Machine Learning in Practice — Competition 1

Marieke van Vreeswijk (s1054488) Naomi Deenen (s1015921)
Daan Brugmans (s1080742)

March 15, 2024

1 Introduction

The task at hand was the participation in a Kaggle competition hosted by Jing et al. [5] called "HMS - Harmful Brain Activity Classification". Initiated by Harvard Medical School, the goal of this competition was to develop a model capable of recognizing and classifying varying types of harmful brain activity, including seizures. Currently, neurologists must manually analyze patients in order to find signs of harmful brain activity. This process requires a great amount of time, energy, and other resources. As a way to ease and hasten this process, there is an increasing interest in using a model capable of detecting and classifying harmful brain activity automatically. The Kaggle competition by Jing et al. aims to produce such a model. Jing et al. provide competitors with a dataset of electroencephalography (EEG) signals which are manually labeled by multiple professionals into one of five harmful brain activity categories and a sixth "other"-category. The model must then, given an EEG signal, predict the likelihoods of the signal belonging to any of the six categories.

2 Method

Our general method was to build upon an existing baseline, distributing tasks among team members. In Kaggle competitions, competitors often make their existing solutions publicly available as to promote participation and advancement of the task. We started with a publicly available baseline solution offered by Bisi consisting of two notebooks: one for training a model on the data provided [2], and one for inferring on an ensemble of models for submissions [3]. We expanded upon these baselines by implementing various features: expanded data normalization, class distribution rebalancing, spectrogram generation from EEG-data, model hyperparameter optimization, and learning predictions from ensembles.

2.1 Baseline

In our baseline, three models were used: ResNet34d, EfficientNetB0, and EfficientNetB1. These models were trained using the provided spectrogram data. Furthermore, for each model, 5 folds were trained to produce the best results. Each spectrogram undergoes four major steps during data preprocessing. First, missing values were set to -1. Second, values greater than e^{10} were set to e^{10}, while values smaller than e^{-6} were set to e^{-6}. Third, a log transformation was done on the spectrogram data. Finally, the data was normalized per instance.

2.2 Data Normalization

In the baseline, the data was normalized per instance. We implemented a new data normalization method, expecting that the models would better understand the differences between the spectrograms. The mean and standard deviation are calculated for the entire training set first. This is accomplished by repeating the preprocessing steps, saving the mean of all values before normalization. The mean of all spectrogram means is then calculated. This mean is then used to calculate the standard deviation of all values in all spectrograms. The data will then be normalized prior to training, with the mean and standard deviation calculated before training begins.

2.3 Class Rebalancing

The number of class labels differs for each spectrogram due to a varying amount of professionals labelling different spectrograms. First, the percentage of votes for each label is calculated and used as a factor. This distribution can be

found in figure 1. However, in the baseline notebook, they group by spectrogram id. This results in the distribution shown in figure 2. 'Other' received more votes than the other labels. The data was already a bit unbalanced, but when grouping on spectrogram id, this imbalance was magnified. This is why we tried to rebalance the data while training the baseline model. The training data was balanced, but the validation data was not. The reason for this is that we did not know how the Kaggle test set was distributed.

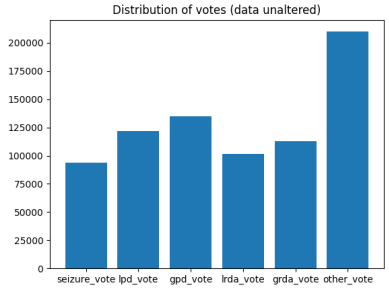

Figure 1: Distribution of votes in original dataset

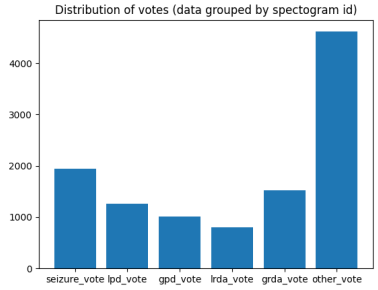

Figure 2: Distribution of votes in grouped dataset

2.4 Spectrogram Generation

On the competition forum, a discussion arose about a formula for creating spectrograms that achieved a better score than the provided Kaggle spectrograms [4]. We implemented this formula to create new spectrograms as input for our EfficientNets. This was done with the signal library of SciPy [7]. To get the right output sizes, which are close to the input size of the network, different values in this function were tried out: length of each segment, number of points to overlap between segments and length of FFT used. Additionally, denoising of the EEG data was tried out before creating the spectrograms, as denoising has been found to improve spectrogram quality [6]. Our denoising was implemented as in the notebook by Yusaku5739 [8]. To compare creating spectrograms using this formula from regular vs. denoised EEG data, for both models, one fold was trained, and the KL-divergence was calculated on the test set. The training of the model by using the new formula spectrograms can be found through this link.

