Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation

Abstract

We present techniques for speeding up the test-time evaluation of large convolutional networks, designed for object recognition tasks. These models deliver impressive accuracy but each image evaluation requires millions of floating point operations, making their deployment on smartphones and Internet-scale clusters problematic. The computation is dominated by the convolution operations in the lower layers of the model. We exploit the linear structure present within the convolutional filters to derive approximatations that significantly reduce the required computation. Using large state-of-the-art models, we demonstrate speedups by a factor of $2-4$x, while keeping the accuracy within 1% of the original model.

1. Introduction

Large neural networks have recently demonstrated impressive performance on a range of speech and vision tasks. However the size of these models can make their deployment at test time problematic. For example, mobile computing platforms are limited in their CPU speed, memory and battery life. At the other end of the spectrum, Internet-scale deployment of these models requires thousands of servers to process the 100's of millions of images per day. The electrical and cooling costs of these servers required is significant.

Training large neural networks (NN) can take weeks, or even months. This hinders research and consequently there have been extensive efforts devoted to speeding up training procedure. However, there are relatively few efforts are improving the *test-time* performance of the models.

In this paper we focus on speeding up the evaluation of *trained* networks, without compromising performance. We consider convolutional neural networks used for computer vision tasks, since they are large and widely used in commercial applications. Within these models, most of the time ($\sim 90\%$) is spent in the convolution operations in the lower layers of the model. The remaining operations: pooling, contrast normalization and the upper fully-connected layers collectively take up the remaning 10%.

We present two novel methods for speeding up the convolution operations. One involves projecting the input image into a set of 1-D color sub-spaces. This allows the filters in the first of layer of the model to be monochromatic (i.e. reducing the color channels from three to one), thereby saving a factor of 3 in computation. The second approach, applied to subsequent convolution layers, involves clustering the filters into a set of low-dimensional linear sub-spaces, each of which is represented by a set of tensor outer-products. Collectively, our techniques speed up execution by factor of $2-4$ while keeping prediction accuracy within 1% of the original model. These gains allow the use of larger, higher performance models than would otherwise be practical.

2. Related Work

(Vanhoucke et al., 2011) explored the properties of CPUs to speed up execution. They present many solutions specific to Intel and AMD CPUs, however some of their techniques are general enough to be used for any type of processor. They describe how to align memory, and use SIMD operations (vectorized operations on CPU) to boost the efficiency of matrix multiplication. Additionally, they propose the linear quantization of the network weights and input. This involves representing weights as 8-bit integers (range $[-1287128]$), rather than 32-bit floats. This approximation is similar in spirit to our approach, but differs in that it is applied to each weight element independently. By contrast, our approximation approach models the structure within each filter. Potentially, the two approaches could be used in conjunction.

The most expensive operations in convolutional networks are the convolutions in the first few layers. The complexity of this operation is linear in the area of the receptive field of the filters, which is relatively large for these layers. However, (Mathieu et al., 2013) have shown that convolution can be efficiently computed in Fourier domain, where it becomes element-wise multiplication (and there is no cost associated with size of receptive field). They report a forward-pass speed up of around $10x$ (depending on the kernel size, number of features etc.). Importantly, this method can be

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

used jointly with most of techniques presented in this paper.

The use of low-rank approximations in our approach is inspired by work of (Denil et al., 2013) who demonstrate the redundancies in neural network parameters. They show that the weights within a layer can be accurately predicted from a small (e.g. $\sim 5\%$) subset of them. This indicates that neural networks are heavily over-parametrized. All the methods presented here focus on exploiting the linear structure of this over-parametrization.

3. Low Rank Approximations

In this section, we give theoretical background on low rank approximations. First, we discuss simplest setting, which is for matrices (two dimensional tensors). We then consider the approximation of 4-dimensional tensors of convolution weights.

3.1. Matrix Low Rank Approximation

Let $X \in \mathbb{R}^{n \times m}$ denote the input to a fully connected layer of a neural network and let $W \in \mathbb{R}^{m \times k}$ denote the weight matrix for the layer. Matrix multiplication, the main operation for fully connected layers, costs $O(nmk)$. However, W is likely to have a low-rank structure and thus have several eigenvalues close to zero. These dimensions can be interpreted as noise, and thus can be eliminated without harming the accuracy of the network. We now show how to exploit this low-rank structure and to XW much faster than $O(nmk)$.

