Deconding EEG Data with Deep Learning

Abraham Canafe
University of California, Los Angeles
Los Angeles, CA 90095
abrahamcanafe@g.ucla.edu

Abstract

In this paper, we approach the EEG decoding problem by utilizing temporal convolutional neural networks (CNNs), CNNs with long short-term memory (LSTMs), and CNNs with multi-head attention. For our given features, we show that temporal convolutions do better than LSTMs and transformer encoders at extracting features related to the classification of the EEG dataset. We also find that adding additional temporal convolutional layers does not guarantee a higher classification accuracy, and we further demonstrate the importance of having sufficient temporal features and a balanced dataset with respect to classification accuracy.

1. Introduction

Inspired by the CNN-based architectures in [7] and [5], we wish to approach the EEG problem through the use of temporal convolutions. Using the EEG dataset from [1], we specifically seek to classify the four output actions with convolutional neural networks (CNNs), CNNs with long short-term memory (LSTM) [3], and CNNs with multi-head attention (MHA) [9]. Since LSTMs and transformers have proven to be useful for the classification of sequential data, we wish to compare the performances of a CNN-LSTM classifier and a CNN-MHA against the performance of a basic CNN for EEG decoding.

We perform several experiments to evaluate the usefulness of CNNs, LSTMs, and transformers in the area of EEG decoding. We first evaluate the classification scores of the **CNN**, **CNN-LSTM**, and **CNN-MHA** architectures shown in Table 3. Next, we explore the effects of adding an additional convolutional layer by evaluating the scores of **DeepCNN**, **DeepCNN-LSTM**, and **DeepCNN-MHA**. Since our architectures primarily rely on temporal convolution, we then explore the effects of training a CNN model with subsets of the original EEG data by varying T_{sub}, which was previously set to 1000 (time steps) for the previous experiments. For this experiment, we use $T_{sub} \in \{400, 600, 800, 1000\}$ timesteps and plot how test accuracy is affected.[1] Finally, we evaluate the multiclassification performances of the **CNN**, **CNN-LSTM**, and **CNN-MHA** architectures after training them with training data only corresponding to class 1.

Section 2.1 summarizes the results of our multiclass classification experiments for the **CNN**, **CNN-LSTM**, and **CNN-MHA** architectures. Section 2.2 summarizes the multiclass classification experiments for the deeper counterparts, **DeepCNN**, **DeepCNN-LSTM**, and **CNN-MHA**. Section 2.3 gives the key findings related to the performance of the CNN model as a function of T_{sub}, and Section 2.4 summarizes the multiclassification results of the **CNN**, **CNN-LSTM**, and **CNN-MHA** architectures trained by data corresponding to only a single label. Section 3 further discusses the results and provides useful insights.

2. Results

2.1. Experiment 1: Optimization of multiclass classification

The training and validation accuracy and loss curves can be found in Figures 5 and 6. Table 3 provides the key metrics for the **CNN**, **CNN-LSTM**, and **CNN-MHA** architectures. From the table, it can be observed that **CNN** yielded a test accuracy and precision, recall, and F1 scores that exceeded those yielded by **CNN-LSTM** and **CNN-MHA**. From the table, it can also be observed that the **CNN-MHA** model outperformed the **CNN-LSTM** model across all four metrics.

2.2. Experiment 2: Convolution depth for multiclass classification

Figures 7 and 8 show plots for the training and validation accuracies and losses of the deeper neural networks. Table 3 also summarizes the performances of **DeepCNN**, **DeepCNN-LSTM**, and **DeepCNN-MHA**. It is particularly interesting to find that **DeepCNN-LSTM** and **DeepCNN-**

[1] Note that we use all data instances in every experiment. We are simply varying the temporal features. See the explanation under Table 3 for more details.

MHA had similar performances with their respective counterparts, even though the **DeepCNN** model yielded accuracy and F1 scores that were worse than those returned by the **CNN** model by a difference of approximately 0.03.

2.3. Experiment 3: Classification accuracy as a function of time

Figure 1 plots the accuracy of the **CNN** model over four different values of T_{sub}. It can be observed that the test accuracy is largest for $T_{sub} \geq 800$.

