

Deep Learning Algorithms for EEG Motor Function Classification

Lukas Hager
105601835
ECE C147

l.h.hager@gmail.com

Samuel Perrott
005930926
ECE C147

sperrott@ucla.edu

Reetnav Das
005504665
ECE C147

reetnav@g.ucla.edu

Aryan
705796873
ECE C147

asingh03@g.ucla.edu

Abstract

This project employs a three-pronged approach to understanding, working with, and classifying EEG data. We begin by developing 7 distinct architectures based on convolutional neural networks (CNNs), recurrent neural networks (RNNs), and a particular random convolutional kernel model called ROCKET, amongst others, to determine which model has the highest classification accuracy. Our findings show that CNN architecture alone is most efficient with a test accuracy of 71%. Subsequently, leveraging this vanilla CNN, we compare the classification accuracy between training the model on a single patient versus training it on all nine patients. Finally, we conduct an analysis of varying sample lengths to determine whether all 1000 time points in our dataset are necessary for constructing accurate models, or if most features can be extracted within shorter time intervals.

1. Introduction

Electroencephalogram (EEG) technology has emerged as a powerful tool in neuroscience, offering insights into brain activity through relatively non-invasive measurement of electrical impulses generated by the brain [10].

To work with it, however, poses a set of challenges due to its high dimensionality, and extensive noise composition. In the past, several CNN-based architectures have been employed to decode EEG data [8, 9]. We explored whether other deep learning architectures could achieve comparable performance. In order to do so, we designed a series of models using Keras, an open source deep-learning Python library, and trained them on a version of BCI's Competition IV dataset [1].

1.1. Data and Preprocessing

The data consists of 2115 training samples and 443 test samples relatively evenly distributed across 9 different patients. There are 4 possible label values corresponds to a patient imagining moving left, right, their foot, or their tongue.

Each sample consists of 1000 millisecond steps of time series collected from 22 different electrodes.

We split the provided training data into two portions, 90% used for training and 10% for validation. For our training set, we added Gaussian noise as a regularization factor. All our data was also maxpooled by a subsampling factor of 2 to make the time series shorter so that our RNN networks could capture more information. For specific architecture details, please refer to section 4.1 of the appendix "Model Architectures."

1.2. Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a type of neural network particularly effective for processing data with a grid-like topology such as images, making them well-suited for a variety of tasks, including image recognition, natural language processing, and bio-signal analysis [7]. In the context of EEG signal processing, CNNs can be tailored to exploit the spatial and temporal characteristics of the data, enhancing the ability to classify or interpret neural signals.

1D Convolutions

In 2016, DeepMind researchers introduced *WaveNet*, a novel architecture that stacked 1D CNN layers, doubling the dilation rate each layer [11]. This allows lower layers to learn short-term patterns and higher layers to learn long-term patterns, effectively allowing CNNs to process time-series data.

2D Convolutions

When applying 2D convolutions to EEG data, the input is often reshaped or transformed to suit the requirements of a 2D convolutional layer. In this case, the input format is (samples, pooled time steps, 1, channels), where the data is treated as a single-row image with varying time steps and multiple channels. This allows one to process temporal sequences across multiple electrodes simultaneously, en-

abling the network to learn patterns that span both time and spatial dimensions of the electrode array.

3D Convolutions

3D convolutions extend the concept further by incorporating an additional spatial dimension into the analysis, making them especially suitable for data that includes spatial relationships along with time. For our EEG data, where spatial relationships between the 22 electrodes provide important context for brain activity, the input format for 3D convolutions is (samples, electrode x coord, electrode y coord, pooled time steps, 1). We used a rough approximation of the electrode’s relative positions in a 7×7 grid to relate them spatially (see Figure 8). This augmented format treats the EEG data as a 3D volume, where the spatial arrangement of electrodes is preserved along two dimensions and the temporal dimension is considered as the third.

