Classification of Motor Imagery EEG data

Fuzail Khan, Sangmin Lim, Derek Nguyen, and Ganesha Srivallabha
UCLA ECE C247 - UIDs: 405428622, 205224428, 905629053, 405291327

Abstract

Our experiments involve exploring data pre-processing techniques, Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs) in order to classify Motor Imagery from EEG data. The data pre-processing techniques include Power Spectral Density, Principal Component Analysis, and Sequential Sampling. The data is then inputted to models utilizing Convolutional, Batch Normalization, Dropout, Pooling, and Long Short Term Memory (LSTM) Layers. Our data pre-processing had no major performance benefits, but allowed for unique insights on the data. Furthermore, LSTMs provided no benefit and models without LSTMs performed the best (Deep CNN with 73% Test Accuracy). Additional experiments were conducted to determine model performance when training over single subjects and model accuracy as input data length is varied.

1. Introduction

With the purpose of achieving the highest accuracy possible, our tasks were categorized in to three parts: 1) Pre-Processing Data, 2) Developing Effective Neural Network (NN) Architectures, and 3) integrate the pre-processing with the NN architectures.

By characterizing the data differently through data-processing techniques, we may be able to emphasize key features in the data. Then by making use of Shallow/Deep Convolutional Neural Networks and Long/Short Term Memory Architectures, we can build effective models. Then by combining data-processing and effective model structures, even more model performance can potentially be achieved.

2. Data Pre-Processing

2.1. Power Spectral Density (PSD)

PSD is effective for time-frequency analysis of non-stationary and non-linear signals. A number of methods based on common space pattern (CSP), time frequency analysis, or both have been studied for feature extraction of EEG signals. There have been numerous studies incorporating PSD as form of pre-processing EEG signals for analysis of Alzheimer's EEG [5], depth of anaesthesia monitoring [1], and other neurological phenomena that humans encounter [3]. Also, a recent study from 2018 shows a method that relates simplistic k-means clustering algorithm with weighted PSD method to achieve high classification accuracy on BCI competition data [2]. Our approach was to incorporate Welch's method in estimating the power of a signal at different frequencies for the EEG signals for pre-processing and apply Deep learning methods. Welch's method was used to convert our EEG signal data into power spectrum. This is an improved method on the standard periodogram spectrum estimating method in that it reduces noise in the estimated power spectra in exchange for reducing the frequency resolution.

2.2. Principal Component Analysis (PCA)

According to [4] the EEG signal has a comparatively low signal-to-noise ratio. The SVD (Singular Value Decomposition) was another data processing technique used, which de-noised the dataset. In each trial,the temporal mean of each electrode's data was subtracted from the raw data. The first five dominant singular values of the covariance matrix explained at least 95% of the variance for all trials. The deviations from mean could therefore be well approximated by a rank 5 matrix. Thus the raw data (which also contains the mean) could be approximated by a rank 6 matrix. It is necessary to note that models trained on this de-noised dataset performed equally or slightly worse than the case when the raw dataset was used for training.

2.3. Sequential Sampling

We experimented with a sequential sampling technique so as to increase the number of data samples available to the model and to gauge whether that has an effect in increasing model performance. Given the sample length ($\mathrm{sample\_len}$) and stride (stride), the total number of samples were equal to $\mathrm{num\_samples} * ((\mathrm{num\_time\_steps} - \mathrm{sample\_len}/\mathrm{stride}) + 1)$
where num_samples was 2115 for training and validation data and 443 for test data. The models were accordingly

modified altered depending on the input dimensions to accomodate the varying sample lengths.

3. Architectures

TensorFlow and PyTorch were both used to create neural network models. We have included our detailed performance data and architecture on Appendices.

3.1. Shallow Convolutional Neural Network (CNN) Architectures

The Shallow CNN architectures are based off of [4] and helped by Guangyuan's sample PyTorch code. The Shallow CNN architecture consisted of a convolutional Layer, FC Layer, Squaring activation, Average Pool, Log activation, another FC Layer, and then a softmax.

