Fast and scalable distributed deep convolutional autoencoder for fMRI big data analytics

Milad Makkie[a], Heng Huang[b], Yu Zhao[a], Athanasios V. Vasilakos[c], Tianming Liu[a,*]

[a] *Computer Science Department, University of Georgia, Athens, GA, United States*
[b] *School of Automation, Northwestern Polytechnical University, Xi'an, China*
[c] *Computer Science Department, Lulea University of Technology, Lulea, Sweden*

A B S T R A C T

In recent years, analyzing task-based fMRI (tfMRI) data has become an essential tool for understanding brain function and networks. However, due to the sheer size of tfMRI data, its intrinsic complex structure, and lack of ground truth of underlying neural activities, modeling tfMRI data is hard and challenging. Previously proposed data modeling methods including Independent Component Analysis (ICA) and Sparse Dictionary Learning only provided shallow models based on blind source separation under the strong assumption that original fMRI signals could be linearly decomposed into time series components with corresponding spatial maps. Given the Convolutional Neural Network (CNN) successes in learning hierarchical abstractions from low-level data such as tfMRI time series, in this work we propose a novel scalable distributed deep CNN autoencoder model and apply it for fMRI big data analysis. This model aims to both learn the complex hierarchical structures of the tfMRI big data and to leverage the processing power of multiple GPUs in a distributed fashion. To deploy such a model, we have created an enhanced processing pipeline on the top of Apache Spark and Tensorflow, leveraging from a large cluster of GPU nodes over cloud. Experimental results from applying the model on the Human Connectome Project (HCP) data show that the proposed model is efficient and scalable toward tfMRI big data modeling and analytics, thus enabling data-driven extraction of hierarchical neuroscientific information from massive fMRI big data.

© 2018 Elsevier B.V. All rights reserved.

1. Introduction

The sheer complexity of the brain has forced the neuroscience community and particularly the neuroimaging experts to transit from the smaller brain datasets to much larger hard-to-handle ones. The cutting-edge technologies in the biomedical imaging field, as well as the new techniques in digitizing, all lead to collect further information from the structural organization and functional neuron activities in the brain through rich imaging modalities like fMRI [1]. Projects such as Human Connectome Project (HCP) [2], 1000 Functional Connectomes [3] and OpenfMRI [4] are the perfect examples of such large neuroimaging datasets. The primary goal of these efforts is to gain a better understanding of the human brain and to diagnose the neurological and psychiatric disorders. Among various neuroimaging methods, task-based functional magnetic resonance imaging, tfMRI, has been widely used to as-

sess functional activity patterns and cognitive behavior of human brain [5–8]. However, the main challenges are to obtain meaningful patterns from the intrinsic complex structure of tfMRI and also lack of clear insight into the underlying neural activities. Given the hierarchical structure of functional networks in human brain, the previously data-driven methods such as Independent component analysis (ICA) [9] and sparse coding for Dictionary Learning [10] as well as model-driven approaches such as General Linear Model (GLM) [11] have been demonstrated to disregard some of the information contained in the rich tfMRI data [12,13]. Thus, these shallow machine learning models are not capable of fully understanding the deep hierarchical structures of functional networks in human brain [12,13]. Consequently, there is an urgent call for more efficient and scalable data analytics and knowledge discovery methods to crack the underlying brain activities.

Recently, new data-driven computational intensive neural network approaches such as deep learning [49,50] have gained increasing interest among researchers, due to their efficiency of extracting meaningful hierarchical features from the low-level raw data. Particularly, Convolutional Neural Network (CNN) is

among the top deep learning methods in the scientific community [14–20], especially in classifying and learning image data [21].

In the context of high dimensional data such as fMRI, however, the large size of training examples (dozens of millions of time series each with hundreds of time points) and the sheer size of model parameters can drastically impact the computational cost and accuracy of learning the fMRI signals. Furthermore, most of the current neural network methods for fMRI analysis are only implemented for local application without any parallelization scheme [28–32]. As indicated by an extensive battery of literature [22–25], many of scaling deep learning applications by using large-scale clusters of GPUs can solve the computational bottleneck for efficient and effective knowledge discovery from fMRI big data.

Following the previous successes in using distributed GPU processing for scaling neural network model, in this work we aim to design a fast and scalable distributed framework and to implement a deep convolutional model, dubbed distributed Deep Convolutional Autoencoder (dist-DCA) to leverage the power of distributed optimization, distributed data partitioning, and multiple GPU processing. The distributed optimizer is based on an asynchronized Stochastic Gradient Descent (SGD) method [22]. In this model, we have used multiple replicas of a single model to optimize parameters, which lead to reducing the training time significantly. For data parallelization, we utilized Apache Spark [27] and Hadoop Distributed File System (HDFS). Considering the computationally intensive operations in tuning the parameter, Spark acts as a fast extract transfer load layer to optimize the data partitioning for the underlying Hadoop ecosystem. This is being accomplished via constructing the Resilient Distributed Dataset, RDD, which provides a functional interface to partitioned data across the cluster. Our major contributions of this work can be summarized as follows.

(1) We implement a distributed deep learning framework using TensorFlow on Spark to take advantage of the power of distributed GPUs cluster.
(2) We propose a distributed deep convolutional autoencoder model to gain meaningful neuroscience insight from the massive amount of tfMRI big data.
(3) We validate our proposed dist-DCA model using a novel high-level sparse dictionary learning method.

Compared to the existing distributed deep learning frameworks such as dist-keras [40], elephas [47] and dl4j [48], our proposed framework has a few critical advantages: (1) the migration from a standalone code to a distributed version can be done with only a few lines of code change. (2) Despite of the previous framework, our framework works efficiently with HDFS, allowing Spark to push datasets. (3) Integrating the model with the current pipeline is easy as Spark is in charge of parallelizing the data. (4) The framework is easy to deploy and scale over the cloud or in-house clusters. We have created an Amazon Machine Image (AMI), which in combination with a spark-ec2 script, can easily scale up the cluster.

The rest of this paper describes our dist-DCA model and architecture in detail. In Section 2, we briefly introduce the primary components in which dist-DCA is implemented. We also review related works in this domain. We then thoroughly describe our deep convolutional model in Section 3. Section 4 is dedicated to data parallelism and distributed optimization. Section 5 describes our scalable experiments in large GPU clusters where we explain how our model can be easily distributed among dozens of GPU nodes to reduce computational time efficiently.

2. Preliminary and related works

Recent advances in building affordable high-performance GPUs with thousands of cores were one of the critical factors in advanc-
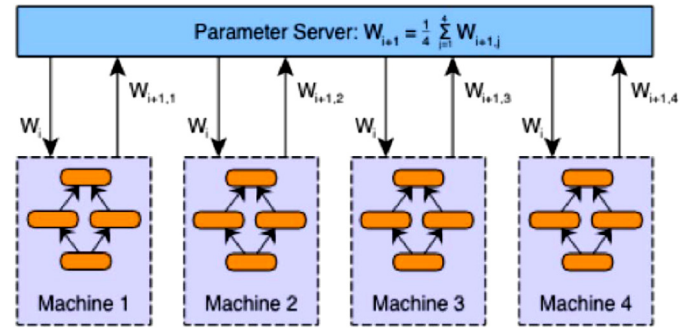

Fig 1. An asynchronous data parallelism model using Asynchronous SGD.

ing large-scale deep learning models. This break-through has also encouraged the scientific community to utilize GPUs more often, as CPU's capacity does not seem to grow in proportion to the rate of increasing demand. However, the limited memory capacity of typical GPUs on the market (usually 8 gigabytes) has become a bottleneck in feeding extensive datasets as far as the training speed is concerned. Therefore, two common approaches of data parallelism and model parallelism are of the researchers' interest.

