FPT UNIVERSITY

AIP391 Project

Hyperparameters tuning of Convolutional neural network for driver drowsiness detection from single-channel EEG

Thach Duc Long, Duong Minh Hieu

Advisor: Phan Duy Hung

Abstract:

Driver drowsiness is one of the main factors leading to road fatalities and hazards in the transportation industry. Electroencephalography (EEG) has been considered as one of the best physiological signals to detect drivers' drowsy states, since it directly measures neurophysiological activities in the brain. However, designing a calibration-free system for driver drowsiness detection with EEG is still a challenging task, as EEG suffers from serious mental and physical drifts across different subjects. By using a compact and interpretable Convolutional Neural Network (CNN) to discover shared EEG features across different subjects for driver drowsiness detection, results show that the model can achieve an average accuracy of 73.22% on 11 subjects for 2-class cross-subject EEG signal classification, which is higher than conventional machine learning methods and other state-of-art deep learning methods. It is revealed by the visualization technique that the model has learned biologically explainable features, e.g…, To better understand how the CNN works, in our project, we calibrate these hyperparameters of this algorithm. Based on these tunes, we can know what role do those hyperparameters play in the algorithm.

Keywords: CNN, driver-drowsiness, EEG, hyper-parameter.

1. Introduction

Driving safety is very important to our everyday life. However, according to the World Health Organization [1] "Global Status Report on Road Safety 2018", the number of road traffic deaths continues to rise steadily, reaching 1.35 million in 2016. ... Road traffic injuries are the eighth leading cause of death for all age groups.

More people now die as a result of road traffic injuries than from HIV/AIDS, tuberculosis or diarrhoeal diseases. Road traffic injuries are currently the leading cause of death for children and young adults aged 5–29 years. In which, the number of accidents caused by drowsiness accounts for a large number.

To solve this problem, many solutions have been proposed. However, the use of EEG to give the warning about sleepiness has brought tremendous effects. By using sensors placed around the head when driving, we get signals from different regions of the brain. Then, we analyze the received signals, thereby giving warnings when drivers drive with signs of drowsiness. Through many times of testing deverse algorithms, researchers have shown that CNN is the optimal algorithm for this problem and also give better results.

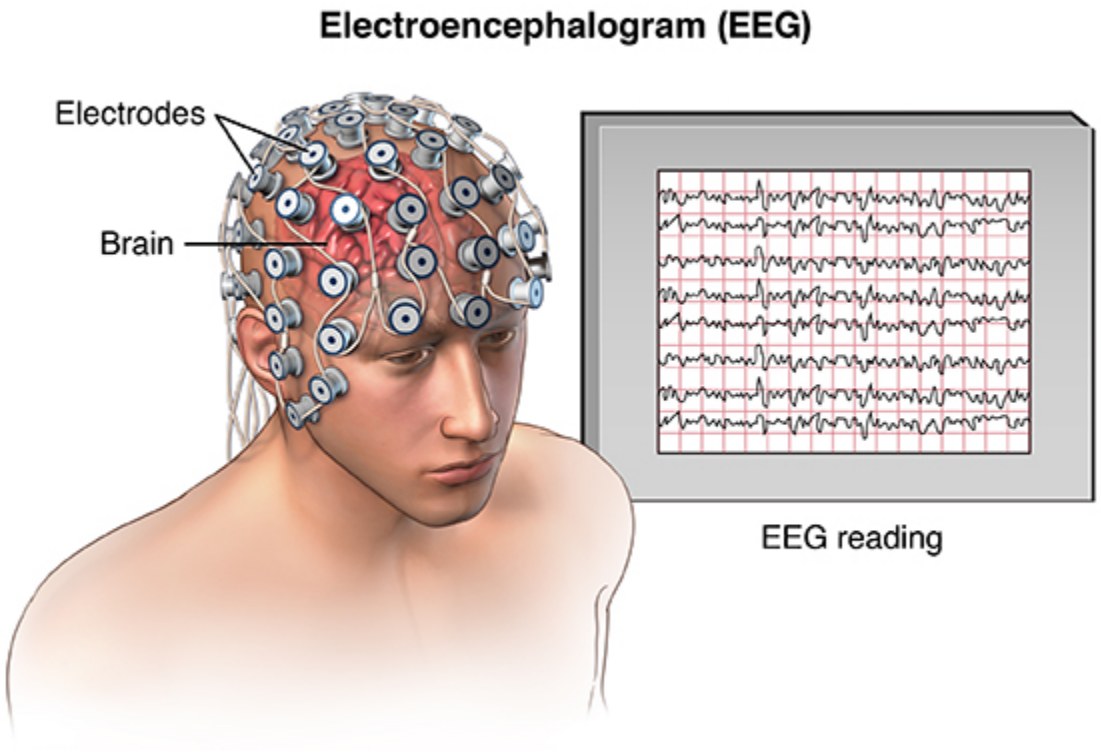


Figure 1: Example of electroencephalogram (EEG) in real life

Our project's goal is to tune the hyperparameters of a Convolutional Neural Network model with high accuracy to observe the changes in learning speed and accuracy to see if any augmentation can improve the model both in the development process and real-time detection performance.

A CNN has a large number of hyperparameters that should be tuned finely for each task it is responsible for. There is currently no deterministic method that can be used to find the best combination of values for these hyperparameters. Normally,

CNN is given on some first guess and tuned later on with little effort. Engineers tend to use experience in guessing and later tuning. Architecture and hyperparameters are determined by guessing and estimating better scenarios. This is called guesstimating.

With the desire to achieve the best possible result in both training and deployment, we want to find a significantly better combination of hyperparameters of a good CNN we found about our concerning problem: driver drowsiness.