1.3. Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are unique in that weights are shared across sequential layers, fostering an web of connections where the output of each computation can influence subsequent computations. This architecture enables the network to maintain a continuous contextual thread throughout the data processing sequence by storing information obtained from all previous time steps into a single input variable $\mathbf{h}_{(t-1)}$ in addition to \mathbf{x}_t . Thus, RNNs are particularly adept at handling time-series data, as their unique configuration allows for the temporal dimension of information to be conserved and integrated into the learning process. In our exploration, we evaluated both simple RNN structures alongside hybrid architectures that combined RNN layers with CNNs. We tested a simple RNN, although it did not perform well.

1.3.1 LSTM

Regular RNNs can suffer from vanishing gradients [4] over many cycles. This decreases the model’s ability to learn long term relationships. Long-Short-Term-Memory (LSTM) networks address this by incorporating enhancements such as forget and input gates and having two inputs, $\mathbf{c}_{(t-1)}$ and $\mathbf{h}_{(t-1)}$, representing the long-term state and the short-term state, in addition $\mathbf{x}_{(t)}$. These LSTM cells allow the model to regulate the transfer of information better, ensuring that the earliest information in the time series is not lost since RNNs often suffer from short term memory [6].

We decided to test two LSTM variants. We created a 3 Conv2D + 1 LSTM and a 4 Conv2D + 2 LSTM. Our 3 Conv2D + LSTM model used a trim of 800 and a batch size of 64. One major change we did with this model compared to the others was to change our MaxPool padding mode from ‘same’ to ‘valid’. This makes the MaxPool layers

downscale the channels instead of keeping them the same, speeding up training time while roughly keeping the same level of performance. We used a fully connected layer after the convolutional layers to decrease the temporal size of the input tensor before passing it into the LSTM layer.

1.3.2 GRU

The *gated recurrent unit* (GRU) cell [2] is a simplified version of the LSTM cell with similar performance [5]. Unlike the standard LSTM, a GRU merges both state vectors into a single vector $\mathbf{h}_{(t)}$, uses a single gate controller for both forget and input gate, and has no output gate. Here, we utilize a hybrid CNN + GRU architecture.

1.4. ROCKET

We tested out a method from sktime called ROCKET, which provides exceptionally fast and accurate time series classification using random convolutional kernels to transform a dataset [3]. From there it can be fitted using a linear classifier. This was a promising idea because of how successful convolutional kernels are at doing time series classification, and our data was time series data. The creators of this module recommend to use SciKit’s RidgeCV classifier, a linear classification model, after transforming the dataset, so that’s what we used.

2. Results

The best performing models were a pure CNN at 71% and a CNN+LSTM at 72%. The pure CNN was able to achieve similar levels of test accuracy in 50 epochs, while the CNN+LSTM achieved it in 200 epochs, demonstrating the efficiency of a multilayer CNN. We found that a learning rate of 0.001 using the Adam optimizer led to most optimal results. We also implemented several regularization techniques such as L_2 regularization, batch normalization, and dropout layers after each convolutional layer. Please see Table 1 to observe how performance changes with model architecture and hyperparameters and Appendix 2 for layer dimensions and kernel sizes used. In general, our stacked CNN layers had increasing number of filters to compensate for the decreasing dimensionality of doing pooling, and the number of parameters was kept to around 700,000 to ensure our models trained efficiently.

2.1. Subject-Specific Training

We also trained our top models on Participant 1 only. We wanted to see which of the two models would perform better when being tested on Participant 1. We found that the CNN + LSTM model overfit more as compared to the vanilla CNN (see Table 2).

2.2. Trim Variation

Finally, we assessed our model’s performance by inputting data from various sized time intervals. To achieve this, we trained ten different models. The first model was trained using only the initial 100 data points across all channels, followed by the second model trained with the first 200 points, and so forth. Our last model incorporated all 1000 data points. We say that a model trained with only the first 100 time samples has Trim 100. Each model utilized a vanilla CNN architecture comprising four two-dimensional convolutional layers, where each model ran for 50 epochs with a learning rate of 0.001. See Table 4.

