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 project is to investigate the time domain in the automatic scoring of sleep stages by combining a convolutional neural network (CNN) with a recurrent neural network (RNN). The raw electroencephalographic signals have been transformed into visual interpretable images by using multi-taper spectral analysis. The six different sleep stages are represented in the transformed images with different visuel patterns. By learning visual sleep patterns, it is possible to automatically classify the current stage of the sleep. The foundation of this project is based upon the article by A. Vilamala et al. (2017) [1]. Their procedures have been re-created in order to create the baseline. The baseline is then compared to a modified network, which combines a CNN and a RNN. Due to the experimental setup and the network implementations in this project, the achieved results are not fully comparable with [1].

There has been applied bootstrapping in order to find the average performance metrics of the two networks. However, the extended model does not out perform the baseline model.

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

1 Introduction

Sleep is the most important part of the human health. It is possible to diagnose several sleep disorders by analyzing the sleep patterns. The current approach of annotating sleep stages is done manually by highly trained professionals and based upon complex transition rules with high probability of a subjective interpretation.

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

The above mentioned bursts 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 divide the different stages of sleep into five categories [1, 2]:

- W: wakefulness to drowsiness. The alpha and delta frequencies are present. The low delta frequency is affected by small eye movements when the eyes switch from open to closed. See the multi-taper frequency spectrum in fig. 1a.
- N1: Non-REM 1. This is the first sleep stage after the transition from W. There are slow eye movements. See fig. 1b.
- N2: Non-REM 2. One or more K-complexes present. See fig. 1c.
- N3-N4: Non-REM 3-4. Slow delta wave activity. Dreaming stage starts here. This is the stage between being fully awake and being fully asleep. The newest definition combines the sleep stages N3 and N4 into one stage. See fig. 1d and 1e.
- R: During the rapid eye movements (REM) stage there are a mix of rhythmic components present in the EEG. The brain activity is similar to W stage. See fig. 1f.

The scope of this project is to create a re-implementation of the CNN [1] in TensorFlow (TF) and use this network as the baseline. The task is to implement and include a RNN when the baseline has been achieved, and hereby learn the transition rules between each of the sleep stages. This research will hopefully provide an improved sleep stage classifier and knowledge about the transition patterns, which is valuable for patients and doctors around the world.

2 Materials and Methods

As a requirement for this project the professor and assistant teachers need to have acces to the running code in order to reproduce the results in this project. The bullets below show links to what you need in order to reproduce the presented results and a link to the demo notebook.

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

2.1 Image Creation

There has been applied multi-taper spectrum analysis in order to turn the EEG-signals into images. An image represents an epoch of 30 $[s]$ of the recorded EEG-signal along its temporal axis. The second axis represents the spectrum of the rhythmic components of interest as mentioned above. The color of the image represents the amplitude of the rhythmic components.

The WFDB Toolbox [3] for Matlab has 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 concepts of multi-taper spectrum analysis, which estimate the images, is not in the scope of this project. The hyperparameters in the preprocessing task, such as the duration (in $[s]$) of an epoch, number of multi-tapers, frequency resolution (in $[Hz]$), etc., used for the estimation of the images, is identical to [1] and is not possible hyperparameter in this project. This ensures that the results of the baseline in this project are comparable with the article [1].

The Matlab toolbox is able to download the data set of interest. The data set used in this projects consists of PSG recordings for 20 subjects. Ten healthy females and ten healthy males between 25 to 34 years of age were included. The subjects have been monitored for two nights except subject 20. There are 38211 images after the preprocessing of the EEG-signals. All images have labeled values which entail a supervised learning approach. The annotated labels follows the old definition where sleep stage N3 and N4 are divided into two. Table 1 illustrates how the labels of the 38211 images are distributed for each of the sleep stages.

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

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

Prior to training of the networks is it considered to use methods to balance the six stages in order to create a state of the art sleep stage classifier.