In model parallelism, different portions of a model computation are done on different computational devices simultaneously for the same batch of examples while sharing the parameters among devices as a single model. This approach, however, is efficient for very large models as splitting a neural network model needs to be done in a case-by-case manner and is very time-consuming. Data parallelism, on the other hand, seems more straightforward for general implementation and can be easily scaled to larger cluster sizes. Fig. 1 demonstrates a data parallelism paradigm. We will discuss our dist-DCA data parallelism scheme in more depths in Section 4.

Our main motivation behind this work is to implement a scalable, asynchronous data parallelism model leveraging TensorFlow on Spark [34] to efficiently learn meaningful hierarchical abstraction of massive size of fMRI data.

2.1. TensorFlow

TensorFlow [46] is a mathematical software and an open-source software library for Machine Intelligence, developed since 2011, by Google Brain Team and initially aimed to machine learning research and deep neural networks. TensorFlow is a numerical computation library using data flow graphs that enables machine learning experts to do more dataintensive computing, e.g., it contains some robust implementations of conventional deep learning algorithms. It offers a very flexible architecture that enables deploying computation to one or more CPUs or GPUs in a standalone, parallel or distributed fashion. We selected TensorFlow in our work as it efficiently supports distributed and parallel GPU processing and it supports Keras. However, having an easy to scale framework is required for running TensorFlow applications when the model and data become large. So, a queuing framework to both seamlessly feed data into the cluster nodes and to schedule and manage tasks efficiently is needed. Pipelining pre-processing, training and inferences steps is a known challenge yet to be addressed by the TensorFlow ecosystem.

2.2. Spark

Since 2009, the Spark framework [27] was developed at the University of Berkeley AMPlab and currently is being maintained by Databricks. This framework addresses deficiencies of MapReduce by introducing resilient distributed datasets (RDD) abstract

where the operations are performed in the memory. Spark compiles the action lineages of operations into efficient tasks, which are executed on the Spark engine. Spark offers a functional programming API to manipulate Resilient Distributed Datasets (RDDs). RDDs represent a collection of items distributed across many computing nodes that can be manipulated in parallel. Spark Core is a computational engine responsible for scheduling, distributing and monitoring applications. It consists of many computational tasks across executor node(s) on a computation node/cluster. Spark's scheduler will execute the duties across the whole cluster. Spark minimizes the repetition of data loading by caching data in memory, which is crucial in complex processes. Spark uses Hadoop filesystem as a core distributed file system (HDFS). Apache Spark is one of the most active Apache projects on GitHub.

In this work, we used a combination of TensorFlow and Spark [46] to leverage the data parallelism and scheduling of Spark, thus enabling direct tensor communication among TensorFlow executors and parameter server(s). Process-to-process direct communication enables TensorFlow program to scale effortlessly. In Section 4, we will describe such communication in more details.o kearb ogtsuiig

2.3. Previous works

In the past few years, there have been multiple studies in adopting neural network methods to model fMRI data and its associated applications. For instance, Chen et al. [28] used convolutional autoencoder in fMRI data aggregation; Plis et al. [29] used deep belief network (DBN) to learn physiologically important representations from fMRI data; Suk et al. [30] combined the Deep Auto-Encoder with Hidden Markov Model to investigate the functional connectivity in resting-state fMRI; Huang et al. [31] used the restricted Boltzmann machine to mine the latent sources in task fMRI data; Ren et al. [32] used convolutional neural networks to classify fMRI-derived functional brain networks, and Wen et al. [33] have used AlexNet to reconstruct the visual and semantic experiences using fMRI data. In the context of applying deep learning applications to fMRI data, however, most works have focused on the classification problem by using a single computation node. Our focus in this paper is the provision of an unsupervised distributed CNN encoder that effectively models the tfMRI big data. This enables us to learn hierarchical feature abstraction while lowering the spatial and temporal noises contained in fMRI data and ensures us to efficiently reduce model training and inferences time by easily scaling cluster of GPUs.

3. Deep convolutional autoencoder

Our main goal of proposing the novel dist-DCA is to introduce our efforts in implementing a distributed deep learning framework using TensorFlow on Spark to take advantage of the power of distributed GPUs cluster. We proposed a deep convolutional autoencoder model to gain meaningful neuroscience insight from the massive amount of tfMRI big data using our abovementioned framework. We furthermore, validated our proposed dist-DCA model using a novel high-level sparse dictionary learning method.

Fig. 2 illustrates both the structure of our proposed dist-DCA model and a validation pipeline based on the online dictionary learning (ODL) algorithm. Part (a) demonstrates the preprocessing step of the tfMRI data including skull removal, motion correction, slice time correction, spatial smoothing, and global drift removal. We used HELPNI [35], a neuroinformatics platform for preprocessing. Part (b) demonstrates the structure of the deep convolutional autoencoder model and its components including all of the hidden layers and features maps. This model is explained later

Table 1
Dist-DCA model summary.

Feature map/filter	Layer 1	Layer 2	Layer 3	Layer4
Encoder	32/21	64/9	128/9	256/9
Decoder	128/9	64/9	32/9	1/21
Total parameters	6,023,549			

in Section 3. This model will be constructed and trained using Keras and Tensorflow APIs. Then the data obtained from preprocessing will be fed into the model that is distributed among multiple computing nodes to reduce training time. In Section 4, we will explain how asynchronous gradient computation reduces the model's training time by communicating and updating parameters values. Part (c) shows the validation study through which we obtain the brain activity pattern. We will describe this pipeline later in Section 5.

A non-distributed version of DCA model is elaborated in [43]. To facilitate the understanding of the model, we recapitulate the model in the following paragraphs. The purpose of autoencoder in DCA is to first encode the input fMRI time series by mapping them into higher level feature maps and then to decode the signals by reversing the process. Throughout this process, we obtain a hierarchical abstraction of fMRI signals while denoising them. As mentioned below, we assume that the model consists of only one convolutional layer both at the encoder and decoder and later we extend it to the real model. A summary of the key model parameters is shown in Table 1.

3.1. Encoder

The Encoder takes one 1D tfMRI signal x as shown in Fig. 2(b) and then by convolving the filters throughout the entire signal generates the feature map in the next layer using the following equation:

$$z_i = f(p_i * x + b_i) \tag{1}$$

Where x is the signal input, and p_i and b_i are the corresponding filter and bias for the ith feature map. f is the activation function. In this paper, except for the output convolutional layer in the decoder layer where we use linear activation function, we use the Rectified Linear Unit (ReLU) as activation functions. The advantages of choosing ReLU in our study is first to reduce the possibility of vanishing gradient and second to represent the signal sparsely as we later use the sparse representation of the hidden layer for data validation. A fully connected layer is used at the end of the encoder to match the encoder final hidden layer feature size with the input signal and to ensure that the hidden states are learned with a full receptive field of input as we use it as the final desired output of the model as mentioned in [43].

$$H = Z * W + C \tag{2}$$

In Eq. (2), the hidden layer states are represented by H, whereas Z, W and C are the feature maps, weight and bias of the fully connected layers, respectively.