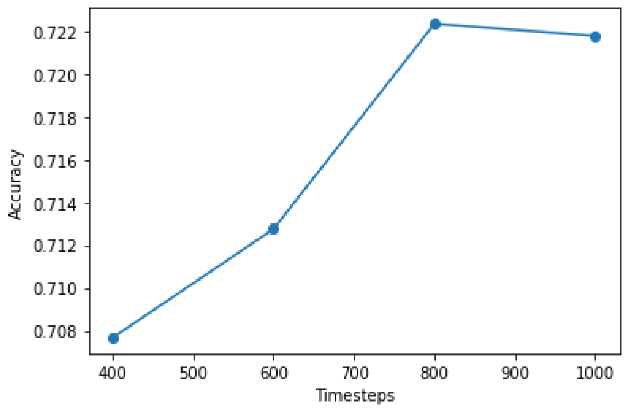


Figure 1. Test accuracy of **CNN** as a function of data timesteps (T_{sub}).

2.4. Experiment 4: Training neural networks with data from a single label

Table 1 summarizes the results of the final experiment, and Figure 2 shows the confusion matrix associated with the **CNN** model. Collectively, **CNN**, **CNN-LSTM**, and **CNN-MHA** yielded accuracy and F1 score metrics.

	CNN	**CNN-LSTM**	**CNN-MHA**
Accuracy	0.25282	0.25282	0.25056
Precision	0.21289	0.18803	0.06292
Recall	0.25182	0.25229	0.25
F1 Score	0.106208	0.10518	0.10054

Table 1. Comparison of model performances across entire EEG dataset

3. Discussion

3.1. Preliminary Discussion

In order to empirically determine the exact architecture design for the models in Table 2, certain intuitive guidelines were followed (particularly for the **CNN** model, whose architecture and hyperparameter set served as the basis for the architectures and hyperparameter sets of the other models).

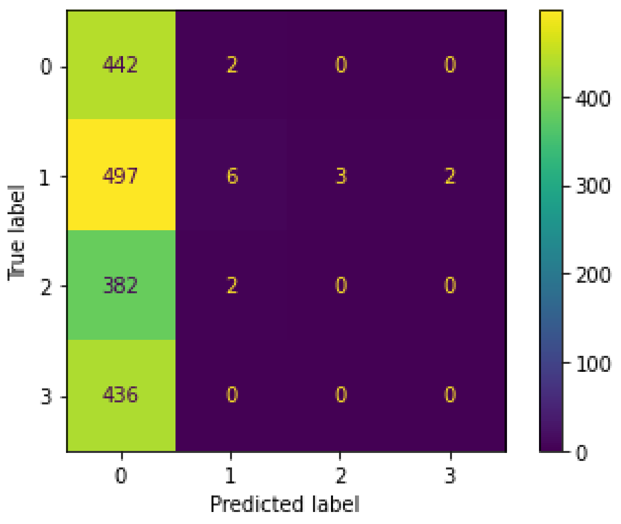


Figure 2. Confusion matrix analysis of multiclass test results for **CNN** (given that the model is trained only by data with labels corresponding to subject 1).

Let us consider the author's rationale behind the design of the **CNN** model. Similar to the authors of [5], we construct a simple CNN consisting of temporal convolutional layers, average pool layers, batchnorm layers, dropout layers, and the ELU activation function. We then begin the hyperparameter tuning process by running the CNN model and ensuring that the training and validation losses are generally decreasing and converging to a sufficiently low cross entropy loss. Since adding deeper convolutional layers increases the receptive field (and since adding more neurons increases the complexity of neural network models, in general), we examined convolutional neural networks that had around 4 convolutional layers. The learning rate was eventually adjusted so that the model initially overfit to the data. After overfitting was observed from the resulting loss and accuracy curves, several methods such as increasing dropout, increasing the weight decay, and reducing the number of model parameters seemed to yield better validation metrics. This iterative process eventually yielded models whose test accuracies were close to (or exceeded) 70% (during the experiments when all data labels and a sufficiently large number of timesteps (T_{sub}) were used.

