Learning connections between layers in deep networks

Eugenio Culurciello[*]
Purdue University
euge@purdue.edu

Jordan Bates
Purdue University
jtbates@purdue.edu

Jonghoon Jin
Purdue University
jhjin@purdue.edu

Aysegul Dundar
Purdue University
adundar@purdue.edu

Clement Farabet
New York University
cfarabet@nyu.edu

Abstract

We present the clustering learning technique applied to learning the connection matrix between layers of multi-layer feedforward deep neural networks. We show that networks and connections trained with clustering learning can obtain similar levels of performance as supervised deep network trained to ....

1 Introduction

Most scientists and engineers are fascinated by the design of an artificial vision system that can reproduce some of the human visual system capabilities in detecting, categorizing, tracking objects in view. The availability of a real-time synthetic vision system with such capabilities would find use in a large variety of applications, such as: autonomous cars, co-robots helpers, smart appliances, cellular-phones, to name a few.

In the recent years the fusion of bio-inspired and neuromorphic vision models with machine learning has dominated the development of artificial vision systems for the categorization of multiple objects in static frames. Bio-inspired deep networks are computer-vision and computational-neuroscience models of the mammalian visual system implemented in deep neural networks [**?**, **?**, **?**, **?**, **?**, **?**]. Most deep network architectures are composed of multiple layers (2, 3 typically), where each layer is composed of: linear two-dimensional filtering, non-linearity, pooling of data, output data normalization [**?**, **?**, **?**]. Recent machine learning research has focused on the task of training such deep networks from the abundant digital data available in the form of image frames and videos. In particular, deep networks need to learn good feature representations for complex visual tasks such as object categorization and tracking of objects in space and time, identifying object presence and absence. These representations usually involve learning the linear filter weight values from labeled and unlabeled input data. Since labeled data is costly and often ridden with human errors [**?**, **?**, **?**], the recent focus is on learning these features purely from unlabeled input data [**?**, **?**, **?**, **?**, **?**]. These recent methods typically learn multiple layers of deep networks by training several layers of features, one layer at a time, with varying complexity of learning models. Traditional work in deep learning for vision has mostly focused on improving the learning algorithms, and rarely it has shown potential

[*]More information on Eugenio Culurciello's laboratory and research can be found here: http://engineering.purdue.edu/elab/. Real time robotic vision systems: http://www.neuflow.org/

for overcoming some of the limitation of these systems, such as reliance on large sets of labeled data, and real-time operation for embedded systems.

Recent techniques based on unsupervised clustering algorithms are especially promising because they use simple learning methods that quickly converge [**?**]. These algorithms are easy to setup and train and are especially suited for robotics research, because less complex knowledge of machine learning is needed, environment-specific data can be collected quickly with a few minutes of video, setup of custom size deep networks is quick and can be adapted to specific tasks. In addition, real-time operation with efficient networks can be obtained with only a few minutes of training and setup, leading to quick and direct experimentation in robotic experiments.

In this paper we present unsupervised clustering algorithms applied to learning the connectivity matrix between layers and filters of deep neural networks for general-purpose vision systems.

The paper presents the following key innovations: (1) use of clustering learning for learning both filters and connections of deep networks with fully unsupervised techniques (Section **??**), (2) the ability of trained network to perform as well as supervised networks trained on still frames, (3) ability to run in real-time?.

2 Methods

In this paper we created and tested a model of unsupervised clustering algorithms in deep neural networks. We extend the results obtained in previous work [http://arxiv.org/abs/1301.2820], aimed at learning filters in multi-layer networks, to also learn the connection matrix between layers.

The connections between layers in a deep neural network are a very important parameter. It has been shown that in many cases random filters perform only slightly worse than fully trained network layers [cite: On Random Weights and Unsupervised Feature Learning, Saxe et all] suggesting that filters (used in the convolutional filtering stage of each layer in a deep network) might not play a dominant role in determining the performance of a deep network. Rather, the connections between layers and the combination of features are mainly responsible for the recent success of supervised deep convolutional networks [cite: hinton krizevsky 2012 et al]. In these networks the connection between layers is learned from the data using global gradient-descent techniques at large scales.

In unsupervised deep networks, one cannot rely on back-propagation techniques to learn the connections between layers. This is because unsupervised networks do not used labeled data, and thus an error signal cannot be computed. In order to use unlabeled data to train large deep networks, we will need a technique that can derive how to group features from one layer into features of the next layers, and at the same time learn the filters needed by the next layer. This is the main contribution of this paper, and what makes it unique. A figure that qualitatively explains the main contribution of the paper is Fig. 1.

