# Learning EEG Representations For Multi-Tasking: From Motor Imagery Classification to Subject Identification

Mai Tran
EPSS Department
UCLA
tranmai@ucla.edu

Kartik Ahuja
ECE Department
UCLA
ahujak@ucla.edu

## Abstract

*In this work, our goal is to learn the representations for the electroencephalogram (EEG) signals, which can be used for multiple prediction tasks. Our primary task is motor imagery classification (MIC). We build models that are simpler than the models proposed in the literature but yet are able to achieve high accuracy for MIC. We provide important insights for building architectures for EEG related tasks. These insights are useful in other settings as we show in the next task where we use these insights to build models for predicting the identity of the subject from the EEG data. We use the model trained for MIC to generate the representations for the EEG signals. We use variational autoencoder (VAE) to learn the distribution of the EEG representations and also provide insights into how the learned representations are very informative for the motor imagery task. We use these representations to carry out transfer learning for predicting subject identities to show the general utility of the representations learned.*

## 1. Introduction

We are given a dataset consisting of EEG recordings from nine subjects. Our primary goal is to train neural networks to predict the task that the subject is imagining. Next, we describe the architectures we choose and why.

### 1.1. Convolutional Neural Networks (CNN)

In the literature, different deep learning architectures [3] [1] have been proposed for motor imagery classification (MIC). In [3] the authors proposed a CNN based architecture. The key highlight of their architecture is that they split the convolution operation in the early layers into a spatial and a temporal convolution. We developed an alternate architecture which does not require any of the special operations built by [3] and can still learn EEG representations as well (we obtain similar or better accuracy for MIC). For the discussion that follows, we assume the number of filters is fixed at entrance layers for ease of exposition. We reduced the temporal and spatial convolution operation introduced by the authors into one simple operation. We created an input tensor with the following dimensions: Length of the time signal $\times$ 1 $\times$ Number of electrodes (width $\times$ height $\times$ depth). The input tensor in [3], in contrast, has dimensions: Length of the time signal $\times$ Number of electrodes $\times$ 1. We treat the number of electrodes as the depth and the authors treat the number of electrodes as the height. We do one convolution instead of the two proposed by the author to produce a tensor with dimension: Length of the signal $\times$ 1 $\times$ Number of filters, which is the same shape as the output tensor in [3] post the two convolutions. In the later section, we explain why this approach works well.

With the entrance layer designed as described above, we explored both shallow and deep CNNs. In the design of deep CNNs, we followed the general principle that the later layers had more number of filters. We used Max Pooling as the downsampling procedure in each layer post the convolution to achieve translational invariance. We investigated the impact of adding dropout and batch normalization on these architectures as well.

In our constructions so far, we used the entire time signal as input to the CNN. But how much of the time signal is needed to be able to do well with MIC? To investigate this, we compare the accuracy we can achieve when we only train the network with the segments of the EEG signal.

### 1.2. Recurrent Neural Network Architectures

EEG is a time signal. Therefore, a natural choice for the architecture is to use Recurrent Neural Networks (RNNs). We used Long Short Term Memory (LSTM) based RNNs. We input the EEG recording from all the electrodes sequentially into the LSTM. We found that the network did not learn and the validation accuracy was stuck at twenty five percent. The long time signal creates long gradient paths in the RNN, which leads to vanishing or exploding gradients at many of the layers, which causes the algorithm not to learn. We propose that instead of using the entire sequence, we used subsequences which were continguously sampled for each person. Each of the continguous subsequences has the same label (as the subject is imagining the same task). Since we used shorter subsequences to train the model it avoids the vanishing and exploding gradient problem. However, using subsequences reduced information from the past, which can limit the ability of the model to learn. To alleviate this problem, we proposed our next architecture which combines CNN with an RNN.

### 1.3. CNN and RNN cascade

We want to input to RNN a lower dimensional summary of the entire input signal. To achieve this, we used CNNs to extract high dimensional features and then projected them into a lower dimensional input for RNN.

