Prediction of finger movements from EEG recordings

Cleres David
Computational Science & Engineering
Email: david.cleres@epfl.ch

Lesimple Nicolas
Computational Science & Engineering
Email: nicolas.lesimple@epfl.ch

Rensonnet Gaëtan
Signal Processing Lab (LTS5)
Email: gaetan.rensonnet@epfl.ch

***Abstract*—This first project for the Deep Learning Course (EE-559) aimed to develop and train a predictor of finger movements from Electroencephalography (EEG) recordings. *Convolutional Neural Networks (CNN)* paired with Data Augmentation seemed to solve this classification task with the highest accuracy. Other classification techniques like *Multilayer Perceptron (MLP)* and different regression-based techniques such as Logistic Regression, Nearest Neighbor and Support Vector Machine (SVM) were also investigated with and without pre-processing and feature extraction but yielded lower accuracy. By putting all the best hyperparamters together it was finally possible to train a model that reached 82.6± 4.01% validation- and 78.6±2.47% test accuracy on average over 15 independent repetitions.**

I. Introduction

Deep learning and Machine Learning methods raised public attention many times in the past years through *moonshot projects* like IBM's TrueNorth [9] or the AlphaGo project from the IT-giant Google. During the past decade, with the increase of mobile devices but also the use of increasingly confiscated medical devices like EEG electrodes, the need to categorize signals from various sources saw a growing interest, especially in the field of Brain Computer Interaction (BCI). This emergence sprang from a combination of factors including inexpensive computer hardware and software which are now available and support the complex high-speed analyses of brain activity that are essential in BCI. Another factor is the greater understanding of the central nervous system and as already mentioned the improved methods for recording these signals in both the short-term and long-term. This project aimed to classify signals from EEG recordings with a trained network in order to predict the laterality of a finger movement (whether it moved to the left or to the right). In other words, it was a standard binary classification problem. The adopted strategy to tackle this challenge was by starting from classic machine learning techniques (Logistic Regression, SVM, ...) and then to investigate models with incremental complexity throughout the project as for instance Multilayer Perceptrons and Convolutional Neural Networks. This report starts with a description of the dataset. Then, the basic principles of CNN are presented. Next, the CNN pros & cons are compared to those of techniques such as logistic regression and Multilayer Perceptron (MLP). Section IV is dedicated to presenting the results of the different neural net architectures and regressions. Finally, discussion and auto-critic based on the results are made in the final part of the report.

II. Dataset

This dataset [2] was recorded from a normal subject during a no-feedback session. The task was to press keys with the index and the little fingers in a self-chosen order and timing 'self-paced key typing'. The experiment consisted of 3 sessions of 6 minutes each. All sessions were conducted on the same day with short breaks in between. Typing was done at an average speed of 1 key per second. The entire dataset consisted of 416 recordings from Nchannels=28 electrodes of 500 ms length each ending 130 ms before a keypress, labeled (**0** for upcoming **left** hand movements and **1** for upcoming **right** hand movements). The dataset was split into a training set of 316 recordings and a test set containing the remaining 100 recordings. The data were provided in the original 1000 Hz sampling, resulting in Nsamples=500 time samples, and in a version down-sampled to 100 Hz, i.e. containing Nsamples=50 time points per recording. Throughout our experiments, only the low-frequency 100-Hz dataset was used. One typical recording for all 28 electrodes during one experiment is shown in Fig. 2.

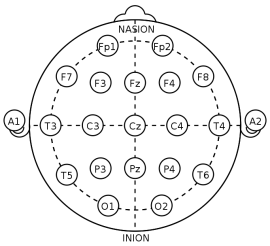

Fig. 1: Electrode locations of the international 10-20 system.

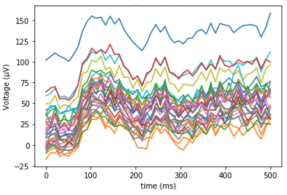

Fig. 2: Raw EEG Data for one observation on one subject and all 28 channels.

III. Methods

A. Preprocessing

Our preprocessing consisted of three steps. First, a cut-band filter was applied in a narrow band around 50Hz to get rid of electrical noise. Second, signal *detrending* was performed by subtracting a best-fit line (in the least-squares sense) from the data. This allowed us to focus on the fluctuations in the data about the trend instead of on the trend itself as systematic shifts could result from sensor (electrode) drift, for example. Third, the dataset was normalized by subtracting the mean and dividing by the standard deviation of each experiment.