In the figure we show two layers of a deep network: layer N and layer $N + 1$. The maps on the left of the figure, are the $N1$ output feature maps of the N-the layer in response to an input image. The maps on the right of the figure, are the $N2$ outputs of the first convolutional module of layer $N + 1$. Our learning technique groups feature maps from layer N into "receptive fields" (RF). These receptive fields are the input of one or multiple maps in layer $N+1$. We form these RF by grouping a small

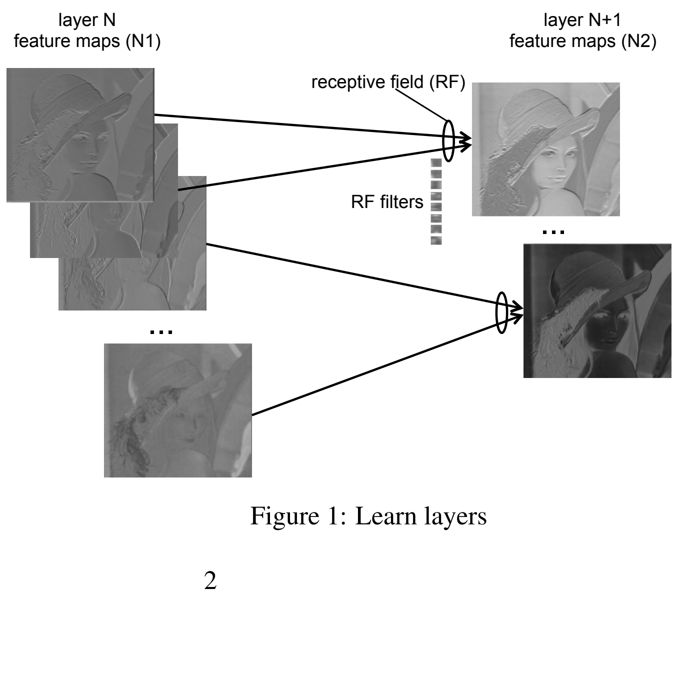

Figure 1: Learn layers

number ”fanin” of layer N
features that co-occur in
one or multiple sets of feature maps from layer N. In other words, we look for features that are highly activated in the same pixel, and we group these features into the RF. The idea behind this step is to group features that occur often together in the data. These RF are then representative of the connections between features of one layer and the next layer, and are completely obtained in a data-driven approach. In order to learn the filters for layer $N + 1$ convolutional module, we perform clustering (Clustering Learning) only within the RF (and not all $N1$ maps). We then obtain both a set of connections from layer N to layer $N + 1$ and the filters used by layer $N + 1$ convolutional module.

There is not much other work in learning connections [cite: adam coates, yi-lan boreau]

We used the Torch7 software for all our experiments [**?**], since this software can reduce training and learning of deep networks by 5-10 times compared to similar Matlab and Python tools.

2.1 Input data

We use input color video sequences of people talking in front of a camera as captured from a computer camera. The input video was resized to 120x 80 pixels and was contrast normalized separately on each RGB channel with a 9x9 gaussian filter using the Torch7 ”nn.Spatial Contrastive Normalization” function.

We use the NYU face detector dataset to test how the unsupervised network would perform on labeled data. The NYU face dataset has a training size of 52,684 32x32 images and a test size of 5,854 32x32 images. The dataset has two classes: ’face’ and ’background’. Input data was contrast normalized separately on each RGB channel with a 9x9 gaussian filter using the Torch7 ”nn.Spatial Contrastive Normalization” function.

Even if other groups showed slight improvements using the YUV color space [**?**], we did not use it. Rather, we kept the images in their original RGB to keep the model closer to biological human vision, where the retina is broadly sensitive to roughly RGB [**?**].

Also we did not use whitening of data (such as ZCA whitening) even if other groups have shown clear advantages of using it. We did not use whitening because of two main reasons: first, it is not applicable for general-purpose vision system where an a-priori dataset cannot be obtained. Second, whitening computation is very expensive (similar to the first layer of a convolutional neural network) and we instead replaced it with local contrast normalization, which is a bio-inspired technique to whiten the input data removing its mean bias and adapting the input dynamic range.

2.2 Network architecture

We experimented by training an unsupervised deep neural network with 2 layers, not counting pooling and normalization operations. The two layers were composed of a two-dimensional convolutional linear filtering stage, a spatial max pooling stage, and a subtractive and/or divisive normalization layer for removing the mean and resetting the std of all outputs to unity. The filters of the first two layers are generated with unsupervised clustering algorithms, as explained below. Using the naming convention in [**?**], for each layer l x_i is an input feature map, y_i is an output feature map. The input of each layer is a 3D array with n_l 2D feature maps of size $n_{l1} \cdot n_{l2}$. Each component (pixel, neuron) is denoted x_{ijk}. The output is also a 3D array, y_i composed of m_l feature maps of size $m_{l1} \cdot m_{l2}$.