**Activation, Filters, MaxPool parameters, Dropout and Learning Rate** For each of the architectures above, we tried activation functions such as ELU and RELU and we empirically found ELU activation to be the best. Hence, we report the results with ELU. We tried three filter sizes (5,1), (10,1) and (15,1). We found the filter (10,1) to be the best, which is what we use in all the models. We tried two pool size (3,1) and (5,1) and two strides (1,1) and (3,1), where we found pool size (3,1) and stride (3,1) to be the best. We optimized the other hyperparameters such as the learning rate and the dropout rate using grid search on the validation data for every architecture separately.

### 1.4. Dissecting the network using VAE

Among all the architectures our deep CNN architecture was the most successful. We analyze the deep CNN to gain some insight into the black-box. We compared the lower dimensional representation of the raw signal versus the lower dimensional representation of the features estimated by the deep CNN. We regard the output of the CNN layers, which is fed into the last fully connected layer, as the representation of the EEG. We trained a Variational Auto Encoder (VAE) to learn a distribution of the latent variables conditional on the input. We provide the representation learned by the CNN as input to the VAE. Next, we trained a VAE with raw data as input. We compared the outputs of the VAEs in both the cases. We are able to show that the learned features are insightful.

### 1.5. Subject Identification Problem

We learned how to choose different architectures for EEG related tasks in the previous sections. Next, we want to investigate if these insights can help other tasks that use EEG data. In this part, we look at the task of subject identification (SI). In [2], the authors proposed an approach to identify subjects based on their EEG recordings.

In our dataset, we have nine subjects. Our task is to use the EEG recording to identify the subject. We used the same architecture of deep CNN that was most successful for MIC.

### 1.6. Transfer Learning: MIC to SI

In the previous sections, we dealt with two very distinct tasks, where our first task was MIC and the second task was SI. We explore if we can use transfer learning techniques for subject identification when we have already learned how to do MIC. We used the deep CNN architecture that we trained for MIC to extract representations of EEG. We trained a softmax classifier with these representations as input to learn for SI.

## 2. Results

**Data Preprocessing** We are given a dataset with 9 subjects. Each subject has 288 trials. Each trial for the subject is 4 seconds long. In each trial the data is recorded from 25 electrodes. The signal at each electrode consists of 1000 data points. In this project, *we use all the 25 electrodes as we had already optimized our architectures before the announcement*. For MIC, each trial carries one label, which corresponds to action that subject imagines. For SI, each trial carries one label, which corresponds to the identity of the subject. The dataset is divided into training, validation and testing as follows. A test data with 443 trials was given to us separately. We randomly split the remaining data consisting of 2115 trials into 1615 trials for training and 500 trials for validation. All the experiments were conducted in Google Colab.

### 2.1. Motor Imagery Classification

In Section 4, we provide details of the architectures. We trained these architectures with the data for all the subjects. In Table 1, we report the accuracy over all the subjects separately and the overall accuracy, i.e., the accuracy computed on the entire test data.

**Results for CNN Architectures** In SCNN (Shallow CNN architecture) architecture (See Section 4.1), we used one convolutional layer followed by MaxPooling. The architecture's performance over all the subjects averaged is 0.48. We added dropout to this architecture in SCNN-dp. The performance marginally increased to 0.51. Next, we also added batch normalization to this architecture in SCNN-dp-bn and the performance shot up to 0.69. Batch-normalization helped alleviate negative effects of covariate shift caused by parallel update at all layers.

We next built a deep convolutional neural network to further improve the performance (DCNN-dp-bn in Section 4). In this architecture, we had four convolutional layers, where the number of filters increased in every layer, followed by one fully connected layer. We were able to improve the performance to up to **0.75 accuracy** with a deeper network.

