DEEP CONVOLUTIONAL AND RECURRENT NEURAL NETWORKS FOR INTERPRETABLE ANALYSIS OF EEG SLEEP STAGE SCORING

Anders Launer Baek

Technical University of Denmark
s160159@student.dtu.dk

ABSTRACT

The purpose of this paper is to investigate the time domain in automatic scoring of sleep stages by combining a recurrent neural network with a convolutional neural network. The raw polisomnography signals have been transformed into visuel interpretable images by using multitaper spectral analysis. The six different sleeping stages are represented in the transformed images with different visuel patterns. By learning those visuel sleeping pattern, it is possible to automatically classify the current stage of the sleep. The basis of this project are based upon the article by [1] and their produces has been re-created in order to create the base line. The base line are here compared to an extended model, which combines a convolutional neural network and a recurrent neural network. Due to the experimentally setup and the implementations of the networks in this project, the results are not comparable with the archived results in [1].

The performances of the two models are close to similar. The extended model does not out perform the base line model.

Index Terms— Convulutional Neural Networks, Recurrent Neural Networks, Sleep Stage Scoring, Computer Vision and Pattern Recognition.

1 Introduction

Sleep is most important part of the human health. it is possible to diagonitise serevel dieasie by analysis the sleeping pattern of a human. The current approach of annotating sleep stages is done manually by highly trained professionals and based upon complex transition with high probability of a subjective interpretion.

It is possible to find the abnormalities within the sleep and hereby recognize some dieasies, by analysis the annotated transitions between the sleeping pattern. The annotated sleep patterns can be visualized by using a hypnogram (see figure 2).

The measurements which is used to classify the sleep stages, are collected during the whole night of sleep. Several of biological signals can be measured during the sleep and the interesting signals for this project is the brain activity. The brain activity can be aquried by using electroencephalography (EEG) method. The main frequencies of the EEG signales are: $delta = 3\,[Hz]$ and below, $theta = 3.5 - 7.5\,[Hz]$, $alpha = 7.5 - 13\,[Hz]$, $beta = 13\,[Hz]$ and above.

The above mentioned burst of rhythmic components are represented in different degrees within each stage of sleep. The newest definition of sleep stages are defined by the American Academy of Sleep Medicine. They defines the sleep stages into five (six) as followed [1, 2]:

- W: wakefulness to drowsiness. The alpha and delta waves are present. The low frequency delta waves is affected by small eye movements, when the eyes switching from open to closed. See the multi-taper frequency spectrum in figure 1a.

- N1: Non-REM 1. This is the first sleep stage after the transition from W. There are slow eye movements, See the multi-taper frequency spectrum in figure 1b.

- N2: Non-REM 2. One or more K-complexes present. See the multi-taper frequency spectrum in figure 1c.

- N3-N4: Non-REM 3-4. Slow delta wave activity. The dreaming starts here. This is the stage between been fully awake and fully asleep. The newest definition combines the sleep stage N3 and N4. See the multi-taper frequency spectrum in figure 1d-1e.

- R: REM is short for rapidly eye movements. Mixed rhythmic components are present in the EEG and the brain activity is similar to W. See the multi-taper frequency spectrum in figure 1f.

The scope of this project is to create a re-implementation of the chosen CNN ([1]) in TensorFlow (TF), in order to get the base line. When the base line has been archived, the task is to implement a recurrent neural network and hereby learn the transitions rule between each of the sleeping stages. This research will hopefully give improve the sleep stage classifier, and be more profitable for patients and doctors around the world.

2 Materials and Methods

As a requirements for this project, the professor and assistant teahters needs to have acces to the running code which produces the results within this paper. The two bullets below links to what you need in order to reproduce the presented results.

- Github: Deep_Learning_Project.git
- DTU SharePoint: Data_dicts_and_Code_models.zip
- DEMO: master/DEMO

2.1 Image Creation

There have been applied multi-taper spectrum analysis in order to turn the EEG signals into images. A given image represents an epoch in seconds of the complete recorded EEG signal along the first axis. The second axis represents the spectrum for the rhythmic components of interest, mentioned above. The third axis (the color) represents the amplitudes of the rhythmic components to a given time.