B. Data Augmentation

Small data sets generally limit the ability of deep learning models to learn discriminative features and lead to overfitting [7]. Given the small size of our dataset, data augmentation was performed by subsampling the richer 1000-Hz dataset in the time domain. For each 1000-Hz recording, 10 different 100-Hz recordings were obtained by taking equally-spaced time samples and shifting the starting point to 10 different positions.

C. Feature Engineering

For each EEG recording, a 2D feature vector was extracted containing 20 scalar metrics from the FT transform, the wavelet transform and the Welch energy of the signal, as those are commonly acknowledged to contain meaningful signal information [1], [10]. The data was standardized by column as the different features did not represent the same quantities. PCA was also performed to determine which of these 20 features contained most of the signal variance.

D. Traditional Machine Learning Methods

1) Nearest Neighbor: In k-nearest neighbors (k-NN) classification, an object is classified by a majority vote of its k nearest neighbors and labeled with the class most common among these k neighbors. If k = 1, then the object is simply assigned to the class of that single nearest neighbor. In our code, each sample was treated as a Nchannels*Nsamples 1D vector and simple 1-nearest-neighbor (1-NN) matching was performed. Principal Component Analysis (PCA) with varying numbers of principal components coupled with 1-NN matching was also investigated. Test samples were classified one by one to avoid out-of-RAM errors.

2) Logistic Regression: Logistic regression is particularly appropriate for binary regression problems as it explains the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables [5].

3) Support Vector Machine (SVM): SVM is a usually more powerful classification model which has traditionally performed well at BCI challenges. The technique was therefore investigated in our problem both on raw samples treated Nchannels*Nsamples 1D vectors and on the feature vectors described above [11].

E. Multi Layer Perceptron & Neural Networks (NN)

A Multilayer Perceptron (MLP) is a neural network consisting of a succession of fully-connected linear layers interspersed with non-linear activation layers. The layers located between the input and the output layers are referred to as hidden layers. We used the rectified linear unit (ReLU) non-linearity and only considered MLPs with identical number of units across all hidden layers.

A total of 15 different MLP architectures were compared resulting from the combination of MLPs with L=1, 2 and 3 hidden layers and hidden layers containing N_h=350, 700, 2100, 2800, 7000 and 14000 units per layer. Since the samples were flattened into 1D vectors with Nchannels*Nsamples=1,400 components, these numbers of units represent between 0.25 and 10 times the dimension of the input data. The network parameters were estimated with vanilla-form stochastic gradient descent (SGD) using the cross-entropy loss function, which is well suited for classification. Each MLP architecture was trained a total of 15 times resulting from the use of three different learning rates (or gradient step sizes), namely 1e-5, 1e-4 and 1e-3, combined with five different batch sizes, namely 10, 15, 20, 25 and 30. Training was stopped as soon as zero classification error was reached on the whole training set.

First, the effect of network depth (or number of hidden layers) on classification performance was examined by reporting, for L=1, 2 and 3, the mean and standard deviation of the test error rate computed over all 75 experiments at fixed L (5 values for N_h and 15 optimization strategies). Second, fixing the network depth L to the optimal value determined in the first step, the impact of the number of hidden units per layer N_h was investigated. The mean and standard deviation of the test error rates were reported over all 15 estimation strategies. Third, fixing L and N_h to the optimal values determined in the two previous steps, the effect of the batch size was investigated. The mean and standard deviation of the test error were computed over the 5 possible values of the learning rate.

The impact of the optimization algorithm was also examined. The network configuration, batch size and learning rate that led to the best test error in the experiments described above were selected and the network was successively trained using the SGD, Adam and Adagrad algorithms.

We finally trained and tested the best network selected by the above procedure with the best optimization strategy and reported the mean test error over 10 independent runs, where the randomness stemmed from weight initialization alone.

F. Convolutional Neural Networks (CNN)

Simple neural networks such as described in the previous paragraph are known for not scaling well to larger inputs, as the number of parameters to reach good accuracy tends to increase dramatically [6]. It is reasonable to assume that convolutional layers may be able to take advantage of the structure of our data using fewer free parameters [12].

In the case of EEG signals recorded from 28 electrodes on the skull, one could make two different interpretations of the data. On one hand, one could take each electrode as one ”color channel” of the same recording/”image” which leaves us with 28 skull channels each being of size 1x50. On the other hand, one could see the input as having 1 ”color channel” and being one ”image” of size 28x50. Both scenarios were explored in the different neural nets that we implemented.