3.2. Decoder

The decoder is following a symmetric pattern and attached to the previous encoder. To reconstruct the input signal, first, the hidden states are mapped and reshaped to a reconstructed version of feature maps Z^t via fully connected layer in the decoder. In Eq. (3), W^t and C^t denote the weights and bias of the fully connected layer in the decoder, respectively.

$$Z^t = H * W^t + C^t \tag{3}$$
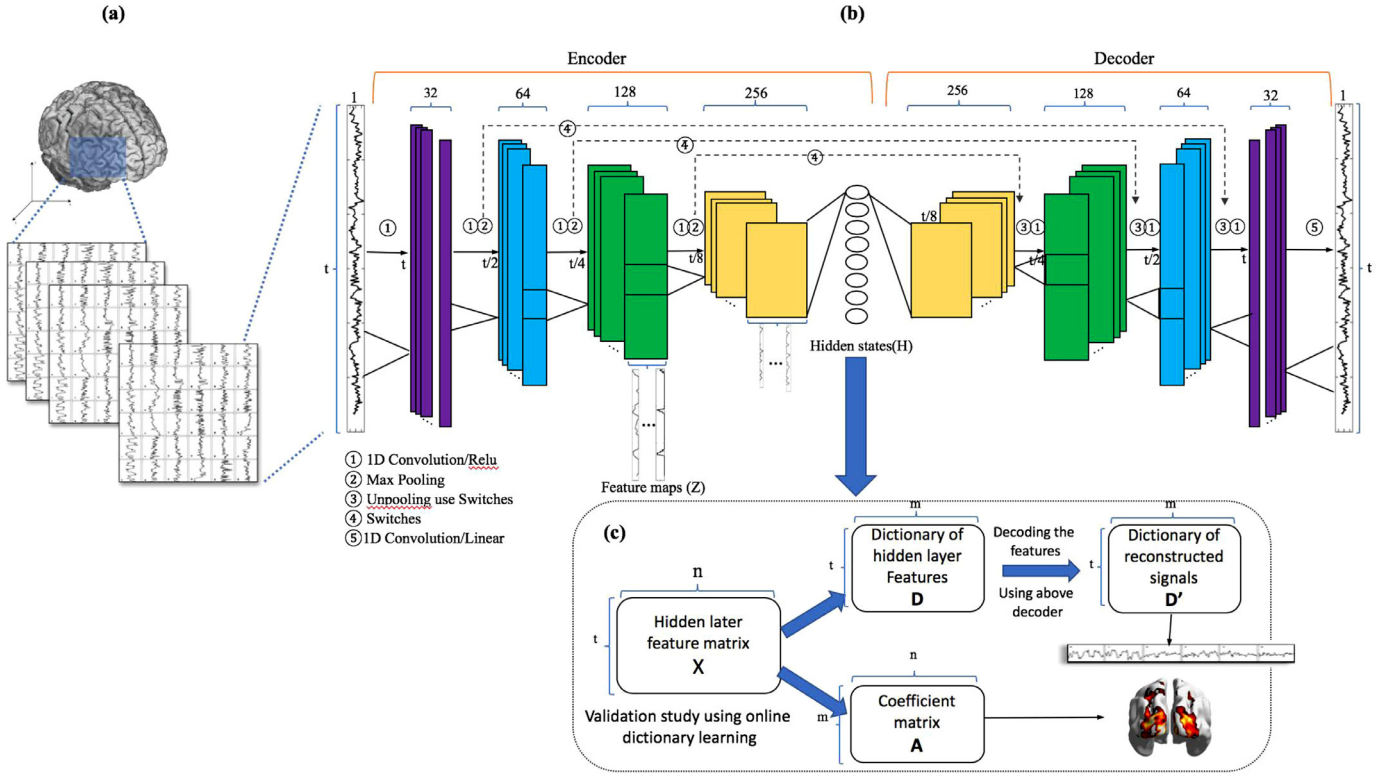
Fig 2. An illustration of the dist-DCA model and the online dictionary learning validation study. (a) demonstrates the preprocessing step of the tfMRI data including signal extraction and normalization. (b) demonstrates the structure of the dist-DCA model and its components including all hidden layers and feature maps. (c) shows the validation study through which we obtain the brain activity pattern.

In the end, input signal will be reconstructed by linearly combining these feature maps, where $\hat{x}$ denotes the reconstructed signal, and p_i^l and b_i^l are the filters and biases as shown in the following equation:

$$\hat{x} = \sum_i \left(p_i^{\ell} * z_i^{\ell} + b_i^{\ell} \right) \tag{4}$$

The same concept is extended to a model with more layers (4 layers in encoder and 4 in the decoder) by transforming the input layer into different feature map in each convolutional layer by a chain rule. To minimize the mean square error between fMRI signals and their reconstructions, we also used an L2 regularization term between feature maps in the top layer of the encoder and the bottom layer of the decoder. Doing so ensures us that the fully connected layer does not randomly shuffle the timing order when reconstructing features maps in the decoder. λ in Eq. (5) controls the significance of the L2 regularization term, and we experimentally set it to 0.006.

$$min \frac{1}{2} \left\| X - \widehat{X} \right\|_2^2 + \frac{1}{2}\lambda \left\| Z - Z^t \right\|_2^2 \tag{5}$$

3.3. Max-pooling and unpooling

The max pooling is applied on each layer after the convolutional layer. This helps first by substantially reducing the computational cost for the upper layer and second, by gaining translation-invariance. The translation-invariance is particularly important in tfMRI due to possible time-shift phenomena while acquiring the raw signal [38,34].

Given the invertible property of max-pooling, we utilized switches [39] in the encoder to memorize the location of the local max in each pooling regions and then we applied the location of the corresponding local max value to its original position. In val-

idation studies (Section 5) when "switches" are not available, we simply use traditional up-sampling.

In the next section, we explain how such a model is replicated among spark executor nodes.

4. Data parallelism and model deployment

In data parallel approaches, a copy of the entire model is sent to each executor, parallelizing processing of gradient descent by partitioning data into smaller subsets. A parameter server then combines the results of each subset and synchronize the model parameters between each executor after receiving gradient delta from each executor. This can be done synchronously or asynchronously. However, in homogeneous environments where nodes share the same hardware specifications and communicate via a reliable network of communication, asynchronous [22] methods outperform for two reasons [22,26]. First, executors do not wait for others to commit before start processing the next batch of data. Second, asynchronous method is more robust to failure of nodes as if one node fails the others will still train their own data partitions and fetch new updates from parameter server.

For example, given a batch size of 100 elements, 5 replicas of the model computes the gradient for 20 elements, and then combine the gradients in a separate node, known as parameter server, and apply parameters updates synchronously, in order to behave exactly as if we were running the sequential SGD algorithm with a batch size of 100 elements.

We have implemented downpour SGD [22] in our distributed framework and have fixed the η fetch and η push of weights and gradients to one, for speeding up convergence and for ease of comparison to simple SGD. Our experiment shows that relaxing consistency requirements are remarkably effective. Downpour SGD comes from the intuition that if we view the gradient descent as a water droplet toward minimizing the error rate, then individual
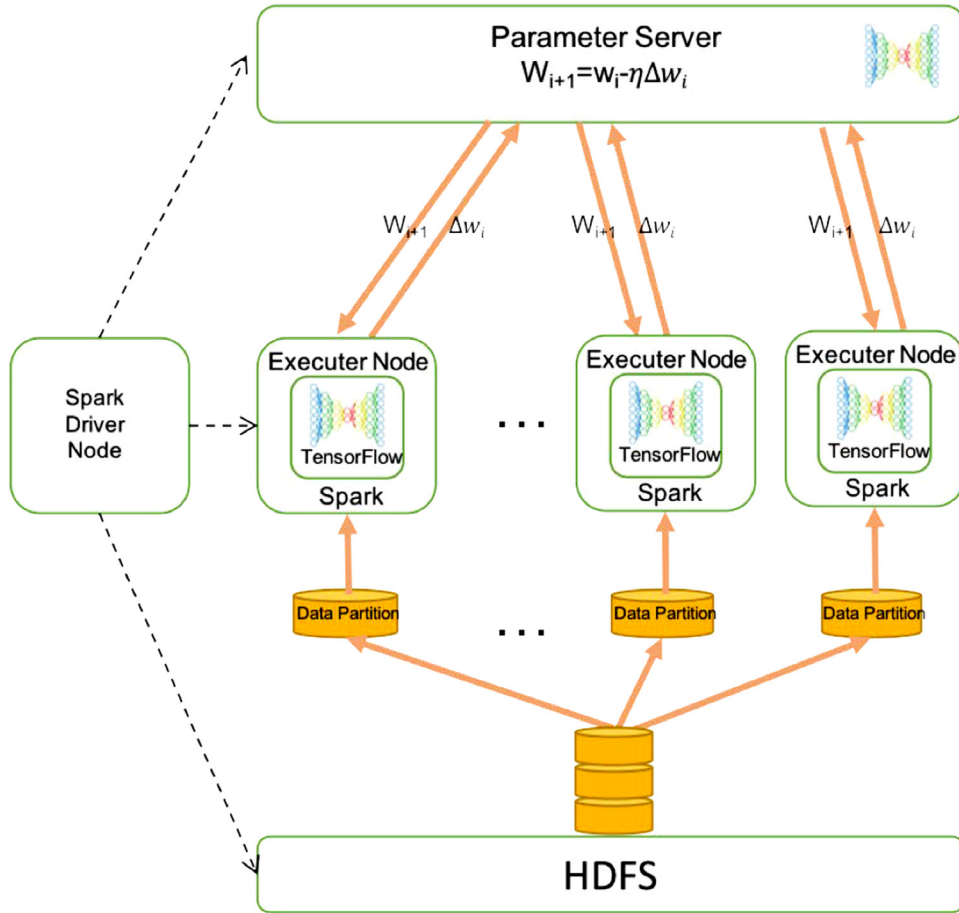
Fig 3. Dist-DCA. Executor nodes asynchronously fetch parameters *w* and push gradients to the parameter server. Spark driver is also in charge of data penalization and task scheduling.

