Deep Neural Network Approximation using Tensor Sketching

Shiva Prasad Kasiviswanathan*† Nina Narodytska‡† Hongxia Jin§

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 [11, 2, 5, 48, 37, 3, 41, 14].[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 [43], 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

*Amazon ML. Work done while the author was at Samsung Research America, Mountain View, CA, USA. kasivisw@gmail.com.

†Equal Contributions.

‡VMware Research, Palo Alto, CA, USA. n.narodytska@gmail.com.

§Samsung Research America, Mountain View, CA, USA. hongxia.jin@samsung.com .

[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.

network NN. Let $\phi : \Gamma \times \Theta \to \Omega$ denote the function (transformation) applied by this layer, where Θ represents the parameter space of the function (generally, a tensor space of some order), Γ and Ω represent the input and output space respectively. Our general idea is to replace ϕ 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 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 θ. 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 Θ by performing random linear projections along different dimensions of Θ, and then perform a simple combination of these sketches. This new operation $\hat{\phi}$ defines a new layer that approximates the functionality ϕ of the layer L. Since $\hat{\phi}$ and ϕ 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 $\widehat{\text{NN}}$, 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 $\widehat{\text{NN}}$? 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 $\widehat{\text{NN}}$ directly on D to get a reduced model $\widehat{\text{NN}}_D$.[2] Our extensive 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, which other state-of-the-art network approximation techniques seem not to succeed on.

1.1 Preliminaries

We denote $[n] = \{1, \ldots, n\}$. 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 $U = Z/\sqrt{k}$.*

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 $\mathbb{E}[MU^\top U] = M\mathbb{E}[U^\top U] = M$.

Tensor Preliminaries. We denote matrices by uppercase letters and higher dimensional tensors by Euler script letters, e.g., $\mathcal{T}$. A real pth 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 $i \in [p]$. As is the case for vectors (where $p = 1$) and matrices (where $p = 2$), we may identify a pth order tensor with the p-way array of real numbers. The different dimensions of the tensor are referred to as *modes*. The $(i_1, \ldots, i_p)$th entry of a tensor $\mathcal{T}$ is denoted by $\mathcal{T}_{i_1 i_2 \ldots i_p}$.

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 $\mathcal{T} \times_n M$ and has dimensions $d_1 \times \cdots \times d_{n-1} \times k \times d_{n+1} \times \ldots d_p$. Elementwise, we have

$$(\mathcal{T} \times_n M)_{i_1 \ldots i_{n-1} j i_{n+1} \ldots i_p} = \sum_{i_n=1}^{d_n} \mathcal{T}_{i_1 i_2 \ldots i_p} M_{j i_n}.$$

[2] There memory footprint of the reduced model $\widehat{\text{NN}}_D$ can be further reduced using various careful operations such as pruning, binarization, quantization, low-rank decomposition, etc., [15, 20, 19, 38, 47, 17, 28, 45, 23, 24, 32, 50], which is beyond the scope of this work.

Note that the operation can also be applied simultaneously to multiple modes. In general, given p matrices $M_1, \ldots, M_p$ where $M_i \in \mathbb{R}^{k_i \times d_i}$, the resulting tensor $\mathcal{T} \times_1 M_1 \times_2 M_2 \cdots \times_p M_p$ is a tensor in $\mathbb{R}^{k_1 \times k_2 \cdots \times k_p}$. For a matrix $W \in \mathbb{R}^{d_1 \times d_2}$ is a matrix, it follows that: $W \times_1 M_1 = M_1 W$ and $W \times_2 M_2 = W M_2^\top$.

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 mat_n) is a matrix whose columns correspond to mode-n fibers of $\mathcal{T}$. For example, in a fourth order tensor $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3 \times d_4}$, $T = \mathrm{mat}_4(\mathcal{T}) \in \mathbb{R}^{d_1 d_2 d_3 \times d_4}$ is a matrix defined as: $T_{(i_1 + d_1(i_2-1) + d_1 d_2 (i_3 - 1))i_4} = \mathcal{T}_{i_1 i_2 i_3 i_4}$, i.e., the (i_1, i_2, i_3, i_4) entry in the tensor $\mathcal{T}$ is assigned to the location $(i_1 + d_1(i_2 - 1) + d_1 d_2 (i_3 - 1), i_4)$ in the matrix T.

The weights of all (two dimensional) filters in a convolutional layer can be denoted by a 4-dimensional tensor in $\mathbb{R}^{d_2 \times w \times h \times d_1}$ where d_1 and d_2 represent the number of output and input feature maps, and h and w represent the height and width of the filter kernels.

2 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., [46]. 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 defined as follows.

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 $\mathcal{S}_n = \mathcal{T} \times_n U_n$.*

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. $W \times_2 U_2 = W U_2^\top$). 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 (proof in Appendix A) analyzes the variance of these estimators. The result will motivate our construction of sketch-based convolutional and fully connected layers in the next section.

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 $S_2 = W U_2^\top (= W \times_2 U_2)$. 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. $\mathbb{E}\left[\left\|U_1^\top S_1 M - WM\right\|_F^2\right] \leq \frac{2 d_1 \|WM\|_F^2}{k}$, *and*

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

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 [16]. 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* $U_{1_i} \in \mathbb{R}^{k \times d_1}$, $U_{2_i} \in \mathbb{R}^{khw \times d_2 hw}$ *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* $\mathcal{I}_{\text{out}}$ *as follows:*

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

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]

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

Here $(\mathcal{S}_{2_i} \odot U_{2_i}^\top)_{xyzs}$ *is the* (x, y, z, s)*th entry,* $\mathcal{S}_{2_{icijs}}$ *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 (see the experimental discussion in Appendix C). 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 $k\ell \leq d_1d_2/(d_1 + d_2)$. With this reduction, the time for computing I_{out}, ignoring dependence on h and w, reduces from $O(h_2w_2d_1d_2)$ (in a traditional CONV layer) to $O(h_2w_2\ell k(d_1 + d_2))$ (in a SK-CONV layer).