The WFDB Toolbox ([3]) for Matlab have been used to download, preprocess and transform the EEG signals into images. The applied script[1] for this process has been provided by the supervisor. The multi-taper spectrum analysis which estimates the images, is not within the scope of the project. The hyperparameters, such as the duration (in $[s]$) of an epoch, number of multi-tapers, frequency resolution (in $[Hz]$), etc., within the image estimation process have been decided to remove from possible hyperparameter in this project. This ensures that the results of the baseline model, in this project, can be comparable with the main article [1] and keep the correct focus.

The Matlab toolbox are able to download the data set of interest and the data which have been used in this projects consists of PSG recordings for 20 Subjects. The Subjects have been monitored for two nights except Subject 20. There is 38211 images after the preprocessing of the EEG signals. All images have labeled-values which employs a supervised learning approach. Table 1 illustrates how the labels of 38211 images are distributed for the sleeping stages.

Sleep Stage	W	N1	N2	N3	N4	R
Dist. (in %)	12	7	46	9	6	20

Table 1: This table summerises the aggregates the distribution of the labels for all 20 Subjects. The distribution of the labels illustrates the sleep stages of subjects during the recordings.

In order to create a state of the art sleep stage classifier, WE need to consider methods to balance the six classes prior to training of the models.

[1]Git repo: "Code/2. from_edf_to_pic.m"

2.2 Neural Network Architectures

The architecture for the base line model is the VGGNet 16 [1, 4] which is a 16-layer network composed of following operations.

- The convolutional operations uses a kernel of $3x3$, which is the smallest possible configuration to capture the notion of left/right, up/down and center. The kernel has a stride of 1, which is the number of pixels the kernel slide at the time. The third hyperparameter within the convolutional operation is the padding. This operation uses padding so the spatial resolution is preserved. This layer performs several linear activations by the kernel, those activations goes through a selected non-linear, ReLU, activation function.
- The max pooling layer reports the maximum non-linear activation value within its rectangular neighborhood. The rectangular neighborhood for this network is $2x2$. The max pooling operation has a stride of 2. The spatial max pooling "decreases the resolution of image" and helps make the representation invariant to small translations of the output from the previous convolution operation [5, sec. 9].
- The last three layers of the VGGNet 16 network is fully connected. The first two have 4096 channels. The last fully connected layer performs a six-channel classification, due to the six classes in this project.
- The final layer of the VGGNet 16 network is the Softmax activation. This activation function is used to calculates the probabilities associated for each class [5, eq. 4.1].

The VGGNet 16 is an acknowledged deep convolution network suitable and well performing network for image recognition in several case studies [4, 6]. The standard input layer takes an $224x224$ RGB image.

2.2.1 Convolutional Neural Network

The baseline model for this project is given in equation 1[2].

$$\begin{aligned} net &= c_{2,64}mc_{2,128}mc_{3,256}mc_{3,512}mc_{3,512}mc_{7,f}dc_{1,f}dc_{1,o} \\ logits &= tf.squeeze(net,[1,2]) \\ probs &= tf.nn.softmax(logits) \end{aligned} \tag{1}$$

where $c_{k,l}$ is a convolutional layer, k is the number of receptive layers and l is the number of channels. $c_{1,f} = c_{1,4096}$ and $c_{1,o} = c_{1,6}$. m is the max pooling. d is a dropout operation, which keeps $p = 50\%$ of the total connection between the previous and next layer.

[2]See implementation in Git repo.: "./Code/master.py"

The TF implementation of the VGGNet 16 is heavily inspired by [7]. There have been added few modifications to the compared to the initial VGGNet 16 [4] in this implementation. The implementation of the VVG16net has transformed the fully connected layers into convolutional 2D layers. It works identically if you change size of the kernel. But then it is necessary to the squeeze the second last layer ($c_{1,o}$) into a 2D-tensor before applying the Softmax activation. Beside this, there have been included L2-regularization, with a weighted decay of 0.0005, between the convolutional operations and included dropouts between the fully connected layers. The modifications are included in eq. 1.

