as a gradient highway. We use convolutional layers to extract localized features from the spacial patterns of electrodes, and capture the temporal features using the LSTM. Combining these two architectures into a ConvLSTM will lead to more accurate classification as opposed to relying solely on either.

#### 2.2.4 CNN + Gated Recurrent Units

Gated Recurrent Units (GRUs) achieve a similar temporal classification effect as LSTMs, but utilizes a simpler gating mechanism allowing for model compactness and iteration efficiency. We experimented with GRUs to test their empirical performance in comparison to LSTMs and utilize the model in our final ensembling.

#### 2.2.5 CNN + Attention

While CNNs excel at extracting local features, they may struggle to capture long-term dependencies within EEG sequences, which are crucial for accurate classification. To compensate for these fallbacks, we introduce multihead attention (MHA) in our model to help capture long-range dependencies in the EEG signals. As proposed in [2], we use a MHA layer after the convolutional layers to attend to different parts of the extracted input sequence, allowing the model to learn complex relationships and dependencies across the time series data.

### 2.3. Model Ensembling

To strengthen our model performance, we ensemble our collection of models. The intuition behind this is that our diverse model architectures will be able to increase generalization as we assume their errors to be independent. However, we found that our CNN + RNN and CNN + Attention models had far lower validation accuracies, so we decided to only ensemble the vanilla CNN with the LSTM and GRU hybrid models.

Our ensembling method uses a majority vote of all model predictions to make the final prediction. We chose this approach over alternatives like averaging the logits, since averaging would directly impact the output probability distribution and give harsher predictions.

## 3. Results

### 3.1. Best Model Performance

The best ensemble classifier we obtained is trained on the first $8$ seconds of the EEG signal. We achieve $73.6\%$ test accuracy on the entire testing dataset (i.e. all subjects). Figures 1-6 display the loss and accuracy plots of training each model in our ensemble.

### 3.2. Training Distribution Experiments

**Result 1** *Training across all subjects improves classification accuracy compared to training on subject 1 alone.*

When training on subject 1 alone, we achieve a test accuracy of $44\%$. When training on all data, we achieve a test accuracy of $74\%$. So, by our empirical results, we can conclude that training across all subjects improves the classifier's performance on subject 1 data. This directly reflects our intuition that when given less data, the model is unable to learn generalizable features and thus performs poorly on unseen data.

### 3.3. Classifier Trends

**Result 2** *Shallow Nets improve classification accuracy.*

We found that our CNN hybrid models tend to overfit with large parameters and deep layers. This was especially true in the LSTM hybrid architecture, as reducing the number of convolutional layers to less than 4 dramatically improved the validation accuracy, as the model was able to consistently achieve above $70\%$ in validation accuracy.

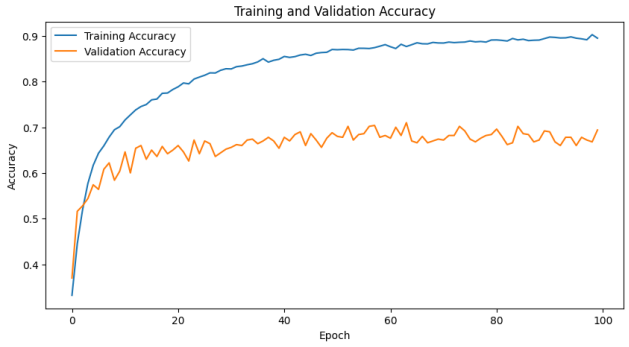


Figure 3. CNN + Long Short-Term Memory Accuracy


Figure 4. CNN + Long Short-Term Memory Loss

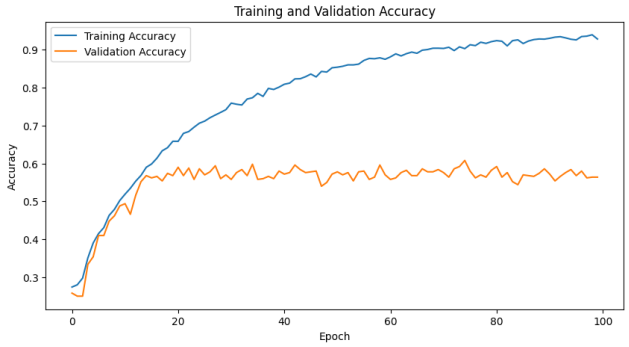


Figure 5. CNN + Gated Recurrent Unit Accuracy

Empirically, we noticed this trend along all our CNN hybrid models, as decreasing the number of layers and their hidden dimensions, allowed for classification accuracy.

