Master in Photonics

MASTER THESIS WORK

HIERARCHICAL REPRESENTATIONS IN MACHINE LEARNING AND MANY-BODY QUANTUM PHYSICS

Raúl Blázquez García

Supervised by Prof. Dr. Antonio Acín (ICREA, ICFO) and Dr. Peter Wittek (ICFO)

Presented on date 27th OCTOBER 2017

Registered at

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Hierarchical Representations in Machine Learning and Many-Body Quantum Physics

Raúl Blázquez García[1]

Supervised by: Peter Wittek[1] **and Antonio Acín**[1,2]

[1]ICFO-Institut de Ciencies Fotoniques,The Barcelona Institute of Science and Technology, 08060 Castelldefels (Barcelona), Spain

[2]ICREA, Passeig Lluis Companys 23, 08010 Barcelona, Spain

E-mail: `raulbzga@gmail.com`

Abstract. Over the past few years a number of proofs have emerged revealing the many connections between the methods used in quantum-many body physics and those in machine learning. In particular, much attention has been given to tensor networks (TNs) and deep learning architectures which exhibit striking similarities. Finding those similarities has helped us gain a better understanding on why deep learning architectures have so much expressive efficiency. Recently, machine learning techniques have been used to approximate many-body physics problems. Conversely, TNs have been used for machine learning tasks. For example, state-of-the-art research has used one-dimensional TNs to solve image recognition problems with very limited scalability. The scalability problem, however, can be overcomed by using a two-dimensional hierarchical TNs and a training algorithm derived from the multipartite entanglement renormalization ansatz (MERA). Here we give further analysis of the inner structural resemblance between such hierarchical tensor networks (TNs) and pretrained convolutional neural networks (CNNs): this was achieved by analyzing their abstraction power layer by layer.

Keywords: Hierarchical representations, CNN,Tree Tensor Networks(TTNs), Manifold Learning, t-SNE.

1. Introduction

Deep neural networks are a valuable asset for a wide variety of tasks [1, 3]. Their ability to approximate most functions much more efficiently than shallow neural networks has been an important factor for this [2]. Recently, CNNs specific type of deep neural networks have shown outstanding results in classification tasks. Since it is difficult to phrase any rule to design the best architecture for a specific problem, gaining a better understanding of CNNs is a topic of primary importance. This has lead to find connections between CNNs and other hierarchical representations. One of them, TNs, have been used for a long time in many-body quantum physics to find coarser representations of quantum states. Being able to give coarser representations of the quantum states is critical in many-body quantum physics problems. The exponential growth of the Hilbert space with the number of particles, makes computations impossible even for systems as small as $\mathcal{O}(10^2)$ particles.[4] Not are only TNs interesting as a powerful numerical tool in many-body quantum physics, but they have also been proven to bear a surprising resemblance with CNNs.

Recent work proves that deep neural networks can efficiently represent most physical states [3]. Likewise, TNs have been applied to machine learning problems [5]. A matter of interest is then how well can TNs perform at classification tasks. The aim of this work is to find further evidence of the equivalence between DNNs and TNs and gain understanding of their architectural features as an integral part of the research conducted in [6]. Here we establish relations between CNNs and hierarchical TNs by analyzing their abstraction power when doing classification tasks. To conduct analysis, we will use manifold learning techniques.

2. Convolutional Neural Networks

CNNs are hierarchical neural networks that have a great performance in identifying particular patterns in data that are common to a certain class. That makes them perform with precision at classification tasks [12]. A CNN has 3 types of layers: convolutional layers, pooling layers, and fully connected layers

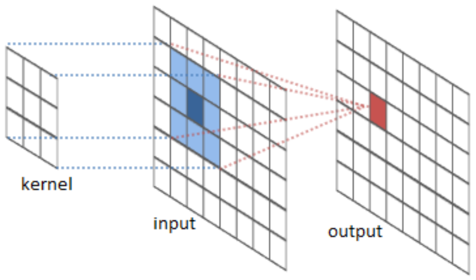


Figure 1: A sketch of the sliding kernel matrix and the input layer generating the output convolutional layer.