2.2 Neural Network Architectures

VGGNet 16 [1, 4] is the architecture of the baseline network. It is composed of a 16-layer network with following operations:

- The convolutional operations use a kernel of $3x3$ pixels, which is the smallest possible configuration to capture the notion of left/right, up/down and center. The kernel has a stride of 1 pixel, which is the number of pixels the kernel slides at the time. The third hyperparameter in the convolutional operation is the padding. This operation use same-padding so the spatial resolution is preserved. This layer performs several linear activations by the kernel. These activations goes through a selected non-linear, e.g. a 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 $2x2$ pixels. The max pooling operation has a stride of 2 pixels. 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 operations [5, sec. 9].
- The last three layers of the VGGNet network is fully connected. The first two have 4096 channels. The final fully connected layer performs a six-channel classification due to the six stages in this project and is activated by the Softmax function. This activation function is used to calculate the probabilities associated to each class [5, eq. 4.1].

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

2.2.1 Convolutional Neural Network

The baseline network is given in eq. 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 layer. d is a dropout operation, which keeps $p = 50\%$ of the total connection between the previous and next layer.

The TF implementation of the VGGNet is heavily inspired by the work of The TensorFlow Authors (2017) [7].

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

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

The current implementation of the VVGNet has transformed the fully connected layers into convolutional 2D layers. By changing the size of the kernel it achieves the same performance. However it is still necessary to squeeze the final layer ($c_{1,o}$) into a 2D-tensor before applying the Softmax function.

L2-regularization with a weighted decay of $5 \cdot 10^{-4}$ has been included between the convolutional operations. Dropout operations between the fully connected layers have been included. The L2-regularization and dropout operations prevent overfitting in this deep complex network. The modifications are included in eq. 1.

2.2.2 Recurrent Neural Network

It is possible to interpret sleep as a sequence entering different stages during the night. By learning these transition rules between the stages can improve the classification.

The most effective RNNs for modelling sequences are called gated RNNs [5, sec. 10.10]. The long short-term memory (LSTM) cell is a network which comprehend the issues of vanishing or exploding gradients. The LSTM cell has an internal recurrence (self-loop) which produce paths where the gradient can flow for long durations and hereby capture long term dependencies.

The main reason for implementing a sequential network is to learn the transition rules between the sleep stages. The first approach to learn the transition rules is to modify the baseline network and use its extracted feature maps as input to the LSTM cell. A second approach could be to use several LSTM cells with regularization in between. A third approach could be to redefine the problem, discretize the EEG-signals and feed the signals into a sequence of LSTM cells. Due to limited time only the first approach has been considered. The network combining 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 the work of [8, 9], where the LSTM cell is 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. The LSTM cell is feed by the final max pooling layer. The input shape to the LSTM cell is static, which can cause discarding of the images in the final mini-batch.

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

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 stages.

The implementation of LSTM cell is heavily inspired by the work of The TensorFlow Authors (2017) [10]. This implementation takes the input tensor from the final max pooling layer in the VGGNet and returns the state tensor. Both are 4D-tensors (batch x height x width x channels).

There has been applied dropout operations on the output tensor from the LSTM cell as illustrated in eq. 2. The tensor is then reshaped in order to fit a fully connected layer, which produce easy separable higher-order feature maps of the six stages. The fully connected layer is activated with the nonlinear Softmax function.

2.3 Network Visualization

There exists many ways of visualizing a neural network in order to better understand how the network interprets different inputs. A common approach in the literature is to visualize the activations of the input along its propagation through the network. The example from [11] gives a solid intuition about how the different operations in each layer is affected by the input data.

Another approach is to calculate sensitivity maps which determines the relative importance to every input feature j [1]. 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 the input image and its label. The mathematical expression is given in eq. 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 $f\left(x\right)$ and its label t.

By applying the function $s^{(j)}$ for an image it is possible to see the relative importance determined by the network for each feature j. Figures 1g-1r illustrate the sensitivity maps for each sleep stage for the CNN and RNN respectively. The sensitivity maps have been created for subject 19 and 20.