3.1.1 Shallow CNN with sequential sampling

An important aspect of the Shallow CNN to deal with EEG data is the split convolution nature of the first layer where a convolution first happens temporally across the 1000 time steps and then spatially across the 22 input electrode channels. The trend of Shallow CNN architectures vs increasing sample lengths from 100 to 800 time steps can be seen in Figure 3. It is seen that the test accuracy increases with increase in sample lengths with marginal differences in accuracies beyond sample lengths of 900 which then become fairly similar to the original sample length of 1000 time steps. This behaviour may explain that a large majority of the original sequence of 1000 time steps needs to be present for the model to distinguish between the different output activities. No indication of overfitting was seen in any of the models with only slight differences in training and validation accuracies with the models which avoided the need of adding additional regularization techniques.
The next step was to try increasing the learning capacity of the shallow CNN by adding more convolutional layers. The original architecture was modified with an additional convolutional layer similar to the configuration in the first layer. This maintained the idea of temporal convolutions across the time step axis with (1,25) sized filters but applied for a second time. ReLU non-linearities were also added to the activations to increase the non-linearities in an attempt to increase the overall learning capacity of the network. The testing accuracy of this model turned out worse than the original architecture by 8% which indicates the learning capacity and parameters of the network may not be an obstacle to the learning of the network.

3.2. Deep CNN architectures

The Deep CNN architectures consists of four repeated layers containing: a Convolutional Layer, a Pooling Layer, a Batchnorm Layer, and a Dropout Layer. Then a FC Layer is used to reduced the outputs to 4 and is soft-maxed. The convolutional layer is chosen because it can learn well, while having much less parameters. Then a pooling layer is used because it does require any additional parameters, yet still allows us to lower the dimensions of our CNN layer outputs. Thus, the pooling layer allows us to focus on important features of the CNN feature maps. Then a batchnorm and dropout layer are used. Finally the FC layer is used to reduce the output size to 4 (corresponding to our number of classes) and then it is softmaxed. The architecture can be seen when viewing Figure 1 and following the yellow path.

3.3. Deep CNN & Long Short Term Memory (LSTM) Architectures

3.3.1 LSTM After Deep CNN

The Deep CNN and LSTM architectures start with the aforementioned Deep CNN. Then an LSTM and softmax layer were added to the end. The LSTM layer is a post-CNN technique, which works by outputting and receiving "hidden" states. The hidden states allow the LSTM to receive all past hidden states and train based off that information. In theory the LSTM technique should allow our model to perform better, because the EEG data could possibly have significant features that are typically near other distinctive features. Thus, the hidden states could capture the previously displayed features and inform future model training. The architecture is shown in Figure 1.

3.3.2 LSTM Before Deep CNN

Architectures where the LSTM is used as the first layer were also explored. Passing a sequence of inputs to an LSTM layer can lead to creation of a feature that encodes information from all time steps in the sequence. Feeding the input data into LSTM layer may be a promising architecture. The following observations were noted:

(1) Layers where the features from LSTM go into very shallow neural networks performed poorly. Skipped connections were tried, where the inputs to the network are also passed to some future layers. In another attempt, the 1000 time steps were separated into sections of 250 timesteps and independently passed through LSTMs, whose outputs were then merged to create features for future layers (the initial part of which is shown in Appendix A, Figure 6). Convolutional LSTMs were tried which perform convolutions on both spatial and temporal dimensions.

All these attempts produced test accuracy in the range of 25%, which is as good as random classification. The reason for the general poor performance might be attributed to the inability of the LSTM layer to learn information about the state of the system from such a small dataset (Each sequence has 1000 timesteps, but there were only 2115 training examples to deal with). For all the architectures in this section, sampling multiple time windows from a single trial (a window of 800 timesteps with stide 50) did not improve test performance. The LSTMs seem to require more information on the time evolution of a system, that cannot be learnt by sampling multiple windows (which are likely to have highly correlated data). The ConvLSTM layer was comparatively better with a test accuracy of about 40%, but comes at the cost of extremely slow training speed. It has to be noted that treating the input at each instant as a map of electrodes leads to poor performance throughout the current section (3.3.2). This seems to support the assumption (as stated in [4]) that the spatial positioning of electrodes is not relevant and that electrodes capture global information from the brain.