executors can be considered as several droplets near each other, all separately flowing down into the same valley. Moreover, we practiced a warm-up phase, wherein a single executor node starts training on its own data partition before starting other executors. This has significantly decreased the probability of diverging of each executor being trapped in its own local optima.

We also chose the Adagrad optimizer [41] to keep the learning rate update for each parameter as the model is training and to ease extending the number of executing nodes. Adagrad uses a separate adaptive learning rate for each parameter. Let $\eta_{i,k}$ be the learning rate of the *i*th parameter at iteration K and $\Delta w_{i,k}$ its gradient, then in Eq. (6) we obtain $\eta_{i,k}$.

$$\eta_{i,k} = \gamma / \sqrt{\sum_{j=1}^{K} \left(\Delta w_{i,j} \right)^2} \qquad (6)$$

Because these learning rates are computed only from the summed squared gradients of each parameter, Adagrad is easily implemented locally within the parameter server. γ is the constant scaling factor for all learning rates, which is larger than the best fixed learning rate used without Adagrad. The use of Adagrad extends the maximum number of model replicas that can productively work simultaneously.

The abovementioned optimization procedures ideally address our problem in two ways. One is by empowering us to process massive fMRI data (nearly 10 millions tfMRI time series each with

284 time points as in this work). And, the other is by allowing us to train our relatively large model, consisting of more than 6 million trainable parameters, faster. As a result, our proposed dist-DCA benefits from asynchronous data parallelism through two main components of distributed data partitioning and distributed parameter optimization as it is shown in the Fig. 3. We use Hadoop as our main distributed file system and Spark for tasks scheduling and data partitioning. Each Spark executor acts as a wrapper of Tensorflow application where one node handles the parameter synchronization and the rest run the Tensorflow application independently just as one single node setup. Each executor commits its gradient delta to parameter server after each processing batch elements and receives the latest parameter from the server. Meanwhile, Spark core efficiently feeds each of the executors through HDFS by partitioning the data based on the number of epochs and dataset size. Fig. 4 shows Spark data partitioning among a cluster of 16 nodes consisting of one driver, two parameter servers, and 13 executors. Spark driver is responsible for handling tasks and for replicating TensorFlow model across a cluster. For each stage and each partition, tasks are created and sent to the executors. If the stage ends with a shuffle, the tasks created will be shuffle-map tasks. After all tasks of a particular stage are completed, the driver creates tasks for the next stage and sends them to the executors, and so on. This repeats until the last stage, where the results return to the driver. With the asynchronized implementation, we ensure that both the model replicas and data partitions are run independently, thus reducing the delays induced by the loaded executors.
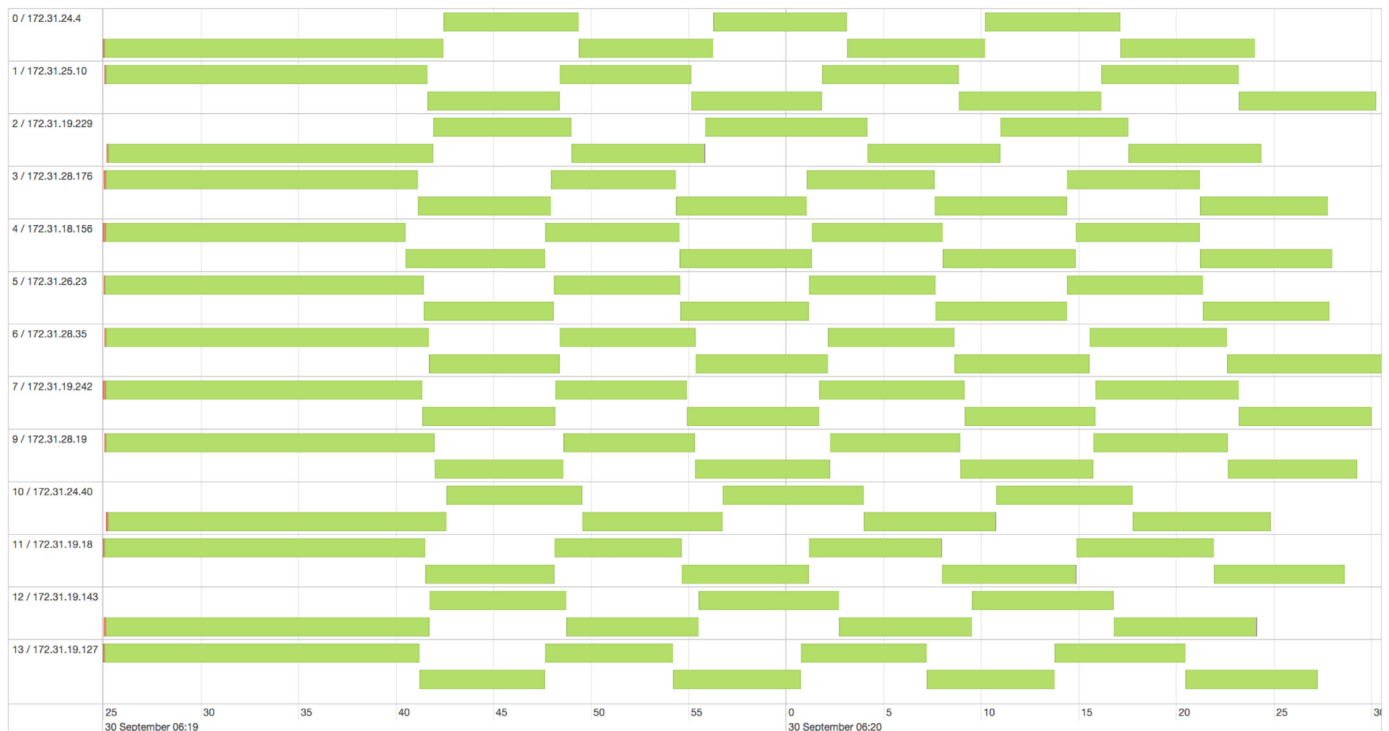
Fig 4. Dist-DCA data partitions. Spark driver keeps track of data partitions and executors' computational times. Here you can see the computational time of all active nodes and how the training tasks are scheduled.

5. Experiments

We evaluated the TensorFlow on Spark performance and scalability by our novel distDCA model using Amazon Elastic Cloud Computing (EC2). We trained this model on 9,658,464 fMRI time series of 48 human subjects and evaluated on 4,024,360 time series of 20 separate subjects.