2.5 Hyperparameter Optimization

We implemented a hyperparameter optimization feature for the models we trained. This was done using the Optuna framework [1]. In Optuna, one defines an objective function that should be optimized, the set of hyperparameters that may be tuned in order to optimize the objective function, and the search space for the hyperparameters that Optuna should explore. Optuna will then handle the optimization by itself by running a pre-specified number of trials. Every trial runs the objective function a single time with a new set of hyperparameter values. For our implementation, we take the baseline's k-fold train loop as the objective, using 2 folds and 2 epochs per fold. We optimize for the validation loss. Our implementation uses a pruning strategy where subsequent trials that perform worse than previous ones are stopped early after the first epoch. In batches of 20 trials, we iteratively constrained the search space of the hyperparameters manually for each round of optimization. We considered seven existing hyperparameters to optimize: EEG-signal lowpass, EEG-signal highpass, learning rate, optimizer betas (1 + 2), optimizer weight decay, and optimizer scheduler. We also added two new hyperparameters: dropout rate and normalization strategy (per instance vs. our method). Our implementation of the hyperparameter optimization feature may be found at this link.

2.6 Learning Ensemble Predictions

Stacking in ensemble learning involves combining predictions from multiple models to improve overall performance. It works by training diverse base models on the training data and then using their predictions as features to train a meta-model. This two-level architecture aims to reduce bias and variance while capturing complex relationships in the data. Various configurations have been experimented with in stacking ensemble learning. Initially, only the predictions of the base models were used as features for training the meta-model. Subsequently, different approaches were explored, including concatenating the prediction vector with statistical measures such as the mean, standard deviation, minimum, maximum, and median derived from the predictions. The meta-model is an MLP implemented in PyTorch, trained using both Cross Entropy (CE) Loss and KL-divergence Loss. The training process involves

splitting the data into training and validation sets, iterating through epochs, and computing both CE and KL losses for each batch. These losses are combined with a weighting factor and optimized using the Adam optimizer. The final output layer comprises six neurons representing class labels, with a Softmax activation function to ensure a normalized output distribution of the classes. The model's performance is evaluated on the validation set, with parameters saved if improvements are observed. If the model does not improve for 20 epochs, training is terminated. This approach ensures the model learns to minimize both CE and KL losses, and prevents overfitting by saving the best model based on validation data. Yet, it does have to be taken into account that the base models were trained on the meta-model validation data, introducing a data leakage. Code can be found at this link.

3 Results

Our final inference notebook containing our best results can be found at this link, with a KL-divergence of 0.34.

3.1 Data Normalization

To determine whether our *datawide* normalization strategy outperformed per-instance normalization, we compared the baseline 5-fold EfficientNetB1 model to a 5-fold EfficientNetB1 model trained with datawide normalization. EfficientNetB1 with 5 folds from the baseline notebook was tested on the Kaggle test set, and the KL-divergence was 0.46. When using the newly trained EfficientNetB1 with 5 folds and datawide normalization, the KL-divergence was 0.41. Next to that, the baseline ensemble network gave a KL-divergence of 0.41. With this new EfficientnetB1 model, the total ensemble became 0.39.

3.2 Class Rebalancing

As previously stated, the baseline 5-fold EfficientNetB1 achieved a KL-divergence of 0.46. When the same model was trained again using class rebalancing, the KL-divergence on the test set rose to 0.53.