**Result 3** *Larger kernel sizes improve convolution performance.*

When optimizing convolutional layer hyperparameters, we noticed a $5\%$ performance boost after changing kernel sizes from $3, 5, 7$ to $11, 13$. This aligns with our intuition that larger kernel sizes provide a larger receptive field, which means that the features from our CNN encoder will incorporate signal information from a larger temporal context.

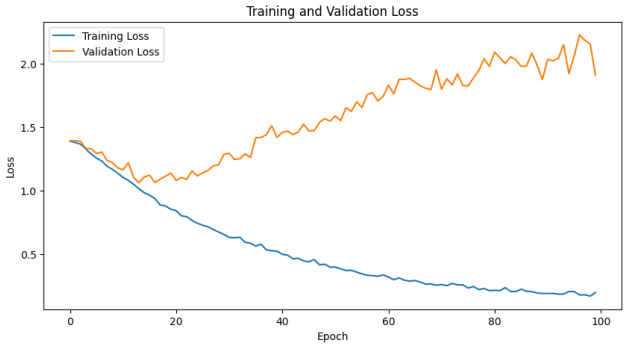


Figure 6. CNN + Gated Recurrent Unity Loss

### 3.4. Classifier Accuracy Over Time

**Result 4** *Training with longer time sequences strengthens model performance.*

Using longer time sequences (i.e., spanning from 0 to 800) from the EEG data has shown improved performance compared to utilizing shorter time window frames, such as those from 0 to 400. This improvement can be attributed to the diverse spatial patterns present within the longer time window. Consequently, the model becomes better at learning and capturing the signal patterns associated with specific motor activities.

## 4. Discussion

One insight we gained in regards to improving model performance was the importance of incorporating weight penalization into our loss functions. Specifically, the base CNN model empirically benefited from the integration of L1 loss, since intuitively it encourages the model to learn a set of sparser weights. This reduces model complexity and strengthens the model's ability to generalize to unseen data.

Through our experimentation, we also determine that a standalone RNN-based architecture is unable to achieve high classification accuracy. Instead, we found that prepending a CNN-based encoder (hybrid architecture) can significantly boost the model performance. The convolutional layers are crucial for the network to extract representative signal features that can be sequentially decoded by the recurrent layers.

Using 1d convolution layers in the CNN architectures to capture the temporal patterns of individual channels, rather than across multiple channels with 2d convolution layers, resulted in better performance. This shows that convolving along the time axis for individual channels was better for capturing the patterns of the test data.

## References


[1] Thomas Moreau Alexandre Gramfort Cédric Rommel, Joseph Paillard. Data augmentation for learning predictive models on eeg: a systematic comparison. *ArXiv*, abs/2206.14483, 2022. 1

[2] Tan-Hsu Tan, Yang-Lang Chang, Jun-Rong Wu, Yung-Fu Chen, and Mohammad Alkhaleefah. Convolutional neural network with multihead attention for human activity recognition. *IEEE Internet of Things Journal*, 11(2):3032–3043, 2024. 2

# 5. Appendix

## 5.1. Model Architectures and Training Details

Below are the summaries for all network architectures used in our experiments.

```
CNN(
  (conv_modules): ModuleList(
    (0): Sequential(
      (0): Conv1d(22, 32, kernel_size=(11,), stride=(1,), padding=(5,))
      (1): LeakyReLU(negative_slope=0.01)
      (2): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (3): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (4): Dropout(p=0.5, inplace=False)
    )
    (1): Sequential(
      (0): Conv1d(32, 64, kernel_size=(11,), stride=(1,), padding=(5,))
      (1): LeakyReLU(negative_slope=0.01)
      (2): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (3): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (4): Dropout(p=0.5, inplace=False)
    )
    (2): Sequential(
      (0): Conv1d(64, 128, kernel_size=(11,), stride=(1,), padding=(5,))
      (1): LeakyReLU(negative_slope=0.01)
      (2): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (3): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (4): Dropout(p=0.5, inplace=False)
    )
  )
  (fc1): Linear(in_features=6400, out_features=128, bias=True)
  (activation1): LeakyReLU(negative_slope=0.01)
  (fc2): Linear(in_features=128, out_features=64, bias=True)
  (activation2): LeakyReLU(negative_slope=0.01)
  (fc3): Linear(in_features=64, out_features=4, bias=True)
)
```