3. Discussion

3.1. Models

3.1.1 CNN

Our standalone CNN performed very well. One possible reason is that stacked CNNs tend to be less affected by changes over time compared to RNNs [6]. By aggregating spatially adjacent features over layers, the CNN can prioritize the most pertinent temporal information, potentially making it less susceptible to noise. Our vanilla CNN also outperformed the CNN+GRU architecture. This can be attributed to the fact that increased model complexity doesn’t necessarily mean improved performance; in fact, overly complex architectures might overfit on the training data, diminishing their ability to generalize, as the ROCKET and RNN architectures do. Our CNN architecture performs well because it is able to effectively process and transform the time-series data as if it were spatial.

3.1.2 CNN+LSTM

Our CNN+LSTM model also performed well. This makes sense because this architecture allows us to combine spatial and temporal information: EEG signals are multidimensional data, typically consisting of multiple channels (electrodes) recording brain activity over time. CNNs can capture spatial information by convolving over different channels, while LSTMs can model temporal dependencies within and across these channels. Combining both CNNs and LSTMs allows for effective utilization of both spatial and temporal information in the EEG data.

3.1.3 ROCKET

The ROCKET (Random Convolutional Kernel Transform) coupled with the RidgeCV linear classifier ended up doing better than the baseline, achieving a training accuracy of 0.95 and a testing accuracy of 0.5. One major limiting factor for the ROCKET model was that we were constrained by

our dataset size leading to overfitting as we saw high training accuracies, but poor testing accuracies in comparison. The issue of overfitting is likely due to the dimensionality expansion caused by running many convolutional kernels. This complexity likely leads to overfitting on small EEG datasets, explaining the performance we got.

3.2. RNN

Our standalone RNN model performed very poorly. One reason the RNN performed poorly is because it suffers from poor long term memory, whereas stacking multiple convolutional layers allows for information to be condensed and propagated through the model. Moreover, while RNNs are suitable for shorter sequential data like time series, EEG signals can have complex temporal dynamics that might not be effectively captured by a simple RNN architecture.

3.3. Subject Specific Training

We found that training on data from all nine participants led to better results than training only on one participant. Training on all nine participants helps the model generalize better, in that the model’s performance improves because it has learned more generalizable patterns from the additional data. Moreover, training on all nine participants can be seen as a form of data augmentation, as it can help diversify the training data, which in turn can prevent overfitting and improve the model’s ability to generalize to new patients. This can also help with noise reduction, allowing our model to hone in on more consistent and reliable patterns in our EEG data.

3.4. Trim Testing

We found that 4 Conv2D was able to reach similar levels of classification accuracy given only the first 300 time steps, instead of using all 1000. We hypothesize that this trend is due to a few reasons. For one, temporal dynamics might come in to play as the brain’s response to stimuli or tasks can be time-sensitive. Certain cognitive processes might occur rapidly and be captured within the first few hundred time points of our EEG recording. Moreover, trimming might also serve as a form of noise reduction, as they may contain less noise compared to the rest of the signal. By focusing on the initial portion, you might be inadvertently excluding sections with higher noise levels, thereby improving signal quality. Initial portions may also encode information for future time signals as well. Finally, trimming reduces the dimensionality of our data, which can help with model training.

References

- [1] Clemens Brunner, Robert Leeb, Gernot Müller-Putz, Alois Schlögl, and Gert Pfurtscheller. Data sets 2a: 4-class motor imagery. Institute for Knowledge Discovery (Laboratory

of Brain-Computer Interfaces), Graz University of Technology, 2008. EEG, cued motor imagery (left hand, right hand, feet, tongue), [22 EEG channels (0.5-100Hz; notch filtered), 3 EOG channels, 250Hz sampling rate, 4 classes, 9 subjects]. Dataset available online: <https://www.bbci.de/competition/iv/>. **1**

