Sleep Staging by Hyperdimensional Dense Networks

Leonardo Hernández-Cano,[1, a)] Alejandro Hernández-Cano,[1, b)] and Ana Leonor Rivera[2, c)]

[1)]*Facultad de Ciencias, UNAM. Av. Universidad 3000. Coyoacán. 04510. Ciudad Universitaria. D.F. México, México*
[2)]*Instituto de Ciencias Nucleares, UNAM. Av. Universidad 3000. Coyoacán. 04510. Ciudad Universitaria. D.F. México, México*

[a)]*Corresponding author: leonardohernandezcano@ciencias.unam.mx*
[b)] *ale.hdz333@ciencias.unam.mx*
[c)] *ana.rivera@nucleares.unam.mx*

Abstract. Traditionally, sleep staging is done by medical experts, but computer aid will improve sleeping evaluation. We propose a mathematically-motivated algorithm based on Dense Convolutional Networks that encodes polysomnography (PSG) recordings into a very high-dimensional vector space to perform sleep-stage scoring. We emphasize the flexibility of our model as it provides a framework to analyze single or multi-channel signals without relying on any statistical information about the dataset. To prove the feasibility of our model we show results attaining comparable or better accuracy than current state-of-the-art models at a fraction of the time and very limited training data.

Keywords: Sleep staging, Machine Learning, Hyperdimensional Computing, Convolutional Neural Networks,

INTRODUCTION

The scoring of sleep into various stages is used to diagnose sleep related disorders. It is generally performed following the Rechtschaffen and Kales (R&K) or the Americal Academy of Sleep Medicine (AASM) guidelines using a polysomnographic (PSG) recording which usually includes electroencephalogram (EEG), electrooculogram (EOG), electromyogram (EMG) and electrocardiogram (ECG) signals [1]. Manual analysis yields low inter-scorer agreement ratios in addition from being a tedious and time-consuming process [2]. Automatic sleep scoring has inspired a lot of research but no widely-accepted solution has emerged [3]. Recently attention has shifted to machine learning, either completely removing human intervention [4, 5, 6] or mixing hand-engineered features with pattern-recognition techniques [2, 7, 8, 9, 10]. In this work we deal with a completely automated approach using raw signal data with the goal to classify sleep stages.

Our algorithm is based on two brain-inspired state-of-the-art machine-learning techniques with available specialized hardware proposed by the Machine Learning community. The first, Hyperdimensional-computing (HD), is a computing model based on projecting data to sparse high-dimensional spaces [11] that has been applied to EEG-based gesture recognition [12]. Here, we propose encoding PSG recordings to a hyperdimensional space using Convolutional Neural Networks (CNNs). CNNs are a class of Artificial Neural Networks in which layers apply a parametrized convolution on their inputs, followed by non-linear activation functions. Dense Convolutional Networks are a novel approach to CNNs that takle the vanishing-gradient problem using a sequence of denseblocks: sequences of convolutional layers which connect each layer to every other layer in a feed-forward fashion [13]. This approach has never been applied to analyzing PSG records. The traditional approach includes using statistical methods to find a transformation that retains as most useful information from the PSG recording as possible while also projecting to a lower dimensionality space (commonly referred to as feature reduction) [10, 14, 15, 16]. The rest of this document is structured as follows: we first motivate and describe the techniques employed, then we share results on the ISRUC-Sleep dataset.

MATERIALS AND METHODS

Dataset

The issue of performance-inflating class-imbalance in widely used datasets has been brought forward recently (see e.g. [2, 8]). In this work we test on the relatively balanced publicly-available ISRUC-Sleep dataset [1], which complied with ethical standards. ISRUC-slepp provides expert AASM scoring on PSG recordings of subjects regarded as healthy as well as on subjects diagnosed with sleep-related disorders.

Proceedings of the XVI Mexican Symposium on Medical Physics
AIP Conf. Proc. 2348, 030002-1–030002-6; https://doi.org/10.1063/5.0051275
Published by AIP Publishing. 978-0-7354-4094-4/$30.00

ISRUC provides two hypnograms for each PSG recording scored by different experts. Previous work has integrated inter-scorer discrepancies directly into the decision algorithm [17], but in this work we make no such attempt, instead our algorithm trains and calculates every metric in respect to "Expert 0".