The L2-regularization and dropout operations prevents overfitting in this deep complex network.

2.2.2 Recurrent Neural Network

It is possible to understand sleep as a sequence of different stages. By learning the transitions between those stages may improve the classification of the sleep stages. Recurrent neural networks (RNN) is the approach to model sequences and the most effective RNNs is called gated RNNs [5, sec. 10.10]. The long short-term memory (LSTM) network is a model which comprehend the issues of vanishing or exploding gradients over time. The LSTM cell has a internal recurrence (self-loop), which produces paths where the gradient can flow for long durations.

The main reason for implementing a sequential network is to learn the transitions rules between the sleeping stages.

The first approach to learn the transitions rules, is to modify the baseline model and use its extracted feature as input for the LSTM. A second approach could be to use several LSTM cells with regularization in between. A third approach could be to redefine the problem, discritize the EEG-signals and feed the LSTM with a 1D-tensor.

Due to limited time only the first approach have been considered. The network which combines the CNN and LSTM cell will further on be mentioned as the RNN.

There are several approaches to feed the LSTM cell. The chosen approach has been inspired by [8, 9], where the the LSTM acting on each of the generated feature maps. Equation 2[3]. illustrates the implementation of the RNN. The first convolutional layers are similar to the VGGNet 16 model. The LSTM cell is feed by the final convolutional layer. The input shape to the LSTM cell is static, which can cause discarding of the final mini-batch.

$$\begin{aligned} net &= c_{2,64}mc_{2,128}mc_{3,256}mc_{3,512}mc_{3,512}mlstm_{7,512}d_pfc_0 \\ logits &= tf.reshape(net, [batch, -1]) \\ logits &= slim.layers.fully_{c}onnected(logits, 6) \\ probs &= tf.nn.softmax(logits) \end{aligned} \tag{2}$$

[3]See implementation in Git repo.: "./Code/rnn.py"

where the notations from eq. 1 are the same and the $lstm_{7,512}$ is the LSTM cell which takes the tensor shape (batch x 7 x 7 x 512). d_p is the dropout operation, which keeps $p = 50\%$ of the total connection between the previous and next layer. fc_0 is the final fully connected layer, which represents the highest order feature maps for the six classes.

The implementation of LSTM cell is heavily inspired by [10]. This implementation takes the input tensor and returns the sate tensor. Both are 4D-tensors (batch x height x width x channels).

As illustrated in eq. 2, there have been apllied an dropout operation on the output tensor from the LSTM cell. Then reshaped in order to feed a fully connected layer, which produces easy separable higher-order feature representations for the six classes discriminate. The final fully connected layer is activated with the non-linear Softmax function.

2.3 Network Visualization

There exits many of ways visualizing a neural network w.r.t. its input data in order to better understand how the network interprets different inputs. A common approach in the literature is to visualizing the activations along its propagation through the network. The example from [11] gives a great intuition about how the different operations in each layer affect the input data along is propagation though the network.

Another approach using the main article ([1]) is to calculate sensitivity maps, which determines the relative importance to every input feature j. j represents the third dimension in the image which is the values of the RGB color. The sensitivity maps is created by calculating the gradients w.r.t. the loss of an input image and its label. The mathematically expression are given in equation 3.

$$s^{(j)} = \frac{1}{N} \sum_{n=1}^{N} \left| \frac{\partial L\left(f\left(x\right), t\right)}{\partial x^{(j)}} \right|_{x=x_n} \tag{3}$$

where L is the loss w.r.t. the input image $f(x)$ and its label t.

By applying the function $s^{(j)}$ for an image, it is possible to see the relative importance in determined by the network for each feature j. Figure 1g-1r illustrates the sensitivity maps for each sleep stage for the CNN and RNN respectively. The Sensitivity maps have been created for subject 1.