2.1. Convolutional layers and filters

In a convolutional layer the value of each of it's neurons is computed as a combination of weighted values of the neurons within a subset of neurons of the previous layer. The process is then repeated for the following neuron of the convolutional layer, using a different subset of input neurons. Some of the neurons on the first subset will belong to the second subset as well (there will be some overlapping). It can be thought as a sliding window that selects a particular part of the input data computes the value of a neuron.

Then it moves a step to compute the next neuron and repeats the process until it has calculated the value of the whole layer. The whole process is repeated with different sets of weights, every set works as a filter that extracts a particular feature of the data. Each filter therefore, generates an output layer. After every convolutional layer the network will spread in as many layers as inputs times filters it has. Convolutional layers can be stack enabling the recognition of higher-level or more abstract features [14, 15].

2.2. Pooling layers

They have the role of extracting the features that better characterize our data instance, and get rid of the rest. Pooling layers have the role of dividing a previous layer in non-overlapping regions and keep only the neurons with maximum value within each region.

2.3. Fully connected Layers

Fully connected layer are always the last step in a CNN classifier and their output is already a class.

3. Hierarchical Tensor Decompositions

3.1. Entanglement entropy

Let us consider a system described by the density matrix ρ which we split into two parts A and A^c. The reduced density matricx ρ_A of the system a is defined by traicing out hte degrees of fredom in the susbsystem A^c,

$$\rho_A = Tr_{A^c}[\rho]. \tag{1}$$

Then the entanglement entropy of subsystem A is given by the *von Neumann entropy* which is defined as

$$\begin{aligned} s(\rho) &= -Tr[\rho\ log(\rho)] \\ &= -\sum_{i=1}^{N} \rho_i\ log(\rho_i). \end{aligned}$$

Entanglement entropy is a measure of how much the grownd state is entangled between the subsstem A and the subsystem A^c.
Entangled states can be non-local, meaning that having full knowledge of a state requires information that depends on parts outside it. Looking only into the state itself would lead to a lack of information about it, that could be avoided by looking at the whole system instead. Von Neuman entropy is a measure of the mixedness of a density matrix. It always takes positive values and is 0 for pure states and log(N) for maximally entangled states. An important property of those systems is the sub-additivity:

$$S(\rho) \leq S(\rho_A) + S(\rho_{A^C}). \tag{2}$$

Note that if the entanglement entropy measures the lack of information of the subsystem, the lack of information has to be greater when we analyze separately the parts of the system, than when we treat them as a whole.

3.2. Real-space Renormalization

Renormalization is a method for finding adequate descriptions off a system when looking at it from different length scales. In many-body physics it can help calculating some properties of the ground state of a system, which are otherwise computationally infeasible.

3.3. Tree Renormalization

For understanding Tree Renormalization (see Fig.1) let us consider a lattice $\mathcal{L}$ with n lattice sites. In order to find the ground state of the system we will do a coarse-graining transformation on it. We will consider a subset of sites of the lattice $\beta \subset \mathcal{L}$. Each lattice point s has a state space V_s therefore the state space of β is

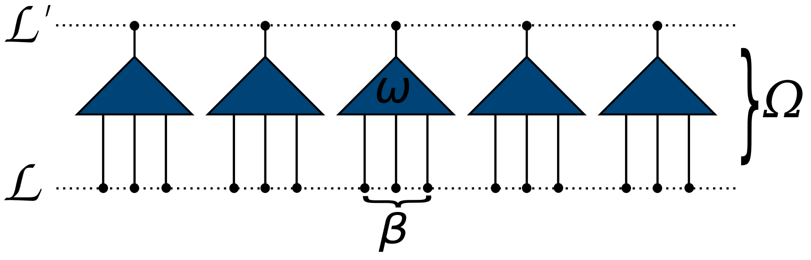


Figure 2: A coarse-graining transformation Ω of a 1-dimensional lattice $\mathcal{L}$ to the coarser lattice $\mathcal{L}'$; where each ω connects a subset β of lattice sites in $\mathcal{L}$ to a single site in $\mathcal{L}'$.

$$V_\beta = \bigotimes_{s \epsilon \beta} V_s. \tag{3}$$

The objective is to find a coarse grained lattice $\mathcal{L}'$ such that each $s' \epsilon \mathcal{L}'$ corresponds to a subset of sites in $\mathcal{L}$ that we defined as β. With that purpose in mind, we introduce a tensor ω.

