Epilepsy Seizure Forecasting from EEG signal using CNN

(MP 062 Kilke AI)

Ossi Arasalo (424709) Tuomas Ylipiha (352813)

***Abstract*—This report investigates the possibility of utilizing Convolutional Neural Networks (CNN) for EEG signal analysis. More specifically, the problem investigated is Epilepsy seizure forecasting from EEG signals. Seizure prediction is an application with interesting tangible real world benefits. This report experiments various CNN architectures for the seizure prediction task, focusing on end-to-end solutions without traditional signal processing methods.**

I. Introduction

Epilepsy seizure forecasting attempts to predict an increased risk of seizure before the beginning of the actual seizure. According to the American Epilepsy Society Seizure Prediction Challenge [1], the brain activity of an epilepsy patient can be divided into four states: Interictal (baseline, no seizure incoming), Preictal (before the seizure), Ictal (during the seizure), and Post-Ica (after the seizure). The purpose of the Seizure Prediction Challenge is to distinguish whether a brain activity was measured during Interictal or Preictal period. The brain activity is provided as 16 electrode EEG signals, either from either Interictal or Preictal period.

The analysis of raw signals, such as EEG signal, often relies on transforming the signal into frequency domain using Fourier transform or Wavelet transform. Focusing on the feature preparation process would normally be a natural first step for problems of this nature, however, we wanted to mainly focus on end-to-end educating ourselves on deep learning approaches and chose Convolutional Neural Networks (CNN) for this task. CNN has shown state-of-the art results for image recognition tasks, which results from its capability to interpret patterns from raw signal. EEG signal is a one dimensional signal containing multiple channels (electrodes), as opposed to two dimensions and three channels of RGB images. Another possibility would've been to utilize Recurrent Neural Networks (RNN), which are a good choice for sequential data, i.e signals over time.

II. Related work

There are multiple papers describing various Deep Learning methods for EEG data. Our models was mostly based on ideas found on [2]. Their implementation is purely based on CNN similarly to ours and the data used is for seizure prediction as well. There is also a good thesis [3] analysing the use of RNN for the seizure prediction task.

The winning solution for this particular Kaggle competition [4] did not utilize Deep Learning in their model. Their work focuses on feature extraction using various signal processing techniques and ensembles multiple base models into the final predictions. Their AUC score was a stunning 0.90315 for the public leaderboard of the competition.

III. Data

The data used in this report is part of a published dataset for Kaggle competition *American Epilepsy Society Seizure Prediction Challenge* [1]. The competition provides multiple datasets, out of which a single dataset titled *"Dog_1.tar.gz"* was analyzed in this report. The dataset contains 480 interictal, 24 preictal and 502 test segments. Each segment contains 10 minutes of EEG samples, sampled at frequency 399.6098 and therefore having a dimensionality of $d = 239766$. There are in total 16 electrodes, which leads to a tremendous total dimensionality of 3836256. Approximately one second samples from both Preictal and Interictal segments can be seen in Figure 1. As seen in Figure 1, preictal and interictal signals can already be distinguished on fairly high frequencies.

Instead of creating predictions for each segment as a whole, we split the segments into sub-segments and create predictions for the sub-segments individually. By splitting the segments, we were able to increase the amount of training samples and reduce the size of the model which made the utilization of CNN feasible. Various sub-segment sizes are tried and reported further in this paper, however, sub-segment sizes between 80 and 512 were tried here. The predictions for the whole segments can later be made by averaging the sub-segment predictions.

In order to enhance convergence, we normalized the signals by subtracting the features by the training set mean and dividing by its standard deviation:

$$\hat{y} = \frac{y - \overline{y}}{\sigma_y} \tag{1}$$

The data is highly imbalanced as the ratio between Interictal and Preictal segments is 20 to 1. The AUC evaluation metric did not suffer from the imbalance during the evaluation, however, during the training the standard cross entropy failed to distinguish between the two classes. For this reason we oversampled the Preictal segments in order to balance the datasets for training. However, we preserved the original ratio for the evaluation set for more comparable results.

IV. Method

We chose to use convolutional neural networks to solve this classification task. Our architecture can be seen in Figure 2.