3 Experimental Evaluation

3.1 Setup

The setup used in training the baseline and the RNN is identical. Both networks are based upon layers from the VGGNet and the weights of the layers are pre-trained on ILSVRC data set for several weeks. It is beneficial to apply the pre-trained weights and use the principle of transfer learning instead of learning the weights from scratch.

According to the literature [12], there are several approaches to perform transfer-learning. Among these is a

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

strategy to remove the final fully connected layer of the network. Then the pre-trained CNN is applied as a feature extractor for a new fully connected layer which fits the number of classes in the data set. A second strategy is carry out the first strategy and fine-tune a selected set of weights in the pre-trained CNN.

The chosen strategy in this setup was to remove and create a new final fully connected layer which fit the number of sleep stages in 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 in order to prevent overfitting.

The weights in the LSTM cell have been trained from scratch. The weights in the forget bias have been initialized to 1 which entails the LSTM cell to have no prior knowledge.

During the training process the networks only have access to the current epoch. This is different from the setup in [1] where the network use two prior epochs and two posterior epochs in order to learn the current epoch. The reason for this choice was due to 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 it can be considered as future research.

Both the CNN and the RNN networks have been optimized in order to minimize their categorical cross-entropy. Their loss function is handled by the AdamOptimizer, provided by TF, which iterates over a mini-batch with a size of 32. The training process has been repeated for 20 training epochs. The learning rate has been initialized to 10^{-5} and its decay rate, first and second order moment have not been changed from default.

Due to the characteristic of the sleep stages, the classes shown in table 1 are not equally distributed. The selected choice for fixing the issue is to randomly down-sample the majority classes to fit the minority class in each training epoch. A similar approach has been applied in [1].

The setup is capable of performing leave-one-out cross-validation due to the relative small number of subjects and a few professionel skilled sleep stage experts. The subjects have been divided into test, train and validation. Subject 19 and subject 20 have been fixed to the validation set in order to get the 10% of the total amount.

The test subject for the first fold is subject one. The train data includes subject 2 to subject 18. Due to time considerations it was chosen only to perform the first fold.

3.2 Results

Table 2 summarizes the performances evaluated for the two validation subjects in both networks. The first column block reports the confusion matrix. The next block reports the normalized confusion matrix. The third block reports the following per-class metrics: Precision, sensitivity, F_1-score and accuracy. The accuracy is not a reliable metric of the performance in this project, yield to misleading results of the imbalanced sleep stages in the validation set [5, sec. 11].

The baseline, reported in table 2, classifies the sleep stage N2 with a sensitivity of 84%. Then followed by R, W, N3, N4 and N1. N1 with a sensitivity of 62% as the most difficult sleep stage to classify. The highest misclassification error is achieved in N3 with a sensitivity of 37%.

The RNN classifies the sleep stage R with the highest sensitivity (87%). Then followed by N2, W, N4, N3 and N1. N1 with a sensitivity of 31% as the most difficult sleep stage to classify.

The precision metrics for the CNN and the RNN are relatively low for sleep stage N3 and sleep stage N4 compared to the other sleep stages. The misclassification rates for N3 and N4 are relatively close to its classification rates. The visual illustrations of the stages are similar, which can be a logical explanation for merging N3 and N4 together into one sleep stage as in the newest definition [2].

Table 3 reports the mean values and its corresponding 95% confident values computed by bootstrapping. There has been applied 100.000 bootstrap iterations with replacement in order to compute the average performance metrics for the two networks. According to table 3 it can be concluded that the RNN does not outperform the baseline network w.r.t. the selected performance metrics.

3.2.1 Sensitivity

The two last rows in fig. 1 illustrate the average representations of the six sleep stages for the two validation subjects. The sensitivity maps have been computed by eq. 3.

The sensitivity maps given in fig. 1j - 1k and fig. 1p - 1q for the CNN and the RNN respectively have more less the same pattern.

The summarized information from the sensitivity maps for sleep stage N3 and N4 provides a valid explanation of the high misclassification error in both networks (see the normalized confusion matrices in table 2).