(2) Further architectures were tried that have an LSTM layer initially, followed by the best case deep CNN layer network (refer to Appendix A, Figure 5). A test accuracy of 53% constant regularization when applied to all layers in the LSTM - CNN network. The maximum test accuracy of 66% (for architectures discussed in this section 3.3.2) was obtained when regularization strength was varied along the depth of the network. The best model was obtained when the regularization coefficient was varied from 0.01 at the first hidden layer to 0.001 at the last convolutional layer (refer to Architecture). This supports the intuition that at the initial layers, general features of the input must be learnt that would then be processed by the deeper layers specifically into features suitable for the classification task. This positive benefit of lowering the regularization strength as we go deeper was verified for multiple LSTM sizes (refer to Performance).

4. Results and Discussion

4.1. Training Motor Imagery Classification

Initial architectures begin with a simple Shallow CNN. By adding dropout, the difference between Test Accuracy (TA) and Validation Accuracy (VA) is greatly reduced, but the overall VA/TA is shown to also decrease. By adding Batchnorm, 2-3% accuracy increases is found. By adding Batchnorm and Dropout, the best VA was found and this architecture was used to implement the Deep CNN architecture.

Initially the Deep CNN architecture did not perform well. A common way to gain more performance is to add a pooling layer. Maxpools performed well, but Avgpools seemed to perform the best. This may have worked because the pooling layers allowed the architecture to focus on more prominent parts of the data. Since this architecture has much more parameters to train, higher epoch 's seemed to provide higher performance from our model (250 epochs seemed to be the optimal number).

Next, LSTM was implemented to attempt to gain more performance. When adding an LSTM, higher epochs allowed the model to gain much higher validation accuracy; however, eventually the test accuracy no longer received any benefit.In fact, LSTM architectures performed very similarly to Non-LSTM architectures (71% vs 73% VA) This may be due to the fact that RNNs require much more data to be effective, and therefore did not help.

By the end of the experimentation, the Deep CNN architectures were found to be the most effective (with a TA of 73.13%). The LSTM models may have been too complex, or not had enough data to perform well. It should be noted that based off the training/validation split and the random variances during training, some models can benefit from randomness and receive above average accuracies.

4.2. Accuracy of models as a function of time

All data consists of 1000 time units of EEG data. The model was trained by data from data ranging from 100 to 1000 time units, in steps of 100. This was done to see how well the model could perform based on the EEG data time length. Figures 2 and 3 show the performances of two models as a function of time. Note that for some models, the pooling layer had to be omitted due to the short time lengths of the data. The figure (numbers) reveal that most learning occurs around the 400th EEG time unit.

4.3. Subject-wise models: One vs One and One vs All

In this regard, in a one-vs-one format, 9 subject-wise models i.e a model trained and tested on each subject were also implemented with the architectures mentioned in Table 6. Considering the difference in overall test accuracies with the architectures, there are patterns seen with similar models doing better/worse where it can be seen that models for subjects 2, 6, 7 and 8 have higher subject-specific test accuracies. Additionally, in a one-vs-all format, we trained a model on each subject and tested this trained model on the entire test data of all subjects to evaluate the generalizability of the trained model. As seen in Table 7, the test accuracies are fairly low in the range of 35%-45% which reiterate the importance of the subject-specific nature of the trained models.

4.4. Analysis of Confusion Matrix

The normalized confusion matrix in Figure 2 between the predicted and true labels for a shallow CNN-based model has a clear observation that the majority of the predictions come under the True Positive (TP) category along the diagonal. It may also be an interesting observation that the network tends to predict the ”Feet” action more often leading to a relatively high number of misclassifications in that column.

References

[1] O. Dressler, G. Schneider, G. Stockmanns, and E. F. Kochs. Awareness and the EEG power spectrum: analysis of frequencies. *BJA: British Journal of Anaesthesia*, 93(6):806–809, 09 2004.

[2] C. Kim, J. Sun, D. Liu, Q. Wang, and S. Paek. An effective feature extraction method by power spectral density of eeg signal for 2-class motor imagery-based bci. *Medical Biological Engineering Computing*, 56:1–14, 03 2018.