**Results for RNN Architectures** We first tried training a standard LSTM with the entire dataset. The network was not able to learn and had accuracy of 0.25. This was possibly due to the vanishing and exploding gradients. Next, we tried training a LSTM with cropped data. We created subsequences from the original trial for each subject. This helped the network learn and get an accuracy up to 0.40 from 0.25. The performance was still limited since the network could not build memory beyond the length of the subsequence, which was very short. Thus, we developed the architecture in the next section.

**CNN-RNN cascaded architectures** We used a deep CNN based architecture followed by an LSTM. We were careful in restricting the input size of the LSTM. Thus, we used a fully connected layer to reduce the dimension of the convolutional layers' output. This architecture achieved accuracy of **0.72**. In Figure 5, we show the training and validation loss for training CNN-LSTM (Due to space limitations we show this plot for one architecture only).

**VAE driven interpretations** For this analysis, we used our best performing architecture, the DCNN-dp-bn. Our goal is to analyze the representation computed by the model and compare with raw data. We used the vector that is output from the last convolutional layer (Layer 4) and fed into the fully connected layer as the representation of EEG. We trained VAEs to estimate lower dimensional features of both the raw data and the representation learned by the network. In Figure 1 and 2, we show the features estimated by VAE for the representation computed by DCNN-dp-bn and for the raw data. We show the different label classes with different colors. In Figure 2, we can see that the four classes are overlapping significantly for the raw data but are separated into four clusters for the learned representation in Figure 1. This demonstrates that the two features estimated by the VAE well summarize a lot of information about what the individual thinks. Hence, analyzing these features can be potentially useful for understanding MIC. Also, notice that the green class (label 2, cue feet) seems to be at the center of the cluster thus indicating it was hard to separate that label from the rest. This leads to the question if the EEG signal corresponding to task label 2 special. Further, the accuracy corresponding to label 2 is 0.70 (lowest among all the labels), which strengthens the claim. Also, note that the red (left) and blue label (right) are also well separated, thus indicating that the signal for left and right are distinct from one another.

**Accuracy as a function of length of the signal** To understand how long do we need to gather the data from a subject to be able to well with the MIC task, we plotted the accuracy as a function of length of the signal segment used in Figure 3. The accuracy increases as a function of the length of the time segment. After about a length of 1.2 seconds the accuracy starts to saturate.

## 2.2. Subject Identification (SI)

In SI, we have the same EEG data but now we use the subject identity as the labels. We found that the DCNN-dp-bp was the best architecture for MIC. We directly use that architecture and train it to do subject identification. We were able to achieve an accuracy of 0.82.

**Transfer Learning** We extracted the representation computed from DCNN-dp-bn, which was also used as input to the VAE. We use this representation as input to a softmax classifier, which we train to classify the subjects. We were able to achieve an accuracy of 0.80. In many circumstances, transfer learning is useful when there is limited data available for the transfer task. In Figure 4, we show that if we only had a fraction of the data, then also transfer learning is able to achieve accuracy for SI comparable or sometimes better than a model (DCNN-dp-bn) trained on the limited raw data directly.

## 2.3. One subject based training

Instead of training on the whole data, we trained the DCNN-dp-bn on each subject and tested its performance overall (and also on each subject separately). We get a range of accuracy values from 30-50 percent as shown in Table 2.

## 3. Discussion

For shallow CNN architectures, we observed that there is a significant improvement in using batch-norm but not so much using dropout. Given the shallow nature of the architecture, overfitting was less of a concern, which is why dropout did not help much. Batchnorm is still useful as the problem of covariate shift and poor initialization plagues shallow architecture as well. We observe that the deep architectures help significantly. A single convolutional layer simply adds the weighted sum of contributions of all the electrodes. Therefore, a shallow CNN does not create features that are based on interactions between electrodes. On the other hand, a deeper network makes more abstract/non-linear feature constructions that are based on interactions between the electrodes possible. Hence, it is able to uncover the patterns that [3] hoped to uncover when using their special convolutions. RNN architecture with cropping alone loses a lot of information. Hence, using a CNN to compress information and present it to LSTM performs better in comparison to LSTM with cropped subsequences.