CNNs offer 2d convolutions which are likely to identify patterns in the signal. Pooling layers can be used after the convolutions to reduce the dimensionality of the problem by down-sampling along the *temporal dimension* and/or *the spatial dimension*. Considering that the input tensor was of dimension 316x1x28x50 (see Section II), performing a convolution along the spatial dimension meant using a vertical filter to learn something on the 28 electrodes which are placed around the skull at a fixed time point. A temporal convolution consists in using a horizontal filter in order to study the EEG signal's evolution over time for one particular electrode. For instance, at 100 Hz, the time between two samples was 10 ms. Hence by taking windows of size (1x5), it was possible to investigate the signal in 50 ms intervals. The convolutions could be specialized to the data by specifying some extra parameters such as striding, padding, and dilation. The **zero-padding** hyperparameter enabled the spatial size of the output volumes to be controlled. The **stride** controlled the size of the output surface. **Dilation** allowed us to specify if at time point t, the network should look at t_{-2}, t_{-1}, t,t_{+1} and t_{+2} (for filter size 5) or rather in t_{-4}, t_{-2}, t, t_{+2} and t_{+4} for a dilation of 2. This was interesting especially in the first layers of convolution since it made it possible to have a larger overview of the signal. Finally, using a Fully Connected (FC) output layer is useful for classification, as described in the previous paragraph. Using all these tools together, CNNs transform the original signal layer by layer from the original values to the final class scores.

Four specific architectures were implemented. The first architecture, EEGNet, is presented in Fig. 6. BCE was used as loss function, Adam as an optimizer (with defaut learning rate). The second one (Model 2) is the CNN depicted in Fig. 7, which was trained with BCE and Adam. The third model, referred to as *Medium CNN* had the architecture shown in Fig. 7 but the number of filters was increased from 4 to 32 in the first layer and 4 to 64 in the second layer. The same was true for the last tested model, the *Heavy model*, where the first layer contained 64 filters and the second layer 128.

IV. Results

A. Pre-Processing and feature engineering

Figure 5 shows that only 3 out of 20 selected features retained 90% of the relative cumulated standard deviation, suggesting redundancy between our features. Classification on feature vectors was therefore not further investigated.

Figures 3 & 4 show examples of EEG recordings preprocessed with band-pass filtering, detrending and normalization, as described in Section 4. Table I suggests that the results of the basic regressions were generally rather poor with test error rates hovering around 50%. Feeding the models with the extracted features from the preprocessing led to an decrease in error rate of up to 12 % in the case of SVM, leading to an optimal test error rate of 39% among all the traditional classifiers tested.

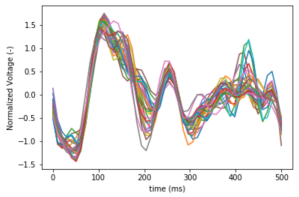

Fig. 3: EEG recording for a left movement - 0 labeled.

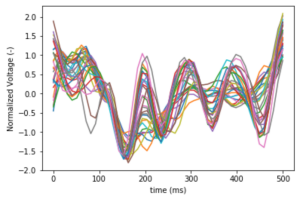

Fig. 4: EEG recording for a right movement - 1 labeled.

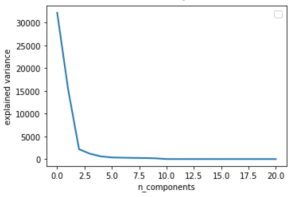

Fig. 5: PCA results showing the explained variance in function of the kept features.