The layers in the clustering learning network used the following sequence of operations:

1. Volumetric Convolution module (1st layer): performing convolutions on images with the learned volumetric CL filters
2. Spatial Convolution module (2nd layer): performing convolutions on images with the learned CL filters: $yc_i = b_j + \sum_i k_{ij} * x_i$, where $*$ is the 2D discrete convolution operator and b_j is a trainable bias parameter.
3. Spatial Max Pooling module: $yp_i = max_{n \times n}(yc_{ij})$ with $n = 2$ in this work.
4. Hyperbolic tangent nonlinearity: $ynl_i = tanh(yp_i)$

5. Spatial Subtractive or Contrastive Normalization: to zero the data mean, reset std to unity. The subtractive normalization operation for a given site ynl_{ijk} computes: $v_{ijk} = ynl_{ijk} - \sum_{ipq} w_{pq} \cdot ynl_{i,j+p,k+q}$, where w_{pq} is a normalized truncated Gaussian weighting window (typically of size 9x9). The divisive normalization computes $y_{ijk} = v_{ijk}/max(mean(\sigma_{jk}), \sigma_{jk})$ where $\sigma_{jk} = (\sum_{ipq} w_{pq} \cdot v^2_{i,j+p,k+q})^{1/2}$. The local contrast normalization layer is inspired by visual neuroscience models.

All networks used 32 filters on the first layer, 64 filters on the second layer. Clustering learning networks used a fully connected input to 1st, and 1st to 2nd layer. We experimented with SLAC connections [**?**]. In this case we used 4x more features than the numbers given above as beginning features, and then we decreased them to the same size of 32,64 in each respective layer. After the Convolution modules we added a SpatialMaxMap module to compute the groups and reduce the feature numbers.

2.3 Learning

We use k-means clustering algorithm (we refer to this step as *Clustering Learning algorithm*) to learn a set of 32 filters in the first layer, and 64 filters in the second layer. The techniques and scripts are general and can quickly modified to learn any number of filters. The filter sizes on both layers was set to 5 x 5 pixels. The input video was sampled to obtain 10,000 patches at random locations of size 5 x 5. These patches were used by the Clustering Learning algorithm to train the first layer, using 15 epochs on the patches dataset and 1,000 mini batch updates for the k-means algorithm. We learned Volumetric filters by sampling the same image location on contiguous N_f frames. Then we grouped the 5 x 5 x N_f frames in one vector and passed it to the Clustering Learning module to perform k-means clustering. This provided volumetric filters that were then unpacked into 5 x 5 patches for each of the input N_f frames. N_f frames were then input into the VolumetricConvolution module to provide 32 output spatio-temporal feature maps. These are also commonly referred as ”motion filters” because they can be used as optical-flow detectors on video sequences.

The input video dataset was processed first by the 1st Volumetric layer, then the output was again sampled by taking 100,000 patches at random locations one random of the 32 feature maps. We then applied the Clustering Learning algorithm again to obtain the required 64 filters for the second layer. These filter were no Volumetric, but were computed on single maps.

Examples of the filters learned with the CL technique on a 1st and 2nd layer are given in figure **??**. The volumetric filters are shown on the left of the figure as 5 x 5 x 2 columns, since we used 2 frames in this example figure. We can provide animated-GIF files upon request. We also show in in figure **??**-middle samples of the processed 1st layer outputs. We also report on the right of the figure the 2nd layer filters. These 2nd layer filters are just a subset of the 32 x 64 filter learned with Clustering Learning. We learned this many filters because layer 1 and 2 are fully connected, and thus can benefit from this large number of different filters. This is different from the approach taken by **??** where the output of the 1st layer was reshaped to a vector and fed to the k-means algorithm, thus doing away with inter-network connection matrices.

Both layer filters were obtained with training for 10 minutes on a modern quad-core Intel i7 laptop. A supervised network of this size would require several tens of thousands of image examples and several hours of training time.

In order to test the network after training, we then used the unsupervised deep network to process a dataset of face from NYU. We passed both the test and train sets through the network. The 30,000 32 x 32 images were processed in about 7 minutes on a modern quad-core Intel i7 laptop running Torch7. We the used a linear layer (linear regression) and trained it on the 64 x 2 x 2 outputs of the processed face dataset. We used the train set for training. We then verified the precision of the network on the test set. As input to the Volumetric networks, we gave the same input to all required frames, but copying the same input samples N_f times. This is not the best test for a Volumetric network that learned spatio-temporal filters, and we are working on more tests with video datasets.