2.4 Hyperparameter

Parameter	Value

Table 2: adas

dropout learning rate weights_regularizer=slim.l2_regularizer(weight_d

Grid searh

3 Experimental Evaluation

3.1 Setup

The setup from training the baseline model and the RNN model is identically. Both model are based upon layers from the VGGNet 16 model. This model have already been trained on ILSVRC data set for several weeks and it is beneficial to applied those archived weights and using the principle of transfer learning instead of the learning all the weights from scratch.

According to the literature [12], there are following approaches to do transfer-learning: Remove the final fully connected layer of the network. Use the pre-trained CNN as an feature extractor for the new fully connected layer, which fits the data set according to number of classes. The second strategy is carry out the first strategy and fine-tune some of the weights in the pre-trained CNN. The chosen strategy in this setup was to remove and create a new final fully connected layer, which fits the data set for both networks. Instead of fine-tuning the weights in the other fully connected layers, it has been chosen to train the dropout operations between those layers. This will prevent overfitting. The weights in the LSTM cell have been trained from scratch. The weights in the forget bias gate have been initialized to 1, which causes the LSTM cell does not have any prior knowledge.

This strategy is different compared to the setup in [1]. They re-train the all the weights in the last three fully connected layers.

During the training process, the networks have only access to the current epoch. This is different from the setup in [1], where the network uses two prior epochs and two posterior epochs in order to learn the current epoch. The reason for this choice was, real time considerations. If the classifier should work in real time, then it does not have any posterior epochs to work with. It is possible to use prior epochs in real time classifications, and may be considered as future research.

Both networks have been optimized in order to minimize their categorical cross-entropy. Their loss function is handled by the AdamOptimizer provided by TF iteration over mini-batches of 32. The learning rate and its decay rate are given in table 2.

scripts are set up to applied creoss validation...

3.2 Results

3.2.1 Performance

3.2.2 sensitivity

3.2.3 Hypnograms

give examples where it totally wrong?

figure 2c around t_{800}... something wired is going on in the RNN

4 Discussion

data targets have not been cross checked by the skill professionals..

how to encounter imblanced data classe:

- random down sample in each epoch.. It can remove useful imformation - use weigthingPenalized Models the the loss function - learn the distribution of the miniorty classes and up sample those his change is called sampling your dataset and there are two main methods that you can use to even-up the classes:

under sampling .. smide information vk... mske kan det give en bedre predectering???

You can add copies of instances from the under-represented class called over-sampling (or more formally sampling with replacement), or You can delete instances from the over-represented class, called under-sampling.

i did not subtract the mean picture in traning... not normalize -¿ failure

batch normalization,

bat size... low -¿ fewer discarding images in the final batch

The CNN reducing spectral variance, this can be an issue for the LSTM...

Issues with the current approach— video and spech are very different

Although using similar architecture, the type of input data we use and the ones in video prediction are quite different. Basically, if you compare one video frame with the next one, it is very likely that you have almost the same image with few changes (displacements) in it. In our case, though, two consecutive epochs can be quite different (in terms of the position of objects), and I guess that the approach proposed for video frames will not work. Notice that videos have 2 spatial dimension and 1 temporal. In our case, we have 1 spatial (frequency) and 1 temporal. If you want to use video frame like approach, it might be more useful to work with EEG scalp maps, when using multi-sensors, but that is a different problem.

We overfitting to one doctor openions.

Personal

		Predicted						Normalized pred. (in %)						Per-class metric (in %)			
		W	N1	N2	N3	N4	R	W	N1	N2	N3	N4	R	Pre.	Sen.	F_1	Acc.
CNN	W	495	145	29	11	1	20	71	21	4	2	0	3	91	71	80	93
	N1	25	211	43	0	0	62	7	62	13	0	0	18	43	62	51	89
	N2	4	51	1313	104	17	68	0	3	84	7	1	4	91	84	88	90
	N3	0	2	11	164	64	0	0	1	5	68	27	0	49	68	57	93
	N4	0	0	0	54	91	0	0	0	0	37	63	0	53	63	57	96
	R	17	80	46	0	0	591	2	11	6	0	0	81	80	81	80	92
RNN	W	578	39	26	7	1	43	83	6	4	1	0	6	89	83	86	95
	N1	38	107	64	0	0	132	11	31	19	0	0	39	55	31	40	91
	N2	8	13	1314	102	28	92	1	1	84	7	2	6	90	84	87	89
	N3	3	0	18	125	95	0	1	0	7	52	39	0	43	52	47	92
	N4	0	0	1	60	84	0	0	0	1	41	58	0	40	58	48	95
	R	19	36	43	0	0	636	3	5	6	0	0	87	70	87	78	90