Hyperdimensional-computing

Hyperdimensional-computing (HD) is a computing model first introduced in the field of neuroscience as an architecture for storing and handling data [18]. HD models project data to very high dimensional vectors (e.g. $D \approx 10,000$) as an alternative to computing directly with numbers. The process used to project data is called *encoding* and it is application-specific. Previous research has proved HD to be a feasible alternative in machine learning problems such as language recognition [19], voice recognition [20] and sensorimotor control learning [21]. An important motivation in HD is the fact that the probability of randomly generated vectors being nearly orthogonal increases as dimensionality increases.

Most research around hyperdimensional computing models (e.g. [20]) has been made assuming binary vectors, that is, each component would be either on or off. More recent work (e.g. [12]) has studied non-binary hypervectors, where each component is capable of holding any real number. In this document we focus on non-binary hypervectors over R^D.

Distance Given two hypervectors A, B we use cosine similarity as the distance function, as it gives us information about the angle formed by A and B in the high dimensional space. Thus if $d(A,B) < d(A,C)$, then the angle formed by C and A is smaller than the one formed B and A. Specifically, our similarity metric is

$$d(A,B) = \frac{A \cdot B}{\|A\|\|B\|}.$$

Addition We implement this operator as the usual element-wise addition of the given vectors and denote it with $A+B$. Addition yields a new hypervector that conserves information stored in A, but is closer in the hyperspace to B.

Encoding functions As is standard, we represent a 30-second epoch of m channels with a length of h with a vector in $R^{h\times\ m}$ (e.g. 30 seconds of EOG and EEG sampled at 100Hz yield a vector of length 30×100 and 2 channels). An effective encoding $f : R^{h\times\ m} \to R^D$ maps epochs with the same label to similar hypervectors. In the next section we shall discuss the problem of finding such effective functions.

Training Given a set of hypervectors $\{v_1, \ldots v_n\} \subseteq R^D$ that are similar , by computing $v_i + v_j$ we obtain a new element that is similar to both v_i and v_j under cosine similarity. This motivates the training procedure. Given an effective encoding function f and a training set E_i of PSG epochs belonging to stage i (e.g. $i =$ REM stage), we compute $V_i \subseteq R^n = \{f(e)|e \in E_i\}$ and find a representative hypervector c_i of class i by adding all elements in V_i:

$$c_i = \sum_{e \in E_i} f(e).$$

CNNs as Non-linear Encoders

CNNs have proven capable of dealing with the sleep-stage classification problem [22], very notably by [4], who showed a relation between the filters learned by their network and AASM staging rules. We begin by framing CNNs as encoders and then detail our strategy for finding effective-encoding function spaces.

Given input x_0, a CNN defines non-linear transformations $H_l(\cdot)$ in each layer l, where l indexes the layer. The output of layer l is denoted with x_l and corresponds to transforming its input via one or more convolution operations and then applying non-linear operations. Examples of popular non-linear operations include application of Rectified Linear Units, Batch Normalization and Pooling. The kernel parameters of the convolution operations are learned during training.

When training a network one is essentially exploring a differentiable space of non-linear functions via, for example, Stochastic Gradient Descent to minimize a loss function. Each parameter update effectively changes the current candidate function for another that may be best suited for classification. That is the key observation in our motivation for using CNNs: they provide the machinery needed to traverse complex spaces of non-linear functions.

Lets assume we have C_1 an l-layer CNN with a hyperdimensional output that takes as inputs PSG recordings of length h and m channels. This induces a function that is a composition of the transformations defined by each layer:

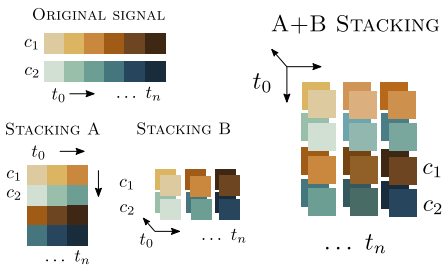


FIGURE 1. Time series stacking and pseudosquaring

$C_1 : R^{h\times\, m} \to R^D = H_1 \circ \ldots \circ H_l$. We then consider $C_2 = C_1 \circ D_c \circ S$ where c is the number of classes (sleep stages), D_c is a dense layer of dimensionality c and S is the *soft-max* function. Recall that D_c defines a function that is essentially a series of dot-products: $D_c(v) = (k_1 \cdot v, \ldots k_c \cdot v)$ where k_i are kernel matrices whose parameters are learned during training. So, there is one hyperdimensional k_i for each class that the input is compared to as the last step of the classification: the class corresponding to the hypervector k_i that outputs the highest dot-product is said to be the class to which C_2 predicts the input to be.

