# Deep Neural Network Approximation using Tensor Sketching

**Shiva Prasad Kasiviswanathan**[1]*, **Nina Narodytska**[2], **Hongxia Jin**[3]
[1] Amazon AWS AI, USA, [2] VMware Research, USA, [3] Samsung Research America, USA
kasivisw@gmail.com, nnarodytska@vmware.com, hongxia.jin@samsung.com

## Abstract

Deep neural networks are powerful learning models that achieve state-of-the-art performance on many computer vision, speech, and language processing tasks. In this paper, we study a fundamental question that arises when designing deep network architectures: Given a target network architecture can we design a "smaller" network architecture that "approximates" the operation of the target network? The question is, in part, motivated by the challenge of parameter reduction (compression) in modern deep neural networks, as the ever increasing storage and memory requirements of these networks pose a problem in resource constrained environments.

In this work, we focus on deep convolutional neural network architectures, and propose a novel randomized tensor sketching technique that we utilize to develop a unified framework for approximating the operation of both the convolutional and fully connected layers. By applying the sketching technique along different tensor dimensions, we design changes to the convolutional and fully connected layers that substantially reduce the number of effective parameters in a network. We show that the resulting smaller network can be trained directly and has a classification accuracy that is comparable to the original network.

## 1 Introduction

Deep neural networks have become ubiquitous in machine learning with applications, ranging from computer vision, to speech recognition, and natural language processing. The recent successes of convolutional neural networks (CNNs) for computer vision applications have, in part, been enabled by recent advances in scaling up these networks, leading to networks with millions of parameters. As these networks keep growing in their number of parameters, reducing their storage and computational costs has become critical for meeting the requirements of practical applications. Because while it is possible to train and deploy these deep convolutional neural networks on modern clusters, their storage, memory bandwidth, and computational requirements make them prohibitive for embedded mobile applications. On the other hand, computer vision applications are growing in importance for mobile platforms. This dilemma gives rise to the following natural question: *Given a target network architecture, is it possible to design a new smaller network architecture (i.e., with fewer parameters), which approximates the original (target) network architecture in its operations on all inputs?* In this paper, we present an approach for answering this *network approximation* question using the idea of *tensor sketching.*

Network approximation is a powerful construct because it allows one to replace the original network with the smaller one for both training and subsequent deployment [Denil *et al.*, 2013; Chen *et al.*, 2015; Cheng *et al.*, 2015; Yang *et al.*, 2015; Sindhwani *et al.*, 2015; Chen *et al.*, 2016; Tai *et al.*, 2016; Garipov *et al.*, 2016].[1] That is, it completely eliminates the need for ever realizing the original network, even during the initial training phase, which is a highly desirable property when working in a storage and computation constrained environments. While approximating any network (circuit) using a smaller network (circuit) is computationally a *hard* problem [Umans, 1998], in this paper, we study the problem of network approximation on convolutional neural networks. To approximate a convolutional neural network NN, we focus on its parametrized layers (the convolutional and fully connected layers). Consider any such layer $L$ in the network NN. Let $\phi : \Gamma \times \Theta \to \Omega$ denote the function (transformation) applied by this layer, where $\Theta$ represents the parameter space of the function (generally, a tensor space of some order), $\Gamma$ and $\Omega$ represent the input and output space respectively. Our general idea is to replace $\phi$ by a randomized function $\hat{\phi} : \Gamma \times \widehat{\Theta} \to \Omega$, such that $\forall \theta \in \Theta, \ \exists \hat{\theta} \in \widehat{\Theta},$ such that for every input $\gamma \in \Gamma, \ \mathbb{E}[\hat{\phi}(\gamma; \hat{\theta})] = \phi(\gamma; \theta)$, where the expectation is over randomness of the function $\hat{\phi}$. In other words, $\hat{\phi}(\gamma; \hat{\theta})$ is an

*Work done while the author was at Samsung Research America.

[1]For clarity, we distinguish between the terms network and model in this paper: network refers to network architecture that describes the transformation applied on the input, whereas model refers to a trained network with fixed parameters obtained by training a network with some training set.

unbiased estimator of $\phi(\gamma;\theta)$. Additionally, we establish theoretical bounds on the variance of this estimator. Ideally, we want the representation length of $\hat{\theta}$ to be much smaller than that of $\theta$. For the construction of $\hat{\phi}$, we introduce a novel randomized tensor sketching idea. The rough idea here is to create multiple sketches of the tensor space $\Theta$ by performing random linear projections along different dimensions of $\Theta$, and then perform a simple combination of these sketches. This new operation $\hat{\phi}$ defines a new layer that approximates the functionality $\phi$ of the layer $L$. Since $\hat{\phi}$ and $\phi$ have the same input and output dimensionality, we can replace the layer $L$ in the network NN with this new (sketch counterpart) layer. Doing so for all the convolutional and fully connected layers in NN, while maintaining the rest of the architecture, leads to a smaller network $\widehat{\mathrm{NN}}$, which approximates the network NN. To the best of our knowledge, ours is the first work that uses the idea of sketching of the parameter space for the task of network approximation.

The next issue is: Can we efficiently train the smaller network $\widehat{\mathrm{NN}}$? We show that, with some changes to the standard training procedure, the parameters (which now represent sketches) of the constructed smaller network can be learnt space efficiently on any training set. Also compared to the original network, there is also a slight improvement in the running time needed for various operations in this smaller network. This allows us to train $\widehat{\mathrm{NN}}$ directly on $D$ to get a reduced model $\widehat{\mathrm{NN}}_D$.[2] Our experimental evaluations, on different datasets and architectures, corroborate the excellent performance of our approach by showing that it increases the limits of achievable parameter number reduction while almost preserving the original model accuracy, compared to several existing approximation techniques. In fact, our technique succeeds in generating smaller networks that provide good accuracy even on large datasets, such as Places2, where other state-of-the-art network approximation techniques fail.

## 2 Preliminaries

We denote $[n] = \{1,\dots,n\}$. Vectors are in column-wise fashion, denoted by boldface letters. For a vector $\mathbf{v}$, $\mathbf{v}^\top$ denotes its transpose and $\|\mathbf{v}\|$ its Euclidean norm. For a matrix $M$, $\|M\|_F$ denotes its Froebnius norm. We use random matrices to create sketches of the matrices/tensors involved in the fully connected/convolutional layers. In this paper, for simplicity, we use random scaled sign (Rademacher) matrices. We note that other families of distributions such as subsampled randomized Hadamard transforms can probably lead to additional computational efficiency gains when used for sketching.