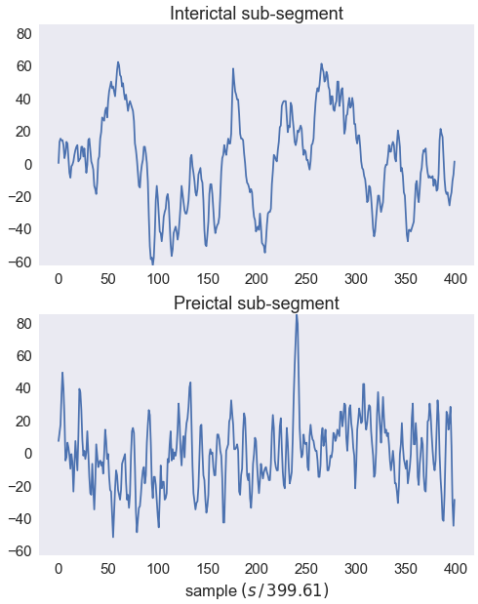


Fig. 1: Visualization of one second of Preictal and Interictal EEG signals

Convolutional neural network consists of convolutional layers that learn to find patterns from the data by creating filters. In our case, layers might learn to find different trendlines. Some filter might be able to detect when the signal is going up, and some other might learn to find the peak values of the signal. Parameter sharing is used to limit the number of weights and to apply the same filter for each part of a single data sample. This is the foundation of a convolutional layer. An operation called convolution operation is performed for each filter to obtain the results discussed previously in this chapter.

There are number of hyperparameters for each convolutional layer that need to tuned. We have to specify the receptive field, that is how large we want our filters to be. In our best model, the receptive field for the first convolutional layer is $(10, 1)$ and the amount of input channels is 4, which means that each of our filters "look" at 10 timesteps of 1D time signal and 4 electrodes at once. Another important hyperparameter is the stride size, that is essentially how we move our receptive field. For instance we use stride of 1 which means that we move our receptive field 1 timestep forward each time. Padding also need to be specified. This means that how we deal with values around the borders of a sample. We are using $VALID$ padding in all of our layers, which means that we drop out the values close to the borders of a data sample. We also have to specify how many filters do we have for each layer.

After the convolutional layer, we do batch normalization. The idea behind batch normalization is to decrease internal covariance shift that is decrease how much our hidden units can change over different batches [5]. Essentially we are scaling and adjusting the the activations by this formula:

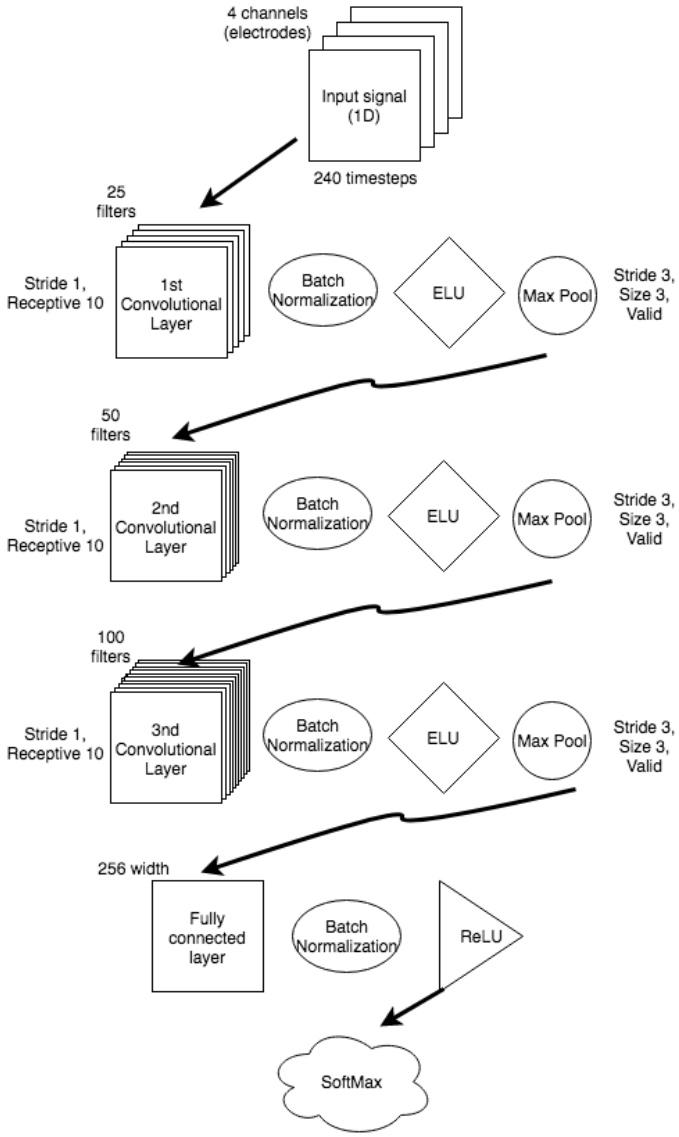


Fig. 2: Model 3: Architecture of our primary model.

$$\frac{\gamma(x-\mu)}{\sigma}+\beta$$

where

$$\gamma = \text{scale}$$
$$\beta = \text{offset}$$
$$x = \text{input (weights)}$$
$$\mu = \text{mean}$$
$$\sigma = \text{variance}$$

Batch normalization can make the learning faster and allow us to use larger learning rates [5].

After each layer, we have to apply a activation function to make the weights non linear for the next layer. We are using ELU, exponential linear units for activations between

convolutional layer and pooling layer, that will be covered shortly. ELU has been shown to perform better than RELU (rectified linear unit) by speeding up the learning [6]. Ther formula for ELU is descriped below:

$$\begin{cases} x, x > 0 \\ \alpha(\exp(x) - 1), x \leq 0 \end{cases}$$

where α determines where the negative values are saturated [6].

ELU is especially fast to calculate, because it's derivate is simple, which makes it suitable for backpropagation. Derivative can be seen below.

$$\begin{cases} 1, x > 0 \\ \alpha(\exp(x) - 1) + \alpha, x \leq 0 \end{cases}$$

After the activation is applied, we do max pooling. The idea behind pooling is to reduce the dimensionality or our output from a convolutional layer. We are using max pooling so we choose the maximum value from a set of weights. There are 3 hyperparameters that controls the behaviour. We specify the pool size, that determines the window size from which we choose to maximum value. The other hyperparamter is stride that operates the same as in convolutional layer. The last one is padding which is the same is in convolutional layer. We are using $VALID$ padding in all our cases.

Softmax cross entropy was used as our loss function. Cross entropy compares the distribution predicted by the model against the the actual distribution.

$$H(y, p) = -\sum_i y_i log(p_i)$$

Where y_i is the label and p_i the predicted label (probability). And the derivative with softmax is [7]:

$$\frac{\partial L}{\partial o_i} = p_i - y_i$$

Adam was used as an optimization function. Adam combines two different optimization methods: RMSProp and AdaGrad. Basically Adam adjusts the learining rate dynamically by calculating the moving average [8]. Adam has 4 hyperparameters, learning rate, β_1 that is the decay rate for the first moment, β_2 that is the decay rate for the second moment and ϵ that is a small value used for numerical stability [8].

V. Experiments

Our primary model consisted of a simple CNN architecture, described in the Method section. We tried multiple variations for this model (Model 1, Model 2, Model 3). In our experiments, we focused on finding the sufficient time window and experimenting how the amount of electrodes improves the results. For regularization, we resulted in early stopping and relying on batch normalization as it does produce some level of regularizaiton.

Model 1 used single electrode input with 80 time step subsegments. This was our starting point where we tried to create a simplest possible model for the task (the model turned out to be insufficiently simple). The fully-connected layer used 1024 nodes, which was probably a cause of heavy overfit.

After Model 1, we created Model 2 by adding more depth to the model with a third convolutional layer and increasing the time dimension to 240 for each sub-segment. This improved the results significantly, but also increased the memory consumption of the model. We also reduced the fully-connected layer width to 256 nodes which seemed a reasonable choice due to large amount of overfitting.

Model 3 included additional electrodes by using in total four electrodes. Otherwise the model architecture was kept the same. In all the cases we used early stopping to avoid overfitting as much as possible. We also compared the convergence of ReLU and ELU for this model, see Figure 4.

We tried two alternative models as well. One promising approach was to perform a wavelet transform on the dataset for a single electrode and then use convolutional neural network to do the classification. Wavelet has been previously used succesfully for EEG classification [9].