3.2. Raw Data Visualization

The raw data were explored examining the raw time series associated with the EEG data. The rolling mean, rolling variance, power spectral density, vanilla short-time Fourier transform, and synchrosqueezed short-time Fourier transform (as well as the wavelet transform) were also examined [2, 8]. The rolling variances shown in figures 3 and 4, along with other related metrics, seem to suggest that the

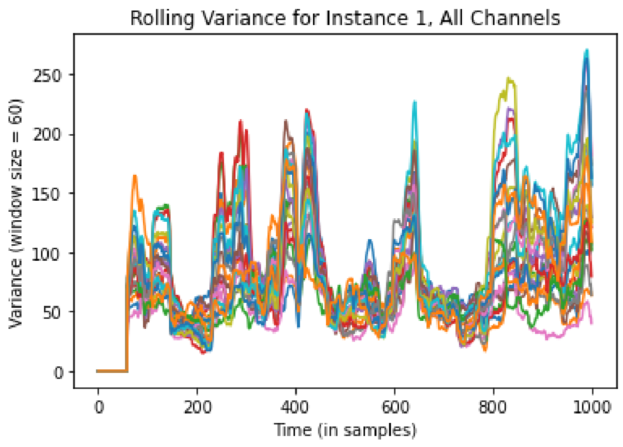


Figure 3. Rolling variances for multiple channels (window size = 60 samples)

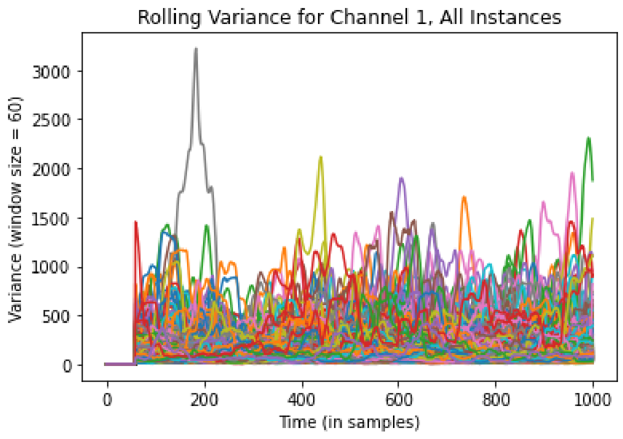


Figure 4. Rolling variances for multiple instances (window size = 60 samples)

channels of a single instance of EEG data are highly correlated, whereas the the collection of data instances associated with a single channel are different enough.

It should be noted that the author made initial attempts to construct 1-dimensional features using the ridges obtained from the time-frequency plot resulting from the synchrosqueezed STFT [4, 8]. The author had reasoned that LSTM and Transformer modules could help solve the EEG decoding problem if the data were transformed into 1-dimensional time series, since both modules tend to perform well when encountering sequential data. However, the author found that the test loss to be unsatisfactory. The author was also unable to fully apply ridge extraction on the spectrograms generated by synchrosqueezed STFT (via the `ssqueezepy` package [6]) because of RAM constraints.

3.3. Experiments 1 and 2 Discussion

While constructing the initial base CNN for experiment 1, it was found that using CNNs with three convolutional layers (using the specific parameters from Table 3) yielded better classification accuracies than CNNs (whose first three convolutional layers were identical) that had deeper convolution layers. Although deeper convolutional neural networks often lend themselves to achieve better classification accuracies, it was determined that adding more than than three conv2d layers to the **CNN** model would lower the classification performance; this observation is evidenced by the worse performance of the deeper **DeepCNN** (Table 2). The intuition that adding too many parameters can cause overfitting also served as the motivating factor behind **CNN-LSTM** and **CNN-MHA** (which only used two convolutional layers) and **DeepCNN-LSTM** and **DeepCNN-MHA** (which only used three convolutional layers).

The results of Table 2 suggest that having an additional temporal convolutional layer is more valuable than adding an LSTM or a transformer encoder. Hence, it may be reasonable to conclude that temporal convolutional layers do better at extracting important features from the engineered features. However, it seems possible transformers and LSTMs could improve classification performance if the EEG data are processed differently, perhaps in a more sequential, interpretable way.