VAE based projections show that the network is able to learn features that are very indicative of the different motor imagery tasks. Also, the plot indicates that the model has difficulty learning label 2 (cue right), which may have interesting ramifications.

Once we learned what is a good architecture for motor imagery tasks, it was easy to build an architecture that does well for a different task, which is the SI task.

Transfer learning is able to do as well as training a full architecture from scratch with raw data. This shows that the representation learned by the CNN are not only informative about the motor imagery task as shown above (in VAE part) but also they are general and can be applied to other tasks.

## References

[1] P. Bashivan, I. Rish, M. Yeasin, and N. Codella. Learning representations from eeg with deep recurrent-convolutional neural networks. *arXiv preprint arXiv:1511.06448*, 2015.

[2] Z. Mao, W. X. Yao, and Y. Huang. Eeg-based biometric identification with deep learning. In *2017 8th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 609–612. IEEE, 2017.

[3] R. Schirrmeister, J. Springenberg, L. 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. arxiv, 2017. *arXiv preprint arXiv:1703.05051*.

## 4. Architectures

Conv2D: 2 Dimensional Convolution, FCNet: Fully Connected Network, BNorm: Batch Normalization, ELU: Exponential Linear Unit. Across all the architectures the padding was kept as 'same' to preserve the dimension of the tensor.

### 4.1. SCNN: Shallow CNN

**Layer 1:** 25 Conv2D filters kernel : (10,1), stride: (1,1) $\rightarrow$ ELU $\rightarrow$ MaxPool: size: (3,1), stride: (3,1) $\rightarrow$ Flatten
**Layer 2:** FCNet with 4 outputs.

### 4.2. SCNN-dp

**Layer 1:** 25 Conv2D filters kernel : (10,1), stride: (1,1) $\rightarrow$ ELU $\rightarrow$ MaxPool: size: (3,1), stride: (3,1) $\rightarrow$ Dropout $\rightarrow$ Flatten

**Layer 2:** FCNet with 4 outputs.

### 4.3. SCNN-dp-bn

**Layer 1:** 25 Conv2D filters kernel : (10,1), stride: (1,1) $\rightarrow$ ELU $\rightarrow$ MaxPool: size: (3,1), stride: (3,1) $\rightarrow$ BNorm $\rightarrow$ Dropout $\rightarrow$ Flatten

**Layer 2:** FCNet with 4 outputs.

### 4.4. DCNN-dp-bn

**Layer 1:** 25 Conv2D filters kernel : (10,1), stride: (1,1) $\rightarrow$ ELU $\rightarrow$ MaxPool: size: (3,1), stride: (3,1) $\rightarrow$ BNorm $\rightarrow$ Dropout

**Layer 2:** 50 Conv2D filters kernel : (10,1), stride: (1,1) $\rightarrow$ ELU $\rightarrow$ MaxPool: size: (3,1), stride: (3,1) $\rightarrow$ BNorm $\rightarrow$ Dropout

**Layer 3:** 100 Conv2D filters kernel : (10,1), stride: (1,1) $\rightarrow$ ELU $\rightarrow$ MaxPool: size: (3,1), stride: (3,1) $\rightarrow$ BNorm $\rightarrow$ Dropout

**Layer 4:** 200 Conv2D filters kernel : (10,1), stride: (1,1) $\rightarrow$ ELU $\rightarrow$ MaxPool: size: (3,1), stride: (3,1) $\rightarrow$ BNorm $\rightarrow$ Dropout $\rightarrow$ Flatten