Figure 7. Vanilla CNN

```
RNN(
  (conv): ModuleList(
    (0): Conv1d(22, 32, kernel_size=(5,), stride=(1,), padding=(2,))
    (1): ELU(alpha=1.0)
    (2): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): Dropout1d(p=0.5, inplace=False)
    (5): Conv1d(32, 64, kernel_size=(5,), stride=(1,), padding=(2,))
    (6): ELU(alpha=1.0)
    (7): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): Dropout1d(p=0.5, inplace=False)
    (10): Conv1d(64, 128, kernel_size=(5,), stride=(1,), padding=(2,))
    (11): ELU(alpha=1.0)
    (12): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (13): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): Dropout1d(p=0.5, inplace=False)
    (15): Conv1d(128, 256, kernel_size=(5,), stride=(1,), padding=(2,))
    (16): ELU(alpha=1.0)
    (17): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (18): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (19): Dropout1d(p=0.5, inplace=False)
  )
  (rnn): RNN(256, 128, batch_first=True, dropout=0.1)
  (out): Linear(in_features=128, out_features=4, bias=True)
)
```

Figure 8. CNN + RNN

```
CNN_Attention_Model(
  (conv1): Conv1d(22, 64, kernel_size=(3,), stride=(1,))
  (conv2): Conv1d(64, 128, kernel_size=(3,), stride=(1,))
  (conv3): Conv1d(128, 256, kernel_size=(3,), stride=(1,))
  (pooling1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1, ceil_mode=False)
  (pooling2): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1, ceil_mode=False)
  (pooling3): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1, ceil_mode=False)
  (activation): ReLU()
  (multi_head_attention1): MultiheadAttention(
    (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features=256, bias=True)
  )
  (dropout): Dropout1d(p=0.5, inplace=False)
  (fc1): Linear(in_features=3328, out_features=18, bias=True)
  (fc2): Linear(in_features=18, out_features=4, bias=True)
)
```

Figure 9. CNN + Attention

```
GRU(
  (conv): ModuleList(
    (0): Conv1d(22, 32, kernel_size=(7,), stride=(1,), padding=(3,))
    (1): ELU(alpha=1.0)
    (2): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): Dropout(p=0.5, inplace=False)
    (5): Conv1d(32, 64, kernel_size=(7,), stride=(1,), padding=(3,))
    (6): ELU(alpha=1.0)
    (7): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): Dropout(p=0.5, inplace=False)
    (10): Conv1d(64, 128, kernel_size=(7,), stride=(1,), padding=(3,))
    (11): ELU(alpha=1.0)
    (12): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (13): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): Dropout(p=0.5, inplace=False)
  )
  (linear): Linear(in_features=128, out_features=256, bias=True)
  (gru): GRU(256, 256, batch_first=True)
  (out): Linear(in_features=256, out_features=4, bias=True)
)
```

Figure 10. CNN + GRU

```
LSTM(
  (conv): ModuleList(
    (0): Conv1d(22, 32, kernel_size=(11,), stride=(1,), padding=(5,))
    (1): ReLU()
    (2): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Dropout(p=0.4, inplace=False)
    (4): Conv1d(32, 64, kernel_size=(11,), stride=(1,), padding=(5,))
    (5): ReLU()
    (6): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (7): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): Dropout(p=0.4, inplace=False)
    (9): Conv1d(64, 128, kernel_size=(11,), stride=(1,), padding=(5,))
    (10): ReLU()
    (11): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): Dropout(p=0.4, inplace=False)
  )
  (lstm): LSTM(128, 64, batch_first=True)
  (out): Linear(in_features=12800, out_features=4, bias=True)
)
```

Figure 11. CNN + LSTM

### 5.2. Training Parameters

| Model | Optimizer | Activation | Learning Rate | Weight Decay | Label Smoothing | Batch Size |
|---|---|---|---|---|---|---|
| Vanilla CNN | Adam | LeakyReLU | $1 * 10^{-4}$ | $1 * 10^{-2}$ | 0.1 | 32 |
| CNN + RNN | Adam | ELU | $1 * 10^{-3}$ | None | None | 64 |
| CNN + Attention | Adam | ReLU | $1 * 10^{-2}$ | None | None | 64 |
| CNN + GRU | Adam | ELU | $1 * 10^{-3}$ | None | None | 64 |
| CNN + LSTM | Adam | ReLU | $1 * 10^{-4}$ | $1 * 10^{-2}$ | None | 32 |

### 5.3. Model Test Accuracies

**Time Window: 0-800**

| Training Scheme | Ensemble |
|---|---|
| Train on All, test on all | $73.6\%$ |
| Train on all, test on 1 | $74\%$ |
| Train on 1, test on all | $38.1\%$ |
| Train on 1, test on 1 | $44\%$ |

**Time Window: 0-400**

| Training Scheme | Ensemble |
|---|---|
| Train on all, test on all | $69.5\%$ |