2. Related Works

We first introduce the currently most used methods for hyperparameter optimization, with a focus on CNN architectures.

An overview of fundamental methods for hyperparameter optimization can be found, for instance, in [2,7]. The best-known methods are Grid Search (GS), Random Search (RS), Bayesian Optimization (BO), Nelder-Mead (NM), Simulated Annealing (SA), Particle Swarm Optimization, and Evolutionary Algorithms. Until recently, the most commonly used hyperparameter optimization strategy was a combination of GS and manual tuning. More complex techniques are now largely available in a variety of software packages.

In current CNN architectures, the number of hyperparameters has increased significantly. These hyperparameters include the number of layers, filter size, number of feature maps, stride, pooling regions, and pooling sizes, and the number of units in each fully-connected layer. The number of these parameters can be in the order of tenths or even hundreds. Not only must a hyperparameter optimization algorithm optimize over variables which are discrete, ordinal, and continuous, but it must simultaneously choose which variables to optimize - a difficult task. Currently, no work covers optimization of every hyperparameter in designing a CNN architecture.

Albelwi et al. recently proposed a general method for CNN hyperparameter optimization [8]. Their approach is based on:

a) a preprocessing step, when they reduce the training set using instance selection;

b) the NM optimization method. For the CIFAR-10, CIFAR-100, and MINIST datasets, the following seven hyperparameters (each with its corresponding range of possible values) were considered: depth, number of fully-connected layers, number of filters, kernel sizes, number of pooling layers, pooling region sizes, number of neurons in fully-connected layers.

Currently, there are many software packages which include hyperparameter optimization methods. The user can choose between several methods (sometimes

called solvers), or continue with the default one. Popular packages in this category are [9]:

- LIBSVM [10], BayesianOptimization, Spearmint, and pyGPGO.
- Hyperopt-sklearn provides automatic algorithm configuration of the scikit-learn library. It can be used for both model selection and hyperparameter optimization. Hyperopt-sklearn has the following implemented solvers [11]: RS, SA, and TPE.
- Optunity, a Python library containing various optimizers for hyperparameter tuning: GS, RS, PSO, NM, Covariance Matrix Adaptation Evolutionary Strategy, TPE, and Sobol sequences

3. Data Preparation

The originality of the electroencephalography dataset used in the paper is an open dataset released in 2019 by Cao et al. [12]. We used the dataset that Jian et al. [13] extracted and filtered for their CNN model. The dataset consists of 3 variables: EEGSample, SubState, and SubIndex.

- **EEGSample** contains 2022 EEG data samples of size 20x384 from 11 subjects. Each sample is a 3 seconds EEG data with 128Hz from 30 channels.

  (Fp1, Fp2, F7, F3, Fz, F4, F8, FT7, FC3, FCZ, FC4, FT8, T3, C3, Cz, C4, T4, TP7, CP3, CPz, CP4, TP8, T5, P3, PZ, P4, T6, O1, Oz, O2)

  The proposed model by authors only use the 28th channel (Oz)

- **SubIndex** is an array of 2022x1. It contains the subject indexes from 1-11 corresponding to each EEG sample.
- **SubState** is an array of 2022x1. It contains the labels of the samples. 0 corresponds to the alert state and 1 corresponds to the drowsy state

  Figure 2 shows how the data is labeled and extracted.

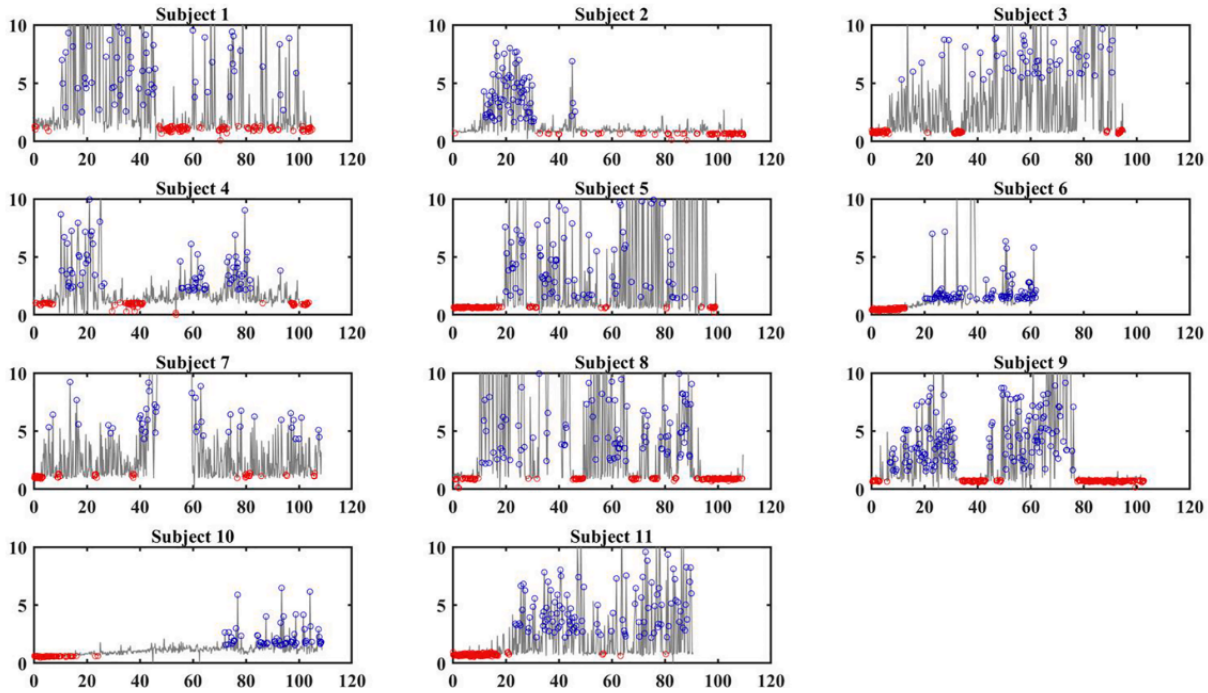


Figure 2: Distribution of the extracted samples against experiment time for 11 subjects. The x-axis is the experimental time (minutes) and the y-axis is the response time (seconds). The red circles indicate where samples in alert states are extracted, while the blue circles indicate where samples in drowsy states are extracted.