The convolution operation can be reduced into a matrix multiplication, an idea that is exploited by many deep learning frameworks [6]. The idea is to reformulate the kernel tensor $\mathcal{K}$ by flattening it along the dimension representing the output feature 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 $I_{\text{in}} \in \mathbb{R}^{h_2w_2 \times d_2hw}$. This construction is quite standard and we refer the reader to [6] for more details. Then it follows that I_{out} defined as $I_{\text{in}}\text{mat}_4(\mathcal{K}) \in \mathbb{R}^{h_2w_2 \times d_1}$ is a reshaping of the output tensor $\mathcal{I}_{\text{out}}$ (i.e., $I_{\text{out}} = \text{mat}_3(\mathcal{I}_{\text{out}})$).

[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 $U_{2_i} \in \mathbb{R}^{khw \times d_2hw}$ (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.

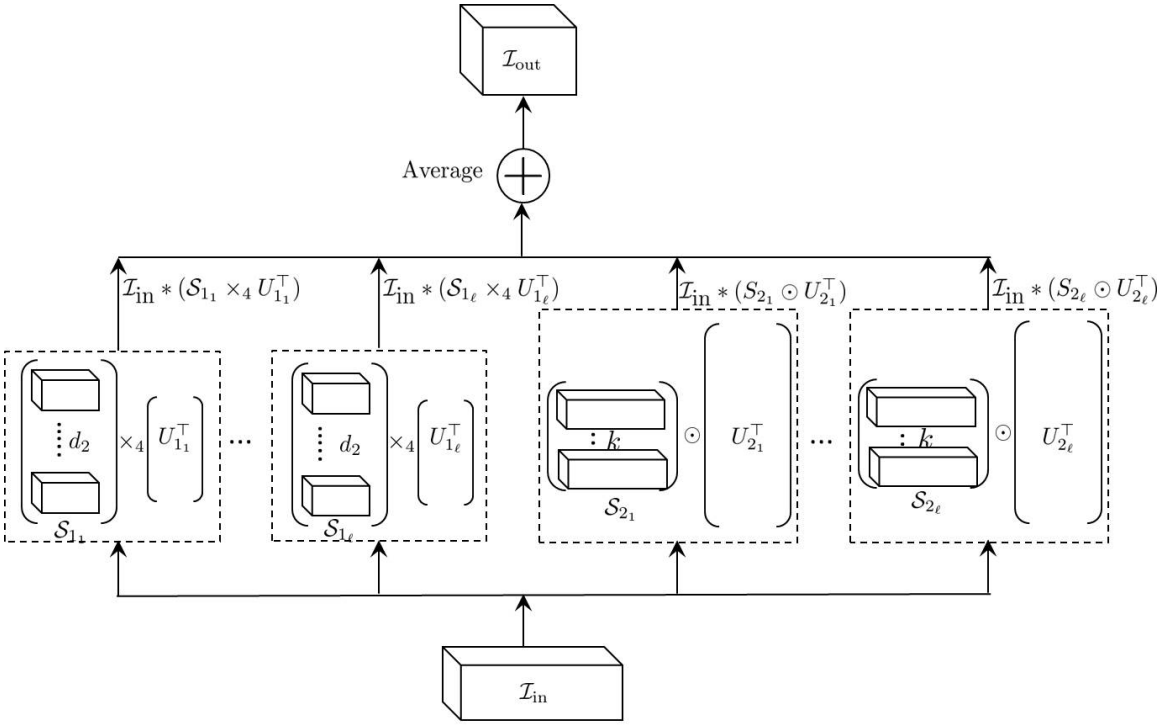

Figure 1: A SK-CONV layer with parameters $(\mathcal{S}_{1_1}, U_{1_1}), \ldots, (\mathcal{S}_{1_\ell}, U_{1_\ell}), (\mathcal{S}_{2_1}, U_{2_1}), \ldots, (\mathcal{S}_{2_\ell}, U_{2_\ell})$.

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:

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

Or in other words,

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

Theoretical Guarantees of a SK-CONV Layer. Given a traditional convolutional layer with kernel tensor $\mathcal{K}$ and independent random scaled sign matrices $U_{1_1}, \ldots, U_{1_\ell}, U_{2_1}, \ldots, U_{2_\ell}$, we can form a corresponding SK-CONV layer by constructing tensors $\mathcal{S}_{1_1}, \ldots, \mathcal{S}_{1_\ell}, \mathcal{S}_{2_1}, \ldots, \mathcal{S}_{2_\ell}$ 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 following theorem (proof in Appendix B), based on Proposition 2.1, analyzes the expectation and the variance of using these sketches as an estimator for $\mathcal{I}_{\text{in}} * \mathcal{K}$ ($\equiv I_{\text{in}} \text{mat}_4(\mathcal{K})$). Since the random matrices are independent of each other, we drop the subscript and perform the analysis for a single instantiation of these sketches. .

Theorem 3.1. *Let $\mathcal{K} \in \mathbb{R}^{d_2 \times h \times w \times d_1}$ be a kernel tensor and $K = \text{mat}_4(\mathcal{K})$. Let $U_1 \in \mathbb{R}^{k \times d_1}$ and $U_2 \in \mathbb{R}^{khw \times d_2 hw}$ be two independent random scaled sign matrices. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be tensors such that $\text{mat}_4(\mathcal{S}_1) = K \times_2 U_1$ and $\text{mat}_4(\mathcal{S}_2) = K \times_1 U_2$. Then for any input matrix $I_{\text{in}} \in \mathbb{R}^{h_2 w_2 \times d_2 hw}$ (formed from an input tensor $\mathcal{I}_{\text{in}} \in \mathbb{R}^{h_1 \times w_1 \times d_2}$):*

1. *Unbiased Estimation: $\mathbb{E}[I_{\text{in}} \text{mat}_4(\mathcal{S}_1) U_1] = I_{\text{in}} K$, and $\mathbb{E}[I_{\text{in}} U_2^\top \text{mat}_4(\mathcal{S}_2)] = I_{\text{in}} K$.*

2. *Variance Bound:*

$$\mathbb{E}\left[\left\|I_{\text{in}}\,\text{mat}_4(\mathcal{S}_1)U_1 - I_{\text{in}}K\right\|_F^2\right] \leq \frac{2d_1\|I_{\text{in}}K\|_F^2}{k}, \text{ and}$$
$$\mathbb{E}\left[\left\|I_{\text{in}}U_2^\top\,\text{mat}_4(\mathcal{S}_2) - I_{\text{in}}K\right\|_F^2\right] \leq \frac{2\|I_{\text{in}}\|_F^2\|K\|_F^2}{khw}.$$

3.1.1 Training a SK-CONV Layer

In this section, we discuss a procedure for training a SK-CONV layer. Let $\text{Loss}()$ 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 the gradient information.