- [2] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014. **2**
- [3] Angus Dempster, François Petitjean, and Geoffrey I Webb. ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, 2020. **2**
- [4] Rian Dolphin. Lstm networks — a detailed explanation. 2020. **2**
- [5] Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, Oct. 2017. **2**
- [6] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O’Reilly, 2023. **2, 3**
- [7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. **1**
- [8] Vernon J Lawhern, Amelia J Solon, Nicholas R Waytowich, Stephen M Gordon, Chou P Hung, and Brent J Lance. Eegnet: a compact convolutional neural network for eeg-based brain–computer interfaces. *Journal of Neural Engineering*, 15(5):056013, jul 2018. **1**
- [9] Robin Tibor Schirrmeister, Jost Tobias Springenberg, Lukas Dominique Josef Fiederer, Martin Glasstetter, Katharina Eggersperger, 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**
- [10] Alois Schlögl, Mel Slater, and Gert Pfurtscheller. Presence research and eeg. 01 2002. **1**
- [11] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016. **1**

4. Appendix

Architecture	Trim	Epochs	Batch Size	Train Accuracy	Validation Accuracy	Test Accuracy
4 Conv2D	550	50	64	0.88	0.67	0.71
3 Conv2D + 1 LSTM	800	200	64	0.79	0.71	0.72
4 Conv2D + 2 LSTM	550	150	32	0.72	0.69	0.68
9 Conv1D + 1 GRU	800	100	64	0.87	0.64	0.62
2 Conv3D + 1 GRU	550	100	64	0.98	0.51	0.52
ROCKET + RidgeCV	1000	—	—	0.95	—	0.50
4 RNN	600	100	64	0.88	0.34	0.35

Table 1. Model accuracies, architecture types, and hyperparameters. Note: all models used a learning rate of 0.001 with the Adam optimizer.

Architecture	Train Accuracy	Validation Accuracy	Accuracy (Subject 1)	Accuracy (Other Subjects)
4 Conv2D	0.96	0.46	0.58	0.38
4 Conv2D + 1 LSTM	0.92	0.67	0.64	0.35

Table 2. Comparison of Top Performing Models Trained on Subject 1.

Training Feature		Notes
Convolutional Layer Regularizations	L_2 Regularization	$\lambda = 0.001$
	Activation	ReLU, elu, swish
	Dropout	0.6, 0.2, 0.5
Epochs	Varied	50, 100, 150
Batch Size	Varied	32, 64
Loss function	Cross Entropy Loss	4 classes
Optimizer	Adam	learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.99$
Data Augmentations	Max Pooling	pooling window size of 2
	Averaging with Noise	averaging window size of 2, noise scale of 0.5
	Subsampling with Noise	subsampling window size of 2, noise scale of 0.5

Table 3. Hyperparameters and Regularizations Tested

Time Cutoff	Testing Accuracy
100	0.5192
200	0.6275
300	0.7201
400	0.7133
500	0.6997
600	0.7065
700	0.6885
800	0.6682
900	0.7088
1000	0.6795

Table 4. Testing Accuracy for Different Trims with 4 Conv2D

4.1. Model Architectures

[illegible]

Figure 1. 4 Conv2D Architecture

Layer (type)	Output Shape	Param #
conv2d_181 (Conv2D)	(None, 488, 1, 251)	1875
max_pooling2d_191 (MaxPooling2D)	(None, 132, 1, 251)	0
batch_normalization_191 (BatchNormalization)	(None, 132, 1, 251)	100
conv2d_190 (Conv2D)	(None, 132, 1, 251)	0
conv2d_192 (Conv2D)	(None, 132, 1, 50)	3000
max_pooling2d_192 (MaxPooling2D)	(None, 44, 1, 50)	0
batch_normalization_192 (BatchNormalization)	(None, 44, 1, 50)	200
conv2d_198 (Conv2D)	(None, 44, 1, 50)	0
conv2d_193 (Conv2D)	(None, 44, 1, 100)	15300
max_pooling2d_193 (MaxPooling2D)	(None, 14, 1, 100)	0
batch_normalization_193 (BatchNormalization)	(None, 14, 1, 100)	400
conv2d_199 (Conv2D)	(None, 14, 1, 100)	0
flatten_47 (Flatten)	(None, 1408)	0
dense_58 (Dense)	(None, 1408)	5048
conv2d_200 (Conv2D)	(None, 48)	0
conv2d_201 (Conv2D)	(None, 1)	0
flatten_48 (Flatten)	(None, 1)	0
dense_59 (Dense)	(None, 1)	100
dense_60 (Dense)	(None, 4)	4