4. Methods

4.1. Compact CNN Structure and model rebuilding

The proposed model is a CNN with a compact design. The structure of the network is described in Figure 2.

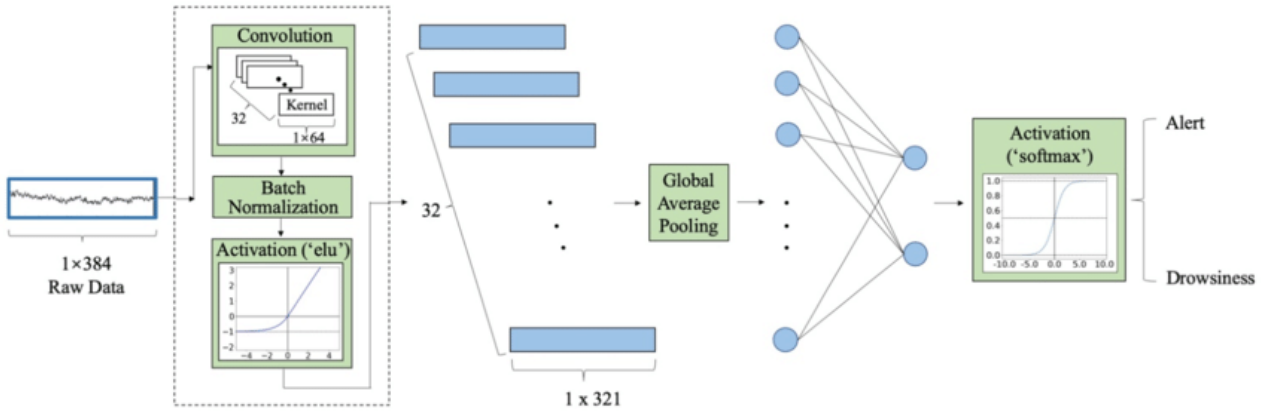


Figure 3: The structure of the proposed compact CNN model.

The first layer consists of 32 one-dimensional convolutional filters of size 64 with stride of 1. Applying these 32 convolutional filters on the one-dimensional EEG data results in a multivariate signal of 32 dimensions. The convolutional layer is followed by a batch normalization layer and an activation layer. The batch normalization operation can remove the internal covariate shift by normalizing each feature dimension across the mini-batch. Preliminary tests show that adding the batch normalization layer after the convolutional layer accelerates convergence and improves the stability of the network. The ELU function was used in the activation layer to add non-linearity transformation to the data.

Finally, a Global Average Pooling (GAP) layer is added at the end of the network followed by the Softmax function. The global pooling operation is equivalent to applying a pooling operation with the window size equal to the size of the signal outputted by the activation layer, and it reduces the filtered signals of 32 dimensions into 32 feature points. This layer dramatically reduces the parameters of the model and thus can effectively prevent overfitting during model training.

We were able to rebuild and run the network with slightly different accuracy from the proposed result. 'The overall accuracy for one run is expected to be 0.7364. However, the results will be slightly different for different computers.' - as said in the authors' code. Therefore we have our baseline as shown in Table 1.

Learning rate	Batch size	Number of epochs	Mean accuracy	Training time (s)
0.01	50	6	0.7191	9.2

Table 1: Model baseline (Training time is measured using jupyter notebook on a 4.3Ghz CPU(single-core mode), Nvidia GTX1660S GPU)

4.2. Learning rate

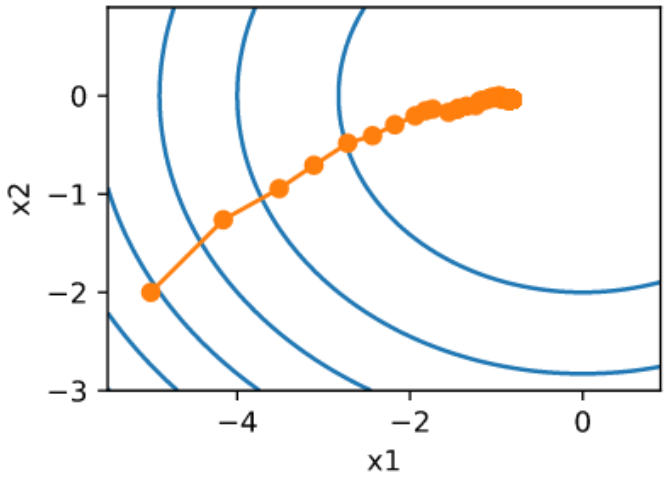


Figure 4: Visualization of a function in Hilbert space

The learning rate is a hyperparameter that controls how the model weight changes based on the error while updating. When using gradient-based optimization, we need to be careful in how we choose the learning rate to achieve convergence. One heuristic for choosing a good learning rate is to start with a small learning rate and gradually increase it. Normally, we need to validate the performance based on the validation dataset to avoid "overfitting", so the validation step will tell us what is the best weight to use.