5.1. Experiments setup

5.1.1. Dataset

We use the Human Connectome Project (HCP) Q1 release dataset [36] containing 68 healthy subjects' tfMRI data. The HCP dataset is advantageous in its high temporal and spatial resolution (TR = 0.72 s, varied temporal length from 176 to 1200 volumes; 2 mm isotropic voxels, to the total of over 201,218 voxels' signals per subject each with the length of 284 time points), which enables more detailed characterization of the brain's functional behavior. We use motor task fMRI data in this study, composed of six most basic motor tasks including visual cues (event 1), tapping left (right) fingers (event 2, 3), squeezing left toes (event 4, 5) and moving tongue (event 6). We divided the Motor task Q1 subjects into two separate subsets of 48 training and 20 validating subjects. For running the dist-DCA model, the preparation steps include fMRI signal pre-processing (gradient distortion correction, motion correction, bias field reduction, and high pass filtering) [37], all implemented using FSL FEAT. Furthermore, we recruited our integrated neuroinformatic platform, HELPNI [35], to facilitate the pre-processing and to integrate different steps of data acquisition using its powerful pipelining ability.

5.1.2. Cloud platform

The dist-DCA model is deployed on Amazon Web Service Elastic Cloud Computing, AWS EC2. EC2 clusters are highly scalable, as the

number of executor nodes could be adjusted effortlessly within the cluster. The pre-processed and converted fMRI data was stored in the cloud through Amazon S3 and accessible by the EC2 clusters. This enables us to pull data to newly initialized instances easily. We used customized scripts along with an AMI containing a pre-configured instance to scale our cluster according to desire. Each cluster's node contains Apache Spark version 2.2.0, Hadoop version 2.6.0, TensorFlow 1.3, Keras 2.08 and python 2.7. To benchmark the scalability and robustness of our proposed framework, we used a variety of node hardware configurations with a different number of node per experiment as summarized in Table 2. The configuration of nodes are as follows. G3 nodes are equipped with High-Frequency Intel Xeon E5-2686 v4 (Broadwell) processors, NVIDIA Tesla M60 GPU, with 2048 parallel processing cores and 8 gigabytes of video memory per GPU with 25 Gbps of aggregate network bandwidth within the cluster. G2 nodes come with Intel Sandy Bridge processors, NVIDIA Kg20 Grid GPU with 1536 CUDA cores and 4 gigabytes of memory per GPU.
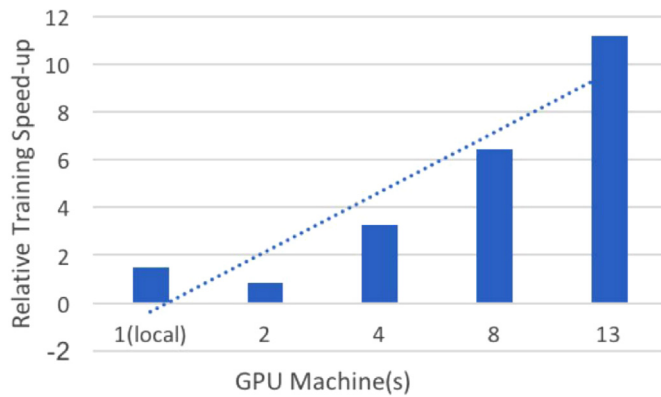
5.2. Performance

We aimed to investigate the performance of our framework with respect to the mean processing time of a single mini-batch (1 fMRI signal) for Downpour SGD with Adagard training as a function of the number of nodes used in a single model instance. To do so, we deployed four clusters of G3 instances with 4, 6, 8 and 16 nodes (correspond to 2, 4, 8 and 13 TensorFlow executors, respectively, as shown above). Given the broadband network communications, except for the 16-node cluster with two parameter servers, we only dedicated one parameter server along with one spark driver. Moreover, to evaluate the effect of network traffic on training speed, we ran a non-distributed version (called DCA) of the model on a single node with the same configuration. In all the experiments, we trained our models (dist-DCA and standalone
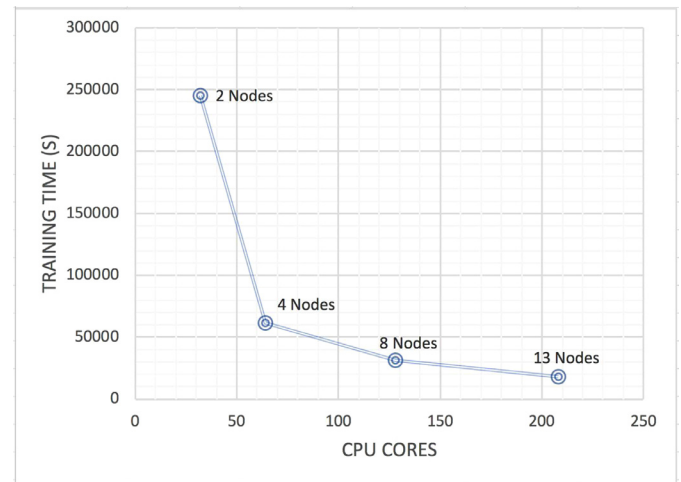
Table 2
Cloud clusters' configuration, each line represents a separate experiment setup.

No. of spark/TF executors	vCPU Cores/node	Used GPU memory per node (GB)	Memory per node (GB)	EC2 node type
1	16	8	122	G3-4x
2	16	8	122	G3-4x
4	4	12	61	P2-x
4	8	4	15	G2-2x
4	16	8	122	G3-4x
4	32	4	60	G2-8x
8	16	8	122	G3-4x
13	16	8	122	G3-4x

Fig 5. Training speed-up versus the numbers of local/spark executors.

Fig 6. Training time of dist-DCA based on the number of CPU cores on different cluster setups.

DCA) on 9,658,464 time series of HCP Q1 data for 1600 batches and 6036 steps per epoch.

Fig. 5 demonstrates the speed of various implementations including our standalone and distributed ones on GPU nodes. Since the standalone DCA has no data-parallelism and no network overhead, it obviously outperforms the two-node cluster. However, clusters with the higher number of executor nodes easily exceed regarding computation time. For example, the cluster with 13 executors outperforms the standalone model with almost seven times. It can be observed that training speed linearly grows as the number of executor nodes increase. However, we expect that the performance drops if we increase the number of executor nodes to more than 13. This happens as network overhead starts to rule over our dist-DCA model performance and as executor nodes will have fewer tasks to do while waiting to fetch new parameters.
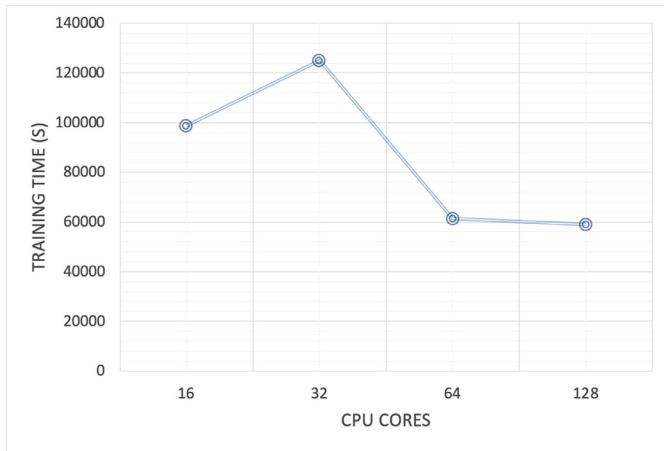
5.3. Scalability