Intuitively, by adding a dense layer after H_l we are guiding the search towards functions whose outputs are hypervectors for which there seem to exist class representatives that allow labeling when compared via *dot-product*, so by training C_2 and removing the added layers back to C_1 we find an effective encoder function.

We now describe how sleep stage classification led us to a family of CNNs based on DenseNets.

Signal Stacking and Pseudosquaring

When using 1D input signals one may work with pre-computed spectograms to treat as images with 2D convolutions or use raw signal data over 1D convolutions. One approach to learning convolutions on raw signal data on multiple channels is to learn per-channel 1D filters [4]. An important property of convolutions if that they work on local information: they depend on the information visible to the kernel. In this section a very simple operation to use 2D convolutions in our network while still using a 1D input and also controlling non-locality is proposed. For simplicity, lets assume we have a 2-channel signal.

A first attempt may begin by stacking channels and forming a long 2D single-channel tensor. This arrangement is unsuitable for our purposes, as even though it will allow 2×2 kernels to see both channels and their values in the time window, it will also compress the 2×2 windows into a single scalar value and will not allow to stack multiple layers to find non-linear filters.

We then propose *stacking A*: as before, we first arrange the 2-channel signal into a single-channel, long, 2-row, 2D tensor, we then cut the tensor into segments of equal length and finally stack the cuts to form a rectangular version of our input. This allows us to stack layers and use larger kernels. In this stacking, time information flows in 2D which allows kernels in convolutional layers to act over multiple signals and their changes in time. One may include non-locality by allowing filters to see values in rows belonging to different time periods by increasing kernel size or changing stride values.

We may instead consider *stacking B* if we want kernels to focus on the overall changes over time instead of the immediate changes in the signals: for jumps in time of size j we compute a long 2D tensor of j channels where the ith row has the values of the ith channel with jumps of size j in time in its first channel: $0, j, 2j, \ldots$, and further channels have an start offset, which allows a computationally efficient way to learn multiple 1D filters over the signal with jumps in time. In this stacking although time flows in 2D, layers are restricted to composing 1D filters, jumps of size j and a time window as long as its kernel.

Should we want to have both the benefits of *stacking A* and *stacking B*, we propose *stacking A+B* (see figure 1): we perform B followed by A. This leads to time flowing in 3D, allowing 2D kernels to compute non-local changes in time and to focus changes in time along multiple channels.

Composition of 2D convolutions is most effective when dealing with square inputs, and in a valid stacking the number of rows has to be a multiple of the channels of the signal. We summarize a procedure we call *pseudosquaring* to find A+B stackings with a maximum jump size of j. Suppose the input signal has length h and c channels, yielding a total of $hc = N$ elements. Consider the set F of 3-number factorizations of N with said restrictions:

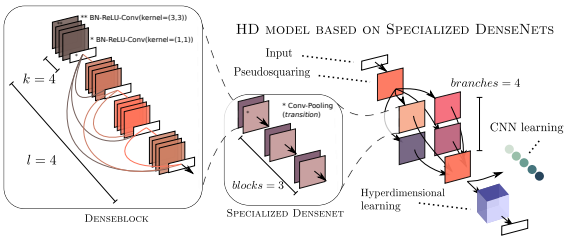


FIGURE 2. A hyperdimensional model based on 1.5D specialized DenseNets and its hyperparameters.

$F = \{(p,q,r)|pqr = N \wedge r \leq j \wedge p \text{ is a multiple of c}\}$ and find $(p,q,r) \in F$ such that $|p-q|$ is minimum. Naturally, this yields the stacking shape closer to a square with depth j or less, which is why we label models using this process "1.5D". Due to computational reasons, after we find said shape $s* = (p,q,r)$ in this work we implement the described arrangement *Psqr* of the signal as an equivalent composition of transpose and reshape operations over tensors:

$$Psqr(t) = \ reshape(t,s*);reshape(t,(p/c,qr,c));transpose(t,(0,2,1));reshape(t,s*);$$

Specialized Dense Convolutional Networks

We now turn to finding CNN architectures that can be used as effective encoding functions. Dense Convolutional Networks (*DenseNets*) are a novel approach to CNNs that tackle the vanishing-gradient problem using a sequence of denseblocks: sequences of convolutional layers which connect each layer to every other layer in a feed-forward fashion [13]. That is, layer x_i of a denseblock receives the feature maps of all its preceding layers $x_0,\ldots,x_{i-1}$. Denseblocks may be connected by *transition layers* consisting of convolution and pooling operations.