3.3 Spectrogram Generation

During model training for our spectrogram generation experiment, we analyzed two loss curves: one for training on denoised spectrograms and one for training on regular spectrograms, both using the new formula. The training of the model with spectrograms created with this new formula, started with a validation loss around 0.65 and ended up at a validation loss around 0.4. For the training of the model with spectrograms created with this new formula, but using denoised EEG-data, the training started around 0.7, but ended up around 0.6. From these loss curves, we deduced that the model which did not use denoising, seems to be learning more of the spectrograms, so that is why we continued with that model and trained it with 5 folds. We ran the ensemble using this model on the test set, resulting in a KL-divergence of 0.44. However, when we ran an ensemble with this model, the hyperparameter-optimized model, and the baseline EfficientNetB0 model, we achieved a KL-divergence of 0.34; a major improvement.

3.4 Hyperparameter Optimization

	Lowpass	**Highpass**	**Learning Rate**	**Scheduler**	**Dropout**	**Normalization**
Optimized	e^{10}	e^{-6}	0.00137	CosineAnnealingLR	18.42%	Datawide
Original	e^{10}	e^{-6}	0.00100	CosineAnnealingLR	0.0%	Instance

Table 1: Table of hyperparameter values for the EfficientNetB1 model. The top row shows the best values that we could find, the bottom row the original values.

To limit our experiment's scope, we only optimized for the EfficientNetB1 model. The set of hyperparameter values we found to produce the lowest validation loss during optimization is shown in table 1. Initial experiments showed that optimizer betas and weight decay rate had little to no influence on the optimization process and were thus kept the same. Lowpass and highpass values for the EEG signals were included in the search space for the first few rounds of optimization, but were later kept unaltered as to measure the influence of the other hyperparameters on optimization better. We found that the baseline was already well-optimized, as our optimization could only improve it slightly. With our optimized model, we were able to reduce ensemble KL-divergence on the test set from

0.41 to 0.39, a slight improvement. We attribute this improvement to the two hyperparameters we introduced to the optimization process: a dropout rate and normalizing *datawide*.

3.5 Learning Ensemble Predictions

Interestingly, the most effective configuration was found to be one where the meta-model took as input not the actual predictions of the base models, but rather the statistical measures derived from these predictions. Specifically, the meta-model utilized the minimum, maximum, mean, standard deviation, and median of the predictions as its input features. However, this approach did not yield an improvement over simply averaging the predictions of all base models for ensemble prediction. Surprisingly, when averaging the predictions post-data collection from the base models, the resulting train KL-divergence was 0.65, indicating worse performance compared to the test data. This discrepancy suggests potential issues with the training data used for the meta-model.

4 Discussion

Results of our experiments are mixed: while some were clear improvements upon the baseline, some seemed to bear no fruit. In particular, the generation of additional spectrograms from existing EEG-data as an additional training data source proved to improve ensemble performance remarkably. Our efforts to enhance ensemble performance, including the implementation of a new data normalization method and optimization of hyperparameters, appeared to yield positive results. However, despite these improvements, employing the meta-model to learn ensemble predictions did not lead to better outcomes, and our experiment with class rebalancing notably worsened the baseline performance.

We expected that training on the extra generated spectrogram data would improve results. Our ensemble comprises of multiple deep, complex CNNs capable of capturing much latent information. Fittingly, the spectrograms upon which our models trained may contain much latent information. We postulate that, by providing the models with more (and more varied) data, they were better capable of capturing and leveraging the data's complexities, improving performance.

The results of our data normalization strategy and hyperparameter optimization also corroborate initial expectations. We propose that the improvements caused by our datawide normalization strategy lie in homogeneity of data: since all data is normalized identically, our models can better compare actual differences between data points, irrespective of instance-specific normalization, allowing for better learning. We were not surprised that our hyperparameter optimization experiment only improved ensemble performance slightly, since we knew that the baseline was likely already well-optimized. Aside from datawide normalization, we attribute the slight improvement to the introduction of dropout during model training.

Nevertheless, using the meta-model to learn ensemble predictions did not yield improved results. We suspect that this lack of improvement could be attributed to potential inaccuracies or errors in the collection of training data for the meta-model. Even taking the mean from this data resulted in inferior performance compared to the test data, suggesting the presence of discrepancies in the training data. Results of our class rebalancing experiment went against initial expectations. We expected that the class imbalance would negatively impact the models' ability to learn. However, it seems that our models learned better on the more imbalanced distribution. We expect that the reason for this phenomenon is that the skewed balance more accurately represents the population distribution.