We tried to tune the network by changing the number of filters per layer, removing and adding new convolutional layers and changing the respective field. Different max pooling sizes were also eperimented mainly ranging between $1-4$ for both axis. Fully connected part of the netwok was also tuned by making it more deep and adding more nodes. Adam was used for training and different learning rates ranging from $1e-1$ to $1e-9$ were tried.

The other approach we tried was based purely on this recent paper [2]. We copied the exact network they used and adapted it to our dataset. Model structure is defined in 6. The most notable detail in this network is that we have a temporal convolutional layer right after the input to reduce dimensionality. The difficulty with this dataset is the fact that we have data from 16 different electrodes. For comparison normal RGB image has 3 channels. The temporal convolutional layer is applied individually for each channel/electrode and the idea is to reduces the time dimensionality. Then the following convolutional layer is applied across all electrodes so the point is to find filters across all electrodes. In other words, this layer tries to find similarities between different electrodes. After the first two layers the dimensionality is significantly decreased.

The following layers are used to find smaller filters. At the end we have fully connected layer.

We tried the model with normal bias and eventually with batch normalization as suggested in the original paper [2]. ELU was used for activation. In other respects, we followed the paper quite strcitly so we didn't tune number filters, strides or receptive field sizes. However, different learning rates were tried.

VI. Results

Model 1 lacked sufficient depth and was incapable of generalizing to test data. The resulted AUC is equal to random guessing. Also, the fully-connected layer was possibly too wide and unregulated, which also reduced the generalization capabilities. Model 2 did manage to create usable predictions,

Fig. 3: Models 1, 2, 3: Visualizaiton of model capacity and convergence.

Model ID	Description	AUC (sub-segment)
Wavelet	Wavelet transformation + CNN	0.512
Schirrmeister et. al.	Architecture based on this paper [2]	0.488
Model 1	Sub-segment length 80, 1 electrode	0.509
Model 2	Sub-segment length 240, 1 electrode	0.572
Model 3	Sub-segment length 240, 4 electrodes	0.644

TABLE I: Results from various experiments, evaluated on sub-segments.

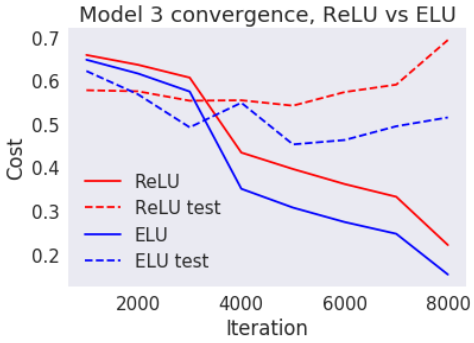


Fig. 4: Comparison of convergence between ELU and ReLU activations for the convolutional layers.

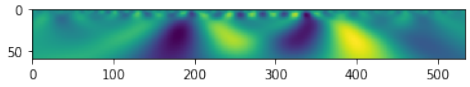


Fig. 5: Wavelet transformation for a single electrode.

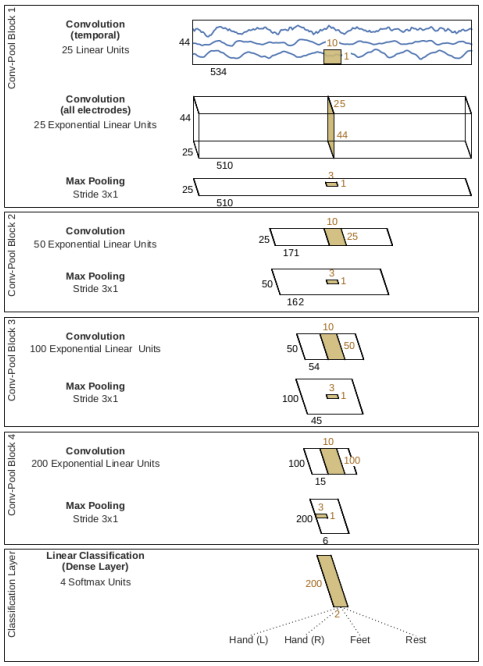


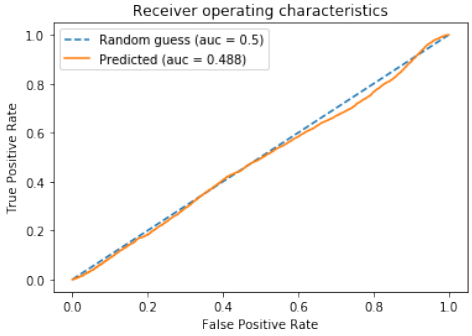
Fig. 6: Schirrmeister et. al. : Architecture visualized in the paper [2]