Layer	Input ($C \times T$)	Operation	Output	Number of Parameters
1	$C \times T$	16 × Conv1D (C×1)	$16 \times 1 \times T$	$16C + 16$
	$16 \times 1 \times T$	BatchNorm	$16 \times 1 \times T$	32
	$16 \times 1 \times T$	Transpose	$1 \times 16 \times T$	
	$1 \times 16 \times T$	Dropout (.25)	$1 \times 16 \times T$	
2	$1 \times 16 \times T$	4 × Conv2D (2×32)	$4 \times 16 \times T$	$4 \times 2 \times 32 + 4 = 260$
	$4 \times 16 \times T$	BatchNorm	$4 \times 16 \times T$	8
	$4 \times 16 \times T$	Maxpool2D (2,4)	$4 \times 8 \times T/4$	
	$4 \times 8 \times T/4$	Dropout (.25)	$4 \times 8 \times T/4$	
3	$4 \times 8 \times T/4$	4 × Conv2D (8×4)	$4 \times 8 \times T/4$	$4 \times 4 \times 8 \times 4 + 4 = 516$
	$4 \times 8 \times T/4$	BatchNorm	$4 \times 8 \times T/4$	8
	$4 \times 8 \times T/4$	Maxpool2D (2,4)	$4 \times 4 \times T/16$	
	$4 \times 4 \times T/16$	Dropout (.25)	$4 \times 4 \times T/16$	
4	$4 \times 4 \times T/16$	Softmax Regression	N	$TN + N$
Total				$16C + N(T + 1) + 840$

Fig. 6: Model 1 - EEGNet: CNN for EEG-based Brain-Computer Interfaces [8].

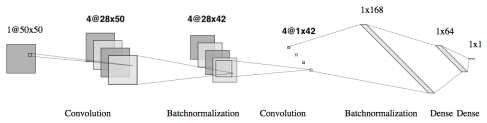

Fig. 7: Model 2: Architecture of the CNN with the highest accuracy and inspired by the knowledge acquired on the MLP designs.

B. MLP

As shown in Table II, single-layer MLPs had an overall lower test error than the MLP models with 2 and 3 hidden layers, although the differences in test error were very small. The standard deviation of the simple 1-layer model was slightly larger than those of the more complex 2- and 3-layer MLPs. Figures 8 and 9 suggest that the optimal number of units per hidden layer was 350 that the optimal batch size was 25, respectively. Table III suggest that SGD peformed slightly better than Adam and Adagrad, in addition featuring a lower standard deviation.

The optimal network architecture was therefore found to be an MLP with a single hidden layer with 350 nodes and the best

Model	Test Error
LR + PCA	0.53
SVM + PCA	0.51
KNN	0.49
LR + PCA + FE	0.51
SVM + PCA + FE	0.39
KNN + FE	0.47

TABLE I: Test accuracy for different traditional machine learning classifiers, with and without feature engineering (FE). LR:logistic regression, SVM:support vector machine, PCA: principal component analysis, KNN: k-nearest neighbors.

optimization strategy was SGD with batch size 25 and learning rate 1e-4 (determined independently). This model reached a test error of 32.9% a standard deviation of 3.41% across all 10 independent runs. Training the model after performing preprocessing and data augmentation yielded a mean test error of 33% with a standard deviation of 2.11%, thus making the model slightly more stable but hardly affecting the overall classification accuracy.

Number of hidden layers	Mean Test Error	Std
1-layer perceptrons (MLP1)	35.560 %	4,027%
2-layer perceptrons (MLP2)	36.386 %	3,605%
3-layer perceptrons (MLP3)	37.903 %	3,071%

TABLE II: Test error vs network depth. Mean error computed over all 75 different experiments (3 numbers of units per layer and 15 optimization strategies).

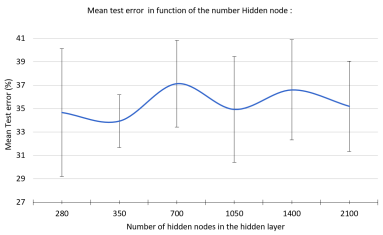

Fig. 8: Effect of the number of units per hidden layer on the test error for the single-layer models

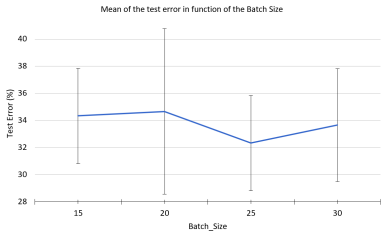

Fig. 9: Effect of batch size on test error for a single-layer MLP with 350 nodes per hidden layer

C. CNN

The data was very noisy and the accuracy of the model depended strongly on the initialization weights. It was therefore decided to define 15 seeds randomly but to use these seeds in the all the experiments so that the average score over the 15

Optimizer	Mean Test Error +- std
SGD	32.9% +- 3.41%
Adam	33.2% +-4.07%
Adagrad	35.6% +- 4.16%

TABLE III: Mean test error and Standard deviation for several optimizers with the following fixed parameters for SLP: loss was Cross Entropy, learning rate was 10^{-4}, batch size is 25 and number of hidden node was 350.