5 Conclusion

We partook in the Kaggle competition "HMS - Harmful Brain Activity Classification". We expanded upon an existing baseline by implementing varying features. Among these, generating new training data, implementing a novel normalization strategy, and optimizing model hyperparameters were particularly effective. As a result, we were able to achieve a KL-divergence score of 0.34.

6 Author Contributions

For this project, we worked individually on realizing various features. We started by brainstorming potential baseline improvements, then collectively assigned one feature per person, and worked on them individually. Due to this approach, it is very clear who realized what. The following list notes which features mentioned in this report were implemented by whom. Additionally, those who implemented a feature also wrote the part of this report regarding that feature, meaning that we all contributed to the final products.

- **Marieke van Vreeswijk**: Learning Ensemble Predictions
- **Naomi Deenen**: Data Normalization, Class Rebalancing, Spectrogram Generation
- **Daan Brugmans**: Hyperparameter Optimization

7 Evaluation of the Process

Looking back, the beginning of this project was a little difficult. We think this is because of the difficulties we had with learning to navigate through and properly use the Kaggle platform. Kaggle was new to all three of us, so it took some time to adjust to that way of working. Next, we had to become familiar with this specific data, the EEG data, and the baseline notebook we used. That took quite some time, but it was okay and fun. Initially, we proposed many ideas, but not all were feasible due to a lack of pre-trained networks or insufficient data. Deciding which features we would try to implement, and which to leave be, also took some time. When we finally realized our first handful of working features, we swiftly found more motivation to continue our efforts. One significant factor slowed us down, however: the baseline notebook had five folds. So, to achieve better results, we also wanted to use five folds. This is why, while we could use one fold for small tests, five folds were required for final model training to determine whether it was truly better. Training 5 folds of EfficientNet models was very costly on time, and trying out different things became difficult as a result. We take this experience to heart and see a potential improvement for the next project. We now have a better understanding of Kaggle, including the notebooks and submissions. We will be able to begin implementing things earlier, because we will have a better understanding of Kaggle and more time to implement and test various ideas. Nonetheless, we are very pleased with how we worked and divided our workload. We all worked on separate features, but were able to use each other's work quite well.

8 Evaluation of the Supervision

We evaluate the supervision provided by the course organizers as positive. We did not feel the need to ask for supervision more than a handful of times, since, in our experience, the course was organized well. It was clear what the goal of the project was, how we would be graded on it, and what foreknowledge was required. We were provided with many tips and suggestions in order to get us into a good position to start realizing features swiftly, and even were offered a lecture helping us understand the domain we would be working in. Supervision was available throughout all phases of the project, even offered when not explicitly asked for. The only thing that we would have liked to received more help for, was the setup on Kaggle: since all of us had no prior experience with the Kaggle platform, and the structure of the data we worked with was quite complex irrespective of platform, it took us quite a while to start implementing features properly. We would have liked an opportunity to learn about the Kaggle platform some more prior to starting the actual project, either through (part of) a lecture, or through an organized tutorial session.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[2] Andreas Bisi. Hms: Train efficientnetb1, 2024. URL https://www.kaggle.com/code/andreasbis/hms-train-efficientnetb1.

[3] Andreas Bisi. Hms: Inference (lb: 0.41), 2024. URL https://www.kaggle.com/code/andreasbis/hms-inference-lb-0-41.

[4] Chris Deotte. Magic formula to convert eeg to spectrograms!, 2024. URL https://www.kaggle.com/competitions/hms-harmful-brain-activity-classification/discussion/469760.

[5] Jin Jing, Zhen Lin, Chaoqi Yang, Ashley Chow, Sohier Dane, Jimeng Sun, and M. Brandon Westover. Hms - harmful brain activity classification, 2024. URL https://kaggle.com/competitions/hms-harmful-brain-activity-classification.

[6] Md. Mamun, Mahmoud Al-Kadi, and Mohd. Marufuzzaman. Effectiveness of wavelet denoising on electroencephalogram signals. *Journal of Applied Research and Technology*, 11(1):156–160, 2013. ISSN 1665-6423. doi: https://doi.org/10.1016/S1665-6423(13)71524-4. URL https://www.sciencedirect.com/science/article/pii/S1665642313715244.

[7] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[8] Yusaku5739. Eeg signal denosing using wavelet transform, 2024. URL https://www.kaggle.com/code/yusaku5739/eeg-signal-denosing-using-wavelet-transform.