To further demonstrates the scalability of our implemented distributed framework, we measured the training time of dist-DCA on the previously discussed dataset. We trained our model in 4 different cluster settings with 2, 4, 8 and 13 G3 executor nodes with a total of 2, 4, 8 and 13 GPUs, respectively. Please note that for the sake of comparison, in all experiments, only one GPU per node was used. Our goal is to obtain minimum loss in the minimum amount of training time. Fig. 6 illustrates that the training time is reduced significantly by almost 51 h in a four-executor with 64 CPU cores (4 GPUs) compared to a two-executor cluster with 32 CPU cores (2 GPUs). However, this increased rate does not hold from the four-node to the eight-node cluster with 128 cores of CPU (8 GPUS). We believe that this is due to network communication overhead and

previously discussed warmup phase. As explained in Section 4, the TensorFlow application (here dist-DCA model) is wrapped inside a spark executor at each node. Executors independently start to train the model by pushing gradients and fetching the new parameters from the parameter server at each stage. These recurring network communications can cause the larger clusters to not linearly scale-up as opposed to the ones with fewer nodes. We can conclude that network can always be a bottleneck in larger clusters.
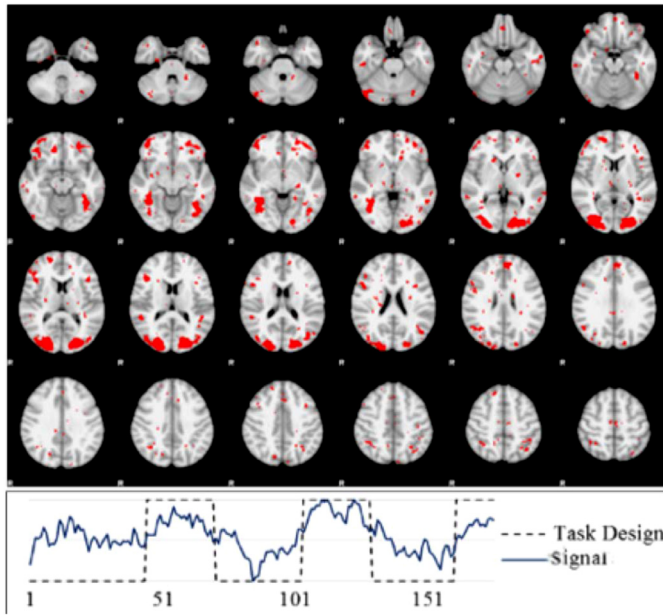
We also performed another experiment solely to evaluate the effect of CPU cores on our proposed framework performance. To do so, we launched 4 clusters each with 4 nodes to train dist-DCA model over our HCP data. In each cluster setup, we used the same environmental setup and one GPU card per node (total of 4 GPUs). Clusters utilized 4, 8, 16 and 32 CPU cores per node, respectively. Demonstrated results in Fig. 7 suggest that increasing only the number of CPUs would not benefit the training speed significantly. A simple comparison of the results in Fig. 5 with Fig. 7 shows that while increasing the number of GPUs in a distributed setup reduces training time significantly; such a conclusion cannot be drawn as opposed to increasing CPU cores.

5.4. Learned features validation and visualization

To validate the learned features of our proposed model, we have performed a validation study on the hidden layer features of the encoder. An illustration of the computational procedure of this validation study is shown in the Fig. 2(c). The rationale behind this is to compare the detected task-related patterns of brain

Fig 7. Training time of dist-DCA based on the number of CPU cores in clusters with the same number of nodes.

Fig 8. Top: spatial pattern visualized on cortical volumetric space of one decomposed network. Bottom: visualization of its temporal pattern.

prehensive collection of concurrent functional networks in the human brain [10]. The spatial and temporal pattern of a sample network decomposed results is demonstrated in Fig. 8.

To draw a fair comparison, we have used the same parameters in both runs. We adopted the parameter-tuning approach that Lv et al. suggested [10]. Both setups learned 400 dictionaries with 0.7 sparsity regularizer [42] to achieve the best performance of the brain network inference. After training, the high-level features of setup one are decomposed as high-level dictionaries and corresponding spatial distributions. Then we use the decoder to project these high-level dictionaries (time series patterns) back to the signal space. The detected patterns are visualized in Fig. 9. As shown on the right side of the figure, although the dictionary learning analysis in both setups has detected all the six motor task patterns, these patterns are mixed with a large number of noises in setup 2, and as a result, the correlation values with task design pattern are quite small. On the other hand, the setup 1 contained much fewer noises in both of the time series patterns and spatial maps. Consequently, we can conclude that our proposed model filters noises in each layer and preserves the useful information of the brain activities. For the sake of simplicity and page limitation, we do not explain the theoretical brain model analysis and reconstruction error analysis. Further details of this comparison can be found at Huang et al. work [43].
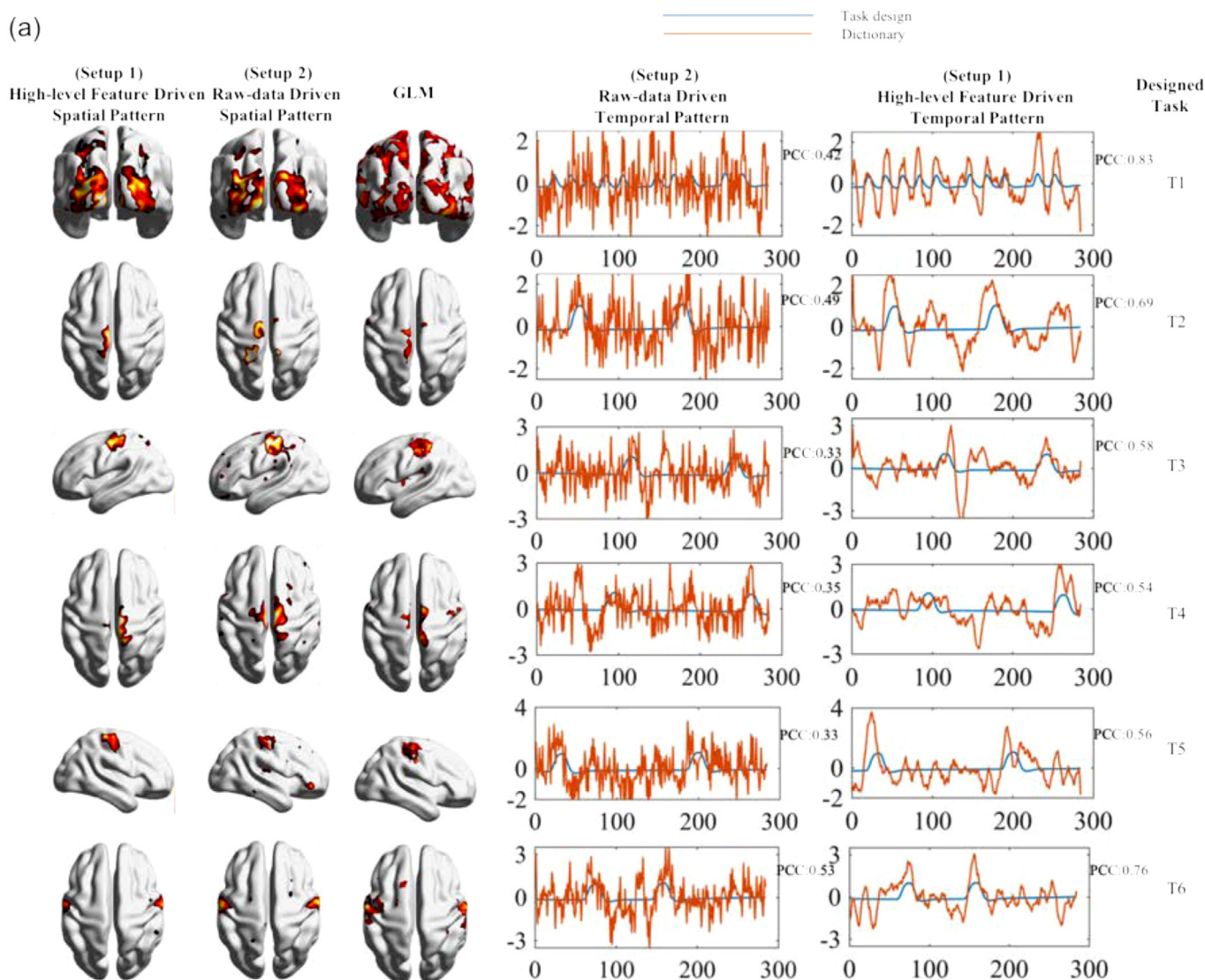
Furthermore, we visualized the filters in each layer. Fig. 10 shows all 32 filters in the first layer of the encoder. The first layer filters summarized the most common sub-shapes of tfMRI time. For example, sinuous and bowl patterns of fMRI are shown with arrows at Fig. 10.