**Definition 1.** *Let $Z \in \mathbb{R}^{k\times d}$ be a random sign matrix with independent entries that are $+1$ or $-1$ with probability $1/2$. We define a random scaled sign matrix $U = Z/\sqrt{k}$.*

[2]There memory footprint of the reduced model $\widehat{\mathrm{NN}}_D$ can be further reduced using various careful operations such as pruning, binarization, quantization, low-rank decomposition, etc., [Gong *et al.*, 2014; Han *et al.*, 2015; Han *et al.*, 2016; Soulié *et al.*, 2015; Wu *et al.*, 2015; Guo *et al.*, 2016; Kim *et al.*, 2016; Wang *et al.*, 2016; Hubara *et al.*, 2016a; Hubara *et al.*, 2016b; Li *et al.*, 2016; Zhu *et al.*, 2016], which is beyond the scope of this work.

Here, $k$ is a parameter that is adjustable in our algorithm. We generally assume $k \ll d$. Note that $\mathbb{E}[U^\top U] = \mathbb{I}_d$ where $\mathbb{I}_d$ is the $d\times d$ identity matrix. Also, by linearity of expectation, for any matrix $M$ with $d$ columns, we have $\mathbb{E}[MU^\top U] = M\mathbb{E}[U^\top U] = M$.

**Notations.** We denote matrices by uppercase letters and higher dimensional tensors by Euler script letters. A real $p$th order tensor $\mathcal{T} \in \otimes_{i=1}^p \mathbb{R}^{d_i}$ is a member of the tensor product of Euclidean spaces $\mathbb{R}^{d_i}$ for $i \in [p]$. The different dimensions of the tensor are referred to as *modes*. The $(i_1,\dots,i_p)$th entry of a tensor $\mathcal{T}$ is denoted by $\mathcal{T}_{i_1 i_2 \dots i_p}$. The mode-$n$ matrix product (for $n \in [p]$) of a tensor $\mathcal{T} \in \mathbb{R}^{d_1\times\cdots\times d_p}$ with a matrix $M \in \mathbb{R}^{k\times d_n}$ is denoted by $\mathcal{T}\times_n M$. Elementwise, we have: $(\mathcal{T}\times_n M)_{i_1\dots i_{n-1} j i_{n+1}\dots i_p} = \sum_{i_n=1}^{d_n} \mathcal{T}_{i_1 i_2\dots i_p} M_{j i_n}$.

A *fiber* of $\mathcal{T}$ is obtained by fixing all but one of the indices of the tensor. A flattening of tensor $\mathcal{T}$ along a mode (dimension) $n$ (denoted by $\mathrm{mat}_n$) is a matrix whose columns correspond to mode-$n$ fibers of $\mathcal{T}$.

**Tensor Sketching.** Our network approximation is based on the idea of tensor sketching. Data sketching ideas have been successfully used in designing many machine-learning algorithms, especially in the setting of streaming data, see e.g., [Woodruff, 2014]. Generally, sketching is used to construct a compact representation of the data so that certain properties in the data are (approximately) preserved. Our usage of sketching is however slightly different, instead of sketching the input data, we apply sketching on the parameters of the function. Also, we want to design sketching techniques that work uniformly for both matrices and higher order tensors. For this, we define a new tensor sketch operation.

**Definition 2** (Mode-$n$ Sketch). *Given a tensor, $\mathcal{T} \in \otimes_{i=1}^p \mathbb{R}^{d_i}$, the mode-$n$ sketch of $\mathcal{T}$ with respect to a random scaled sign matrix $U_n \in \mathbb{R}^{k\times d_n}$ for $n \in [p]$, is defined as the tensor $\mathcal{S}_n = \mathcal{T}\times_n U_n$, where $\times_n$ denotes the mode-$n$ matrix product.*

Since, we generally pick $k \ll d_n$, the space needed for storing the sketch $\mathcal{S}_n$ is a factor $d_n/k$ smaller than that for storing $\mathcal{T}$. In the case of matrices, the sketches are created by pre- or post-multiplying the matrix with random scaled sign matrices of appropriate dimensions. For example, given a matrix $W \in \mathbb{R}^{d_1\times d_2}$, we can construct mode-1 sketch (resp. mode-2 sketch) of $W$ as $W\times_1 U_1 = U_1 W$ (resp. $W\times_2 U_2 = WU_2^\top$). Given a sketch $S_1 = W\times_1 U_1$ (resp. $S_2 = W\times_2 U_2$) of a matrix $W$ and another matrix $M \in \mathbb{R}^{d_2\times d_3}$, it is natural to use $U_1^\top S_1 M$ (resp. $S_2 U_2 M$) as an estimator for the matrix product $WM$. It is easy to see that both these estimators are unbiased. The second part of the following proposition analyzes the variance of these estimators. The result will motivate our construction of sketch-based convolutional and fully connected layers in the next section. We omit the proof due to space limitations.

**Proposition 2.1.** *Let $W \in \mathbb{R}^{d_1\times d_2}$. Let $U_1 \in \mathbb{R}^{k\times d_1}$ and $U_2 \in \mathbb{R}^{k\times d_2}$ be two independent random scaled sign matrices. Let $S_1 = U_1 W (= W\times_1 U_1)$ and $S_2 = WU_2^\top (= W\times_2 U_2)$. Then for any matrix $M \in \mathbb{R}^{d_2\times d_3}$:*

1. $\mathbb{E}[U_1^\top S_1 M] = WM$, *and* $\mathbb{E}[S_2 U_2 M] = WM$.
2. $\mathbb{E}\left[\left\|U_1^\top S_1 M - WM\right\|_F^2\right] \le \frac{2d_1\|WM\|_F^2}{k}$, *and*

$$\mathbb{E}\left[\|S_2 U_2 M - WM\|_F^2\right] \leq \frac{2\|W\|_F^2 \|M\|_F^2}{k}.$$

Notice that the variance terms decrease as $1/k$. The variance bound can also be plugged into Chebyshev's inequality to get a probability bound. Also, the variance bounds are quantitatively different based on whether the sketch $S_1$ or $S_2$ is used. In particular, depending on $W$ and $M$, one of the variance bounds could be substantially smaller than the other one, e.g., if the columns in $M$ are in the null space of $W$ then $WM$ is a zero matrix, so while one bound gives a tight zero variance the other one does not.