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 $G = \frac{\partial \text{Loss}}{\partial I_{\text{out}}} \in \mathbb{R}^{h_2w_2 \times d_1}$. For $i \in [\ell]$,[7]

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

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:

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

Typically, the FC layer is followed by application of some non-linear activation function. As in the case of convolutional 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* $(S_{1_1}, U_{1_1}), \ldots, (S_{1_\ell}, U_{1_\ell}), (S_{2_1}, U_{2_1}), \ldots, (S_{2_\ell}, U_{2_\ell})$ *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* $\mathbf{h}_{\text{in}} \in \mathbb{R}^{d_2}$ *performs the following operation:*

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

Note that $\mathbf{a}$ in the above definition could be equivalently represented as:

$$\mathbf{a} = \frac{1}{2\ell}\sum_{i=1}^{\ell}(S_{1_i} \times_1 U_{1_i}^\top)\mathbf{h}_{\text{in}} + \frac{1}{2\ell}\sum_{i=1}^{\ell}(S_{2_i} \times_2 U_{2_i}^\top)\mathbf{h}_{\text{in}} + \mathbf{b}.$$

The number of free parameters overall in all the S_{1_i} and S_{2_i} matrices put together is $\ell k(d_1 + d_2)$. Therefore, 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 $k\ell \leq d_1d_2/(d_1 + d_2)$.

[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.

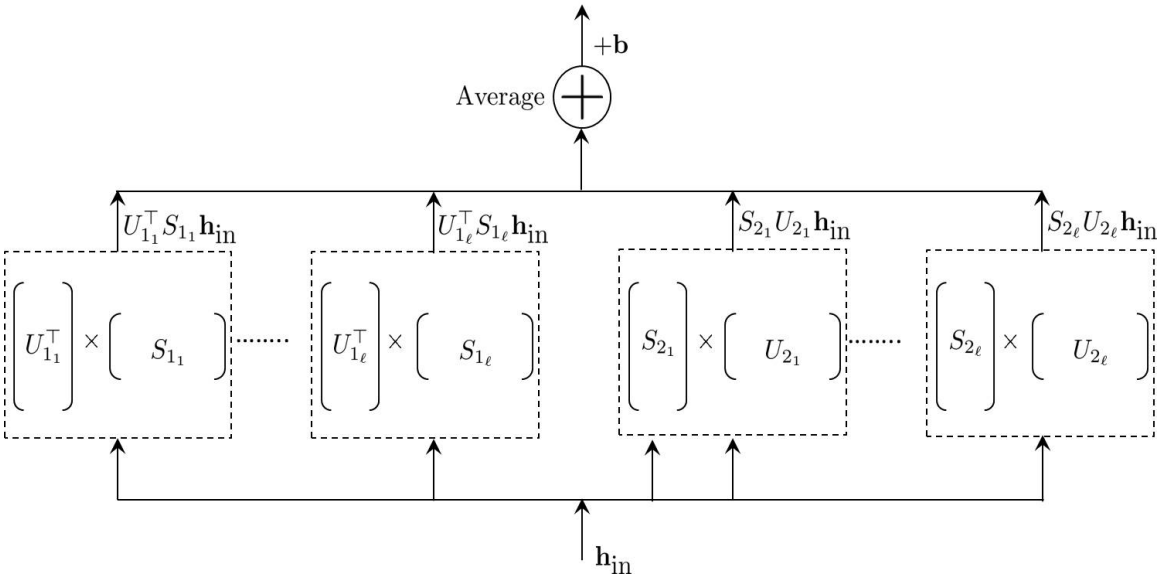

Figure 2: A SK-FC layer with parameters $\mathbf{b}, (S_{1_1}, U_{1_1}), \ldots, (S_{1_\ell}, U_{1_\ell}), (S_{2_1}, U_{2_1}), \ldots, (S_{2_\ell}, U_{2_\ell})$.

Another advantage is that the time needed for computing the pre-activation value ($\mathbf{a}$ 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 ℓ 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 $U_{1_1}, \ldots, U_{1_\ell}, U_{2_1}, \ldots, U_{2_\ell}$, 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 $W\mathbf{h}_{\text{in}} + \mathbf{b}$ for a vector $\mathbf{h}_{\text{in}} \in \mathbb{R}^{d_2}$. Since the random matrices are independent of each other, we drop the subscript and perform the analysis for a single instantiation of these sketches.