$$\omega : V_{s'} \to V_\beta \;\Rightarrow\; \omega^\dagger : V_\beta \to V_{s'} \tag{4}$$

We will also have a tensor Ω such:

$$\Omega = \bigotimes_{s' \epsilon \mathcal{L}'} \omega \tag{5}$$

$$\Omega : V_{\mathcal{L}'} \to V_{\mathcal{L}} \;\Rightarrow\; \Omega^{\dagger} : V_{\mathcal{L}} \to V_{\mathcal{L}'} \tag{6}$$

A TTN is a stack of coarsgrainig operations in a hierarchical structure (see Fig. 3).

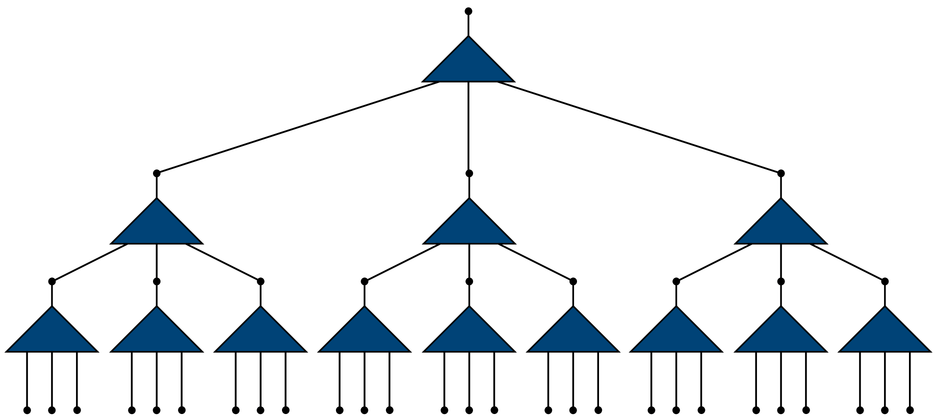


Figure 3: TTN scheme where entanglement between sites is treated diferently depending on the position.

3.3.1. Hilbert spaces The Hilbert space of a state is the complex vector space where the state lives. The dimension of the Hilbert space, is equivalent to the maximum number of linear independent states in which we can find the system. When treating many-body systems, computing states that live in high dimensional Hilbert spaces is costly. Usually, the systems treated in many body physics show short range entanglement, which we can exploit for efficient numerical calculations.

3.4. Entanglement Renormalization

In Tree Renormalization (see Fig. 3.) every β is a subsystem of the hole system meaning that we are loosing information due to the entanglement of nearest neighbors belonging to different subsets β of $\mathcal{L}$. In some way we are treating entanglement between different lattice points differently. That leads to a propagation of the entanglement between consecutive sites that belong to different coarse-graining subsets β over subsequent coarse-graining transformations Ω. That means that the state we have to coarse grain can not be factorized in product states, and therefore it lives in a bigger Hilbert space. That makes computations harder and longer or forces us to truncate additional information for keeping calculations feasible.

A way of overcoming the problem is the use of disentanglers. Disentangler are unitary operations that remove entanglement between consecutive lattice sites that are coarse-grained separately to the next level(See Fig. 4). by doing that we can keep the size of the Hilbert space under control.

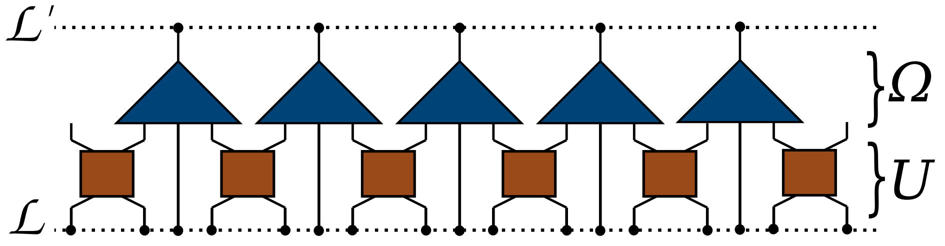


Figure 4: The disentanglers are now aded to the coarse-graning isometries.