To achieve high-dimensionality we use independent *DenseNets* as branches after applying *pseudosquarization*. When concatenated and flattened the output becomes hyperdimensional. Specifically, given a PSG signal $e \in R^{n\times m}$, our *pseudosquaring* layer *Psqr*, the usual flattening function $flatten$, and DenseNets $D_1,\ldots D_b$, our encoding function (figure 2) is

$$E(e) = \ flatten((D_1(Psqr(e)),\ldots,D_b(Psqr(e)))$$

Note that each dimension of $E(e)$ is computed by exactly one branch D_i. One may also achieve high-dimensionality by increasing the number of kernels in the convolutional layers after applying *Psqr*.

We now consider E and its training extension H_2 in respect to the remarks made in previous sections. When training H_2, kernel parameters in the convolutional layers are first initialized randomly. As mentioned, the hyperdimensionality of the output $E(e)$ will make vectors orthogonal almost certainly, initially projecting the input data into a very sparse hyperspace. This hypervectors will be compared with hypervectors of the final dense layer. As training progresses, the high connectivity provided by the DenseNets architecture allows information to be propagated efficiently, thus enabling learning to occur. After training H_2, a hyperdimensional model H is trained with the output of E.

RESULTS

We focus on testing the effectiveness of the learning process by using a quick (when on specialized hardware) 15-epoch training on limited data to challenge the so-called *curse of dimensionality*: typically enormous amounts of time and training data are needed as dimensionality grows. Accuracy in state-of-the-art results are most frequently in the range from 75% to 85% [5, 6, 7, 8] and vary greatly depending on the dataset, technique, amount of sleep stages considered, amount of preprocessing, etc. For example, [3] reaches 54% percent using symbolic fusion to formalize the decision process and [23] achieve 83% and 90% using 5 and 11 channels respectively on 86 ISRUC recordings to train and 10 for testing. To challenge the effectiveness of the learning process we will train on only 2 channels (EOG E1-M2 and EEG F3-M2) from 40% of the data from the 10 recordings in subgroup III leaving 60% for testing.

We follow [13] and define $H_l(\cdot)$ as a composite function of Batch Normalization, followed by a Rectified Linear Unit, and a 3×3 convolution; we set $k = 16, blocks = 5, l = 4$ in the multi-branch model and $k = 32, blocks = 3, l = 4$

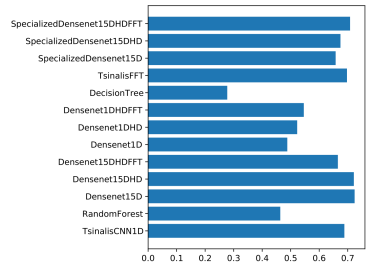


FIGURE 3. Sub-Dep

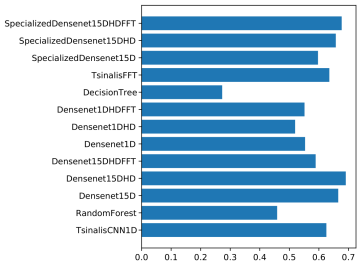


FIGURE 4. Sub-Ind

in the single-branch model and batch size of 32 with an Adam optimizer function. Implementation was done using `tensorflow` in keras [24] which allowed training on an Nvidia GTX 1070 GPU.

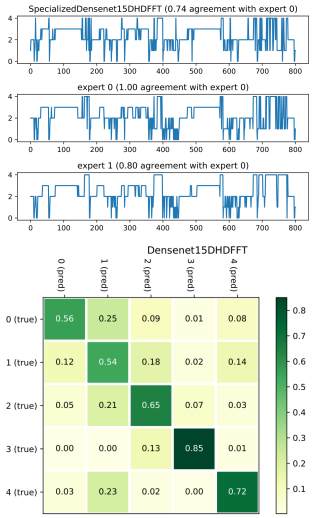
We compare possibilities of the proposed architectures along with our implementation of the work in [4] as described in their paper (label "TsinalisCNN1D"). We will focus on 5-branch and single-branch models as described in the preceding sections (respectively, labels starting with "SpecializedDenseNet" and "DenseNet"). As an ablation test we compare between classification performed with (suffix "HD") and without cosine similarity and representative hypervectors (note that in both cases data is projected to hyperdimensional-space; but in the latter case classification is performed by the layer of the network added for training); and also with and without *pseudosquaring* (labels "1.5D" and "1D" respectively). Inspired by [2, 25], we also compare with Random Forests and Random Trees (as baselines) as implemented in `scikit-learn` [26], along with a transformation to frequency-domain via Fast Fourier Transform (label "FFT").