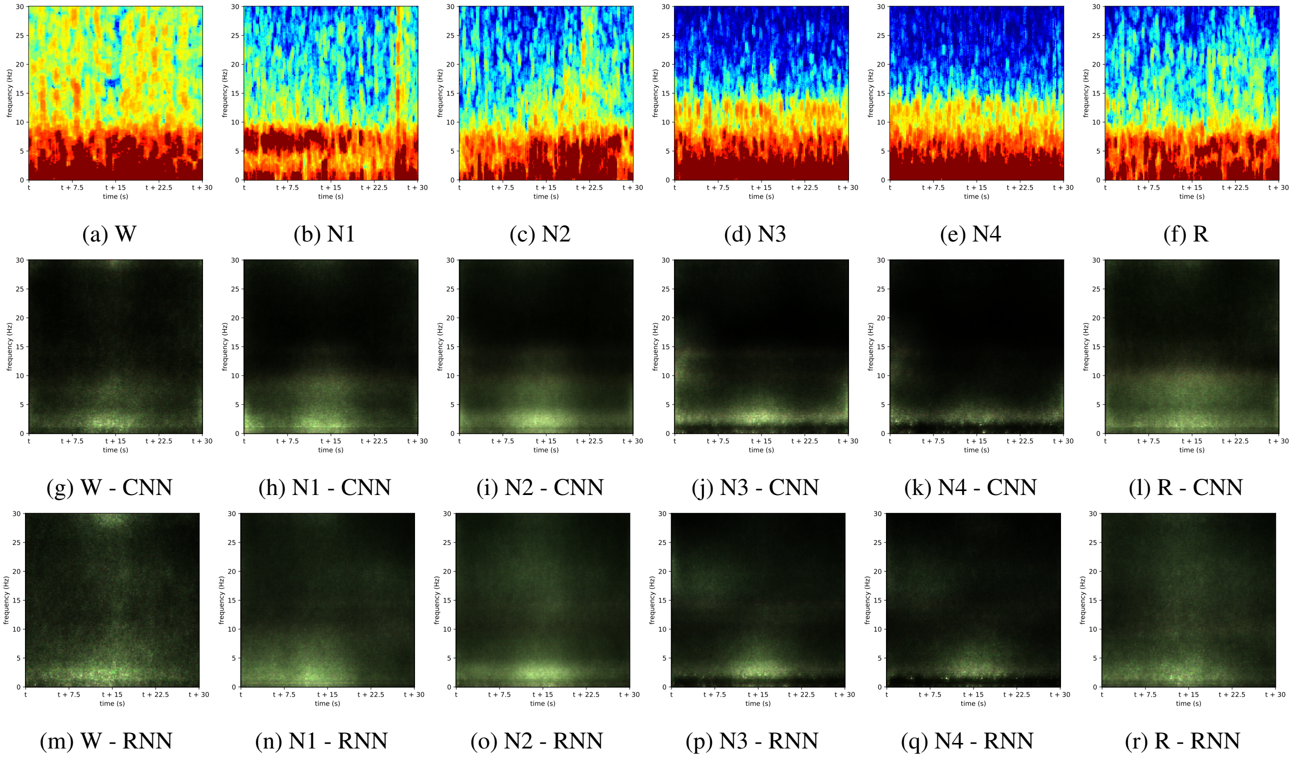
4 Discussion

The chosen approach to handle the imbalanced stages, randomly down-sampled the training data to fit the minority stage. By during so, it is possible to discard information, e.g. the spatial variation within each class. Although, the discarded information can be captured by repeating several training epochs and train the classifier with different balanced permutations of the sleep stages. An alternate approach to solve the imbalanced stages is to penalize the loss function w.r.t. to the sleep stage distribution in each mini-batch.

The relative low average performance can be caused by a lack of normalization of the input images. A possible solu-

		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 2: This table report the confusion matrix, its normalized confusion matrix and selected performances metrics for the CNN and RNN network.