Theorem 3.2. *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 $S_2 = W U_2^\top (= W \times_2 U_2)$. Then for any $\mathbf{h}_{\text{in}} \in \mathbb{R}^{d_2}$ and $\mathbf{b} \in \mathbb{R}^{d_1}$:*

1. *Unbiased Estimation: $\mathbb{E}[U_1^\top S_1 \mathbf{h}_{\text{in}} + \mathbf{b}] = W\mathbf{h}_{\text{in}} + \mathbf{b}$, and $\mathbb{E}[S_2 U_2 \mathbf{h}_{\text{in}} + \mathbf{b}] = W\mathbf{h}_{\text{in}} + \mathbf{b}$.*
2. *Variance Bound:*

$$\mathbb{E}\left[\left\|U_1^\top S_1 \mathbf{h}_{\text{in}} + \mathbf{b} - (W\mathbf{h}_{\text{in}} + \mathbf{b})\right\|^2\right] \leq \frac{2d_1 \|W\mathbf{h}_{\text{in}}\|^2}{k},$$

$$\mathbb{E}\left[\left\|S_2 U_2 \mathbf{h}_{\text{in}} + \mathbf{b} - (W\mathbf{h}_{\text{in}} + \mathbf{b})\right\|^2\right] \leq \frac{2\|W\|_F^2 \|\mathbf{h}_{\text{in}}\|^2}{k}.$$

3.2.1 Training a SK-FC Layer

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

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

Similarly, the gradient with respect to $\mathbf{h}_{\text{in}}$ can be calculated as:

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

Now let $\mathbf{a} = U_1^\top S_1 \mathbf{h}_{\text{in}} + \mathbf{b} = (S_1^\top U_1)^\top \mathbf{h}_{\text{in}} + \mathbf{b}$. Again let $\mathbf{g} = \frac{\partial \text{Loss}}{\partial \mathbf{a}}$. Applying chain-rule gives

$$\frac{\partial \text{Loss}}{\partial S_1} = \sum_{i=1}^{d_1} \frac{\partial \text{Loss}}{\partial a_i} \frac{\partial a_i}{\partial S_1},$$

where a_i denotes the ith entry of $\mathbf{a}$. We can compute $\frac{\partial a_i}{\partial S_1}$ as:

$$\frac{\partial a_i}{\partial S_1} = \frac{\partial\, \mathbf{u}_{1_i}^\top S_1 \mathbf{h}_{\text{in}}}{\partial S_1} = \mathbf{u}_{1_i} \mathbf{h}_{\text{in}}^\top,$$

where $\mathbf{u}_{1_i}$ is the ith column in U_1. Therefore, we get

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

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

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

Putting together (5), (6), (7), and (8) gives the necessary gradients for the SK-FC layer (where $\mathbf{a}$ is defined using (4)). Let $\mathbf{g} = \frac{\partial \text{Loss}}{\partial \mathbf{a}}$. 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-parametrized, and there is significant redundancy in deep learning networks [11]. There have been several previous attempts to reduce the complexity of deep neural networks under a variety of contexts.

Approximating only the Fully Connected Layers. A set of techniques have focused on approximating only the fully connected layers in some reduced form. Yang *et al.* [48] use the *Fastfood* transformation technique of [30] to approximate the fully connected layers. The HashedNets architecture, proposed by Chen *et al.* [2], uses a hash function to enforce parameter sharing between random groups of parameters in a fully connected layer to reduce the number

of effective parameters. Cheng *et al.* [5] achieve parameter reduction by imposing a circulant matrix structure on fully connected layers. Sindhwani *et al.* [37] generalize this construction by proposing a broad family of structured parameter matrix structure and showing its effectiveness on the fully connected layers. Choromanska *et al.* [7] provide some theoretical justifications for using structured hashed projections in these layers. While some of these techniques are highly effective on the fully connected layers, they fall short of achieving a significant reduction in the number parameters for modern CNNs which are dominated by convolutional layers [40, 21].[8] Therefore, any *effective* technique for parameter reduction on CNNs should *also* act on convolutional layers.

Approximating both the Convolutional and Fully Connected Layers. Most relevant to our paper is a line of work on approximating both the fully connected and convolutional layers. Denil *et al.* [11], 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.* [41] 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 [25]) 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. Very recently, Garipov *et al.* [14], building upon a work by [36], used a tensor factorization technique, called tensor train decomposition, to uniformly approximate both the fully connected and convolutional layers. However, constructing an exact tensor factorization (even computing the tensor rank) is in general a challenging NP-hard problem, whereas our approach relies only on simple linear transformations. Chen *et al.* [3] combine the hashing idea from [2] 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 [27], 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.

Other Related Work. 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) [15, 20, 19, 38, 47, 17, 28, 45, 23, 24, 50, 32]. 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 solve the network approximation problem 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 (which is beyond the scope of this paper).

Hinton *et al.* [22] and Ba *et al.* [1] proposed approaches to learn a "distilled" model, training a more compact neural network to reproduce the output of a larger network. The general idea is to train a large network on the original training labels, then learn a much smaller distilled model on a weighted combination of the original labels and the softmax output of the larger model. Note that with our network approximation approach, we do not need to train the original large network. Also unlike distillation-based approaches where a separate distilled model has to be formed with each dataset, our approach produces a single reduced network that can be then trained on any dataset.

Other techniques proposed for parameter reduction include inducing zeros in the parameter matrices via sparsity regularizers [8] and storing weights in low fixed-precision formats [18, 9]. These ideas can be readily incorporated with our approach, potentially yielding further reductions in the model memory size. Daniely *et al.* [10] generate sketches of the input and show that it can lead to compact neural networks. Our approach, based on sketching the parameters of the deep network, is complementary to this idea, and the two approaches can be used in conjunction.

Several works apply related approaches to speed up the evaluation time with CNNs [26, 12, 31, 13]. The focus of

[8] Some recent studies [34, 33] have suggested that removing fully connected layers and replacing them with convolutions and pooling could be beneficial for certain computer vision applications.

this line of work is not on parameter reduction but rather decreasing the evaluation time during testing. In each of these results, any resulting storage reduction comes as a side effect. Other techniques for speeding up convolutional neural networks include use of Winograd or FFT-based convolutions [29, 35, 44]. Again, unlike here, parameter reduction is not a focus of these results.

5 Experimental Evaluation

In this section, we experimentally demonstrate the effectiveness of our proposed network approximation approach. Our goal through the experiments is *not to test the limits of reduction* possible in deep neural networks, but rather to demonstrate that through our tensor sketching approach it is possible to design a substantially smaller network that achieves almost the same performance as the original network on a wide-range of datasets. We used the Torch machine learning framework and all the experiments were performed on a cluster of GPUs using a single GPU for each run. Additional experimental results are presented in Appendix C.