4.3. Batch size

Batch size refers to the number of training examples utilized in one iteration. To have a batch, we take randomly the data point in the dataset until we have enough batch size. In theory algorithms, the best performance will be taken when we use the whole dataset for a training step, however, it costs a lot of computational resources. Besides, if we choose a too small batch size, the model will easily be "overfitting" because of the bias when we randomly sample from the training dataset. For faster training and saving resources purpose, choosing the right batch size is quite important.

To demonstrate the effectiveness of batch size, assume that each time our CNN takes a d^2 complexity d is the size of the training batch (the size of the data called D). To have the model convergence, we need to use the whole dataset. Following the probability theory, if we use the number of iterations follow the formula:

$$Iteration = \frac{total\_data}{batch\_size}$$

So the total complexity is :$\frac{D}{d} * d^2$. It is clear to see that if d = D, the computation is extremely bigger than a small d.

4.4. Grid Search

Since the dataset is small and training time for each run is low, we were able to perform parameter search with a grid search method.

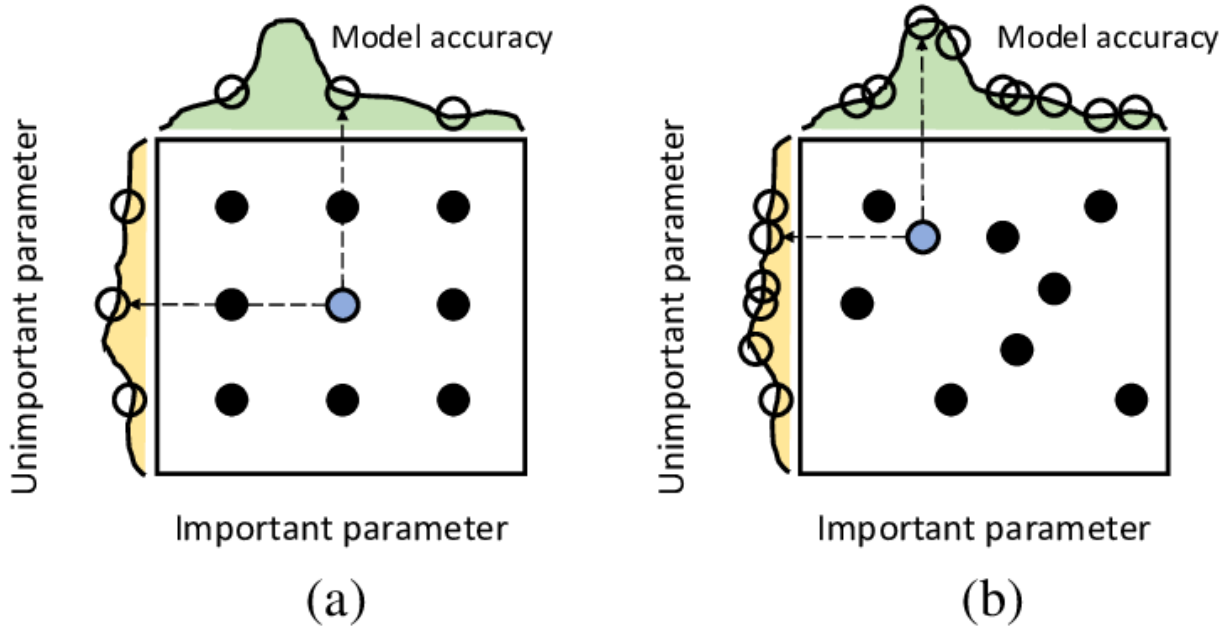


Figure 5: The difference between (a) Grid Search and (b) Random Search in hyper-parameters tuning.

We set up a basic GridSearchCV-like code to implements our parameters search. This is due to the fact that the authors' model does not follow scikit-learn CNN guidelines. The loop was fed by a set of parameters (4 learning rate values and 4 batch size values). Cross-validation value for grid search was set to 11 because the authors of the model also use leave-one-out method for cross validation on 11 subjects. We used the same 'negative log loss' scoring for unbias judge on parameters impact. To ensure the model train enough for convergence, we determine the number of epochs by following the formula:

$$n_{epochs} = \frac{400}{n_{train\_sample} / \ batch\_size}$$

5. Results and Discussion

5.1. Manually tuning learning rate and batch size

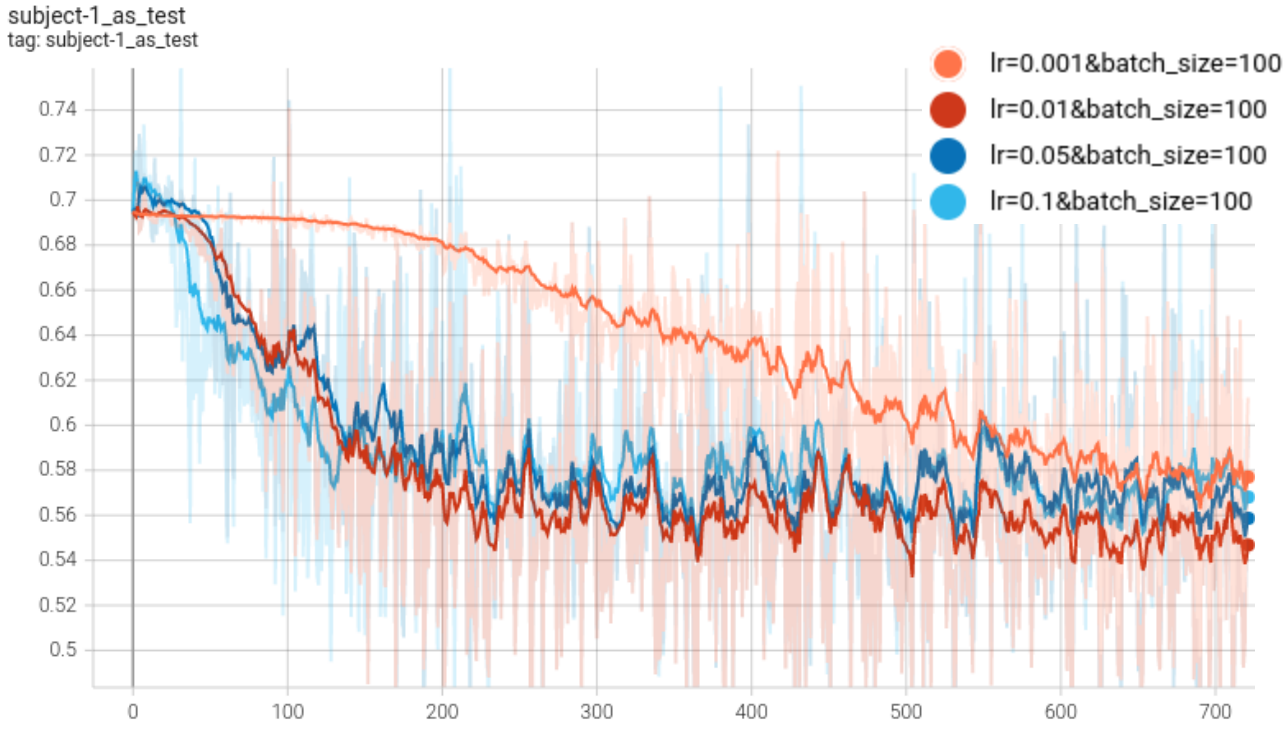


Figure 6: Result of using subject 1 as test data with learning rate [0.1, 0.05, 0.01, 0.001], batch size = 100 and number of epochs = 38.

lr=0.001&batch_size=100
lr=0.01&batch_size=100
lr=0.05&batch_size=100
lr=0.1&batch_size=100

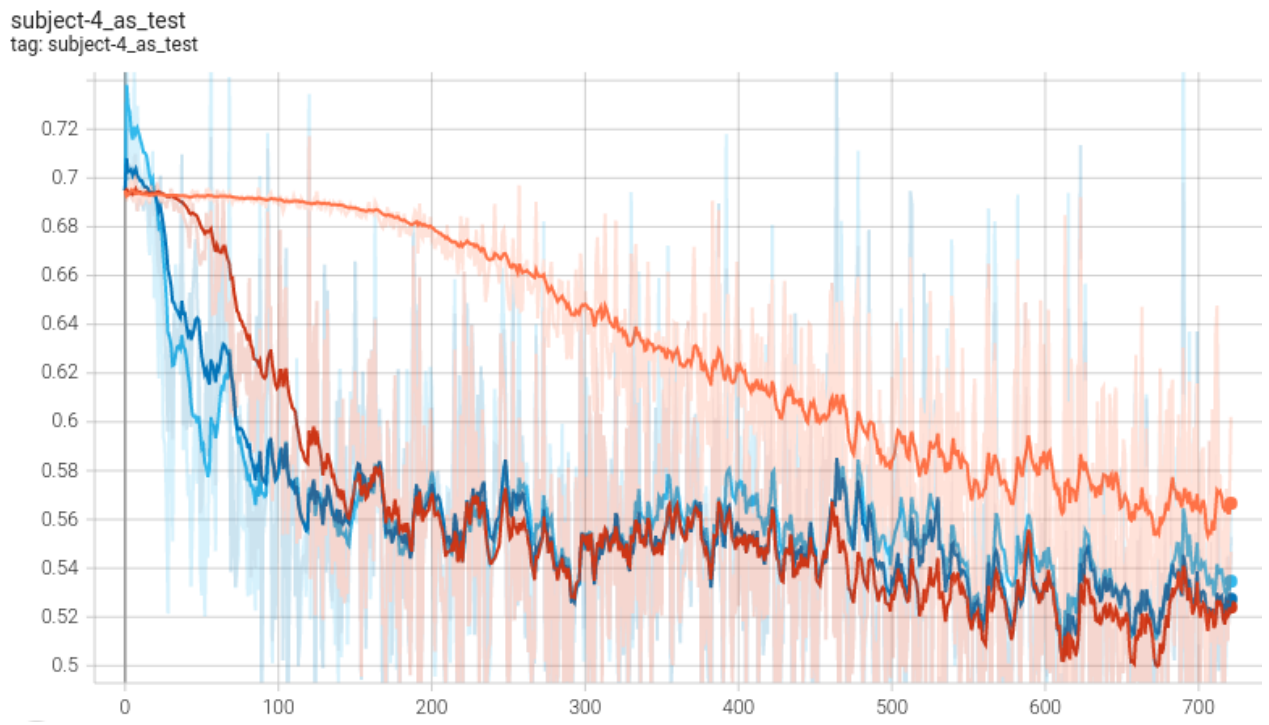


Figure 7: Result of using subject 4 as test data with learning rate [0.1, 0.05, 0.01, 0.001], batch size = 100 and number of epochs = 38