[3] Z. Y. Ong, A. Saidatul, and Z. Ibrahim. Power spectral density analysis for human eeg-based biometric identification. In *2018 International Conference on Computational Approach in Smart Systems Design and Applications (ICASSDA)*, pages 1–6, 2018.

[4] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball. Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human eeg. *Human Brain Mapping*, 38:5391–5420, 11 2017.

[5] R. Wang, J. Wang, H. Yu, X. Wei, C. Yang, and B. deng. Power spectral density and coherence analysis of alzheimer's eeg. *Cognitive neurodynamics*, 9:291–304, 06 2015.

Appendix A Performance

Model	Validation Accuracy	Test Accuracy
Shallow CNN	60.99%	N/A
Shallow CNN-DP	58.62%	N/A
Shallow CNN-BN	63.35%	65.01%
Shallow CNN-BN-DP	64.06%	N/A
Deep CNN-BN-DP	59.57%	N/A
Deep CNN-BN-DP-MaxPool	70.21%	69.07%
Deep CNN-BN-DP-AvgPool (epoch #: 100;250)	73.99%;76.595%	70.20%;73.13%
Deep CNN-BN-DP-AvgPool-LSTM (epoch #: 250;350;450)	72.34%;74.23%;77.30%	70.65%;71.55%;71.10%
Deep CNN-BN-DP-AvgPool-2xLSTM	68.55%	63.88%
PSD-Shallow CNN-BN-DP	33.96%	35.84%
PSD-Deep CNN-BN-DP-AvgPool	42.92%	42.21%
PSD-Deep CNN-BN-DP-AvgPool-LSTM	43.39%	37.02%
Shallow CNN (300 epochs)	78.53%	72.46%
Shallow CNN with Second Conv + Relu Layer	77.05%	58.51%
LSTM-CNN_adaptive reg -1,2,3 (LSTM units 22,6,15) (L1 and L2 reg goes from 0.1 to 0.05 to 0.01 to 0 with depth)	62.64%, 67.37%, 67.61%	66.13%, 65.46%, 62.97%
LSTM-CNN-2_constant reg (reg = 0.01)	48.93%	53.27%

Table 1: Motor Classification Accuracies

Subject #	0	1	2	3	4	5	6	7	8
Shallow CNN w/ Sequential Sampling	64.8%	33.6%	82.8%	45.6%	47.7%	45.7%	68.4%	78.0%	80.0%
Shallow CNN + 1 Conv layer	67.2%	53.6%	83.6%	50.4%	62.1%	54.7%	76.8%	79.6%	80.4%
LSTM-CNN_adaptive reg -1	62.00%	57.99%	69.99%	47.99%	72.34%	57.14%	69.99%	56.00%	70.21%

Table 2: Training for One Subject; Testing on One Subject

Subject #	0	1	2	3	4	5	6	7	8
LSTM-CNN_adaptive reg -1	38.60%	43.34%	41.98%	43.79%	46.50%	47.30%	46.72%	40.85%	39.95%
Shallow CNN-BN	41.3%	39.27%	36.79%	41.31%	37.47%	41.76%	41.30%	42.21%	39.27%