## 3 Sketch-based Network Architecture

We now describe our idea of approximating a network using tensor sketching. Our approach, in almost identical fashion, can be used to reduce the number of parameters involved in both the convolutional and the fully connected layers without significantly affecting the resulting accuracy.

### 3.1 Sketching Convolutional Layers

A typical convolutional layer in a CNN transforms a 3-dimensional input tensor $\mathcal{I}_{\text{in}} \in \mathbb{R}^{h_1 \times w_1 \times d_2}$ into a output tensor $\mathcal{I}_{\text{out}} \in \mathbb{R}^{h_2 \times w_2 \times d_1}$ by convolving $\mathcal{I}_{\text{in}}$ with the kernel tensor $\mathcal{K} \in \mathbb{R}^{d_2 \times h \times w \times d_1}$, where $h_2$ and $w_2$ depends on $h, w, h_1, w_1$ and possibly other parameters such as stride, spatial extent, zero padding [Goodfellow *et al.*, 2016]. We use $*$ to denote the convolution operation, $\mathcal{I}_{\text{out}} = \mathcal{I}_{\text{in}} * \mathcal{K}$. The exact definition of the convolution operator ($*$) that depends on these above mentioned additional parameters is not very important for us, and we only rely on the fact that the convolution operation can be realized using a matrix multiplication as we explain below.[3] Also, a convolutional layer could be optionally followed by application of some non-linear activation function (such as ReLU or tanh), which are generally parameter free, and do not affect our construction.

We use the tensor sketch operation (Definition 2) to reduce either the dimensionality of the input feature map ($d_2$) or the output feature map ($d_1$) in the kernel tensor $\mathcal{K}$. In practice, the dimensions of the individual filters ($h$ and $w$) are small integers, which we therefore do not further reduce. The motivation for sketching along different dimensions comes from our mathematical analysis of the variance bounds (Theorem 3.1), where as in Proposition 2.1 based on the relationship between $\mathcal{I}_{\text{in}}$ and $\mathcal{K}$ the variance could be substantially smaller in one case or the other. Another trick that works as a simple boosting technique is to utilize multiple sketches each associated with an independent random matrix. Formally, we define a SK-CONV layer as follows (see also Figure 1).

**Definition 3.** *A* SK-CONV *layer is parametrized by a sequence of tensor-matrix pairs* $(\mathcal{S}_{1_1}, U_{1_1}), \ldots, (\mathcal{S}_{1_\ell}, U_{1_\ell})$, $(\mathcal{S}_{2_1}, U_{2_1}), \ldots, (\mathcal{S}_{2_\ell}, U_{2_\ell})$ *where for* $i \in [\ell]$ $\mathcal{S}_{1_i} \in \mathbb{R}^{d_2 \times h \times w \times k}$, $\mathcal{S}_{2_i} \in \mathbb{R}^{k \times h \times w \times d_1}$ *and* $U_{1_i} \in \mathbb{R}^{k \times d_1}$, $U_{2_i} \in$

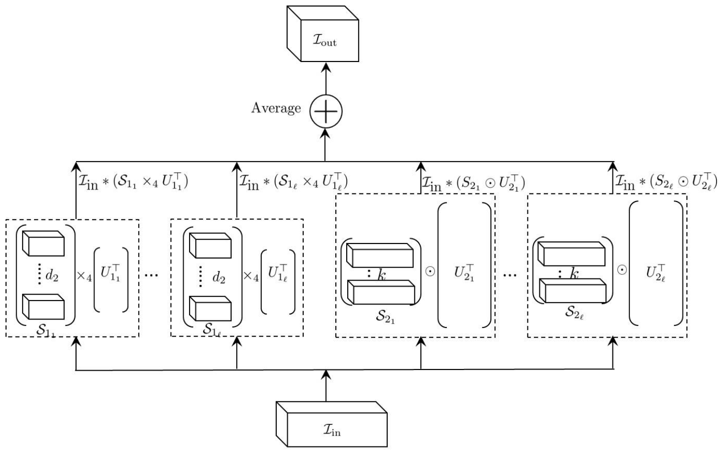
$\mathcal{I}_{\text{out}}$
Average
$\mathcal{I}_{\text{in}} * (\mathcal{S}_{1_1} \times_4 U_{1_1}^\top)$
$\mathcal{I}_{\text{in}} * (\mathcal{S}_{1_\ell} \times_4 U_{1_\ell}^\top)$
$\mathcal{I}_{\text{in}} * (S_{2_1} \odot U_{2_1}^\top)$
$\mathcal{I}_{\text{in}} * (S_{2_\ell} \odot U_{2_\ell}^\top)$
$d_2$
$\times_4$
$U_{1_1}^\top$
$\mathcal{S}_{1_1}$
$d_2$
$\times_4$
$U_{1_\ell}^\top$
$\mathcal{S}_{1_\ell}$
$k$
$\odot$
$U_{2_1}^\top$
$\mathcal{S}_{2_1}$
$k$
$\odot$
$U_{2_\ell}^\top$
$\mathcal{S}_{2_\ell}$
$\mathcal{I}_{\text{in}}$

Figure 1: A SK-CONV layer with parameters $(\mathcal{S}_{1_1}, U_{1_1}), \ldots, (\mathcal{S}_{1_\ell}, U_{1_\ell}), (\mathcal{S}_{2_1}, U_{2_1}), \ldots, (\mathcal{S}_{2_\ell}, U_{2_\ell})$.