experiments was able to show which model was the best. A comparison between all the models is shown in table V. Figure 10shows that as the number of epochs increased, the prediction score increased as well. There was, however, a trade-off as the training time was proportional to the number of epochs, while the score saturated and the model started to over-fit from about 40 epochs. Figure 11 shows that the size of the batches

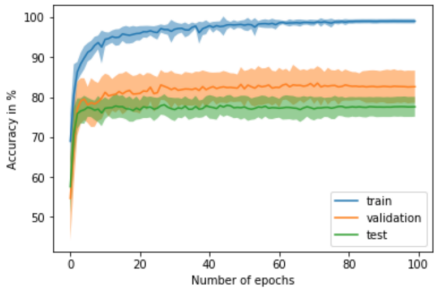

Fig. 10: CNN - Effect of number of epochs on the test accuracy. Mean score with standard deviation of the best model over 15 independent runs. The chosen model was the CNN model using BCE as loss function and Adam as optimizer with batch size 15

had an influence on the precision of the model with batches of 7 samples giving the best results, as shown in Fig. 10. Moreover, the best dropout percentage has been found being 0.2 (cf. Fig.12). Finally, by putting all the best hyperparamters together it was possible to train a model that reached 82.6± 4.01% validation- and 78.6±2.47% test accuracy on average over 15 repeats with different random seeds. However, it was sometimes possible to reach up to 85.6±0.% validation- and 81.6±% test error with a specific random seed and weight initialization. Adding extra features to the more complex CNN models (medium and heavy CNN as described in V) allowed the standard deviation to decrease from 4.01% to 1.98% and 2.5% for the validation set with the Heavy model and similarly for the test dataset. However, being more reliable did not make the larger models more accurate, confirming the observations made on MLPs in the previous paragraph.

V. Discussion

One of the major drawbacks of using sophisticated CNNs was their high time consumption, especially as the number of input data (by data augmentation) or layers increased. However, it was also observable, in this specific learning task that the very complex model did not perform well when it came to learning the important features. As observed on the MLP experiments and then confirmed by the different trials

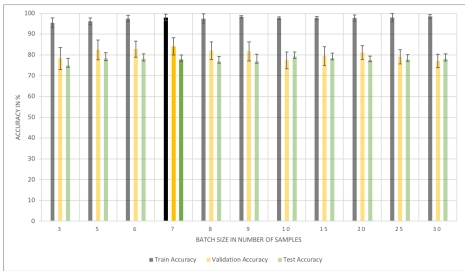

Fig. 11: CNN - Effect of the batch size on the train accuracy (gray), validation accuracy (yellow) and test accuracy (green). The tested model was *Model 2 CNN*.

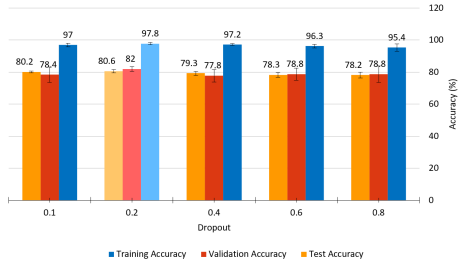

Fig. 12: CNN - Effect of dropout on the test accuracy (orange), validation accuracy (red) and train accuracy (blue). The tested model was *Model 2 CNN*.

on the CNNs, too much complexity killed the prediction score.

Pre-processing and feature engineering helped traditional machine learning models but did not significantly improve the performance of the neural networks. This could be expected since one of the major breakthroughs of deep networks was their ability to "learn by themselves" basic operations like for instance, filtering, normalization and to a certain extent basic feature engineering. Data augmentation on the other hand had a globally positive impact on the prediction capabilities of our neural networks. This is a recurrent feature in deep learning where models perform best with larger datasets.

In order to assess the robustness and the accuracy of the model, it was necessary to compute several repetitions of the same code with different random seeds for the model's weight initialization. Nevertheless, for the figures of this report, it was necessary to fix the all the different random seeds in order to prove that the produced plots did not happen by chance when it came to the best prediction score.

VI. Conclusion & Perspectives

This project demonstrated the difficulty of handling all the different parameters of a Deep Learning Model in order to compute a reliable prediction. In particular, it shed light on the importance of the weight initialization procedure. By gradually increasing the complexity of our classifiers, we have gone from an average test error rate of 39% with SVM (paired with feature engineering and PCA), down to 32.9% with an MLP and finally 21.4% using a relatively simple CNN.