3.4.1. Matrix Entranglement Renormalization Anzats (MERA) MERA is an extention of TTN in which disentanglers are used to avoid a continuous grouth of the Hilbert Space along the network. Although MERA can be a case of interest in 1D systems it becomes too complex to work with in 2D. Since we will be working with images that are 2D systems, the suitable choice is TTNs. [7, 8]

4. Manifold learning

It is difficult to interpret relations between classes of data instances when we are treating high-dimensional data. Lowering its dimensionality by assigning each data point a location in a lower dimensional map gives an intuitive representation of the inherent relations within the data. Manifold learning is a set of algorithms that try to find the distance between high dimensional objects and represent them in a lower dimmensional space. The most widely used algotrithms are: Principal Component Ananlysis (PCA), Multidimensional Scaling (MDS) and t-Stochastic Neghbor Embedding (t-SNE). We choose to use t-SNE because in contrast to the others it preserves the local structure of the data. [10]

4.1. Stochastic Neghbor Embedding (SNE)

To perfom SNE the conditional probabilityes $p_{j|i}$ must be calculated, where i and j refer to different data instances.

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k\neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)}. \tag{7}$$

The shorter the distance between two points the larger the conditional probability $p_{j|i}$.

$$q_{j|i} = \frac{exp(-||y_i - y_j||^2)}{\sum_{k\neq i} exp(-||y_i - y_k||^2)}. \tag{8}$$

Here authors asume σ_i to be 2 in the lower dimensional space. The objective then becomes to find y_i and y_j such that $q_{j|i}$ it's close to $p_{j|i}$

$$C = \sum_i \sum_j p_{j|i} log(\frac{p_{j|i}}{q_{j|i}}). \tag{9}$$

The perplexity it's a parameter that gives an idea of the number of close neighbours that each point should have:

$$Perp(P_i) = 2^{H(P_i)}, \tag{10}$$

$$H(P_i) = -\sum_j p_{j|i}(\sigma_i) log_2 p_{j|i}(\sigma_i). \tag{11}$$

The parameter σ_i is given by a fixed perplexity value and changes for every data point. Gradient descent is used to find the minimum of the cost function C. There is also a momentum term that avoids getting stuck in local minima:

$$\frac{\delta C}{\delta y_i} = 2\sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j). \tag{12}$$

In SNE, there is a high cost for using widely separated points in the low dimensional space to represent points that are nearby in the high dimensional space, whereas there is a small cost for using nearby datapoints to represent points that are far away in the original space. Therefore, SNE focuses on mantaining the local structure of the data.

4.2. t-SNE

In t-SNE, the conditional probabilities are simetrized, meaning that $p_{i|j} = p_{j|i}$ and $q_{i|j} = q_{j|i}$. Then equation (11) can be rewriten as (12):

$$\frac{\delta C}{\delta y_i} = 4\sum_j (p_{j|i} - q_{j|i})(y_i - y_j) \tag{13}$$

Whereas in SNE the low dimensional data points are represented by a normal distribution, in t-SNE, they are represented by a t distribution.

The t-distribution does not have exponentials and therefore it is easier to compute:

$$q_{j|i} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k\neq i}(1 + ||y_i - y_k||^2)^{-1}}, \tag{14}$$

and the gradient of the cost function becomes:

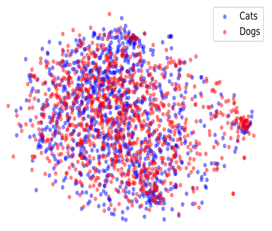
$$\frac{\delta C}{\delta y_i} = 4\sum_j (p_{j|i} - q_{j|i})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}. \tag{15}$$

4.2.1. Algorithm The first step is to calculate the affinities (conditional probabilities) in the high dimensional space $p_{i|j}$ Then the initial solution $y_1, y_2 ... y_n$ from a normal distribution and we comupute the affinites in the low dimensional space $q_{i|j}$. Those afinites are used to calculate the gradient of the cost funtion [9]. The gradient descent formula is used:

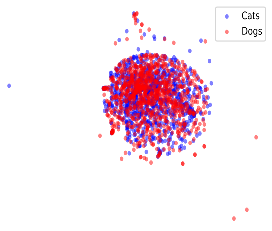
$$Y^{(t)} = Y^{(t-1)} + \eta\frac{\delta C}{\delta y} + \alpha(t)(Y^{(t-1)} - Y^{(t-2)}) \tag{16}$$