clearly increasing the input data dimensionality and adding depth to model improved the results to AUC 0.572. Even further increasing the complexity in Model 3 by adding more electrodes seemed to improve results as well. It seems that the electrodes are not redundant and possibly the full 16 electrodes should be used. Still, the improvement was significant even with this data of manageable dimension.

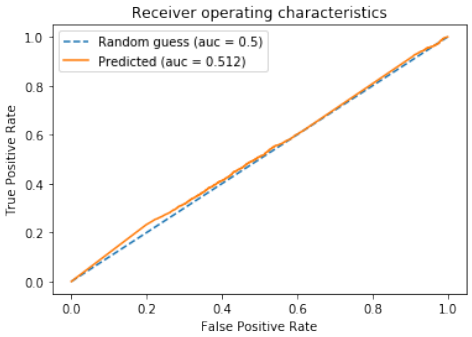
Another interesting detail was that ELU activations for the convolutional layers seemed to outperform ReLU by converging to more optimal minima as well as converging faster, see 4.

Preprocessing the data with wavelet transform did not produce any good results as can be seen from the table. AUC curve is nearly a straight line as in figure 7b so the predictive power of this approach is nearly useless. Loss convergences rapidly and increasing the number of epochs does not seem to help at all.

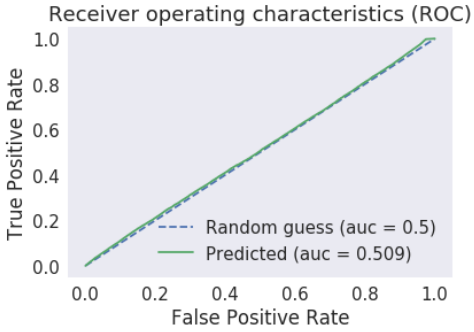
CNN architecture which was based on this recent paper [2] did not produce that good results in our experiments. The network is quite complex so it had a vast amount of hyperparameters that can be tuned. The result achieved was useless, because the AUC is below 0.5 as in the plot 7a. Changing the learning rate or penalizing the loss function for



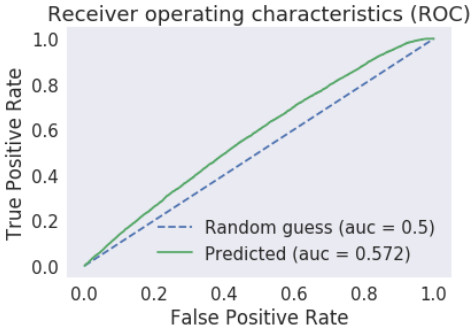
(a) Schirrmeister et. al.: ROC curve for model from the recent paper [2].



(b) Wavelet: ROC curve for CNN with wavelet transform.

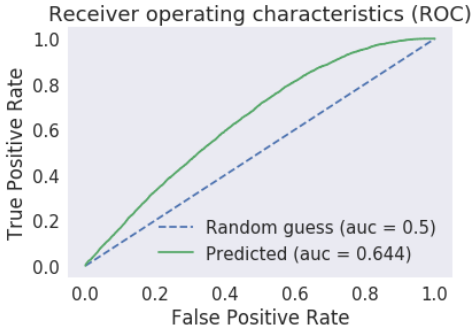


(c) Model 1: ROC curve for Model 1

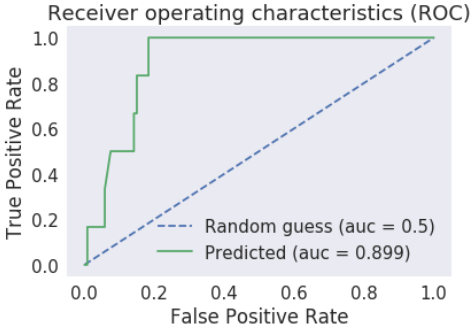


(d) Model 2: ROC curve for Model 2

Fig. 7: ROC curves of sub-segment predictions for various experiments.