$\mathbb{R}^{khw \times d_2 hw}$ *are independent random scaled sign matrices,*[4] *which on input* $\mathcal{I}_{\text{in}} \in \mathbb{R}^{h_1 \times w_1 \times d_2}$ *constructs* $\hat{\mathcal{I}}_{\text{out}}$ *as follows:*

$$\hat{\mathcal{I}}_{\text{out}} = \frac{1}{2\ell} \sum_{i=1}^{\ell} \mathcal{I}_{\text{in}} * (\mathcal{S}_{1_i} \times_4 U_{1_i}^\top) + \mathcal{I}_{\text{in}} * (\mathcal{S}_{2_i} \odot U_{2_i}^\top), \quad (1)$$

*where* $\mathcal{S}_{2_i} \odot U_{2_i}^\top \in \mathbb{R}^{d_2 \times h \times w \times d_1}$ *is defined as*[5]

$$(\mathcal{S}_{2_i} \odot U_{2_i}^\top)_{xyzs} = \sum_{c=1}^{k} \sum_{i=1}^{h} \sum_{j=1}^{w} \mathcal{S}_{2_{i_{cijs}}} U_{2_{i(cij)(xyz)}}.$$

*Here* $(\mathcal{S}_{2_i} \odot U_{2_i}^\top)_{xyzs}$ *is the* $(x, y, z, s)$*th entry,* $\mathcal{S}_{2_{i_{cijs}}}$ *is the* $(c, i, j, s)$*th entry, and* $U_{2_{i(cij)(xyz)}}$ *is the* $(cij, xyz)$*th entry in* $(\mathcal{S}_{2_i} \odot U_{2_i}^\top)$, $\mathcal{S}_{2_i}$*, and* $U_{2_i}$*, respectively.*

By running multiple sketches in parallel on the same input and taking the average, also results in a more stable performance across different choices of the random matrices The number of free parameters overall in all the $\mathcal{S}_{1_i}$ and $\mathcal{S}_{2_i}$ tensors put together equals $\ell hwk(d_1 + d_2)$.[6] Therefore, with a SK-CONV layer, we get a reduction in the number of parameters compared to a traditional convolutional layer (with $hwd_1d_2$ parameters) if $k\ell \leq d_1 d_2/(d_1 + d_2)$. With this reduction, the time for computing $\hat{\mathcal{I}}_{\text{out}}$, ignoring dependence on $h$ and $w$, reduces from $O(h_2 w_2 d_1 d_2)$ (in a traditional CONV layer) to $O(h_2 w_2 \ell k(d_1 + d_2))$ (in a SK-CONV layer).

**Implementing a SK-CONV Layer with Matrix Multiplications.** We next discuss how to implement a SK-CONV layer using just matrix multiplications. The convolution operation can be reduced into a matrix multiplication, an idea that is exploited by many deep learning frameworks [Chetlur *et al.*, 2014]. The idea is to reformulate the kernel tensor $\mathcal{K}$ by flattening it along the dimension representing the output feature

[3]In a commonly used setting, with stride of 1 and zero-padding of 0, $h_2 = h_1 - h + 1$ and $w_2 = w_1 - w + 1$, and $\mathcal{I}_{\text{out}} \in \mathbb{R}^{(h_1 - h + 1) \times (w_1 - w + 1) \times d_1}$ is defined as: $\mathcal{I}_{\text{out}_{xys}} = \sum_{i=1}^{h} \sum_{j=1}^{w} \sum_{c=1}^{d_2} \mathcal{K}_{cijs}\, \mathcal{I}_{\text{in}_{(x+i-1)(y+j-1)c}}$.

[4]We define $U_{2_i} \in \mathbb{R}^{khw \times d_2 hw}$ (instead of $U_{2_i} \in \mathbb{R}^{k \times d_2}$) for simplifying the construction.

[5]Let $\mathcal{O}_i = \mathcal{S}_{2_i} \odot U_{2_i}^\top$. The $\odot$ operation can be equivalently defined: $\text{mat}_4(\mathcal{O}_i) = U_{2_i}^\top \text{mat}_4(\mathcal{S}_{2_i})$.

[6]The random matrices, once picked are not changed during the training or deployment.

map, which in our setting is represented along the fourth dimension of $\mathcal{K}$. The input tensor $\mathcal{I}_{\text{in}}$ is used to form a matrix $I_{\text{in}} \in \mathbb{R}^{h_2 w_2 \times d_2 hw}$. This construction is quite standard and we refer the reader to [Chetlur *et al.*, 2014] for more details. Then it follows that $\hat{I}_{\text{out}}$ defined as $I_{\text{in}} \text{mat}_4(\mathcal{K}) \in \mathbb{R}^{h_2 w_2 \times d_1}$ is a reshaping of the output tensor $\hat{\mathcal{I}}_{\text{out}}$ (i.e., $\hat{I}_{\text{out}} = \text{mat}_3(\hat{\mathcal{I}}_{\text{out}})$).

Using this equivalence and simple algebraic observations ($\text{mat}_4(\mathcal{S}_{1_i} \times_4 U_{1_i}^\top) = \text{mat}_4(\mathcal{S}_{1_i}) U_{1_i}$ and $\text{mat}_4(\mathcal{S}_{2_i} \odot U_{2_i}^\top) = U_{2_i}^\top \text{mat}_4(\mathcal{S}_{2_i})$), we can re-express the operation in (1) as:

$$\hat{I}_{\text{out}} = \frac{1}{2\ell} \sum_{i=1}^{\ell} I_{\text{in}} \text{mat}_4(\mathcal{S}_{1_i}) U_{1_i} + I_{\text{in}} U_{2_i}^\top \text{mat}_4(\mathcal{S}_{2_i}). \quad (2)$$

Or in other words, $\hat{I}_{\text{out}}$ equals to

$$\frac{1}{2\ell} \sum_{i=1}^{\ell} I_{\text{in}} (\text{mat}_4(\mathcal{S}_{1_i}) \times_2 U_{1_i}^\top) + I_{\text{in}} (\text{mat}_4(\mathcal{S}_{2_i}) \times_1 U_{2_i}^\top).$$

We use this matrix representation ($\hat{I}_{\text{out}}$) of $\hat{\mathcal{I}}_{\text{out}}$ in our implementation of a SK-CONV layer both in the forward pass and when deriving the gradients during back-propagation.

**Theoretical Guarantees of a SK-CONV Layer.** Given a traditional convolutional layer with kernel tensor $\mathcal{K}$ and independent random scaled sign matrices $U_{1_1}, \dots, U_{1_\ell}, U_{2_1}, \dots, U_{2_\ell}$, we can form a corresponding SK-CONV layer by constructing tensors $\mathcal{S}_{1_1}, \dots, \mathcal{S}_{1_\ell}, \mathcal{S}_{2_1}, \dots, \mathcal{S}_{2_\ell}$ such that $\text{mat}_4(\mathcal{S}_{1_i}) = \text{mat}_4(\mathcal{K}) U_{1_i}^\top$ and $\text{mat}_4(\mathcal{S}_{2_i}) = U_{2_i} \text{mat}_4(\mathcal{K})$ for $i \in [\ell]$. The next theorem based on Proposition 2.1, analyzes the expectation and the variance of using these sketches as an estimator for $\mathcal{I}_{\text{out}} = \mathcal{I}_{\text{in}} * \mathcal{K}$ ($\equiv I_{\text{in}} \text{mat}_4(\mathcal{K})$).