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. Compression rate < 1 indicates compression with smaller values indicating higher compression. The top-1 error (denoted by ERRTOP-1) for a trained model on a test set captures the percentage of images in the test set misclassified by the model. To get a more stable picture of the model performance, ERRTOP-1 is computed by averaging the test error after each of the last 10 training epochs.

Datasets. We use 5 popular image datasets: CIFAR10 (objects recognition dataset with 3×32×32 images), SVHN (digits recognition dataset with 3×32×32 images), STL10 (objects recognition dataset with 3×96×96 images), ImageNet10 objects recognition dataset with 3×256×256 images, a subset of ImageNet1000 dataset that we created[9], and Places2 (scene understanding dataset with 365 classes and about 8 million images in the training set). Note that, Places2 is a big and challenging dataset that was used in the recent ILSVRC 2016 "Scene Classification" challenge.

Network Architectures. We ran our experiments on four different network architectures. 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 in this area is that reducing the number of parameters in convolutional layers seems to be a much more challenging problem than that for fully connected layers. The first network architecture that we experiment with is the popular Network-in-Network (NinN) [33] with minor adjustments for the corresponding image sizes (we used strides of the first layer to make these adjustments). Network-in-Network is a moderately sized network which attains good performance on medium sized datasets, e.g. CIFAR10 [49]. For this network, we did not employ batch normalization [25] or dropout [39] to have a uniform set of experiments across different techniques. The second network that we consider is the same as NinN with only one change that the last convolution layer is replaced by a fully connected layer (we denote it as NinN+FC). Following [2], the third network that we experiment is a simple shallow network, which we refer to as TestNet, with only 2 convolution layers and 2 fully connected layers which allows us to easily test the efficacy of our approximation technique for each layer individually. We describe the construction of TestNet in more detail in Appendix C. Table 1 shows the original (uncompressed) top-1 error (ERRTOP-1) for NinN and NinN+FC. The number of parameters are about 966K for NinN and 1563K for NinN+FC for all datasets. The statistics about TestNet are presented in Figure 2 (Appendix C). The final network that we consider is GoogLeNet [40] with batch normalization, which we use for the Places2 dataset. This network has a top-1 error of 32.3% on the Places2 dataset.

Network	CIFAR10	STL10	SVHN	ImageNet10
NinN	17.7	43.2	6.0	27.1
NinN+FC	16.9	41.2	5.4	26.0

Table 1: Top-1 error of the NinN architecture and its variant on different datasets.

Baseline Techniques. As discussed in Section 4 there are by now quite a few techniques for network approximation.

[9] We used following classes: bottle, cat, grasshopper, grocery, truck, chair, running shoes, boat, stove, and clock. The training set consists of 13000 images and the test set consists of 500 images

We compare our proposed 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 [3], low-rank decomposition technique of [11] (LOWRANK$_1$), and tensor decomposition technique of [41] (LOWRANK$_2$). While using the FreshNets technique, we also use the HashedNets technique of feature hashing [2] for compressing the fully connected layers as suggested by [3]. We used open-source implementations of all these techniques: HashedNets, FreshNets, and LOWRANK$_1$ are from [4] and LOWRANK$_2$ from [42]. We set the required parameters to ensure that all the compared approaches achieve about the same compression rate.

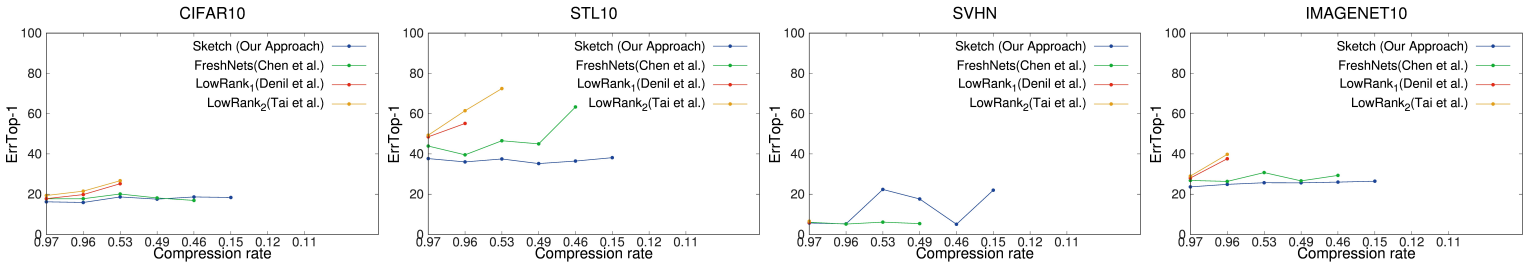

Figure 3: Top-1 error for the NinN architecture as we decrease the compression rate by compressing one convolutional layer at a time each by a factor of 10. The x-axis is not to scale.

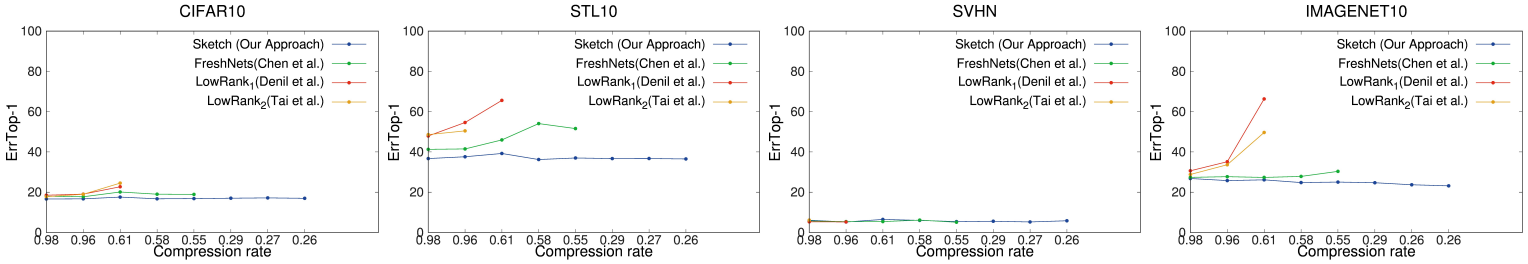

Figure 4: Top-1 error for the NinN architecture as we decrease the compression rate by compressing one convolutional layer at a time each by a factor of 4. The x-axis is not to scale.