5. Procedure and results

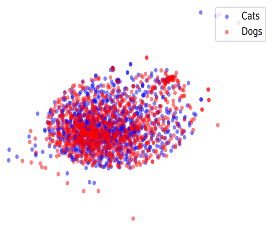
In the present section, we describe the implementation details of t-SNE both in a CNN and a TTN. For this particular case, a pretrained CNN classifier, trained on the Cifar-10 dataset is used[13]. The classifier is strucutred in three blocks. Each of them consists of two convolutional layers and a pooling layer. We perfom t-SNE in each layer to visualize the expressivenes of the model along the different layers. By applying t-SNE in each of the layers, we expect to see an improvement in the separation of the classes as we go through the layers of the network [11].



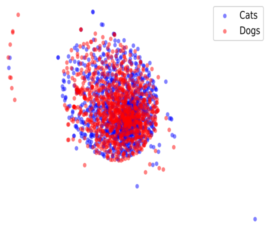
(a) Block 1 convolution 1



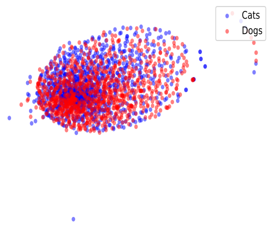
(b) Block 1 convolution 2



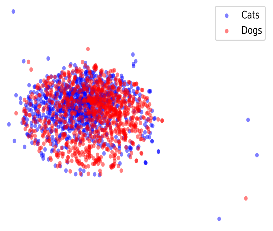
(c) Block 1 pooling



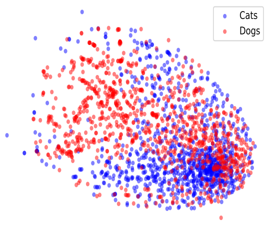
(d) Block 2 convolution 1



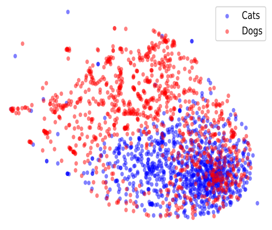
(e) Block 2 convolution 2



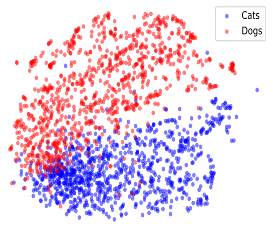
(f) Block 2 pooling



(g) Block 3 convolution 1



(h) Block 3 convolution 2

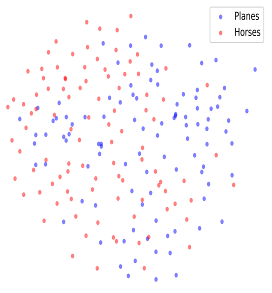


(i) Block 3 pooling

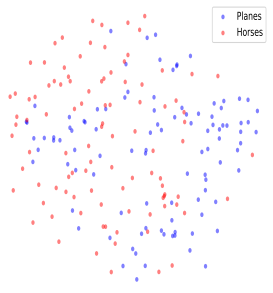
Figure 5: t-SNE results for every layer of a CIFAR classifier. Being (a) the closests layer to the inputs and (i) the closest to the outputs. Every spot in the graph represents the relative position of the output of a layer for one of the images. Blue and red dots correspond to outputs generated by images of cats and dogs respectively.

The CNN initialized with the pretrained weights and it is loaded with 2000 images of two classes, which are classified one by one. For every image going trough the network, we store the outputs of every one of the layers. Once we have the output of all the images for a certain layer, we are ready to perform t-SNE on the outputs of every layer.It is easy to see the progress in the class separation along the network. (See Fig. 5). In the first layer the network is confused and there is no distinction between the two categories. In the last layers the separation becomes clear.

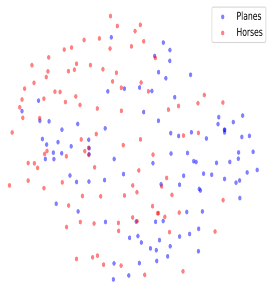
Similarly, t-SNE is used in the layers of a TTN trained as a binary image classifier of pictures of horses and airplaines. The TTN consists of 5 layers, and is loaded with 2000 images. In the first layer, we find the raw images, followed by their coarsegrained version in the subsequential layers and a final output layer. In the TTN training the parameters that are stored are the values of the coarse graining operations between layers(see Fig. 6).