Table 3: My caption

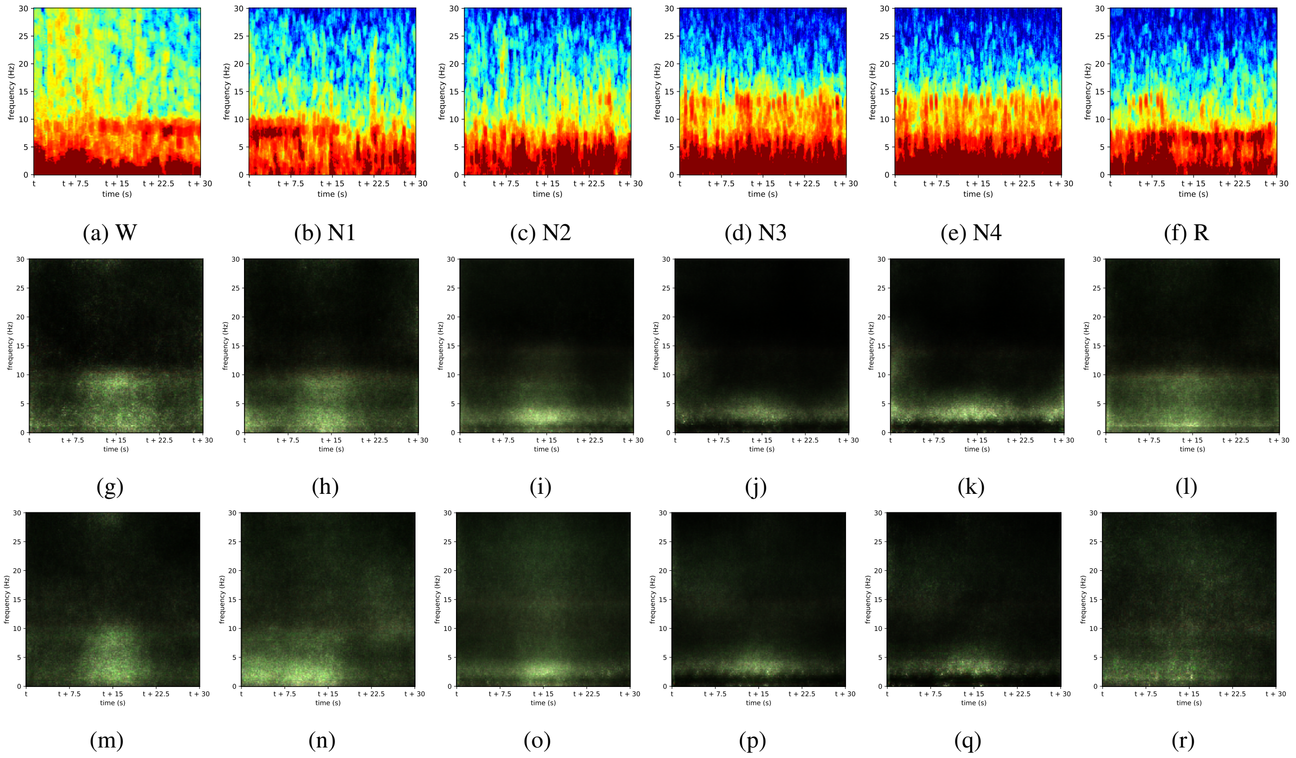
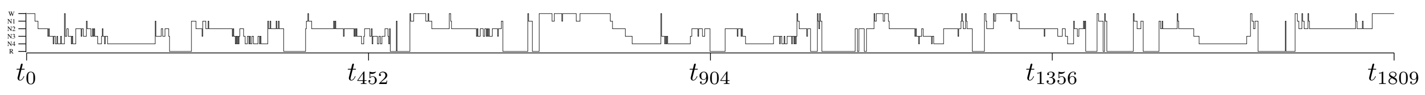