3.4. Experiment 3 Discussion

Even though experiment 3 suggests that utilizing T_{sub}=800 (out of T_0=1000) timesteps is sufficient to obtain a reasonbly classification accuracy score, Figure 1 assumes that the subset of EEG data is extracting using the timestep indices $t = 1, 2, \ldots, T_{sub}$. While this choice of data partitioning seems reasonable (considering points that are close in time may have temporal characteristics that can be extracted by the convolutional layers), it might also be worthwhile to randomly select T_{sub} indices before applying feature engineering and before training the models. Doing this could perhaps, on average, reduce the number of timesteps T_{sub} needed to sufficiently train the neural network models.

3.5. Experiment 4 Discussion

Since the **CNN**, **CNN-LSTM**, and **CNN-MHA** models were only trained using a small, imbalanced subset of the data, it was not surprising to find that yielded accuracies close to 0.25 and poor F1 scores. In fact, it maybe be observed that the models in Table 1 merely yielded accuracy scores that were not much better than a naive classifier that returns an output based on a uniform distribution of labels.

References

[1] Clemens Brunner, Robert Leeb, Gernot Müller-Putz, Alois Schlögl, and Gert Pfurtscheller. Bci competition 2008–graz data set a. *Institute for Knowledge Discovery (Laboratory of Brain-Computer Interfaces), Graz University of Technology*, 16:1–6, 2008. 1

[2] Ingrid Daubechies and Stephane Maes. A nonlinear squeezing of the continuous wavelet transform based on auditory nerve models. In *Wavelets in medicine and biology*, pages 527–546. Routledge, 2017. 2

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 1

[4] Dmytro Iatsenko, Peter V. E. McClintock, and Aneta Stefanovska. On the extraction of instantaneous frequencies from ridges in time-frequency representations of signals. *CoRR*, abs/1310.7276, 2013. 3

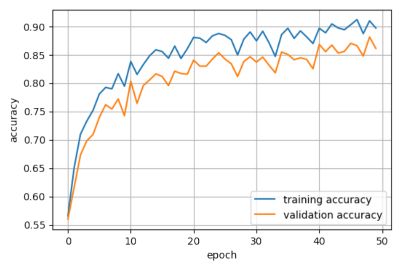
[5] Vernon J Lawhern, Amelia J Solon, Nicholas R Waytowich, Stephen M Gordon, Chou P Hung, and Brent J Lance. Eegnet: a compact convolutional neural network for eeg-based brain–computer interfaces. *Journal of neural engineering*, 15(5):056013, 2018. 1, 2

[6] John Muradeli. ssqueezepy. *GitHub. Note: https://github.com/OverLordGoldDragon/ssqueezepy/*, 2020. 3

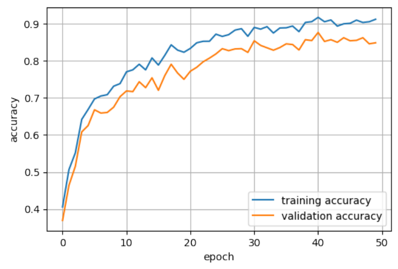
[7] Robin Tibor Schirrmeister, Jost Tobias Springenberg, Lukas Dominique Josef Fiederer, Martin Glasstetter, Katharina Eggensperger, Michael Tangermann, Frank Hutter, Wolfram Burgard, and Tonio Ball. Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human EEG. *CoRR*, abs/1703.05051, 2017. 1

[8] Gaurav Thakur and Hau-Tieng Wu. Synchrosqueezing-based recovery of instantaneous frequency from nonuniform samples. *SIAM Journal on Mathematical Analysis*, 43(5):2078–2095, jan 2011. 2, 3

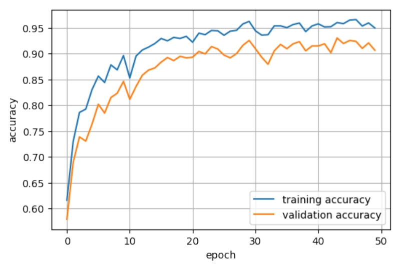
[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. 1



(a) Training and validation accuracies for CNN model.