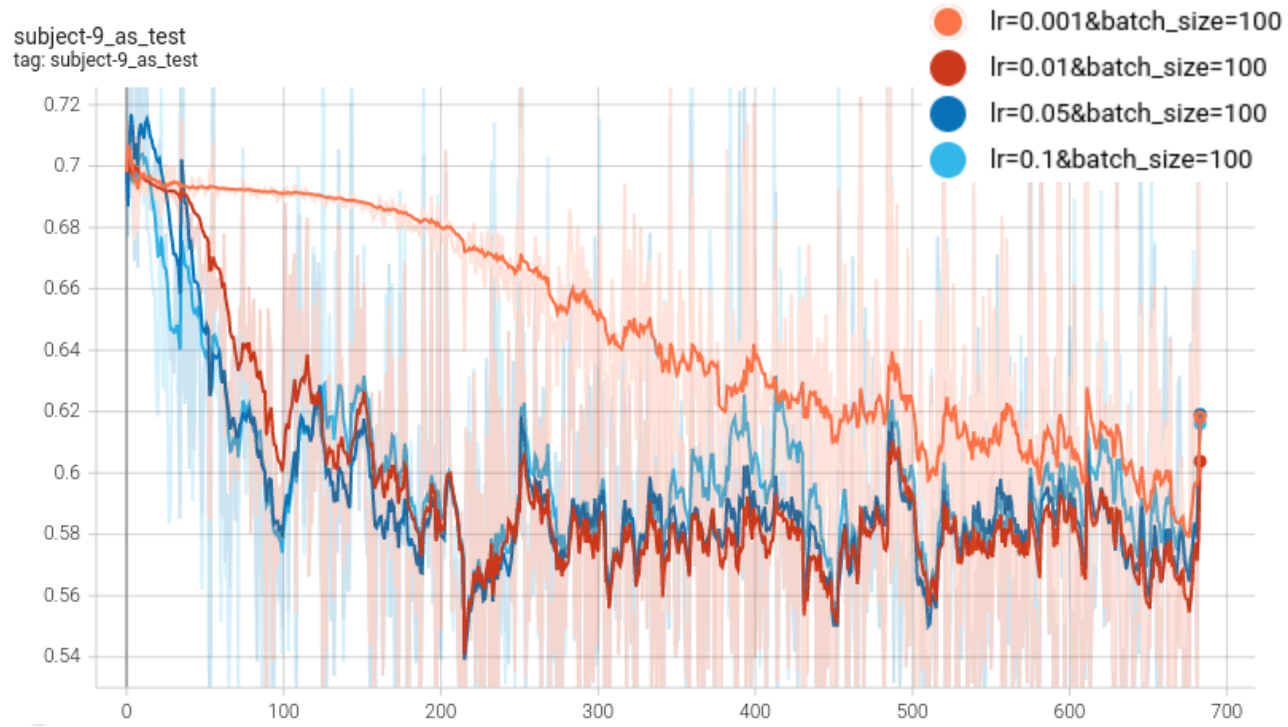
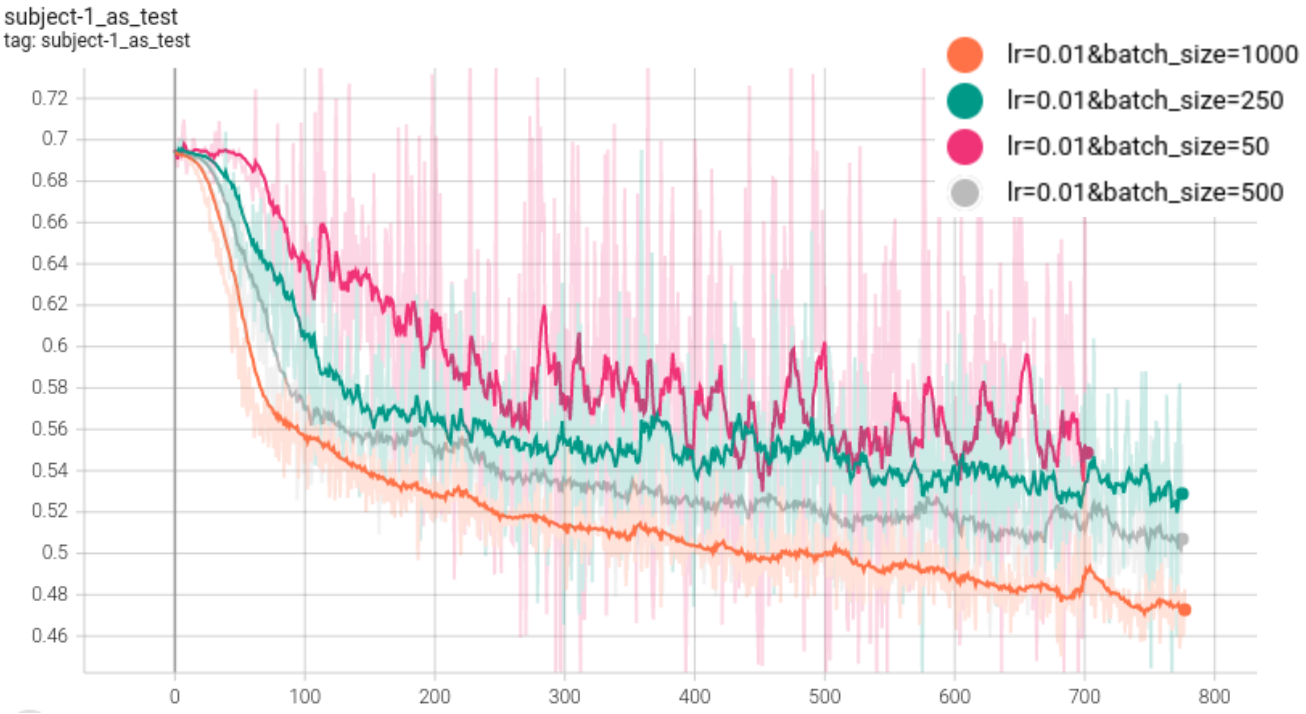