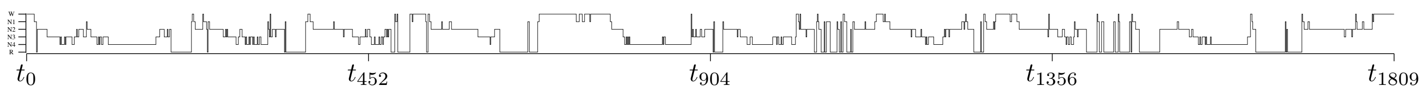
Fig. 1: This figure contains plots of each annotated sleeping stage for subject 1. The plots are given columnwise from left to right according to the previous mentioned sequence of the sleeping stages. Fig. 1a to 1f illustrates an random epoch of the multi-taper spectrum for each sleeping stage for subset 1. It is clear to see a high similarity of sleeping stage N3 and N4, fig. 1d and fig. 1e. Second and third row, fig 1g to 1r shows the sensitivity maps from the CNN and the RNN respectively.

NN	Precision	Sensitivity	F_1-score	Accuracy
CNN	65.4-**67.9**-70.4	70.9-**71.3**-71.8	67.5-**68.8**-70.0	92.3-**92.3**-92.4
RNN	61.9-**64.6**-67.2	63.2-**65.9**-68.6	61.8-**64.2**-66.6	92.2-**92.2**-92.2

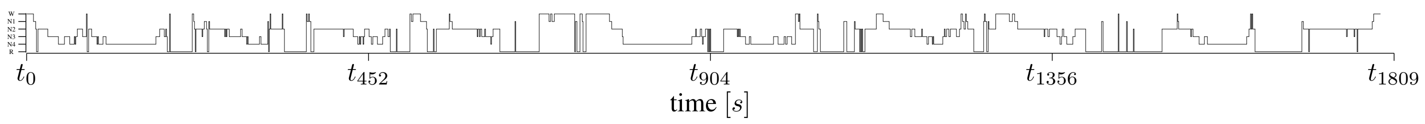
Table 4: My caption



(a)



(b)



(c)

Fig. 2: A figure with two subfigures

5 Conclusion

6 Acknowledgment

7 References

[1] A. Vilamala, K. H. Madsen, and L. K. Hansen, "Deep Convolutional Neural Networks for Interpretable Analysis of EEG Sleep Stage Scoring," *ArXiv e-prints*, Oct. 2017.

[2] Kevin McAffee, "The AASM Manual for the Scoring of Sleep and Associated Events: Version 2.0," 2017.

[3] Physionet.org, "wfdb-app-matlab," 2017.

[4] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[5] Yoshua Bengio Ian Goodfellow and Aaron Courville, "Deep learning," Book in preparation for MIT Press, 2016.

[6] Stanford CS, "CS231n: Convolutional Neural Networks for Visual Recognition - Conv," 2017.

[7] The TensorFlow Authors, "Contains model definitions for versions of the Oxford VGG network.,master/research/slim/nets/vgg.py, Git SHA: cbb6247," 2017.

[8] Hrayr Harutyunyan and Hrant Khachatrian, "Combining CNN and RNN for spoken language identification," 2016.

[9] Tara Sainath, Oriol Vinyals, Andrew Senior, and Hasim Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *ICASSP*, 2015.

[10] The TensorFlow Authors, "Convolutional LSTM implementation.,master/research/video_prediction/lstm_ops.py, Git SHA: f87a58c," 2017.

[11] Stanford CS, "CS231n: Convolutional Neural Networks for Visual Recognition," 2017.

[12] Stanford CS, "CS231n: Convolutional Neural Networks for Visual Recognition - Transfer learning," 2017.