**Theorem 3.1** (Theoretical Guarantees of a SK-CONV Layer)**.** *Let $\mathcal{K} \in \mathbb{R}^{d_2 \times h \times w \times d_1}$ be a kernel tensor. Let $U_{1_1}, \dots, U_{1_\ell} \in \mathbb{R}^{k \times d_1}$ and $U_{2_1}, \dots, U_{2_\ell} \in \mathbb{R}^{khw \times d_2 hw}$ be a set of independent random scaled sign matrices. Let $\mathcal{S}_{1_1}, \dots, \mathcal{S}_{1_\ell}, \mathcal{S}_{2_1}, \dots, \mathcal{S}_{2_\ell}$ be tensors defined as above. Then for any input tensor $\mathcal{I}_{\text{in}} \in \mathbb{R}^{h_1 \times w_1 \times d_2}$ with $\mathcal{I}_{\text{out}} = \mathcal{I}_{\text{in}} * \mathcal{K}$, we have:*

1. *Unbiased Estimation:* $\mathbb{E}[\hat{\mathcal{I}}_{\text{out}}] = \mathcal{I}_{\text{out}}$.
2. *Variance Bound:*

$$\mathbb{E}\left[\left\|\hat{\mathcal{I}}_{\text{out}} - \mathcal{I}_{\text{out}}\right\|_F^2\right] \leq \frac{d_1 \|\mathcal{I}_{\text{in}} * \mathcal{K}\|_F^2}{\ell k} + \frac{\|\mathcal{I}_{\text{in}}\|_F^2 \|\mathcal{K}\|_F^2}{\ell khw}.$$

**Training a SK-CONV Layer**

In this section, we discuss a procedure for training a SK-CONV layer. Let Loss() denote some loss function for the network. For computational and space efficiency, our goal will be to perform the training without ever needing to construct the complete kernel tensor ($\mathcal{K}$). We focus on deriving the gradient of the loss with respect to the parameters in a SK-CONV layer, which can then be used for back-propagating gradients.

We can again exploit the equivalence between the convolution operation and matrix multiplication. Consider the operation performed in the SK-CONV layer as defined in (2). Let $G = \frac{\partial \text{Loss}}{\partial \hat{I}_{\text{out}}} \in \mathbb{R}^{h_2 w_2 \times d_1}$. For $i \in [\ell]$,[7]

$$\frac{\partial \text{Loss}}{\partial \, \text{mat}_4(\mathcal{S}_{1_i})} = \frac{I_{\text{in}}^\top G U_{1_i}^\top}{2\ell}, \quad \frac{\partial \text{Loss}}{\partial \, \text{mat}_4(\mathcal{S}_{2_i})} = \frac{U_{2_i} I_{\text{in}}^\top G}{2\ell}, \text{ and}$$

$$\frac{\partial \text{Loss}}{\partial I_{\text{in}}} = \sum_{i=1}^{\ell} \frac{G U_{1_i}^\top \, \text{mat}_4(\mathcal{S}_{1_i})^\top}{2\ell} + \sum_{i=1}^{\ell} \frac{G \, \text{mat}_4(\mathcal{S}_{2_i})^\top U_{2_i}}{2\ell}.$$

Notice that all the required operations can be carried out without ever explicitly forming the complete $d_2 \times h \times w \times d_1$ sized kernel tensor.

## 3.2 Sketching Fully Connected Layers

Neurons in a fully connected (FC) layer have full connections to all activations in the previous layer. These layers apply a linear transformation of the input. Let $W \in \mathbb{R}^{d_1 \times d_2}$ represent a *weight* matrix and $\mathbf{b} \in \mathbb{R}^{d_1}$ represent a *bias* vector. The operation of the FC layer on input $\mathbf{h}_{\text{in}}$ can be described as:

$$\mathbf{a} = W \mathbf{h}_{\text{in}} + \mathbf{b}. \quad (3)$$

Typically, the FC layer is followed by application of some non-linear activation function. As in the case of CONV layers, our construction is independent of the applied activation function and we omit further discussion of these functions. Our idea is to use the tensor sketch operation (Definition 2) to sketch either the columns or rows of the weight matrix.

**Definition 4.** *A SK-FC layer is parametrized by a bias vector $\mathbf{b} \in \mathbb{R}^{d_1}$ and a sequence of matrix pairs $(S_{1_1}, U_{1_1}), \dots, (S_{1_\ell}, U_{1_\ell}), (S_{2_1}, U_{2_1}), \dots, (S_{2_\ell}, U_{2_\ell})$ where for $i \in [\ell]$, $S_{1_i} \in \mathbb{R}^{k \times d_2}$, $S_{2_i} \in \mathbb{R}^{d_1 \times k}$ and $U_{1_i} \in \mathbb{R}^{k \times d_1}$, $U_{2_i} \in \mathbb{R}^{k \times d_2}$ are independent random scaled sign matrices, which on input $\mathbf{h}_{\text{in}} \in \mathbb{R}^{d_2}$ constructs $\hat{\mathbf{a}}$ as follows:*

$$\hat{\mathbf{a}} = \frac{1}{2\ell} \sum_{i=1}^{\ell} U_{1_i}^\top S_{1_i} \mathbf{h}_{\text{in}} + \frac{1}{2\ell} \sum_{i=1}^{\ell} S_{2_i} U_{2_i} \mathbf{h}_{\text{in}} + \mathbf{b}. \quad (4)$$

Note that $\hat{\mathbf{a}}$ in the above definition could be equivalently represented as $\hat{\mathbf{a}} = \frac{1}{2\ell} \sum_{i=1}^{\ell} (S_{1_i} \times_1 U_{1_i}^\top) \mathbf{h}_{\text{in}} + \frac{1}{2\ell} \sum_{i=1}^{\ell} (S_{2_i} \times_2 U_{2_i}^\top) \mathbf{h}_{\text{in}} + \mathbf{b}$. The number of free parameters overall in all the $S_{1_i}$ and $S_{2_i}$ matrices put together is $\ell k(d_1 + d_2)$. Hence, compared to a traditional weight matrix $W \in \mathbb{R}^{d_1 \times d_2}$, we get a reduction in the number of parameters if $k\ell \leq d_1 d_2 / (d_1 + d_2)$. Another advantage is that the time needed for computing the pre-activation value ($\hat{\mathbf{a}}$ in (4)) in a SK-FC layer is $O(\ell k(d_1 + d_2))$ which is smaller than the $O(d_1 d_2)$ time needed in the traditional FC setting if the values of $k$ and $\ell$ satisfy the above condition.