Every matrix $W \in \mathbb{R}^{m \times k}$ can be expressed using singular value decomposition:

$$W = USV^\top, \text{ where } U \in \mathbb{R}^{m \times m}, S \in \mathbb{R}^{m \times k}, V \in \mathbb{R}^{k \times k}$$

S is has eigenvalues on the diagonal, and zeros elsewhere. W can be approximated by choosing the t largest eigenvalues from S. We can write the approximation as

$$\hat{W} = \tilde{U}\tilde{S}\tilde{V}^\top, \text{ where } \tilde{U} \in \mathbb{R}^{m \times t}, \tilde{S} \in \mathbb{R}^{t \times t}, \tilde{V} \in \mathbb{R}^{t \times k}$$

Now the computation $X\tilde{W}$ can be done in $O(nmt + nt^2 + ntk)$, which, for sufficiently small t can be significantly smaller than $O(nmk)$.

3.2. Tensor Low Rank Approximations

In typical object recognition architectures, the convolutional layer weights at the end of training exhibit strong redundancy and regularity across all dimensions. A particularly simple way to exploit such regularity is to linearly compress the tensors, which amounts to finding low-rank approximations.

Convolution weights can be described as a 4-dimensional tensor. Let $W \in \mathbb{R}^{C \times X \times Y \times F}$ denote such a weight tensor. C is the number of number of input channels, X and Y are the special dimensions of the kernel, and F is the target number of feature maps. Let $I \in \mathbb{R}^{C \times N \times M}$ denote an input signal where C is the number of input maps, and N and M are the spatial dimensions of the maps. The computation performed by a generic convolutional layer is defined as

$$I * W(f, x, y) = \sum_{c=1}^{C} \sum_{x'=1}^{X} \sum_{y'=1}^{Y} I(c, x + x', y + y') W(c, x', y', f)$$

We would like to approximate W with a low rank tensor that has a particular structure that allows for a more efficient computation of the convolution. The approximations will be more efficient in two senses: both the number of floating point operations required to compute the convolution output and the number of parameters that need to be stored will be dramatically reduced.

A standard first convolutional layer will receives three color channels, typically in RGB or YUV space, as input whereas later hidden layers typically receive a much larger number of feature maps that have resulted from computations performed in previous layers. As a result, the first layer weights often have a markedly different structure than the weights in later convolutional layers. We have found that different approximation techniques are well suited to the different layers which we now describe. The first approach, which we call the monochromatic filter approximation can be applied to the weights in the first convolutional layer. The second approach, which we call the bi-clustering approximation, can be applied to later convolutional layers where the number of input and output maps is large.

3.3. Monochromatic filters

Let $W \in \mathbb{R}^{C \times X \times Y \times F}$ denote the first convolutional layer weights of a trained network. The number of input channels, C, is 3 and each channel corresponds to a different color component (either RGB or YUV). We have found that the color components of weights from a trained convolutional neural network have low dimensional structure. In particular, the weights can be well approximated by projecting the color dimension down into a 1D subspace. Figure ?? shows the original first layer convolutional weights of a trained network and the weights after the color dimension has been projected into 1D lines.

The approximation is computed as follows. First, for every output feature, f, we consider consider the matrix $W_f \in \mathbb{R}^{C \times XY}$, where the spatial dimensions have been combined, and find the singular value decomposition,

$$W_f = U_f S_f V_f^\top$$

where $U_f \in \mathbb{R}^{C \times C}, S_f \in \mathbb{R}^{C \times XY}, V_f \in \mathbb{R}^{XY \times XY}$. We then take the rank-1 approximation to W_f

$$\tilde{W}_f = \tilde{U}_f \tilde{S}_f \tilde{V}_f^\top$$

where $\tilde{U}_f \in \mathbb{R}^{C \times 1}, \tilde{S}_f \in \mathbb{R}, \tilde{V}_f \in \mathbb{R}^{1 \times XY}$.