**Layer 5:** FCNet with 4 outputs.

### 4.5. LSTM-crop

**Layer 1:** LSTM$\rightarrow$ tanh $\rightarrow$ Dropout

**Layer 2:** FCNet with 4 outputs.

### 4.6. CNN-LSTM

**Layer 1:** 25 Conv2D filters kernel : (10,1), stride: (1,1) $\rightarrow$ ELU $\rightarrow$ MaxPool: size: (3,1), stride: (3,1) $\rightarrow$ BNorm $\rightarrow$ Dropout

**Layer 2:** 50 Conv2D filters kernel : (10,1), stride: (1,1) $\rightarrow$ ELU $\rightarrow$ MaxPool: size: (3,1), stride: (3,1) $\rightarrow$ BNorm $\rightarrow$ Dropout

**Layer 3:** 50 Conv2D filters kernel : (10,1), stride: (1,1) $\rightarrow$ ELU $\rightarrow$ MaxPool: size: (3,1), stride: (3,1) $\rightarrow$ BNorm $\rightarrow$ Dropout

**Layer 4:** 50 Conv2D filters kernel : (10,1), stride: (1,1) $\rightarrow$ ELU $\rightarrow$ MaxPool: size: (3,1), stride: (3,1) $\rightarrow$ BNorm $\rightarrow$ Dropout $\rightarrow$ Flatten

**Layer 5:** FCNet with 50 outputs $\rightarrow$ ELU

**Layer 6:** LSTM with 20 outputs $\rightarrow$ tanh

**Layer 7:** FCNet with 4 outputs.

### 4.7. VAE

**Layer 1:** FCNet with 500 outputs $\rightarrow$ ELU $\rightarrow$ BNorm

**Layer 2:** FCNet with 100 outputs $\rightarrow$ ELU $\rightarrow$ BNorm

**Layer 3:** FCNet with 2 outputs

**Layer 4:** FCNet with 100 outputs $\rightarrow$ ELU $\rightarrow$ BNorm

**Layer 5:** FCNet with 500 outputs $\rightarrow$ ELU $\rightarrow$ BNorm

**Layer 6:** FCNet with output of same shape as input to Layer 1.

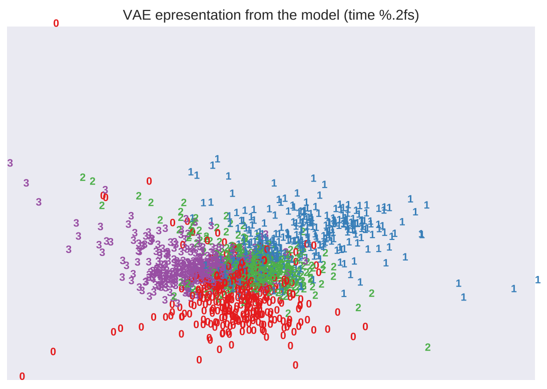

Figure 1. 2-dimensional features estimated by the VAE trained for the hidden layer features learned by CNN. Each class has a different color. Red Class (Label 0: Left), Blue Class (Label 1: Right), Green Class (Label 2: Foot), Purple Class (Label 3: Tongue)

| Subject → | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | Overall | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Model ↓ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| SCNN | 39 | 6 | 50 | 6 | 37 | 5 | 49 | 5 | 66 | 4 | 40 | 6 | 70 | 6 | 49 | 6 | 42 | 8 | 49 | 2 |
| SCNN-dp | 43 | 11 | 51 | 13 | 44 | 8 | 48 | 18 | 65 | 18 | 48 | 9 | 61 | 15 | 54 | 18 | 49 | 13 | 51 | 12 |
| SCNN-dp-bn | 59 | 3 | 68 | 2 | 58 | 5 | 65 | 3 | 88 | 3 | 60 | 6 | 84 | 2 | 71 | 2 | 66 | 3 | 69 | 0.9 |
| DCNN-dp-bn | 72 | 4 | 73 | 4 | 61 | 3 | 66 | 4 | 89 | 1 | 72 | 2 | 79 | 1 | 80 | 2 | 79 | 3 | 75 | 1 |
| LSTM-crop | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | 40 | 1 |
| CNN-LSTM | 63 | 5 | 72 | 4 | 61 | 3 | 64 | 4 | 89 | 1 | 62 | 5 | 80 | 5 | 80 | 3 | 73 | 5 | 72 | 2 |