**Theoretical Guarantees of SK-FC Layer.** Given a traditional FC layer with weight matrix $W$ (as in (3)), and independent random scaled sign matrices $U_{1_1}, \dots, U_{1_\ell}, U_{2_1}, \dots, U_{2_\ell}$, we can form a corresponding SK-FC layer by setting $S_{1_i} = U_{1_i} W$ and $S_{2_i} = W U_{2_i}^\top$. We now analyze certain properties of this construction. The following theorem, based on Proposition 2.1, analyzes the expectation and the variance of using these sketches as an estimator for $\mathbf{a} = W \mathbf{h}_{\text{in}} + \mathbf{b}$ for a vector $\mathbf{h}_{\text{in}} \in \mathbb{R}^{d_2}$.

[7]The gradients computed with respect to $\text{mat}_4(\mathcal{S}_{1_i})$ and $\text{mat}_4(\mathcal{S}_{2_i})$ can also be converted into a tensor by reversing the $\text{mat}_4()$ operator.

**Theorem 3.2.** *Let $W \in \mathbb{R}^{d_1 \times d_2}$. Let $U_{1_1}, \dots, U_{1_\ell} \in \mathbb{R}^{k \times d_1}$ and $U_{2_1}, \dots, U_{2_\ell} \in \mathbb{R}^{k \times d_2}$ be a set of independent random scaled sign matrices. Let $S_{1_i} = U_{1_i} W (= W \times_1 U_{1_i})$ and $S_{2_i} = W U_{2_i}^\top (= W \times_2 U_{2_i})$ for $i \in [\ell]$. Then for any $\mathbf{h}_{\text{in}} \in \mathbb{R}^{d_2}$ and $\mathbf{b} \in \mathbb{R}^{d_1}$ with $\mathbf{a} = W\mathbf{h}_{\text{in}} + \mathbf{b}$:*

1. *Unbiased Estimation:* $\mathbb{E}[\hat{\mathbf{a}}] = \mathbf{a}$
2. *Variance Bound:*

$$\mathbb{E}\left[\|\hat{\mathbf{a}} - \mathbf{a}\|^2\right] \leq \frac{d_1 \|W\mathbf{h}_{\text{in}}\|^2}{\ell k} + \frac{\|W\|_F^2 \|\mathbf{h}_{\text{in}}\|^2}{\ell k}.$$

**Training a SK-FC Layer**

In this section, we discuss a procedure for training a network containing SK-FC layers. Let Loss() denote some loss function for the network. Let $\mathbf{c} = S_2 U_2 \mathbf{h}_{\text{in}} + \mathbf{b}$. Let $\mathbf{g} = \frac{\partial \text{Loss}}{\partial \mathbf{c}}$. In this case, using chain-rule of calculus

$$\frac{\partial \text{Loss}}{\partial S_2} = \mathbf{g}\mathbf{h}_{\text{in}}^\top U_2^\top = (\mathbf{g}\mathbf{h}_{\text{in}}^\top) \times_2 U_2. \quad (5)$$

Similarly, the gradient with respect to $\mathbf{h}_{\text{in}}$ is:

$$\frac{\partial \text{Loss}}{\partial \mathbf{h}_{\text{in}}} = (S_2 U_2)^\top \mathbf{g} = (S_2 \times_2 U_2^\top)^\top \mathbf{g}. \quad (6)$$

Now let $\mathbf{c} = U_1^\top S_1 \mathbf{h}_{\text{in}} + \mathbf{b} = (S_1^\top U_1)^\top \mathbf{h}_{\text{in}} + \mathbf{b}$. Again let $\mathbf{g} = \frac{\partial \text{Loss}}{\partial \mathbf{c}}$. Applying chain-rule gives $\frac{\partial \text{Loss}}{\partial S_1} = \sum_{i=1}^{d_1} \frac{\partial \text{Loss}}{\partial c_i} \frac{\partial c_i}{\partial S_1}$, where $c_i$ denotes the $i$th entry of $\mathbf{c}$. We can compute $\frac{\partial c_i}{\partial S_1}$ as $\frac{\partial c_i}{\partial S_1} = \frac{\partial\, \mathbf{u}_{1_i}^\top S_1 \mathbf{h}_{\text{in}}}{\partial S_1} = \mathbf{u}_{1_i} \mathbf{h}_{\text{in}}^\top$, where $\mathbf{u}_{1_i}$ is the $i$th column in $U_1$. Therefore, we get

$$\frac{\partial \text{Loss}}{\partial S_1} = \sum_{i=1}^{d_1} g_i \mathbf{u}_{1_i} \mathbf{h}_{\text{in}}^\top = U_1 \mathbf{g}\mathbf{h}_{\text{in}}^\top = (\mathbf{g}\mathbf{h}_{\text{in}}^\top) \times_1 U_1, \quad (7)$$

where $g_i$ denotes the $i$th entry of $\mathbf{g}$. Finally, the gradient with respect to $\mathbf{h}_{\text{in}}$ in this case equals:

$$\frac{\partial \text{Loss}}{\partial \mathbf{h}_{\text{in}}} = (S_1^\top U_1)\mathbf{g} = (S_1 \times_1 U_1^\top)^\top \mathbf{g}. \quad (8)$$

Putting together (5), (6), (7), and (8) gives the necessary gradients for the SK-FC layer (where $\hat{\mathbf{a}}$ is defined using (4)). Let $\mathbf{g} = \frac{\partial \text{Loss}}{\partial \hat{\mathbf{a}}}$. For $i \in [\ell]$,

$$\frac{\partial \text{Loss}}{\partial S_{1_i}} = \frac{U_{1_i} \mathbf{g}\mathbf{h}_{\text{in}}^\top}{2\ell}, \frac{\partial \text{Loss}}{\partial S_{2_i}} = \frac{\mathbf{g}\mathbf{h}_{\text{in}}^\top U_{2_i}^\top}{2\ell}, \text{ and}$$
$$\frac{\partial \text{Loss}}{\partial \mathbf{h}_{\text{in}}} = \sum_{i=1}^{\ell} \frac{S_{1_i}^\top U_{1_i} \mathbf{g}}{2\ell} + \sum_{i=1}^{\ell} \frac{U_{2_i}^\top S_{2_i} \mathbf{g}}{2\ell}.$$

Note that all the above computations can be performed without ever explicitly forming the complete $d_1 \times d_2$ weight matrix.