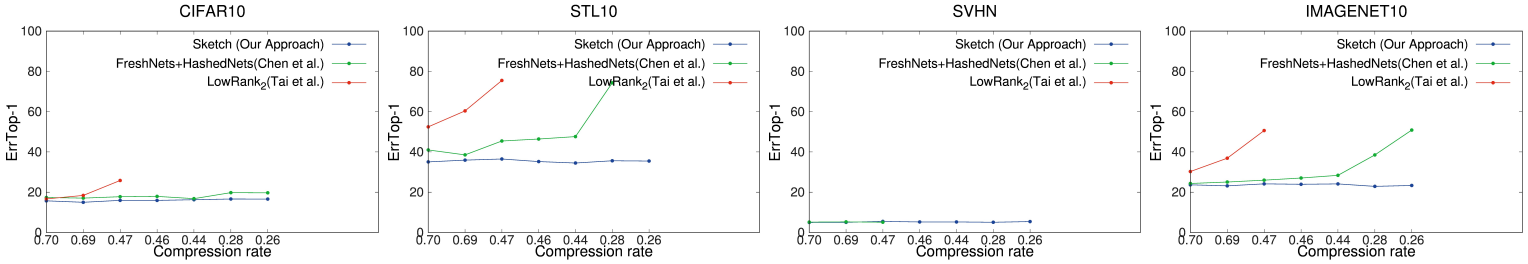

Figure 5: Top-1 error for the NinN+FC architecture. The size of FC layer is about half of the total size of convolutional layers CONV$_2$ to CONV$_8$. We compress the fully connected layer by a factor of 4. We then use a similar experimental setup as in Figure 4 of reducing the number of parameters in the convolutional layers (CONV$_2$ to CONV$_8$) each by a factor of 4. The x-axis is not to scale.

Compression on the Convolutional Layers. We performed a set of experiments to evaluate the performance of our scheme only on the convolutional layers. We used the NinN architecture for this purpose. NinN is, essentially, a sequence of nine convolution layers (labeled as CONV$_1$ to CONV$_9$). We compress these layers one by one, starting from CONV$_2$ and finishing at CONV$_9$ by reducing the number of parameters in each layer by a factor of r which is set as 10. When all these 8 convolution layers are compressed the achieved network compression rate is approximately[10] equal to $1/r$.

[10]We do not compress the first layer that takes input.

Figures 3 and 4 shows the results of our experiments. 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 such as in Figure 3 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. Effects of this become more pronounced as we decrease the compression rate. In most cases, we 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 probably also adds a regularization effect during the training.

Compression 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 768×768 followed by a classifier layer of size 768×10. In Figure 5, 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.

Places2 Dataset. 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. Again here, as in all the above cases, model storage sizes can be further reduced by taking this reduced model and using certain post-processing operations as detailed in Section 4, which is outside the scope of this evaluation.

Parameter Sensitivity. In Appendix C, we present experiments that highlight the role of parameters k and ℓ in our proposed approach. In general, we observe that the accuracy of the compressed models improve as we increase k or ℓ (this happens because we are increasing the effective size of the constructed sketches). Also, due to the averaging effect, increasing ℓ decreases the variance of top-1 error with respect to the randomization that arises from the use of random matrices.

Computational Efficiency. While our primary focus is on network approximation (i.e., designing networks with a smaller set of parameters), an added bonus is that the networks generated through our tensor sketching approach are also computationally more efficient. For example, at the compression rate of 0.15 the wall-clock testing time, of our reduced NinN is on average between 1.6-2x smaller compared to the original network across all the tested datasets. Since the sketch tensors in our construction are dense, further efficiency gains are possible by better exploiting the dense matrix capabilities of modern GPUs.

References

[1] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

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

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

[4] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. Freshnets. http://www.cse.wustl.edu/~ychen/FreshNets, 2016.

[5] 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.

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

[7] Anna Choromanska, Krzysztof Choromanski, Mariusz Bojarski, Tony Jebara, Sanjiv Kumar, and Yann LeCun. Binary embeddings with structured hashed projections. In *ICML*, 2016.

[8] MD Collins and P Kohli. Memory-bounded deep convolutional neural networks. In *ICASSP*, 2013.

[9] Matthieu Courbariaux, Jean-Pierre David, and Yoshua Bengio. Low precision storage for deep learning. In *ICLR*, 2015.

[10] Amit Daniely, Nevena Lazic, Yoram Singer, and Kunal Talwar. Sketching and neural networks. *ArXiv*, 2016.

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

[12] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.

[13] Michael Figurnov, Dmitry Vetrov, and Pushmeet Kohli. Perforatedcnns: Acceleration through elimination of redundant convolutions. In *NIPS*, 2016.

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

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

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

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

[18] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *ICML*, 2015.

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

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

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[22] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *ArXiv*, 2015.

[23] 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.

[24] 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.

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

[26] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *ArXiv*, 2014.

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

[28] 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.

[29] Andrew Lavin. Fast algorithms for convolutional neural networks. *ArXiv*, 2015.

[30] Quoc Le, Tamás Sarlós, and Alex Smola. Fastfood-approximating kernel expansions in loglinear time. In *ICML*, 2013.

[31] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *ArXiv*, 2014.

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

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

[34] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[35] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *ArXiv*, 2013.

[36] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *NIPS*, 2015.

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

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

[39] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1), 2014.

[40] 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.

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

[42] Cheng Tai, Tong Xiao, Xiaogang Wang, et al. Lowrankcnn. https://github.com/chengtaipu/lowrankcnnl, 2016.

[43] 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.

[44] Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A gpu performance evaluation. *ArXiv*, 2014.

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

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

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

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

[49] S. Zagoruyko. Cifar-10 in torch. http://torch.ch/blog/2015/07/30/cifar.html, 2015.