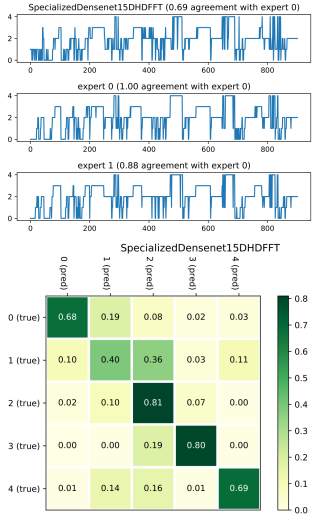
Figures labeled with "Sub-Dep" refer to the subject-dependent test (i.e. testing epochs may be from a recording from which some samples were drawn for training) and those labeled with "Sub-Ind" refer to the subject-independent test (i.e. testing epochs belong a record from which no sample was drawn for training). Hypnograms come from the subject independent test. No manipulation was performed using information from the testing set. No sample used in training is used to compute performance metrics.

First, we acknowledge the performance of "TsinalisCNN1D" (which is also the model with the most parameters): it attained near-state-of-the-art results while being under trained. That model was outperformed by "SpecializedDensenet1.5DHDFFT", "DenseNet1.5DHD", and "DenseNet1.5D", which also outperformed all other models even though they were equally under trained. No FFT-based model was the top performer in any of the two tests.

We now analyze the top-performer ("DenseNet1.5DHD") and compare our results with a state-of-the-art Artificial Neural Network-based (ANN) approach [23]. We used around 4.65% of the data employed by [23] but reached 92% of their reported 5-channel recall (0.70 vs 0.76) and 83% of their accuracy (0.69 vs 0.83) with a comparable testing set size (60%) over the same dataset. This shows that performance comparable to state-of-the-art ANN models can be reached with a fraction of the data by using hyperdimensional CNN encoders. That the "HD" version of the models generalized better to the subject independent test than non-"HD" versions in all but one case ("DenseNet1D") shows that classification using cosine similarity and representative hypervectors was beneficial.

Thus, our results show that HD-encoding CNNs outperform standard baselines and attain performance comparable to state-of-the-art models with limited training data. Furthermore, classification in HD-space via cosine-similarity and representative hypervectors improves accuracy compared to classifying with an HD-encoding CNN alone.





(Example output hypnograms of "SpecializedDensenet1.5DHDFFT") The overall evolution of the PSG recordings matches the expert scorers. When there is expert disagreement the model tends to misclassify epochs.

(Example confusion matrices) Confusion matrices from subject-dependent tests show a tendency for misclassification of N1 (label "1"), the hardest stage to classify (see e.g. [2]). The examples on the left show that "branching" improves accuracy in the case of transformation via FFT.

CONCLUSION

We propose a scheme to use CNNs as encoders to hyperdimensional spaces where classification can be performed, which leads to models that attain comparable or better accuracy than current state-of-the-art automatic sleep-staging approaches at a fraction of the time and with very limited training data.