Table 1. Mean test accuracy $\mu$ and standard deviation of test accuracy $\sigma$ when the models were trained on the entire data. The models with highest accuracy are colored in red. In LSTM-crop we had randomly subsampled the sequences which did not guarantee that all the subjects were present in the subsampled data. Hence, we cannot report the accuracy over the subjects separately.

| Subject → | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | Overall | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Model | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| DCNN-dp-bn | 32 | 5 | 37 | 6 | 33 | 3 | 38 | 5 | 51 | 2 | 37 | 5 | 47 | 2 | 46 | 3 | 37 | 6 | 40 | 6 |

Table 2. Mean test accuracy $\mu$ and standard deviation of test accuracy $\sigma$ when the models were trained on one subject. We averaged the performance of all the models trained on each subject separately.

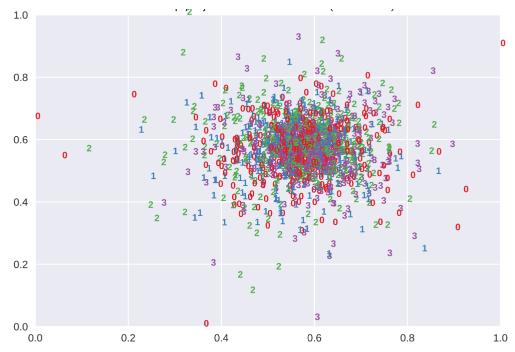

Figure 2. 2-dimensional features estimated from the VAE trained for the raw data. Red Class (Label 0: Left), Blue Class (Label 1: Right), Green Class (Label 2: Foot), Purple Class (Label 3: Tongue)

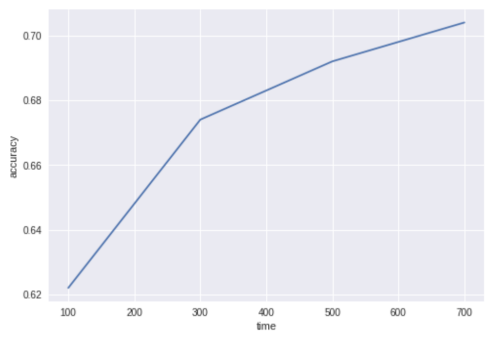

Figure 3. Accuracy vs time. After 300 data points the accuracy starts to saturate.

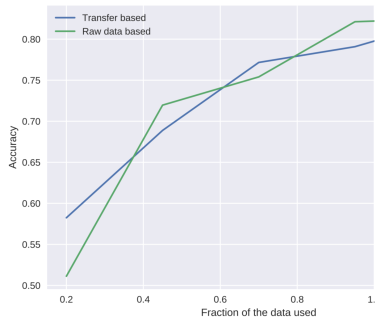

Figure 4. Transfer learning vs Direct learning. Compare accuracy for subject identification task when the amount of datasize available varies.

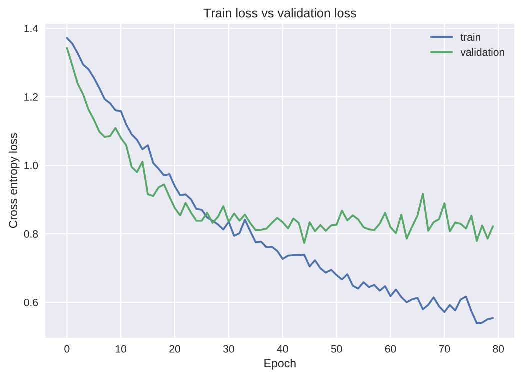

Figure 5. Training vs Validation loss for CNN-LSTM architecture