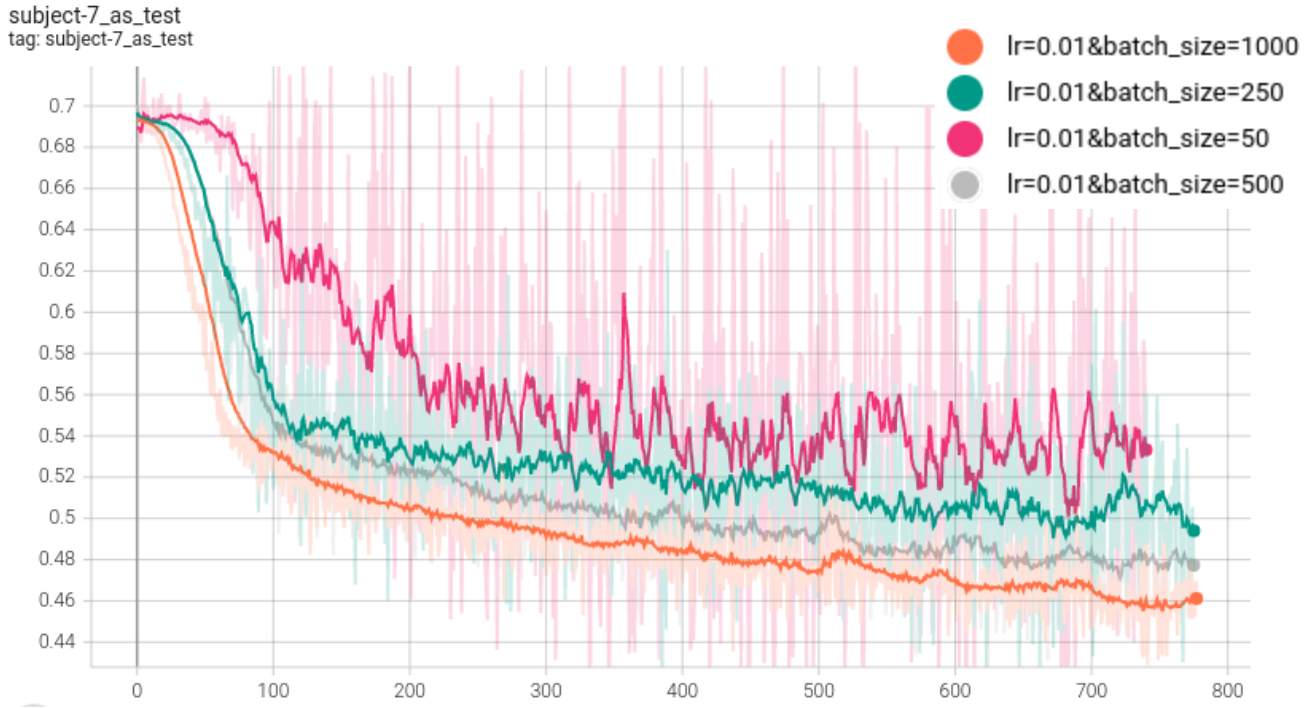
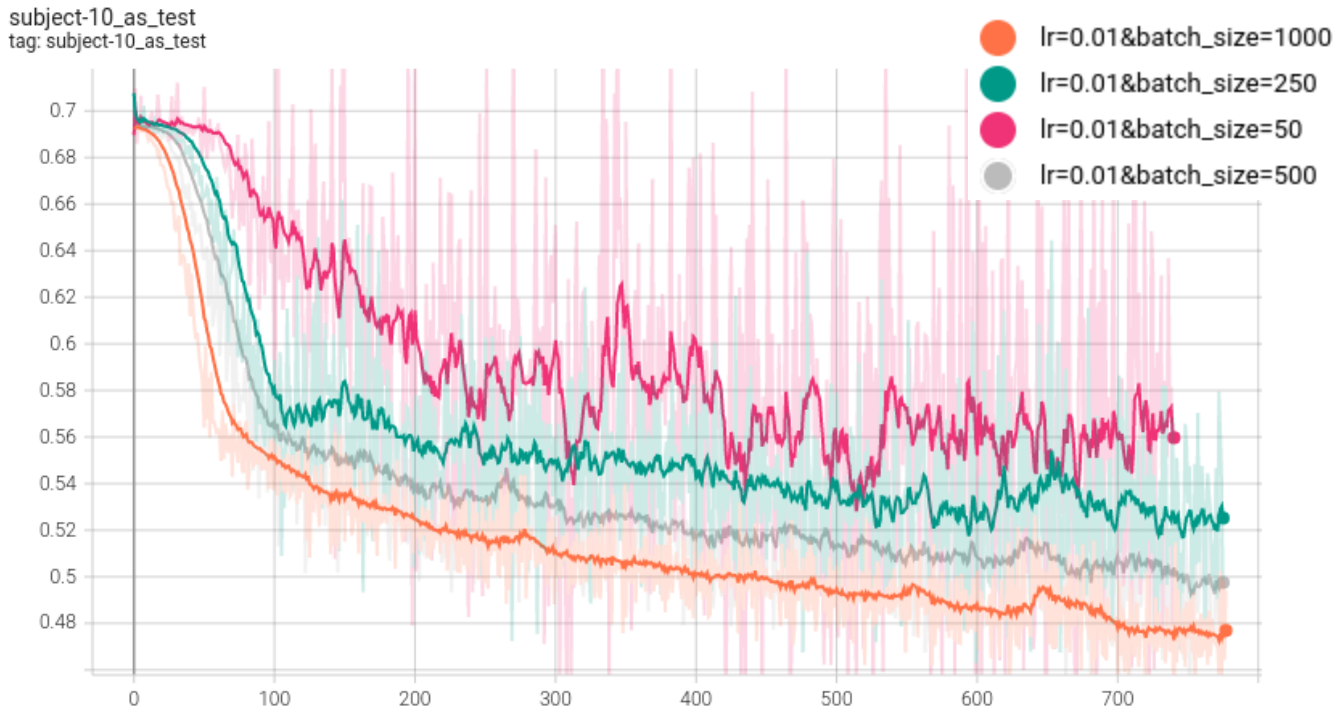