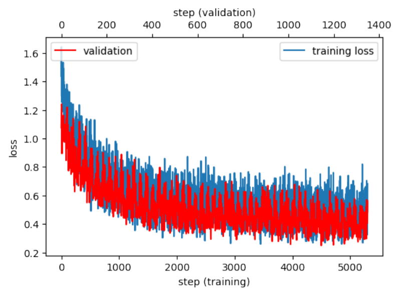


(b) Training and validation accuracies for CNN-LSTM model.

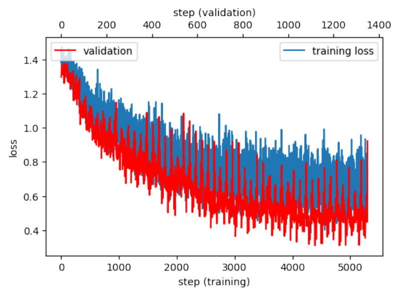


(c) Training and validation accuracies for CNN-MHA model.

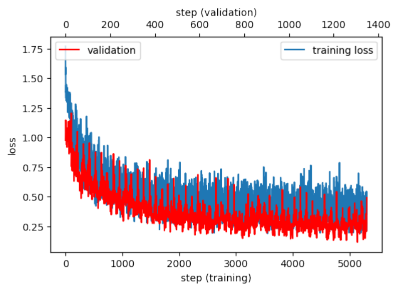
Figure 5. Training and validation accuracies of architectures over 50 epochs (batch size = 64).



(a) Training and validation accuracies for CNN model.

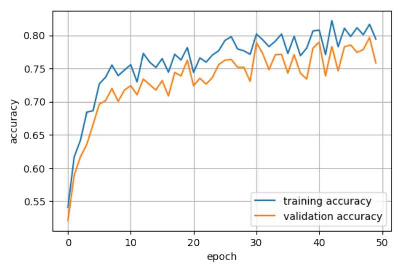


(b) Training and validation accuracies for CNN-LSTM model.

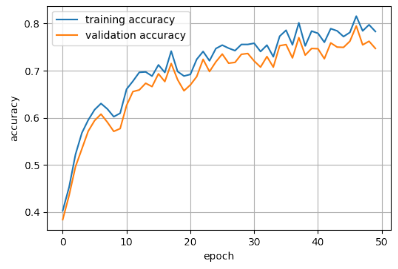


(c) Training and validation accuracies for CNN-MHA model.

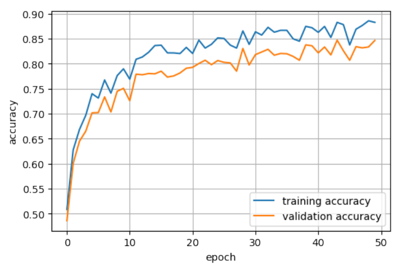
Figure 6. Training and validation losses of architectures over 50 epochs (batch size = 64).



(a) Training and validation accuracies for DeepCNN model.

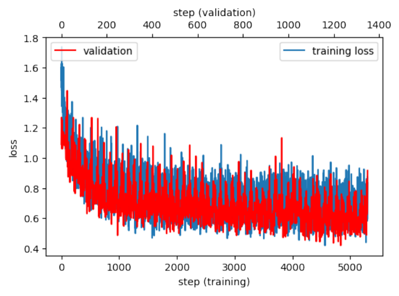


(b) Training and validation accuracies for DeepCNN-LSTM model.

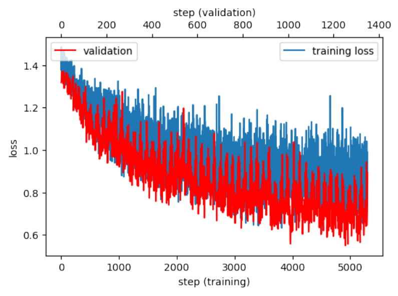


(c) Training and validation accuracies for DeepCNN-MHA model.