Table 3: Training for One Subject; Testing on All Data

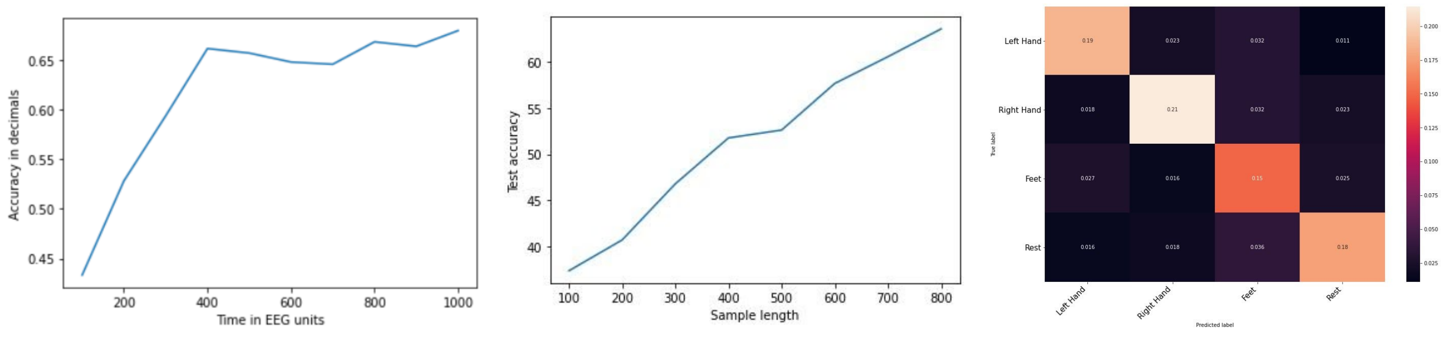


Figure 2(a/b): Accuracy as a Function of Time: Shallow CNN-BN(a;left),Sequential Sampling(b;middle);Confusion Matrix between True and Predicted Labels(c;right)

Appendix B Architecture

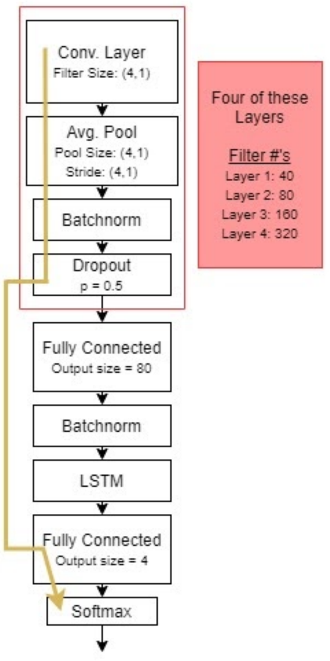
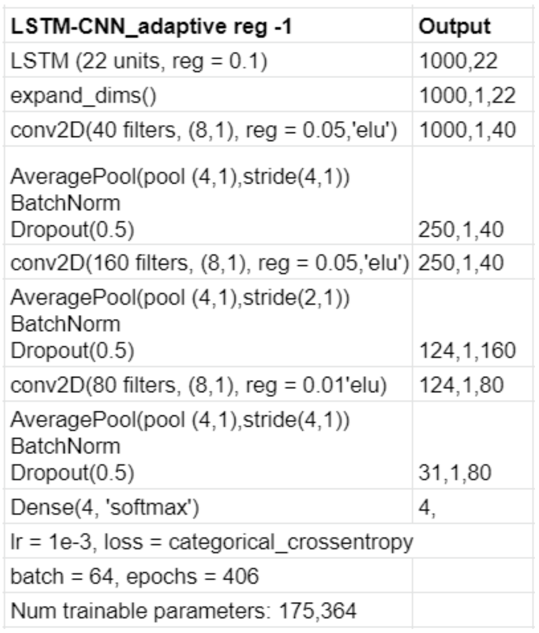


Figure 4: Deep CNN+LSTM Architecture



LSTM-CNN_adaptive reg -1	Output
LSTM (22 units, reg = 0.1)	1000,22
expand_dims()	1000,1,22
conv2D(40 filters, (8,1), reg = 0.05,'elu')	1000,1,40
AveragePool(pool (4,1),stride(4,1)) BatchNorm Dropout(0.5)	250,1,40
conv2D(160 filters, (8,1), reg = 0.05,'elu')	250,1,40
AveragePool(pool (4,1),stride(2,1)) BatchNorm Dropout(0.5)	124,1,160
conv2D(80 filters, (8,1), reg = 0.01'elu)	124,1,80
AveragePool(pool (4,1),stride(4,1)) BatchNorm Dropout(0.5)	31,1,80
Dense(4, 'softmax')	4,
lr = 1e-3, loss = categorical_crossentropy	
batch = 64, epochs = 406	
Num trainable parameters: 175,364	

Figure 5: LSTM+Deep CNN Architecture

General Details

1. All Conv Layers and the first FC Layer: ELU
2. Last FC layer: Softmax
3. Optimizer: Adam
4. Loss: Categorical Cross Entropy
5. Train-Test split: 80/20 split with no replacement
6. Training batch size: 32/64

Shallow CNN

Follows the yellow path;red boxed area done once.