Figure 8: Result of using subject 9 as test data with learning rate [0.1, 0.05, 0.01, 0.001], batch size = 100 and number of epochs = 38







5.2. Grid Search result

Learning rate, Batch size, Epoch	Accuracy	Training time
lr=0.1, batch_size=50, n_epoch=11	0.7131	15.713 seconds
lr=0.1, batch_size=250, n_epoch=55	0.7190	55.403 seconds
lr=0.1, batch_size=500, n_epoch=111	0.7032	112.756 seconds
lr=0.1, batch_size=1000, n_epoch=222	0.6892	229.354 seconds
lr=0.05, batch_size=50, n_epoch=11	0.7221	16.440 seconds
lr=0.05, batch_size=250, n_epoch=55	0.7147	56.791 seconds
lr=0.05, batch_size=500, n_epoch=111	0.7143	112.849 seconds
lr=0.05, batch_size=1000, n_epoch=222	0.7015	229.624 seconds
lr=0.01, batch_size=50, n_epoch=11	0.7228	15.827 seconds
lr=0.01, batch_size=250, n_epoch=55	0.7259	56.103 seconds

lr=0.01, batch_size=500, n_epoch=111	0.7181	113.334 seconds
lr=0.01, batch_size=1000, n_epoch=222	0.7109	230.593 seconds
lr=0.001, batch_size=50, n_epoch=11	0.6900	15.611 seconds
lr=0.001, batch_size=250, n_epoch=55	0.7365	56.224 seconds
lr=0.001, batch_size=500, n_epoch=111	0.7334	113.483 seconds
lr=0.001, batch_size=1000, n_epoch=222	0.7358	230.147 seconds

Discussion: In section 4.2 and 4.3 we chose the parameters that lead to lowest loss and most stable loss line by hand. This may be a good way to find a single parameter but not really correct for multiple circumstances when we need to find a combination of parameters.

The result of the grid search method shows that learning rate 0.001 leads to higher loss but when combined with batch_size of 250 and 55 epochs, it gives better results and acceptable run time. We suggest that authors should increase the number of epochs to ensure convergence.

6. Conclusion and Perspectives

In this project, we were able to analyze the dataset of EEG data that shows driver drowsiness while driving. We dive into 3 basic hyper-parameters of the CNN model: learning rate, batch_size and number of epochs. Grid Search evidently shows us the score for each combination of hyper-parameters and helps us achieve 0.7365 accuracy over 0.7191 of Jian et al.'s baseline. In future work, we want to dive deeper into scoring and CNN model's structure to achieve better results.

Acknowledgement

References

[1] F. Sagberg, P. Jackson, H.-P. Kr¨uger, A. Muzet, and A. J. Williams, Fatigue, Sleepiness and Reduced Alertness as Risk Factors in Driving.Oslo: Institute of Transport Economics, 2004.

[2] Bergstra, J.; Bardenet, R.; Bengio, Y.; Kégl, B. (2011b). Algorithms for hyper-parameter optimization. In

[3] Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F. C. N., and Weinberger, K. Q., editors, NIPS, 2546–2554, 2011.
[4] Bergstra, J.; Komer, B.; Eliasmith, C.; Yamins, D.; Cox, D. D. (2015). Hyperopt: a Python library for model selection and hyperparameter optimization Computational Science and Discovery, 8(1), 014008,2015.
[5] Bergstra, J.; Yamins, D.; Cox, D. D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML'13, JMLR.org, I–115–I–123, 2013.
[6] Chang, C.-C.; Lin, C.-J. (2011). LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011.
[7] Domhan, T.; Springenberg, J. T.; and Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15, AAAI Press, 3460–3468, 2015.
[8] Albelwi, S.; Mahmood, A. (2017). A framework for designing the architectures of deep convolutional neural networks. Entropy, 19(6), 2017.
[9] Andonie, R. (2019). Hyperparameter optimization in learning systems. Journal of Membrane Computing ,2019.
[10] Hinz, T.; Navarro-Guerrero, N.; Magg, S.; Wermter, S. (2018). Speeding up the hyperparameter optimization of deep convolutional neural networks. International Journal of Computational Intelligence and Applications, 17(02), 1850008, 2018.
[11] Bergstra, J.; Komer, B.; Eliasmith, C.; Yamins, D.; Cox, D. D. (2015). Hyperopt: a Python library for model selection and hyperparameter optimization. Computational Science and Discovery, 8(1), 014008,2015.
[12] Z. Cao, C.-H. Chuang, J.-K. King, and C.-T. Lin, "Multi-channel EEG recordings during a sustained-attention driving task," *Scientific Data*, vol. 6, no. 1, Apr. 2019.
[13] Jian Cui, Zirui Lan, Yisi Liu, Ruilin Li, Fan Li, Olga Sourina, Wolfgang Müller-Wittig. A compact and interpretable convolutional neural network for cross-subject driver drowsiness detection from single-channel EEG.

Appendix B. Source code & Data

Data and code is available at
https://drive.google.com/drive/folders/1meyMh26cqQQwO2QMjv4IxWEGE07sUsGl