Total params: 77839 (344.40 KB)
Trainable params: 77840 (344.40 KB)
Non-trainable params: 558 (13.87 KB)

Figure 2. 3 Conv2D + 1 LSTM Architecture

[illegible]

Figure 3. 4 Conv2D + 2 LSTM Architecture

[illegible]

Figure 4. 9 Conv1D + 1 GRU Architecture

Leaf Name	Original Name	Parent P
base_000_000000	None	0
base_000_000001	None	1
base_000_000002	None	2
base_000_000003	None	3
base_000_000004	None	4
base_000_000005	None	5
base_000_000006	None	6
base_000_000007	None	7
base_000_000008	None	8
base_000_000009	None	9
base_000_000010	None	10
base_000_000011	None	11
base_000_000012	None	12
base_000_000013	None	13
base_000_000014	None	14
base_000_000015	None	15
base_000_000016	None	16
base_000_000017	None	17
base_000_000018	None	18
base_000_000019	None	19
base_000_000020	None	20
base_000_000021	None	21
base_000_000022	None	22
base_000_000023	None	23
base_000_000024	None	24
base_000_000025	None	25
base_000_000026	None	26
base_000_000027	None	27
base_000_000028	None	28
base_000_000029	None	29
base_000_000030	None	30
base_000_000031	None	31
base_000_000032	None	32
base_000_000033	None	33
base_000_000034	None	34
base_000_000035	None	35
base_000_000036	None	36
base_000_000037	None	37
base_000_000038	None	38
base_000_000039	None	39
base_000_000040	None	40
base_000_000041	None	41
base_000_000042	None	42
base_000_000043	None	43
base_000_000044	None	44
base_000_000045	None	45
base_000_000046	None	46
base_000_000047	None	47
base_000_000048	None	48
base_000_000049	None	49
base_000_000050	None	50
base_000_000051	None	51
base_000_000052	None	52
base_000_000053	None	53
base_000_000054	None	54
base_000_000055	None	55
base_000_000056	None	56
base_000_000057	None	57
base_000_000058	None	58
base_000_000059	None	59
base_000_000060	None	60
base_000_000061	None	61
base_000_000062	None	62
base_000_000063	None	63
base_000_000064	None	64
base_000_000065	None	65
base_000_000066	None	66
base_000_000067	None	67
base_000_000068	None	68
base_000_000069	None	69
base_000_000070	None	70
base_000_000071	None	71
base_000_000072	None	72
base_000_000073	None	73
base_000_000074	None	74
base_000_000075	None	75
base_000_000076	None	76
base_000_000077	None	77
base_000_000078	None	78
base_000_000079	None	79
base_000_000080	None	80
base_000_000081	None	81
base_000_000082	None	82
base_000_000083	None	83
base_000_000084	None	84
base_000_000085	None	85
base_000_000086	None	86
base_000_000087	None	87
base_000_000088	None	88
base_000_000089	None	89
base_000_000090	None	90
base_000_000091	None	91
base_000_000092	None	92
base_000_000093	None	93
base_000_000094	None	94
base_000_000095	None	95
base_000_000096	None	96
base_000_000097	None	97
base_000_000098	None	98
base_000_000099	None	99
base_000_000100	None	100
base_000_000101	None	101
base_000_000102	None	102
base_000_000103	None	103
base_000_000104	None	104
base_000_000105	None	105
base_000_000106	None	106
base_000_000107	None	107
base_000_000108	None	108
base_000_000109	None	109
base_000_000110	None	110
base_000_000111	None	111
base_000_000112	None	112
base_000_000113	None	113
base_000_000114	None	114
base_000_000115	None	115
base_000_000116	None	116
base_000_000117	None	117
base_000_000118	None	118
base_000_000119	None	119
base_000_000120	None	120
base_000_000121	None	121
base_000_000122	None	122
base_000_000123	None	123
base_000_000124	None	124
base_000_000125	None	125
base_000_000126	None	126
base_000_000127	None	127
base_000_000128	None	128
base_000_000129	None	129
base_000_000130	None	130
base_000_000131	None	131
base_000_000132	None	132
base_000_000133	None	133
base_000_000134	None	134
base_000_000135	None	135
base_000_000136	None	136
base_000_000137	None	137
base_000_000138	None	138
base_000_000139	None	139
base_000_000140	None	140
base_000_000141	None	141
base_000_000142	None	142
base_000_000143	None	143
base_000_000144	None	144
base_000_000145	None	145
base_000_000146	None	146
base_000_000147	None	147
base_000_000148	None	148
base_000_000149	None	149
base_000_000150	None	150
base_000_000151	None	151
base_000_000152	None	152
base_000_000153	None	153
base_000_000154	None	154
base_000_000155	None	155
base_000_000156	None	156
base_000_000157	None	157
base_000_000158	None	158
base_000_000159	None	159
base_000_000160	None	160
base_000_000161	None	161
base_000_000162	None	162
base_000_000163	None	163
base_000_000164	None	164
base_000_000165	None	165
base_000_000166	None	166
base_000_000167	None	167
base_000_000168	None	168
base_000_000169	None	169
base_000_000170	None	170
base_000_000171	None	171
base_000_000172	None	172
base_000_000173	None	173
base_000_000174	None	174