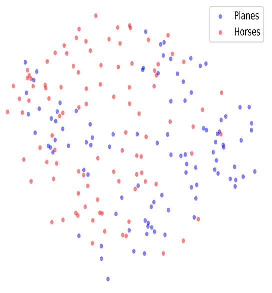
(a) Layer 0



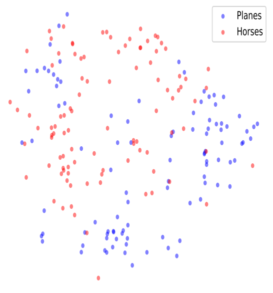
(b) Layer 1



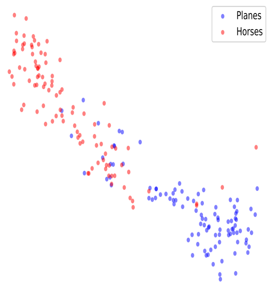
(c) Layer 2



(d) Layer 3



(e) Layer 4



(f) Layer 5

Figure 6: t-SNE results for every layer of a Tree Tensor Network. Being (a) input layer and (f) the ouput layer. Every spot in the graph represents the relative position of the output of a layer for one of the images. Blue and red dots correspond to outputs generated by images of Horses and Airplanes respectively.

As expected we can observe an improvement in class separation over the distinct layers of the network (see Fig. 6). The TTN trained as a binary classifier shows a hierarchy in levels of abstraction over its layers that resembles very much the growing feature extraction power of CNNs trough the network.

6. Summary and Conclussion

A number of ideas have been outlined above, it would be fruitfull to hilight them here in a summary form:

- In our work we have shown further evidence of the resemblance between TNs and deep learning architectures.
- By using manifold learning we have been able to give an intuitive idea of the hierarchical levels of abstraction within CNNs and TTNs.
- We conclude that hierarchical TNs bear the representation power of deep CNNs. Morover, they exibit the same increase on the level of abstraction along the network.

References

[1] D. Rolnick and M. Tegmark, "The power of deeper networks for expressing natural functions," *arXiv:1705.05502 (2017).*

[2] N. Cohen, O. Sharir, Y. Levine, R. Tamari, D. Yakira and A. Shashuna , "Analysis and Design of Convolutional Networks via Hierarchical Tensor Decompositions," *arXiv:1705.02302 (2017).*

[3] X. Gao and L. Duan, "Efficient Representation of Quantum Many-body States with Deep Neural Networks," *arXiv:1701.05039 (2017).*

[4] R. Orús, "A Practical Introduction to Tensor Networks: Matrix Product States and Projected Entangled Pair States," *arXiv:1306.2164 (2013).*

[5] E. M. Stoudenmire, D. Schwab, "Supervised Learning With Quantum-Inspired Tensor Networks", *Advances in Neural Information Systems 29, 4799-4807 (2016) arXiv:1603.05775.*

[6] D. Liu, S. Ran, P. Wittek, C. Peng, R. Blázquez García, G. Su, M. Lewenstein, "Machine Learning by Two-Dimensional Hierarchical Tensor Networks: A Quantum Information Theoretic Perspective on Deep Architectures," *arXiv:1710.04833 (2017).*

[7] Jacob C. Bridgeman, Christopher T. Chubb, "Hand-waving and Interpretive Dance: An Introductory Course on Tensor Networks," *arXiv:1603.03039 (2016).*

[8] M. Hauru, "Marster's Thesis: Multiscale Entanglement Renormalisation Ansatz," *University of Helsinki, (2013).*

[9] L. van der Maaten, G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research 9 (2008).*

[10] https://distill.pub/2016/misread-tsne/

[11] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

[12] https://keras.io/datasets/

[13] https://github.com/dseuss/ml-experiments/releases/download/cifar10/cifar10_ reduced.h5

[14] http://colah.github.io/posts/2014-07-Conv-Nets-Modular/

[15] http://colah.github.io/posts/2014-07-Understanding-Convolutions/