6. Conclusion

Providing an effective model to represent the large scale tfMRI data to break down the intrinsic complex structure of tfMRI signals has been highly demanded yet challenging. A novel deep learning model along with distributed computing are the keys to transforming our understanding of some of the most complicated brain signals [43]. In this work we presented a novel scalable distributed deep convolutional autoencoder that hierarchically models largescale tfMRI time series data while gaining a higher level abstraction of the tfMRI signal. We used Apache Spark and TensorFlow as the computational engines to parallelize millions of fMRI time series and to train our model over large cluster of GPUs. Our experiment results showed that such a model can effectively scale-up to dozens of computation nodes, processing extensive dataset over hundreds of computational cores. The significance of network overhead, however, can severely impact the training time. Furthermore, our results showed that the high-level features are superior in task-related regions detection. The proposed autoencoder was also able to denoise the tfMRI signal as the learned dictionary atoms by our novel high-level sparse dictionary learning suggests. In general, our work contributes a novel deep convolution autoencoder framework for fMRI data modeling with significant application potentials in cognitive and clinical neuroscience in the future.

In our future work, we plan to perform further tests to implement a parallel version of our model to use the computational power of multi-GPU on a multi-node distributed setting to maximize the performance. We also plan to use the 1200+ available subjects of all HCP releases including acquisitions of different types of tasks to identify brain areas in a wide range of neural systems (such as Relation, Working Memory, Language, Social Interaction, Motor, etc.). This will benefit from our proposed distributed

activity through a sparse dictionary learning method in two setups. One by feeding the high-level features of the hidden layer (setup 1) contained by dist-DCA and the other with the raw tfMRI signals (setup 2). Sparse dictionary learning as an unsupervised learning algorithm aims at finding a sparse representation of input data in the form of a linear combination of basic elements, known as dictionaries along with their corresponding coefficients. [10,44,45]. This goal is achieved by aggregating fMRI signals into an overcomplete dictionary matrix and a corresponding coefficient matrix through an effective online dictionary learning algorithm [42]. The time series of each over completed dictionary represents the temporal activity of a brain network, and its corresponding reference weight vector stands for the spatial map of every network. This method is recognized as an efficient method for inferring a com-

Fig 9. Validation study of the dist-DCA. Comparing the temporal and spatial patterns of 6 motor tasks driven from high-level features and raw-data. GLM is for reference. Pearson correlation of the designed tasks with learned dictionary atoms is shown as PCC value.
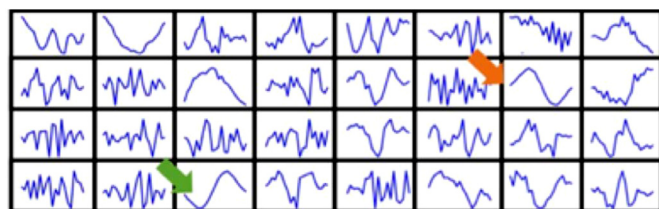
Fig 10. All 32 filters in the first layer of encoder. Arrows show the most common pattern of tfMRI time series.

model, enabling data-driven hierarchical neuroscientific discovery from massive fMRI big data in the future.

Acknowledgments

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.neucom.2018.09.066.

References

[1] S. Choudhury, J.R. Fishman, M.L. McGowan, E.T. Juengst, Big data, open science and the brain: lessons learned from genomics, Front. Hum. Neurosci. 8 (2014) 239.

[2] D.C. Van Essen, S.M. Smith, D.M. Barch, T.E. Behrens, E. Yacoub, K. Ugurbil, WU-Minn HCP Consortium, The WU-Minn human connectome project: an overview, Neuroimage 80 (2013) 62–79.

[3] M. Mennes, B.B. Biswal, F.X. Castellanos, M.P. Milham, Making data sharing work: the FCP/INDI experience, NeuroImage 82 (2013) 683–691, doi:10.1016/j.neuroimage.2012.10.064.

[4] R.A. Poldrack, D.M. Barch, J.P. Mitchell, T.D. Wager, A.D. Wagner, J.T. Devlin, C. Cumba, O. Koyejo, M.P. Milham, Toward open sharing of task-based fMRI data: the OpenfMRI project, Front. Neuroinform. 7 (2013) 12, doi:10.3389/fninf.2013.00012.

[5] V. Abolghasemi, S. Ferdowsi, S. Sanei, Fast and incoherent dictionary learning algorithms with application to fMRI, Signal Image Video Process. 9 (1) (2015) 147–158.

[6] B.A. Ardekani, I. Kanno, Statistical methods for detecting activated regions in functional MRI of the brain, Magn. Reson. Imaging 16 (10) (1998) 1217–1225.

[7] A.H. Andersen, D.M. Gash, M.J. Avison, Principal component analysis of the dynamic response measured by fMRI: a generalized linear systems framework, Magn. Reson. Imaging 17 (6) (1999) 795–815.

[8] P.A. Bandettini, A. Jesmanowicz, E.C. Wong, J.S. Hyde, Processing strategies for time-course data sets in functional MRI of the human brain, Magn. Reson. Med. 30 (2) (1993) 161–173.

[9] M.J. McKeown, T.J. Sejnowski, Independent component analysis of fMRI data: examining the assumptions, Hum. Brain Mapp. 6 (1998) 368–372.

[10] J. Lv, X. Jiang, X. Li, D. Zhu, S. Zhang, S. Zhao, H. Chen, T. Zhang, X. Hu, J. Han, J. Ye, L. Guo, T. Liu, Holistic atlases of functional networks and interactions reveal reciprocal organizational architecture of cortical function, IEEE Trans. Biomed. Eng. 62 (2015) 1120–1131.

[11] K.J. Friston, A.P. Holmes, K.J. Worsley, J.-P. Poline, C.D. Frith, R.S.J. Frackowiak, Statistical parametric maps in functional imaging: a general linear approach, Hum. Brain Mapp. 2 (1995) 189–210.

[12] L. Ferrarini, I.M. Veer, E. Baerends, M.J. van Tol, R.J. Renken, N.J. van der Wee, D.J. Veltman, A. Aleman, F.G. Zitman, B.W. Penninx, M.A. van Buchem, Hierarchical functional modularity in the resting-state human brain, Hum. Brain Mapp. 30 (7) (2009) 2220–2231.

[13] D. Meunier, R. Lambiotte, A. Fornito, K. Ersche, E.T. Bullmore, Hierarchical modularity in human brain functional networks, Front. Neuroinform. 3 (2009) 37.

[14] W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bull. Math. Biophys. 5 (4) (1943) 115–133.

[15] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. (2012) 1097–1105.

[16] J. Sun, H. Li, Z. Xu, Deep ADMM-Net for compressive sensing MRI, Adv. Neural Inf. Process. Syst. (2016) 10–18.

[17] D.C. Cireşan, A. Giusti, L.M. Gambardella, J. Schmidhuber, Mitosis detection in breast cancer histology images with deep neural networks, in: Proceedings of the International Conference on Medical Image Computing and Computer-assisted Intervention, Springer, Berlin, Heidelberg, 2013, pp. 411–418.