base_000_000175	None	175
base_000_000176	None	176
base_000_000177	None	177
base_000_000178	None	178
base_000_000179	None	179
base_000_000180	None	180
base_000_000181	None	181
base_000_000182	None	182
base_000_000183	None	183
base_000_000184	None	184
base_000_000185	None	185
base_000_000186	None	186
base_000_000187	None	187
base_000_000188	None	188
base_000_000189	None	189
base_000_000190	None	190
base_000_000191	None	191
base_000_000192	None	192
base_000_000193	None	193
base_000_000194	None	194
base_000_000195	None	195
base_000_000196	None	196
base_000_000197	None	197
base_000_000198	None	198
base_000_000199	None	199
base_000_000200	None	200
base_000_000201	None	201
base_000_000202	None	202
base_000_000203	None	203
base_000_000204	None	204
base_000_000205	None	205
base_000_000206	None	206
base_000_000207	None	207
base_000_000208	None	208
base_000_000209	None	209
base_000_000210	None	210
base_000_000211	None	211
base_000_000212	None	212
base_000_000213	None	213
base_000_000214	None	214
base_000_000215	None	215
base_000_000216	None	216
base_000_000217	None	217
base_000_000218	None	218
base_000_000219	None	219
base_000_000220	None	220
base_000_000221	None	221
base_000_000222	None	222
base_000_000223	None	223
base_000_000224	None	224
base_000_000225	None	225
base_000_000226	None	226
base_000_000227	None	227
base_000_000228	None	228
base_000_000229	None	229
base_000_000230	None	230
base_000_000231	None	231
base_000_000232	None	232
base_000_000233	None	233
base_000_000234	None	234
base_000_000235	None	235
base_000_000236	None	236
base_000_000237	None	237
base_000_000238	None	238
base_000_000239	None	239
base_000_000240	None	240
base_000_000241	None	241
base_000_000242	None	242
base_000_000243	None	243
base_000_000244	None	244
base_000_000245	None	245
base_000_000246	None	246
base_000_000247	None	247
base_000_000248	None	248
base_000_000249	None	249
base_000_000250	None	250
base_000_000251	None	251
base_000_000252	None	252
base_000_000253	None	253
base_000_000254	None	254
base_000_000255	None	255
base_000_000256	None	256
base_000_000257	None	257
base_000_000258	None	258
base_000_000259	None	259
base_000_000260	None	260
base_000_000261	None	261
base_000_000262	None	262
base_000_000263	None	263
base_000_000264	None	264
base_000_000265	None	265
base_000_000266	None	266
base_000_000267	None	267
base_000_000268	None	268
base_000_000269	None	269
base_000_000270	None	270
base_000_000271	None	271
base_000_000272	None	272
base_000_000273	None	273
base_000_000274	None	274
base_000_000275	None	275
base_000_000276	None	276
base_000_000277	None	277
base_000_000278	None	278
base_000_000279	None	279
base_000_000280	None	280
base_000_000281	None	281
base_000_000282	None	282
base_000_000283	None	283
base_000_000284	None	284
base_000_000285	None	285
base_000_000286	None	286
base_000_000287	None	287
base_000_000288	None	288
base_000_000289	None	289
base_000_000290	None	290
base_000_000291	None	291
base_000_000292	None	292
base_000_000293	None	293
base_000_000294	None	294
base_000_000295	None	295
base_000_000296	None	296
base_000_000297	None	297
base_000_000298	None	298
base_000_000299	None	299
base_000_000300	None	300
base_000_000301	None	301
base_000_000302	None	302
base_000_000303	None	303
base_000_000304	None	304
base_000_000305	None	305
base_000_000306	None	306
base_000_000307	None	307
base_000_000308	None	308
base_000_000309	None	309
base_000_000310	None	310
base_000_000311	None	311
base_000_000312	None	312
base_000_000313	None	313
base_000_000314	None	314
base_000_000315	None	315
base_000_000316	None	316
base_000_000317	None	317
base_000_000318	None	318
base_000_000319	None	319
base_000_000320	None	320
base_000_000321	None	321
base_000_000322	None	322
base_000_000323	None	323
base_000_000324	None	324
base_000_000325	None	325
base_000_000326	None	326
base_000_000327	None	327
base_000_000328	None	328
base_000_000329	None	329
base_000_000330	None	330
base_000_000331	None	331
base_000_000332	None	332
base_000_000333	None	333
base_000_000334	None	334
base_000_000335	None	335
base_000_000336	None	336
base_000_000337	None	337
base_000_000338	None	338
base_000_000339	None	339
base_000_000340	None	340
base_000_000341	None	341
base_000_000342	None	342
base_000_000343	None	343
base_000_000344	None	344
base_000_000345	None	345
base_000_000346	None	346
base_000_000347	None	347
base_000_000348	None	348
base_000_000349	None	349
base_000_000350	None	350
base_000_000351	None	351
base_000_000352	None	352
base_000_000353	None	353
base_000_000354	None	354
base_000_000355	None	355
base_0		