This approximation corresponds to shifting from C color channels to 1 color channel for each output feature. We can further exploit the regularity in the weights by sharing the color component basis between different output features. We do this by clustering the F left singular vectors, $\tilde{U}_f$, of each output feature f into C' different equal sized clusters, where C' is much smaller than F. Then, for each of the $\frac{F}{C'}$ output feature, f, that is assigned to cluster c, we can approximate W_f with

$$\tilde{W}_f = U_c \tilde{S}_f \tilde{V}_f^\top$$

where $U_c \in \mathbb{R}^{C \times 1}$ is the cluster center for cluster c and $\tilde{S}_f$ and $\tilde{V}_f$ as as before.

This low-rank approximation allows for a more efficient computation of the convolutional layer output. By decomposing the approximated weights into two tensors. Let $W_C \in \mathbb{R}^{C' \times C}$ denote the color transform matrix where the rows of W_c are the cluster centers $U_c^\top$. Let $W_{mono} \in \mathbb{R}^{X \times Y \times F}$ denote the monochromatic weight tensor containing $\tilde{S}_f \tilde{V}_f^\top$ for each of the F output features. Given this decomposition, we can compute the output of the convolutional layer by first transforming the input signal, $I \in \mathbb{R}^{C \times N \times M}$ into a different basis using the color transform matrix: $\tilde{I} = W_c$.

3.4. Bi-clustering of hidden layer weights

4. Numerical Experiments

4.1. Testing time

on GPU: Michael can help.

on CPU

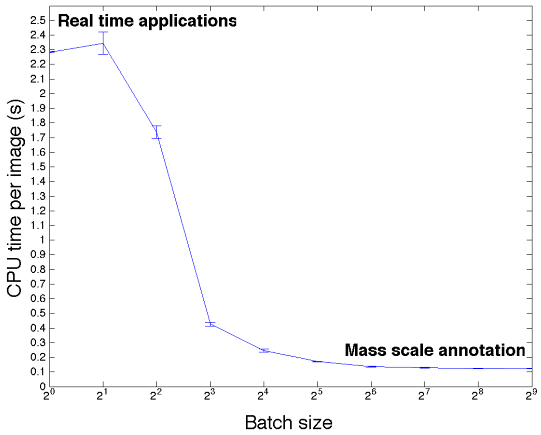

Figure 1. CPU computational time per image for various batch sizes.

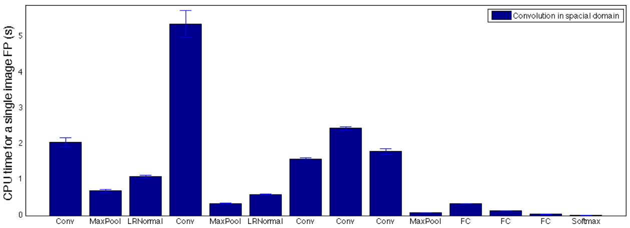

Figure 2. Per layer breakdown of execution time for mini batch of size 128. Such size of mini batch gives optimal per image CPU time.

4.1.1. Monochromatic

4.1.2. Linear combination of filters

4.1.3. Separable filters

4.2. Denoising

5. Implications

5.1. Denoising Aspect

we can improve training by simple linear denoting.

5.2. Low-Rank training

Low-rank to avoid over-fitting.

6. Discussion

7. Discussion

References

Denil, Misha, Shakibi, Babak, Dinh, Laurent, Ranzato, Marc'Aurelio, and de Freitas, Nando. Predicting parameters in deep learning. *arXiv preprint*

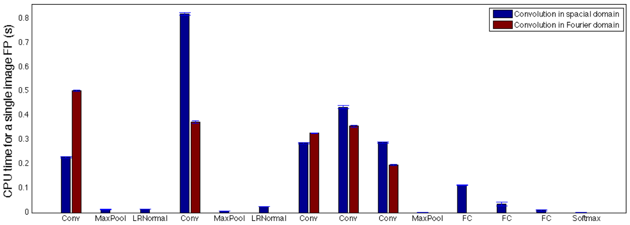

Figure 3. Per layer breakdown of execution for mini batch of size 1. Use for real time applications.