## 3.3 Final Construction of $\widehat{\text{NN}}$

Given a convolutional neural network NN, construct $\widehat{\text{NN}}$, an approximation of NN, by replacing the convolutional layers (resp. fully connected layers) with SK-CONV layers (resp. SK-FC layers). A nice feature about this construction is that, based on need, we can also choose to replace only some of the layers of the NN with their sketch counterpart layers.

## 4 Comparison to Previous Work

Deep neural networks are typically over-parameterized, and there is significant redundancy in deep learning networks [Denil *et al.*, 2013]. There have been several previous attempts to reduce the complexity of deep NN under a variety of contexts.

Most relevant to our paper is a line of work on approximating both the fully connected and convolutional layers. Denil *et al.* [Denil *et al.*, 2013], suggested an approach based on learning a low-rank factorization of the matrices (tensors are viewed as a matrix) involved within each layer of a CNN. Instead of learning both the factors of a factorization during training, the authors suggest techniques for carefully constructing one of the factors (called the dictionary), while only learning the other one. Our sketching-based approach is related to low-rank factorization, however using sketching we eliminate the overhead of carefully constructing the dictionary. Tai *et al.* [Tai *et al.*, 2016] achieve parameter reduction using a tensor decomposition technique that is based on replacing the convolutional kernel with two consecutive kernels with a lower rank. The issue with this approach is that with the increased depth of the resulting network, training becomes more challenging, and the authors rely on *batch normalization* (proposed by [Ioffe and Szegedy, 2015]) to overcome this issue. In our proposed approach, the depth of the reduced network remains equal to that of the original network, and the reduced network can be trained with or without batch normalization. Chen *et al.* [Chen *et al.*, 2016] combine the hashing idea from [Chen *et al.*, 2015] along with the discrete cosine transform (DCT) to compress filters in a convolutional layer. Their architecture, called FreshNets, first converts filter weights into frequency domain using discrete cosine transform and then uses the hashing idea to randomly group the resulting frequency parameters into buckets. Our sketches are created by using random projections which is related to the hashing trick used in these results, however, our techniques are naturally attractive for convolutional neural networks as they are known to be preserve spatial locality [Johnson and Lindenstrauss, 1984], a property that is not preserved by simple hashing. Also, in contrast to FreshNets, our architectures require just simple linear transformations for both fully connected and convolutional layers, and do not require special routines for DCT, Inverse DCT, etc. Additionally, we provide theoretical bounds on the quality of approximation that is missing in these previous studies.

There is a long line of work on reducing model memory size based on post-processing a trained network (with sometimes further fine-tuning of the compressed model) [Gong *et al.*, 2014; Han *et al.*, 2015; Han *et al.*, 2016; Soulié *et al.*, 2015; Wu *et al.*, 2015; Guo *et al.*, 2016; Kim *et al.*, 2016; Wang *et al.*, 2016; Hubara *et al.*, 2016a; Hubara *et al.*, 2016b; Zhu *et al.*, 2016; Li *et al.*, 2016]. Techniques such as pruning, binarization, quantization, low-rank decomposition, etc., are intermingled with training of a network on a dataset to construct a reduced model. These results do not achieve a direct network approximation as the training happens on the original network. In practice, one can combine our approach with some of the above proposed model post-processing techniques to further reduce the storage requirements of the trained model.

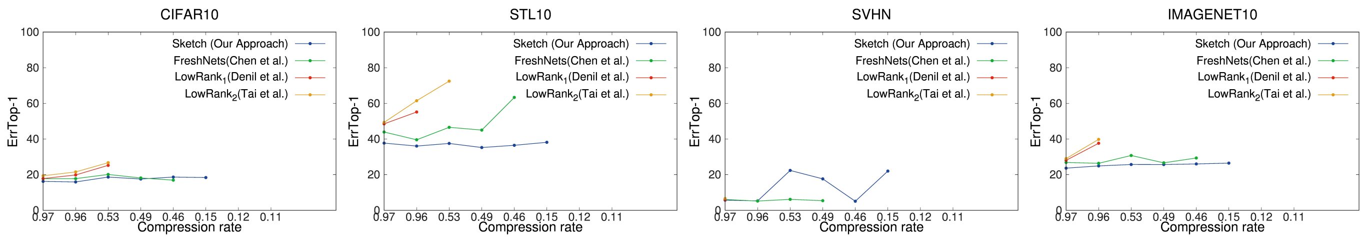

Figure 2: Top-1 error for the NinN architecture as we decrease the compression rate by compressing one convolutional layer at a time each by a factor of 10. The x-axis is not to scale.

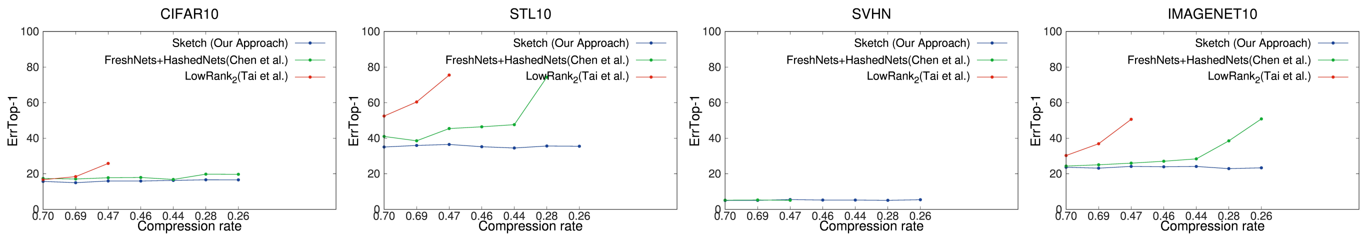

Figure 3: Top-1 error for the NinN+FC architecture as we decrease the compression rate by compressing one convolutional layer at a time each by a factor of 4 and we compress the fully connected layer by a factor of 4. The x-axis is not to scale. The size of FC layer is about half of the total size of convolutional layers CONV$_2$ to CONV$_8$.

## 5 Experimental Evaluation

In this section, we experimentally demonstrate the effectiveness of our proposed network approximation approach.