(a) W (b) N1 (c) N2 (d) N3 (e) N4 (f) R

(g) W - CNN (h) N1 - CNN (i) N2 - CNN (j) N3 - CNN (k) N4 - CNN (l) R - CNN

(m) W - RNN (n) N1 - RNN (o) N2 - RNN (p) N3 - RNN (q) N4 - RNN (r) R - RNN

Fig. 1: This figure contain plots of each sleep stage for the two validation subjects. The visualizations are given columnwise from left to right according to the previous sequence of the sleep stages. Fig. 1a to 1f illustrate a random epoch of the multi-taper spectrum for each of the sleep stages. There is high similarity between sleep stage N3 and N4. The second and the third row, fig 1g to 1r illustrates the average sensitivity maps of the CNN and of the RNN respectively for the two validation subjects.

tion to this could be to subtract the mean image of the entire dataset from the input image [1, 4].

The size of the mini-batch was set to 32 images. This hyperparameter variates within the literature. Due to the static implementation of the LSTM cell, the discarded images in the final mini-batch need to be as few as possible. A lower batch size will entail a faster converging in the weights by frequently applying back-propagation. However by using too small a mini-batch size the estimation of the gradient is less accurate

and vice versa. .....Positiv and negat with a lar mini-batc size. GPU memory i sues

Learning long term dependencies between the sleep stages, the vanishing or exploding gradient issue is only

Study	Precision	Sensitivity	F_1-score	Accuracy
CNN	65-**68**-70	71-**71**-72	67-**69**-70	92-**92**-92
RNN	62-**65**-67	63-**66**-69	62-**64**-67	92-**92**-92

Table 3: **Mean** and corresponding 95% confident values computed by 100.000 bootstrap iterations with replacement.

solved by the gated RNNs such as the LSTM cell. The implementation of the LSTM cell is inspired by a video frame prediction method [10]. In this method there are few displacements for two consecutive frames. The CNN reduces the spatial variance in the image despite the temporal variance can be huge. This challenge can be studied into further details.

The implementation only includes one LSTM cell. The LSTM cells can easily be stacked in multiple layers which possibly can capture the long term dependencies in a better way however it will add a significant amount of learnable parameters.

It is possible to gain information regarding the training process by visualizing the loss as a function of training iterations. Hereby it is possible to achieve knowledge about the hyperparameters such as the learning rate, the effect of the dropout operations and how accurate the estimate of the gradient is. The current setup uses a relative low learning rate (10^{-5}).

hertil

The training process should be analyzed in order to check whether the learning rate is efficient for learning the weights of the new LSTM cell. Another advantage of the analysis of the learning process can provide an idea of how much the performance metrics changes as a function of the amount of training.

One of the challenging issues with the applied data set is the annotation of the sleep stages. The annotation of a sleep stages for each of the subjects has been done only one expert. This mean that the classifier is trained to the subjective differences of each of the experts. The quality of the annotation can be enhanced by introducing more experts to each of the subjects.

It was chosen to follow and train a classifier according to the old definition of the sleep stages because the annotations in the data set was created based upon the old definition. Sticking to the old definition of the sleep stages has been experienced to be less appropriate. The confusion matrices and the sensitivity maps for both the CNN and the RNN reports troubles in the separation of sleep stage N3 and sleep stage N4.

5 Conclusion

This project has successfully reproduced the VGGNet in [1] in TF which was one of the objectives. The network has successfully fine-tuned the dropout operations and the weights in its final fully connected layer. The VGGNet network have been used as the baseline. The chosen implementation of the RNN does note outperform the baseline in average performance metrics (see table 3), despite the RNN does achieve better classification sensitivity in the sleep stage W and in the sleep stage R (see table 2).

Further improvements in this projects can be achieved by merging sleep stage N3 and sleep stage N4 together. Study the affect of stacking multiple LSTM cells and apply the LSTM cell from layers with a lower-level feature representation and a higher spatial variance.

6 Acknowledgment

I do appreciate the supervision from Albert Vilamala and Sirin Gangstad, and the great help from the competent teaching assistants in the course.

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.