[18] S.C. Turaga, et al., Convolutional networks can learn to generate affinity graphs for image segmentation, Neural Comput. 22 (2) (2010) 511–538.

[19] Y. Taigman, et al., Deepface: closing the gap to human-level performance in face verification, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014.

[20] Y. Pu, Z. Gan, R. Henao, X. Yuan, C. Li, A. Stevens, L. Carin, Variational autoencoder for deep learning of images, labels and captions, Adv. Neural Inf. Process. Syst. (2016) 2352–2360.

[21] J. Masci, U. Meier, D. Cireşan, J. Schmidhuber, Stacked convolutional auto-encoders for hierarchical feature extraction, in: Proceedings of the International Conference on Artificial Neural Networks, Springer, Berlin, Heidelberg, 2011, pp. 52–59.

[22] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A.Y. Ng, Large scale distributed deep networks, Adv. Neural Inf. Process. Syst. (2012) 1223–1231.

[23] D.C. Ciresan, U. Meier, L.M. Gambardella, J. Schmidhuber, Deep big simple neural nets excel on handwritten digit recognition, CoRR (2010).

[24] A. Coates, A. Ng, H. Lee, An analysis of single-layer networks in unsupervised feature learning, in: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, 2011, pp. 215–223.

[25] Q.V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, A.Y. Ng, On optimization methods for deep learning, in: Proceedings of the Twenty-eighth International Conference on International Conference on Machine Learning, Omnipress, 2011, pp. 265–272.

[26] S. Zhang, A.E. Choromanska, Y. LeCun, Deep learning with elastic averaging SGD, Adv. Neural Inf. Process. Syst. (2015) 685–693.

[27] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing, in: Proceedings of NSDI, 2012, pp. 15–28.

[28] P.H. Chen, X. Zhu, H. Zhang, J.S. Turek, J. Chen, T.L. Willke, P.J. Ramadge, A convolutional autoencoder for multi-subject fmri data aggregation (2016) arXiv:1608.04846.

[29] S.M. Plis, D.R. Hjelm, R. Salakhutdinov, E.A. Allen, H.J. Bockholt, J.D. Long, V.D. Calhoun, Deep learning for neuroimaging: a validation study, Front. Neurosci. 8 (2014) 229.

[30] H.-I. Suk, C.-Y. Wee, S.-W. Lee, D. Shen, State-space model with deep learning for functional dynamics estimation in resting-state fMRI, Neuroimage 129 (2016) 292–307.

[31] H. Huang, X. Hu, J. Han, J. Lv, N. Liu, L. Guo, T. Liu, Latent source mining in FMRI data via deep neural network, in: Proceedings of the 2016 IEEE Thirteenth International Symposium on Biomedical Imaging (ISBI), IEEE, 2016, pp. 638–641.

[32] D. Ren, Y. Zhao, H. Chen, Q. Dong, J. Lv, T. Liu, 3-D functional brain network classification using convolutional neural networks, in: Proceedings of the 2017 IEEE Fourteenth International Symposium on Biomedical Imaging (ISBI 2017), IEEE, 2017, pp. 1217–1221.

[33] H. Wen, et al., Neural encoding and decoding with deep learning for dynamic natural vision, arXiv preprint (2016). arXiv:1608.03425

[34] A. Feng, TensorFlowOnSpark, GitHub repository, 2016. https://github.com/yahoo/TensorFlowOnSpark

[35] M. Makkie, et al., HAFNI-enabled largescale platform for neuroimaging informatics (HELPNI), Brain Inform. 2 (4) (2015) 225–238.

[36] D.C. Van Essen, S.M. Smith, D.M. Barch, T.E.J. Behrens, E. Yacoub, K. Ugurbil, The WU-Minn human connectome project: an overview, NeuroImage 80 (0) (2013) 62–79, doi:10.1016/j.neuroimage.2013.05.041.

[37] D.M. Barch, G.C. Burgess, M.P. Harms, S.E. Petersen, B.L. Schlaggar, M. Corbetta, D. Nolan, Function in the human connectome: task-fMRI and individual differences in behavior, Neuroimage 80 (2013) 169–189.

[38] K.J. Friston, L. Harrison, W. Penny, Dynamic causal modelling, Neuroimage 19 (4) (2003) 1273–1302.

[39] M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: Proceedings of the European Conference on Computer Vision, Springer, Cham, 2014, pp. 818–833.

[40] J. HermansCERN IT-DB, Distributed Keras: Distributed Deep Learning with Apache Spark and Keras,, GitHub repository, 2016. https://github.com/JoeriHermans/dist-keras.

[41] J.C. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (2011) 2121–2159.

[42] J. Mairal, F. Bach, J. Ponce, G. Sapiro, Online dictionary learning for sparse coding, in: Proceedings of the Twenty-sixth Annual International Conference on Machine Learning, ACM, 2009, pp. 689–696.

[43] H. Huang, X. Hu, Y. Zhao, M. Makkie, Q. Dong, S. Zhao, T. Liu, Modeling task fMRI data via deep convolutional autoencoder, in: Proceedings of the IEEE Transactions on Medical Imaging, 2017.

[44] M.W. Weiner, D.P. Veitch, P.S. Aisen, L.A. Beckett, N.J. Cairns, R.C. Green, D. Harvey, et al., The Alzheimer's disease neuroimaging Initiative: a review of papers published since its inception, Alzheimer's Dementia 9 (5) (2013) e111–e194.

[45] X. Li, M. Makkie, B. Lin, M.S. Fazli, I. Davidson, J. Ye, T. Liu, S. Quinn, Scalable fast rank-1 dictionary learning for fmri big data analysis, in: Proceedings of the Twenty-second ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 511–519.

[46] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, S. Ghemawat, Tensorflow: large-scale machine learning on heterogeneous distributed systems, arXiv preprint (2016).

[47] Max Pumperla, elephas (2015), GitHub repository https://github.com/maxpumperla/elephas.

[48] Deeplearning4j Development Team. Deeplearning4j: open-source distributed deep learning for the JVM, Apache Software Foundation License 2.0. http://deeplearning4j.org

[49] N. Zeng, Z. Wang, H. Zhang, W. Liu, F.E. Alsaadi, Deep belief networks for quantitative analysis of a gold immunochromatographic strip, Cognit. Comput. 8 (4) (2016) 684–692.

[50] N. Zeng, H. Zhang, B. Song, W. Liu, Y. Li, A.M. Dobaie, Facial expression recognition via learning deep sparse autoencoders, Neurocomputing 273 (2018) 643–649.

Milad Makkie received his Ph.D. in computer science from the University of Georgia. He earned his bachelors and Master's degree in biomedical engineering, from the Azad University and the Amirkabir University of Technology respectively. In 2009, he started leading a company that became one of the top tele-medicine and e-health providers. His research interests include financial data analytics, distributed processing, cloud computing, machine learning and big-data analytics.

Heng Huang is a Ph.D. student of Northwestern Polytechnical University, China. His research Interests include fMRI, Data mining and artificial intelligence.

Yu Zhao is a Ph.D. at the Computer Science department in the University of Georgia and is working as a research assistant in CAID lab. His research interests focus on brain functional network analysis, using computer science techniques including big data solutions, machine learning and deep learning.

Tianming Liu is a Distinguished Research Professor of Computer Science at The University of Georgia. His research interest focuses on brain imaging and mapping. He has published over 280 peer-reviewed papers in this area. Dr. Liu is the recipient of the NIH Career Award and the NSF CAREER award, both in the area of brain mapping.

Athanasios V. Vasilakos received his Ph.D. degree in computer engineering from the University of Patras, Patras, Greece, in 1988. He served or is serving as an Editor or/and a Guest Editor for many technical journals, such as the IEEE Transactions on Network and Service Management, the IEEE Transactions on Cloud Computing and many more.