**Metrics.** We define *compression rate* as the ratio between the number of parameters in the reduced (compressed) network architecture and the number of parameters in the original (uncompressed) network architecture. The top-1 error of a trained model is denoted by ERRTOP-1.

**Datasets.** We use 5 popular image datasets: CIFAR10, SVHN, STL10, ImageNet10 (a subset of ImageNet1000 dataset), and Places2. Note that, Places2 is a challenging dataset that was used in the ILSVRC 2016 "Scene Classification" challenge.

**Network Architectures.** We present our experiments on two different network architectures: Network-in-Network [Lin *et al.*, 2014] (NinN) and GoogLeNet [Szegedy *et al.*, 2015] (which we use for the Places2 dataset). The choice of architectures was done keeping in mind limited computational resources at our disposal and a recent trend of moving away from fully connected layers in CNNs. A common observation is that reducing the number of parameters in convolutional layers seems to be a much more challenging problem than that for fully connected layers. NinN achieves a baseline top-1 error of 17.7, 43.2, 6.0, and 27.1 on the CIFAR10, STL10, SVHN, and ImageNet10 datasets respectively. Similarly, GoogLeNet achieves a baseline top-1 error of 32.3% on the Places2 dataset.

**Baseline Techniques.** We compare our approach with four state-of-the-art techniques that approximate *both* the convolutional and the fully connected layers: FreshNets technique that uses hashing in the frequency domain to approximate the convolutional layer [Chen *et al.*, 2016], low-rank decomposition technique of [Denil *et al.*, 2013] (LOWRANK$_1$), and tensor decomposition technique of [Tai *et al.*, 2016] (LOWRANK$_2$). While using the FreshNets, we also use the HashedNets technique of feature hashing [Chen *et al.*, 2015] for compressing the fully connected layers as suggested by [Chen *et al.*, 2016].

**Results.** Figure 2 shows the results of our first set of experiments. In this case, we use the NinN architecture. If a point is missing in the plots then the corresponding network training failed. We expect the error to go up as we decrease the compression rate, i.e., increase the parameter reduction. We observe this general trend in almost all our plots, with minor fluctuations on the SVHN dataset. We make two main observations from these plots. First, our method was always able to get to a better compression rate compared to other techniques, in that these comparative techniques started failing sooner as we kept decreasing the compression rate. For example, our approach consistently achieves a compression rate of 0.15 that none of the other techniques even get close to achieving. Second, our approach also almost always achieves better accuracy when compared to other techniques. As explained in Section 4, our approach has some advantages over the compared techniques, especially in terms of its ability to approximate (compress) the convolutional layers.

Next we consider results on both the convolutional and fully connected layers. We now add fully connected layers into the mix. To do so, we used a modified NinN architecture (denoted as NinN+FC) in our experiments where we replaced the last convolution layer (CONV$_9$) with a fully connected layer of size $768 \times 768$ followed by a classifier layer of size $768 \times 10$. In Figure 3, we present the results of these experiments. Our approach again outperforms other techniques in terms of both accuracy and the maximum achievable compression rate. The results demonstrate the effectiveness of proposed approach on both the convolutional and fully connected layers.

An interesting observation from our experiments is that we can gain up to 4% or lose up to 2% of accuracy compared to original network accuracy. The fact that sometimes our reduced network was able to gain a bit of accuracy over the original network suggests that our randomized technique also acts as an implicit regularizer (thus preventing overfitting) during training.

To evaluate our approach on a large dataset, we ran additional experiments on the Places2 dataset (using a centered crop). Here we used the GoogLeNet architecture with batch normalization. Due to limited computational resources, we ran a single experiment where we compressed all but the first layer to achieve a compression rate of about 0.2. At this compression level, training for none of the competitor methods succeeded, whereas, our approach gave a top-1 error of 36.4%. Note that the top-1 error of the original GoogLeNet on this dataset is 32.3%. This demonstrates that our approach manages to generate smaller networks that perform well even on large datasets.

## References

[Chen *et al.*, 2015] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *ICML*, 2015.

[Chen *et al.*, 2016] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. Compressing convolutional neural networks in the frequency domain. In *KDD*, 2016.

[Cheng *et al.*, 2015] Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *ICCV*, pages 2857–2865, 2015.

[Chetlur *et al.*, 2014] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *ArXiv*, 2014.

[Denil *et al.*, 2013] Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In *NIPS*, pages 2148–2156, 2013.

[Garipov *et al.*, 2016] Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: compressing convolutional and fc layers alike. *ArXiv*, 2016.

[Gong *et al.*, 2014] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *ArXiv*, 2014.

[Goodfellow *et al.*, 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[Guo *et al.*, 2016] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *NIPS*, 2016.

[Han *et al.*, 2015] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NIPS*, 2015.

[Han *et al.*, 2016] Song Han, Huizi Mao, and William J Dally. A deep neural network compression pipeline: Pruning, quantization, huffman encoding. In *ICLR*, 2016.

[Hubara *et al.*, 2016a] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

[Hubara *et al.*, 2016b] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*, 2016.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.

[Johnson and Lindenstrauss, 1984] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[Kim *et al.*, 2016] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *ICLR*, 2016.

[Li *et al.*, 2016] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[Lin *et al.*, 2014] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *ICLR*, 2014.

[Sindhwani *et al.*, 2015] Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. In *NIPS*, 2015.

[Soulié *et al.*, 2015] Guillaume Soulié, Vincent Gripon, and Maëlys Robert. Compression of deep neural networks on the fly. *ArXiv*, 2015.

[Szegedy *et al.*, 2015] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[Tai *et al.*, 2016] Cheng Tai, Tong Xiao, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. In *ICLR*, 2016.

[Umans, 1998] Christopher Umans. The minimum equivalent dnf problem and shortest implicants. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 556–563. IEEE, 1998.

[Wang *et al.*, 2016] Yunhe Wang, Chang Xu, Shan You, Dacheng Tao, and Chao Xu. Cnnpack: Packing convolutional neural networks in the frequency domain. In *NIPS*, 2016.

[Woodruff, 2014] David P. Woodruff. Sketching as a tool for numerical linear algebra. *FnT-TCS*, 2014.

[Wu *et al.*, 2015] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. *ArXiv*, 2015.

[Yang *et al.*, 2015] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *ICCV*, 2015.

[Zhu *et al.*, 2016] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.