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

A Proof of Proposition 2.1

Proposition A.1 (Proposition 2.1 Restated)**.** *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 $S_2 = W U_2^\top (= W \times_2 U_2)$. 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. $\mathbb{E}\left[\left\|U_1^\top S_1 M - WM\right\|_F^2\right] \leq \frac{2d_1 \|WM\|_F^2}{k}$, *and*
   $\mathbb{E}\left[\|S_2 U_2 M - WM\|_F^2\right] \leq \frac{2\|W\|_F^2 \|M\|_F^2}{k}$.

Proof. Part 1 follows by simply using linearity of expectation.

We focus on Part 2 which investigates the variance bounds for $U_1^\top S_1 M$ and $S_2 U_2 M$. For this, we use some standard ideas from the matrix sketching literature [46].

Consider first $\mathbb{E}\left[\|S_2 U_2 M - WM\|_F^2\right]$. We start by noting,

$$S_2 U_2 M - WM = W U_2^\top U_2 M - WM = \frac{1}{k} W Z^\top Z M - WM,$$

where $U_2 = Z/\sqrt{k}$. Let $\mathbf{w}_a, \mathbf{z}_b, \mathbf{m}_c$ denote the a, b, c-th columns of $W^\top$, Z, and M respectively. We have

$$\left\|W Z^\top Z M\right\|_F^2 = \sum_{a,c} \left(\mathbf{w}_a^\top \left(\sum_{b=1}^k \mathbf{z}_b \mathbf{z}_b^\top \right) \mathbf{m}_c \right)^2.$$

Therefore, we get

$$\begin{aligned}
\left\| \frac{1}{k} W Z^\top Z M - WM \right\|_F^2 &= \sum_{a,c} \left(\frac{1}{k} \mathbf{w}_a^\top \left(\sum_{b=1}^k \mathbf{z}_b \mathbf{z}_b^\top \right) \mathbf{m}_c - \mathbf{w}_a^\top \mathbf{m}_c \right)^2 \\
&= \sum_{a,c} \left(\frac{1}{k} \mathbf{w}_a^\top \left(\sum_{b=1}^k \mathbf{z}_b \mathbf{z}_b^\top \right) \mathbf{m}_c - \frac{1}{k} \sum_{b=1}^k \mathbf{w}_a^\top \mathbf{m}_c \right)^2 \\
&= \sum_{a,c} \left(\sum_{b=1}^k \frac{\mathbf{w}_a^\top \mathbf{z}_b \mathbf{z}_b^\top \mathbf{m}_c - \mathbf{w}_a^\top \mathbf{m}_c}{k} \right)^2. \qquad (9)
\end{aligned}$$

Let $y_{abc} = \frac{\mathbf{w}_a^\top \mathbf{z}_b \mathbf{z}_b^\top \mathbf{m}_c - \mathbf{w}_a^\top \mathbf{m}_c}{k}$ which can be re-expressed as:

$$y_{abc} = \frac{1}{k} \sum_{\substack{r,s \\ r \neq s}} W_{ar} Z_{rb} Z_{sb} M_{sc},$$

where W_{ar} is the (a,r)th entry in W, Z_{rb} and Z_{sb} are the (r,b)th and (s,b)th entries in Z respectively, and M_{sc} is the (s,c)th entry in M. Using this notation, we can re-express (9) as:

$$\begin{aligned}\left\|\frac{1}{k}WZ^\top ZM - WM\right\|_F^2 &= \sum_{a,c}\left(\sum_{b=1}^{k} y_{abc}\right)^2 \\ &= \sum_{a,c}\sum_{b,b'} y_{abc}\, y_{ab'c} \\ &= \frac{1}{k^2}\sum_{a,c}\sum_{b,b'}\sum_{r\neq s} W_{ar}Z_{rb}Z_{sb}M_{sc}\sum_{r'\neq s'} W_{ar'}Z_{r'b'}Z_{s'b'}M_{s'c}.\end{aligned}$$

Taking expectation,

$$\mathbb{E}\left[\|S_2U_2M - WM\|_F^2\right] = \frac{1}{k^2}\sum_{\substack{a,c,b,b',r,s,r',s' \\ r\neq s \\ r'\neq s'}} W_{ar}W_{ar'}M_{sc}M_{s'c}\,\mathbb{E}[Z_{rb}Z_{sb}Z_{r'b'}Z_{s'b'}].$$

Now, $\mathbb{E}[Z_{rb}Z_{sb}Z_{r'b'}Z_{s'b'}]$ is non-zero only if either: 1) $r=r'$, $s=s'$, and $b=b'$ or 2) $r=s'$, $s=r'$, and $b=b'$. Therefore, we can simplify $\mathbb{E}\left[\|S_2U_2M - WM\|_F^2\right]$ as,

$$\begin{aligned}\mathbb{E}\left[\|S_2U_2M - WM\|_F^2\right] &\leq \frac{2}{k^2}\sum_{a,c}\sum_{b}\sum_{r} W_{ar}^2\sum_{s} M_{sc}^2 \\ &= \frac{2}{k}\sum_{a,r} W_{ar}^2\sum_{c,s} M_{sc}^2 \\ &= \frac{2}{k}\|W\|_F^2\|M\|_F^2. \end{aligned} \tag{10}$$

This proves the claimed bound for $\mathbb{E}\left[\|S_2U_2M - WM\|_F^2\right]$.

Now we bound $\mathbb{E}\left[\|U_1^\top S_1M - WM\|_F^2\right]$. We start by re-expressing the result in (10). Start by noting that $S_2 = WU_2^\top$. Therefore, from (10),

$$\mathbb{E}\left[\left\|WU_2^\top U_2M - WM\right\|_F^2\right] \leq \frac{2}{k}\|W\|_F^2\|M\|_F^2.$$

Now by setting, $W = \mathbb{I}_{d_1}$ in this result and by noting $\|\mathbb{I}_{d_1}\|_F^2 = d_1$, we get that for any matrix $A \in \mathbb{R}^{d_2\times d_3}$,

$$\mathbb{E}\left[\left\|U_1^\top U_1A - A\right\|_F^2\right] \leq \frac{2d_1}{k}\|A\|_F^2, \tag{11}$$

where the expectation is now over $U_1 \in \mathbb{R}^{k\times d_1}$.