Optimizer	Loss Function	Validation Error	Test Error
SGD	BCE	22.9	21.2
Adam	**BCE**	**20.6**	**21.4**
Adamax	BCE	22.9	22.6
SGD	CE	30.3	23.5
Adam	CE	20.6	21.4
Adamax	CE	23.2	21.8
SGD	MSE	21.2	21.4
Adam	MSE	22.4	21.8
Adamax	MSE	25.5	21.6
SGD	L1	37.6	28.7
Adam	L1	22.5	21.4
Adamax	L1	24.8	21.8

TABLE IV: Mean accuracy (in %) for different optimizer and loss functions for batch size B=15 samples. The test architecture was the one for fig. 7

Model	Validation err.	Test err.
EGGNet CNN	27.9±1.98	28.3±2.5
EGGNet CNN + DA	30,8±4.314	30.5±4.34
Model 2 CNN	18,1±4.21	21,68±2.64
Model 2 CNN + DA	**17.4**±4.01	**21.4**±2.47
Medium CNN + DA	23.8±1.98	21.7±1.76
Heavy CNN + DA	26.2±2.5	19.7±1.49

TABLE V: Best prediction accuracies for different model architectures for 10 fixed seeds, batch size 15, BCE loss function and Adam optimizer. DA: Data Augmentation

'Smart' feature engineering was able to improve traditional models but for deep learning, only data augmentation had a positive effect. From the MLP and the CNN experiments, we generally conclude the the simpler models performed best.

Due to time reasons, some other architectures could not be implemented. According to literature Recurrent Convolutional Neural Networks (RCNN) [3] are considered as being the state-of-art for certain EEG signal processing applications as well as Long Short-Term Memory Models (LSTM) [4]. Given that neural activity measured by EEG can be regarded as highly dynamic, non-linear time series data, recurrent neural networks and particularly LSTM appear to be a tool of choice for modeling the temporal characteristics of brain activity. However, sticking to CNN could also be another solution. It would have been interesting to compute two networks in parallel - one deep CNN as presented in the report and a second one based on the extracted features from the feature engineering.

REFERENCES

[1] Amjed S Al-Fahoum and Ausilah A Al-Fraihat. Methods of eeg signal features extraction using linear analysis in frequency and time-frequency domains. *ISRN neuroscience*, 2014, 2014.

[2] Benjamin Blankertz, Gabriel Curio, and Klaus-Robert Müller. Classifying single trial eeg: Towards brain computer interfacing. In *Advances in neural information processing systems*, pages 157–164, 2002.

[3] Juri Fedjaev. Decoding eeg brain signals using recurrent neural networks.

[4] Nihal Fatma Güler, Elif Derya Übeyli, and Inan Güler. Recurrent neural networks employing lyapunov exponents for eeg signals classification. *Expert systems with applications*, 29(3):506–514, 2005.

[5] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

[6] Serena Yeung Justin Johnson, Fei-Fei Li. Cs231n: Convolutional neural networks for visual recognition. In *Stanford Course CS231n*. Stanford University, 2017.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F Pereira, C J C Burges, L Bottou, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[8] Vernon J Lawhern, Amelia J Solon, Nicholas R Waytowich, Stephen M Gordon, Chou P Hung, and Brent J Lance. Eegnet: A compact convolutional network for eeg-based brain-computer interfaces. *arXiv preprint arXiv:1611.08024*, 2016.

[9] Ewan Nurse, Benjamin S Mashford, Antonio Jimeno Yepes, Isabell Kiral-Kornek, Stefan Harrer, and Dean R Freestone. Decoding eeg and lfp signals using deep learning: heading truenorth. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 259–266. ACM, 2016.

[10] Abdul-Bary Raouf Suleiman, Toka Abdul-Hameed Fatehi, et al. Features extraction techniqes of eeg signal for bci applications. *Faculty of Computer and Information Engineering Department College of Electronics Engineering, University of Mosul, Iraq*, 2007.

[11] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[12] Schirrmeister Robin Tibor, Springenberg Jost Tobias, Fiederer Lukas Dominique Josef, Glasstetter Martin, Eggensperger Katharina, Tangermann Michael, Hutter Frank, Burgard Wolfram, and Ball Tonio. Deep learning with convolutional neural networks for EEG decoding and visualization. *Human Brain Mapping*, 38(11):5391–5420.