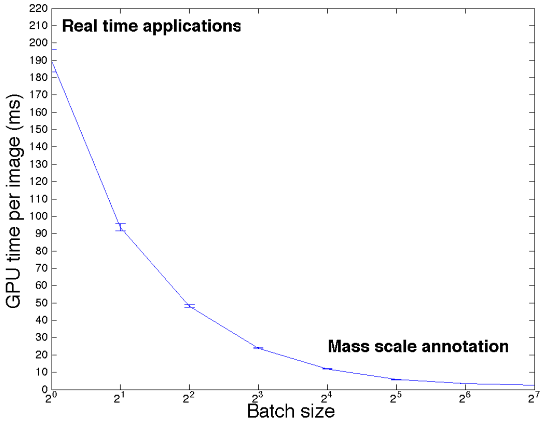

Figure 4. GPU computational time per image for various batch sizes.

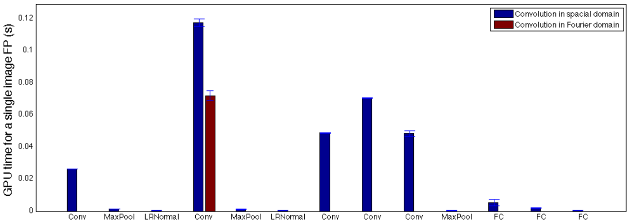

Figure 5. Per layer breakdown of execution time for mini batch of size 128. Such size of mini batch gives optimal per image GPU time.

arXiv:1306.0543, 2013.

Hinton, Geoffrey E, Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Le, Q. V., Ngiam, J., Chen, Z., Chia, D., Koh, P. W., and Ng, A. Y. Tiled convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2010.

Le, Quoc V, Ranzato, Marc'Aurelio, Monga, Rajat, Devin, Matthieu, Chen, Kai, Corrado, Greg S, Dean, Jeff, and Ng, Andrew Y. Building high-level features using large scale unsupervised learning. *arXiv preprint arXiv:1112.6209*, 2011.

Lowe, David G. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pp. 1150–1157. Ieee, 1999.

Mathieu, Michael, Henaff, Mikael, and LeCun, Yann. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

Vanhoucke, Vincent, Senior, Andrew, and Mao, Mark Z. Improving the speed of neural networks on cpus. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.

Zeiler, Matthew D, Taylor, Graham W, and Fergus, Rob. Adaptive deconvolutional networks for mid and high level feature learning. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2018–2025. IEEE, 2011.

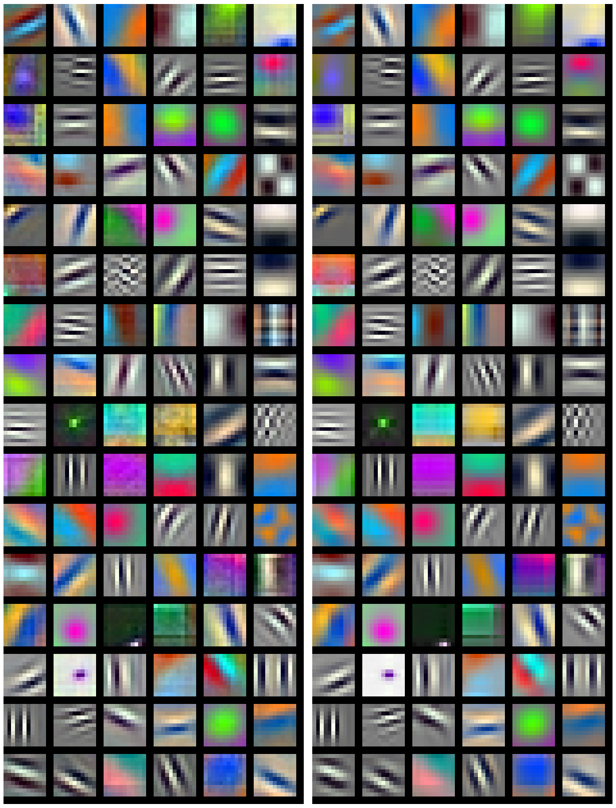

Figure 6. (Left) Original filters, (Right) approximated filters. (this pictures are too big, and should contain white separation).