Figure 5. 2 Conv3D + 1 GRU Architecture

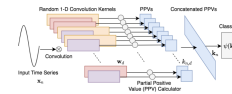
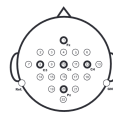


Figure 6. ROCKET + RidgeCV Architecture

Layer type	Original Shape	Param #
conv1a_1 (Conv2D)	(None, 28, 28, 64)	5568
batch_normalization_12 (Batch Normalization)	(None, 28, 28, 64)	256
max_pooling1d_1 (MaxPooling1D)	(None, 112, 64)	0
conv2a_1 (Conv2D)	(None, 112, 64)	0
conv1a_2 (Conv2D)	(None, 112, 64)	8256
batch_normalization_13 (Batch Normalization)	(None, 112, 64)	256
max_pooling1d_2 (MaxPooling1D)	(None, 75, 64)	0
conv2a_2 (Conv2D)	(None, 75, 64)	0
conv1a_3 (Conv2D)	(None, 75, 64)	8256
batch_normalization_14 (Batch Normalization)	(None, 75, 64)	256
max_pooling1d_3 (MaxPooling1D)	(None, 37, 64)	0
conv2a_3 (Conv2D)	(None, 37, 64)	0
conv1a_4 (Conv2D)	(None, 65)	8256
batch_normalization_15 (Batch Normalization)	(None, 64)	256
conv2a_4 (Conv2D)	(None, 64)	0
conv1a_5 (Conv2D)	(None, 42)	288

Figure 7. 4 RNN Architecture



7		2				
	8		3		1	
14		9		4		
	15		10		5	
19		16		11		6
	20		17		12	
22		21		18		13

(A) Electrode Placement (B) Electrode Numbering

Figure 8. EEG Electrode Placement and Numbering