Deep CNN

Follows the yellow path, where the red boxed area is done 4x.

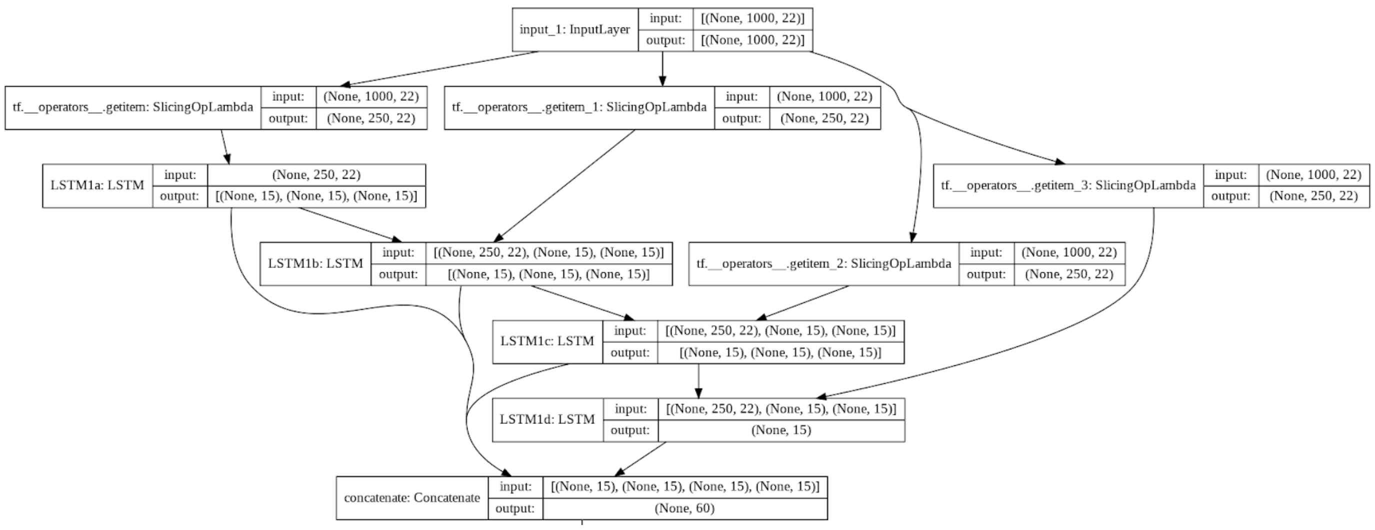
Deep CNN-LSTM

Follows the diagram completely. LSTM output: 80

Deep CNN-2xLSTM

Follows the diagram, but repeats the LSTM layer twice. LSTM output size: 40.

Figure 6: Sample custom architecture


input_1: InputLayer | input: [(None, 1000, 22)] | output: [(None, 1000, 22)]
tf.__operators__.getitem: SlicingOpLambda | input: (None, 1000, 22) | output: (None, 250, 22)
tf.__operators__.getitem_1: SlicingOpLambda | input: (None, 1000, 22) | output: (None, 250, 22)
tf.__operators__.getitem_3: SlicingOpLambda | input: (None, 1000, 22) | output: (None, 250, 22)
LSTM1a: LSTM | input: (None, 250, 22) | output: [(None, 15), (None, 15), (None, 15)]
LSTM1b: LSTM | input: [(None, 250, 22), (None, 15), (None, 15)] | output: [(None, 15), (None, 15), (None, 15)]
tf.__operators__.getitem_2: SlicingOpLambda | input: (None, 1000, 22) | output: (None, 250, 22)
LSTM1c: LSTM | input: [(None, 250, 22), (None, 15), (None, 15)] | output: [(None, 15), (None, 15), (None, 15)]
LSTM1d: LSTM | input: [(None, 250, 22), (None, 15), (None, 15)] | output: (None, 15)
concatenate: Concatenate | input: [(None, 15), (None, 15), (None, 15), (None, 15)] | output: (None, 60)