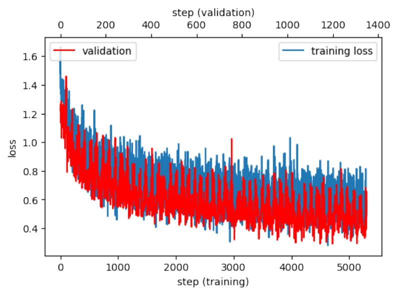
Figure 7. Training and validation accuracies of deep architectures over 50 epochs (batch size = 64).



(a) Training and validation losses for Deep-CNN model.



(b) Training and validation losses for DeepCNN-LSTM model.



(c) Training and validation losses for DeepCNN-MHA model.

Figure 8. Training and validation losses of deep architectures over 50 epochs (batch size = 64).

	Accuracy	**Precision**	**Recall**	**F1 Score**
CNN	0.72178	0.73221	0.71598	0.72045
CNN-LSTM	0.66930	0.67202	0.67292	0.66731
CNN-MHA	0.70598	0.70548	0.70331	0.70418
DeepCNN	0.68905	0.69490	0.68893	0.68792
DeepCNN-LSTM	0.66422	0.68305	0.67139	0.66402
DeepCNN-MHA	0.71049	0.71070	0.70704	0.70829

Table 2. Comparison of model performances across entire EEG dataset

EEG Classifier Architectures					
CNN	**CNN-LSTM**	**CNN-MHA**	**DeepCNN**	**DeepCNN-LSTM**	**DeepCNN-MHA**
Input $(N \times C \times T)$					
conv2d-50	conv2d-50	conv2d-50	conv2d-50	conv2d-50	conv2d-50
ELU					
Batchnorm					
AvgPool2d					
Dropout (p=0.5)					
conv2d-100	conv2d-100	conv2d-100	conv2d-100	conv2d-100	conv2d-100
ELU					
Batchnorm					
AvgPool2d					
Dropout (p=0.5)					
conv2d-200			conv2d-200	conv2d-200	conv2d-200
ELU			ELU	ELU	ELU
Batchnorm			Batchnorm	Batchnorm	Batchnorm
AvgPool2d			AvgPool2d	AvgPool2d	AvgPool2d
Dropout (p=0.5)			Dropout (p=0.5)	Dropout (p=0.5)	Dropout (p=0.5)
	Linear-100	Linear-100	conv2d-400		
			ELU		
			Batchnorm	Linear-100	Linear-100
			AvgPool2d		
Linear-4			Dropout (p=0.5)		
	LSTM	TransformerEncoderLayer	Linear-4	LSTM	TransformerEncoderLayer
	Linear-4			Linear-4	
ELU					
Softmax					

Table 3. Summary of architectures

PyTorch was used to construct the architectures in Table 3. It should be noted that the dimensions $(N \times C \times T)$ of the input layer denote the number of data instances, the number of channels, and the selected effective time steps of the EEG dataset. Let $N_0 \times C \times T_0$ denote the dimensions of an unprocessed training (or testing) EEG dataset, and let $N_0 \times$ C $\times$ T_{sub} be the dimensions of a subset of the EEG data (with respect to the time steps). We preprocess the data by concatenating maxpooled features, averaged features with additional Gaussian noise, and two disjoint partitions of the original data with added Gaussian noise. Hence, our effective dataset has dimensions $(N \times C \times T) = (4N_0 \times C \times T_{sub}/2)$, where $T_{sub} \leq T_0$.

Models were trained using a batch size of 64 over 50 epochs. The Adam optimizer was used to optimize cross entropy loss, using a learning rate of $\gamma = 0.0005$, a weight decay $\lambda = 0.01$, and beta values $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The notation "Linear-$\langle$output$\rangle$" and "conv2d-$\langle$output channels$\rangle$" are used to denote the outputs of the linear and convolutional layers used in Table 3. All conv2d layers use zero padding and utilize 1×10 temporal convolutional kernels with a stride of 1. The AvgPool2d layers use 1×3 kernels at a stride of 1. The LSTM layers use a hidden state size of 10 and dropout of p = 0.2. The TransformerEncoderLayer blocks consist of self-attention (10 heads) and a feedforward network with dropout p = 0.5.