(a) Model 3: Sub-segmet AUC



(b) Model 3: Full segment AUC, segment predictions averaged from sub-segement predictions.

Fig. 8: ROC curves for our best model

label imbalance did not produce any accuracy benefits.

Loss function penalization for wavelet model worked better than the original wavelet model or the architecture based on the paper. Even though AUC value was not as good as in model 3, it was similar to model 2 (0.57).

VII. Discussion

We wanted to compare the results against the public leaderboards of the mentioned American Epilepsy Society Seizure Prediction Challenge Kaggle competition [1], however, in order to submit for the competition we would've needed to make predictions for all the datasets, which totalled over 70GB in size. This turned out to be too tedious and therefore we only evaluated our results against a test dataset taken from the labeled training data.

Our experiments were mostly compared using AUC of sub-segment predictions. These sub-predictions produce presumably lower AUC compared to the AUC of the full segment predictions obtained by averaging the sub-segments, however full segment predictions are very likely quite imprecise due to low amount of samples available. Also, it can be expected that the full segment predictions correlate highly with sub-segment predictions.

The public leaderboard winner had 0.9 AUC. Approxi-

mately top 20% of the entries on the public leaderboard got above 0.7 AUC. Comparing our results to these results is not feasible due to (i) low amount of full segments in our test set (ii) we only used 'Dog 1' dataset. However, our best model had full segment AUC 0.899 and 0.644 sub-segment AUC. The full-segment AUC was averaged form the sub-segments. The sub-segment AUC is much more reliable as the amount of samples in sub-segment test set is much higher due to splitting (in the order of hundreds of thousands vs hundreds for full-segments). We made sure the test set did not contain sub-segments from segments used in training. Also, seeing how our initial models failed to predict the exactly same data at all (AUC ≈ 0.5) gives us some confidence in our model.

VIII. Conclusions

This mini-project introduced us the challenges of applying Deep Learning to real world data. Choosing between various architectures and hyperparameter choices is time consuming and non-trivial. As an example, we realized there are various options for preventing model overfitting (dropout, early stopping, weight penalty, network dimensions,..) all of which can solve the problem in a given scenario. Selecting the best strategy seems to be highly empirical and sometimes confusing.

The best strategy for creating a usable model was to first define the evaluation dataset and evaluation metrics. After this initial setup, it makes sense to start with a simple model and move towards more complex models by adding layers and input features.

As for the results, CNNs seem to be suitable for the EEG prediction task. The EEG signals carry meaningful information at sub-second wave-lengths, making CNNs with manageable sizes useful. It would be interesting to experiment RNNs as well for the same task and compare the performance of the two.

In general, CNNs are be quite heavy in terms of required RAM and having a proper GPU is needed for larger networks, especially for the state-of-the art results presented in the papers. However, interesting results can be obtained even with modest equipment by simplifying the network architectures and applying data normalization and hyperparameter tuning.

IX. Roles of the authors

References

[1] Kaggle.com. (2014) American epilepsy society seizure prediction challenge. [Online]. Available: https://www.kaggle.com/c/seizure-prediction

[2] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball, "Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human EEG," *CoRR*, vol. abs/1703.05051, 2017. [Online]. Available: http://arxiv.org/abs/1703.05051

[3] M. Larmuseau, "Epileptic seizure prediction using deep learning," 2016.

[4] A. Barachant, A. Temko, F. Li, and G. T. Junior, "Kaggle seizure prediction challenge 2016," https://github.com/alexandrebarachant/kaggle-seizure-prediction-challenge-2016, 2016.

[5] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[6] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *CoRR*, vol. abs/1511.07289, 2015. [Online]. Available: http://arxiv.org/abs/1511.07289

[7] P. Dahal, "Classification and loss evaluation - softmax and cross entropy loss," https://deepnotes.io/softmax-crossentropy, 2017.

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[9] A. Subasi, "Eeg signal classification using wavelet feature extraction and a mixture of expert model," *Expert Systems with Applications*, vol. 32, no. 4, pp. 1084–1093, 2007.