REFERENCES

1. S. Khalighi, T. Sousa, J. M. Santos, and U. Nunes, "Isruc-sleep: A comprehensive public dataset for sleep researchers." Computer Methods and Programs in Biomedicine (2016).
2. M. M. Rahman, M. I. H. Bhuiyan, and A. R. Hassan, "Sleep stage classification using single-channel eog." Computers in Biology and Medicine (2018).
3. A. Ugon, A. Kotti, B. Séroussi, K. Sedki, J. Bouaud, J.-G. Ganascia, P. Garda, C. Philippe, and A. Pinna, "Knowledge-based decision system for automatic sleep staging using symbolic fusion in a turing machine-like decision process formalizing the sleep medicine guidelines." Expert Systems With Applications **114**, 414 – 427 (2018).
4. O. Tsinalis, P. M. Matthews, Y. Guo, and S. Zafeiriou, "Automatic sleep stage scoring with single-channel eeg using convolutional neural networks." Annals of Biomedical Engineering (2016).
5. A. Sors, S. Bonnet, S. Mirek, L. Vercueil, and J.-F. Payen, "A convolutional neural network for sleep stage scoring from raw single-channel eeg." Biomedical Signal Processing and Control **42**, 107 – 114 (2018).
6. A. B. Klok, J. Edin, M. Cesari, A. N. Olesen, P. Jennum, and H. B. D. Sorensen, "A new fully automated random-forest algorithm for sleep staging." Conference Proceedings: ... Annual International Conference Of The IEEE Engineering In Medicine And Biology Society. IEEE Engineering In Medicine And Biology Society. Annual Conference **2018**, 4920 – 4923 (2018).
7. P. Ghasemzadeh, H. Kalbkhani, and M. G. Shayesteh, "Sleep stages classification from eeg signal based on stockwell transform." IET Signal Processing (2019).
8. D. Jiang, Y.-n. Lu, Y. Ma, and Y. Wang, "Robust sleep stage classification with single-channel eeg signals using multimodal decomposition and hmm-based refinement." Expert Systems With Applications (2019).
9. Z. Liu, J. Sun, Y. Zhang, and P. Rolfe, "Sleep staging from the eeg signal using multi-domain feature extraction." Biomedical Signal Processing and Control **30**, 86 – 97 (2016).
10. Y.-L. Hsu, Y.-T. Yang, J.-S. Wang, and C.-Y. Hsu, "Automatic sleep stage recurrent neural classifier using energy features of eeg signals." Neurocomputing **104**, 105 – 114 (2013).
11. A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing." International Symposium on Low Power Electronics & Design , 64 (2016).
12. Y. Kim, M. Imani, and T. S. Rosing, "Efficient human activity recognition using hyperdimensional computing," in *IOT* (2018).
13. G. Huang, Z. Liu, K. Q. Weinberger, and L. v. d. Maaten, "Densely connected convolutional networks." 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017).
14. S. I. Dimitriadis, C. Salis, and D. Linden, "A novel, fast and efficient single-sensor automatic sleep-stage classification based on complementary cross-frequency coupling estimates." Clinical Neurophysiology **129**, 815 – 828 (2018).
15. T. Padma Shri and N. Sriraam, "Comparison of t-test ranking with pca and sepcor feature selection for wake and stage 1 sleep pattern recognition in multichannel electroencephalograms." Biomedical Signal Processing and Control **31**, 499 – 512 (2017).
16. T. Lajnef, S. Chaibi, P. Ruby, P.-E. Aguera, J.-B. Eichenlaub, M. Samet, A. Kachouri, and K. Jerbi, "Learning machines and sleeping brains: Automatic sleep stage classification using decision-tree multi-class support vector machines." Journal of Neuroscience Methods **250**, 94 – 105 (2015).
17. A. Ugon, A. Kotti, B. Séroussi, K. Sedki, J. Bouaud, J.-G. Ganascia, P. Garda, C. Philippe, and A. Pinna, "Knowledge-based decision system for automatic sleep staging using symbolic fusion in a turing machine-like decision process formalizing the sleep medicine guidelines." Expert Systems With Applications **114**, 414 – 427 (2018).
18. P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," Cognitive Computation **1** (2009), 10.1007/s12559-009-9009-8.
19. M. Imani, J. H. Hwang, T. S. Rosing, A. Rahimi, and J. M. Rabaey, "Low-power sparse hyperdimensional encoder for language recognition," IEEE Design & Test **34**, 94–101 (2017).
20. M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," (2017) pp. 1–8.
21. A. Mitrokhin, P. Sutor, C. Fermüller, and Y. Aloimonos, "Learning sensorimotor control withneuromorphic sensors: Toward hyperdimensional active perception," Sci. Robot.4 (2019), 10.1126/scirobotics.aaw6736.
22. K. S. Prabhudesai, L. M. Collins, and B. O. Mainsah, "Automated feature learning using deep convolutional auto-encoder neural network for clustering electroencephalograms into sleep stages." 2019 9th International IEEE/EMBS Conference on Neural Engineering (NER), Neural Engineering (NER), 2019 9th International IEEE/EMBS Conference on , 937 (2019).
23. C. Zhihong, Z. Xiangwei, S. Xuexiao, and C. Lizhen, "Automatic sleep stage classification based on convolutional neural network and fine-grained segments." Complexity (2018).
24. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," (2015), software available from tensorflow.org.
25. E. A. Khourshee and A. S. Essa, "Eegs feature extraction by multi-level dwt with different numbers of principal components." 2019 International Conference on Advanced Science and Engineering (ICOASE), Advanced Science and Engineering (ICOASE), 2019 International Conference on , 1 (2019).
26. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research **12**, 2825–2830 (2011).