In order to test the ability of our network to learn frequently occurring objects in the video scene, we tested the output of the network after processing the entire NYU face dataset with it. We then found the neuron that in overall responded the most to a face, but not a background. We found the neuron most responsive to face tn by running obtaining the maximum of all output neuron of the network net to the face set, as can be seen in the equation below.

$$bf = \sum_i tn(i) \cdot dataset(i), tn = arg_n max(\sum_i net_n(faceset(i))) \tag{1}$$

We then computed a weighted average of tn with the output of the network for each sample of the dateaset, be that a face or a background. The weighted average neuron bf also reported in the equation above. The optimal stimulus for the neuron bf is reported in figure **??**.

3 Results

We report the results of our experiments in table **??**. We here compared several version of the unsupervised 2-layer deep network.

The size of the network, the number of each layer of filters, is given in the table. The type of network was: full = fully connected; LCN = local contrast normalization at the output of each layer (as opposed to a simpler subtractive normalization of the means only); SLAC = single-link agglomerate clustering technique from [**?**] to reduce the filters from 128 to 32 on the 1st layer and from 256 to 64 in the 2nd layer; random = test of a random network to compare the learning techniques; noVC does not compute volumetric filters, but rather uses the same static filters at each frame ;supervised CNN = standard 2-layer convolutional neural network used as face detector with 8 features on the 1st layer and 32 on the second layer, max pooling of 4 x 4, connection table with fan in of 8, followed by a linear classifier; linear on pixels = test of a linear classifier on the bare 32 x 32 input pixels without the use of any deep network for feature extraction.

With only one frame as input SLAC 1-layer with LCN provided the best results, followed closely by the fully connected LCN network. It is clear that proper normalization of the standard deviation at each layer is key to the highest performance. This is due to the fact that this step prepares the best data spread for the nonlinearity at each layer, thus preserving the maximum information content.

Similar results are reported by [**?**]. The SLAC algorithm also performed well only on the 1st layer, and reported a loss of 2% points when used in the second layer. This issue is not well understood and under investigation.

Random networks gave surprisingly high results, as has been reported by several other groups [**?**]. But the Clustering Learning algorithm was in overall superior. The random network architecture was identical to a 32, 64 full LCN model, with weights initialized with a random uniform distribution by the Torch7 package.

A linear layer on the input image was computed as a baseline to give indication of the difficulty of the task. As expected, a face-detection task is hardly the best example for a deep network with much larger capacity. nevertheless the ease of availability of data and the fact that our network was able to learn face features from video is a satisfactory result. In any case the linear network performed much worse at 79% recognition rate on faces. almost 20 points lower than Clustering Learning networks.

The supervised CNN gave the top results with above 99% recognition rates, and by using a smaller network. This is also expected, as gradient descent algorithms are the best at adapting a network architecture to a single task. But we remind the reader that our goal is a general network with little need for supervision, so we are willing to accept a less efficient network for these great advantages.

The best results of this study was achieved with a fully-connected network with LCN and without the use of Volumetric filters. This network instead computed static filters and used them at each frame. This idea overcomes the problems of feeding a video as input and training on a static dataset, and allows to show the full power of Clustering Learning applied to videos: the ability to cluster features even across frames, be those frames from video or still images.

Notice also that in the case of the fully connected network, we also trained and tested a Volumetric network with 1,2,4 input frames. 2 frames showed increased recognition results, due to the fast that the two frames give higher signal-to-noise ratio at the input of the network. 4 frames instead did not perform as well. Partly this is due to the fact that our test dataset was not a video, so most of the learnt spatio-temporal filters were not effectively used in testing.

Table **??** reports the execution time of some of the tested networks reported in table **??**.

Increasing the size of the filters to 64 and 128 also gave really high results, but with a slower computation time of about 1s.

The supervised CNN has the lowest computational time, together with the highest performance, as it is fully numerically optimized to the task and dataset by gradient descent methods, which currently perform the best on fixed dataset. If only fixed dataset performance would translate to real-life performance, there would not be a need for unsupervised training. But unfortunately this is not the case and supervised CNN perform quite badly when used in a different setting from which they were trained for [**?**]. We thus advocate to take the performance of supervised CNN with a grain of salt: remember that its very high performance, at least its precision, will not be available as part of a general-purpose vision system.

4 Discussion

We presented results on clustering learning algorithms trained on video sequences for general-purpose vision system. These algorithms can be used to train multi-layer feed-forward neural networks from videos in minutes. We show results on

Acknowledgments

We are especially grateful to the the Torch7 developing team, in particular Ronan Collobert, who first developed this great, easy and efficient tool, Clement Farabet, Koray Kavukcuoglu, Leon Bottou. We could not have done any of this work without standing on these giants shoulders.