Since $U_1^\top S_1M = U_1^\top U_1WM$. Therefore, $U_1^\top S_1M - WM = U_1^\top U_1WM - WM$. The idea is to invoke (11) with $A = WM$. We get,

$$\mathbb{E}\left[\left\|U_1^\top U_1WM - WM\right\|_F^2\right] \leq \frac{2d_1}{k}\|WM\|_F^2.$$

This completes the proof of this theorem. □

B Proof of Theorem 3.1

Theorem B.1 (Theorem 3.1 Restated). *Let $\mathcal{K} \in \mathbb{R}^{d_2\times h\times w\times d_1}$ be a kernel tensor and $K = \mathrm{mat}_4(\mathcal{K})$. Let $U_1 \in \mathbb{R}^{k\times d_1}$ and $U_2 \in \mathbb{R}^{khw\times d_2hw}$ be two independent random scaled sign matrices. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be tensors such that $\mathrm{mat}_4(\mathcal{S}_1) = K \times_2 U_1$ and $\mathrm{mat}_4(\mathcal{S}_2) = K \times_1 U_2$. Then for any input matrix $I_{\text{in}} \in \mathbb{R}^{h_2w_2\times d_2hw}$ (formed from an input tensor $\mathcal{I}_{\text{in}} \in \mathbb{R}^{h_1\times w_1\times d_2}$):*

1. *Unbiased Estimation:* $\mathbb{E}[I_{\text{in}} \operatorname{mat}_4(\mathcal{S}_1)U_1] = I_{\text{in}}K$, *and* $\mathbb{E}[I_{\text{in}}U_2^\top \operatorname{mat}_4(\mathcal{S}_2)] = I_{\text{in}}K$.
2. *Variance Bound:*

$$\mathbb{E}\left[\|I_{\text{in}} \operatorname{mat}_4(\mathcal{S}_1)U_1 - I_{\text{in}}K\|_F^2\right] \leq \frac{2d_1\|I_{\text{in}}K\|_F^2}{k}, \text{ and}$$
$$\mathbb{E}\left[\left\|I_{\text{in}}U_2^\top \operatorname{mat}_4(\mathcal{S}_2) - I_{\text{in}}K\right\|_F^2\right] \leq \frac{2\|I_{\text{in}}\|_F^2\|K\|_F^2}{khw}.$$

Proof. First note that, by definition,

$$I_{\text{in}} \operatorname{mat}_4(\mathcal{S}_1)U_1 = I_{\text{in}}KU_1^\top U_1.$$

Using an analysis similar to Proposition 2.1 gives,

$$\mathbb{E}[I_{\text{in}} \operatorname{mat}_4(\mathcal{S}_1)U_1] = I_{\text{in}}K, \text{ and}$$
$$\mathbb{E}\left[\|I_{\text{in}} \operatorname{mat}_4(\mathcal{S}_1)U_1 - I_{\text{in}}K\|_F^2\right] \leq \tfrac{2d_1\|I_{\text{in}}K\|_F^2}{k}.$$

Similarly, by definition of $\operatorname{mat}_4(\mathcal{S}_2)$, we have:

$$I_{\text{in}}U_2^\top \operatorname{mat}_4(\mathcal{S}_2) = I_{\text{in}}U_2^\top U_2 K.$$

Again relying on an analysis similar to Proposition 2.1 gives,

$$\mathbb{E}[I_{\text{in}}U_2^\top \operatorname{mat}_4(\mathcal{S}_2)] = I_{\text{in}}K, \text{ and}$$
$$\mathbb{E}\left[\left\|I_{\text{in}}U_2^\top \operatorname{mat}_4(\mathcal{S}_2) - I_{\text{in}}K\right\|_F^2\right] \leq \tfrac{2\|I_{\text{in}}\|_F^2\|K\|_F^2}{khw}.$$

This completes the proof of this theorem. □

C Additional Experimental Results

In this section, we present some additional experimental results that investigate the role of various parameters in our proposed approach. We start by describing the TestNet architecture that we use for the following experiments.

Images	CONV$_1$	MAXPOOL$_1$	CONV$_2$	MAXPOOL$_2$	FC$_1$	FC$_2$
3×32×32	$d_2 = 3, d_1 = 30, f = 5\times5$	$f = 2\times2$	$d_2 = 30, d_1 = 30, f = 5\times5$	$f = 4\times4$	$d_2 = 480, d_1 = 250$	$d_2 = 250, d_1 = 10$
3×96×96	$d_2 = 3, d_1 = 20, f = 7\times7$	$f = 5\times5$	$d_2 = 20, d_1 = 40, f = 5\times5$	$f = 5\times5$	$d_2 = 1960, d_1 = 500$	$d_2 = 500, d_1 = 10$
3×256×256	$d_2 = 3, d_1 = 20, f = 7\times7$	$f = 11\times11$	$d_2 = 20, d_1 = 30, f = 9\times9$	$f = 7\times7$	$d_2 = 3000, d_1 = 500$	$d_2 = 500, d_1 = 10$

Figure 6: TestNet Architecture.

TestNet Architecture. TestNet is a simple shallow network with only 2 convolution layers and 2 fully connected layers. This allows us to easily test the efficacy of our approximation technique for each layer individually. Figure 6 shows parameters of TestNet for different image sizes.

A ReLU layer is used after each fully connected and convolutional layer. For example, consider images of size 3×32×32. The first convolutional layer takes 3 input feature maps ($d_2 = 3$) and produces 30 output feature maps ($d_1 = 30$) using filters of size 5 by 5 ($f = 5\times5$), and we represent it as a 4-dimensional tensor in $\mathbb{R}^{3\times5\times5\times30}$. Note that in TestNet the fully connected layers contain much more network parameters than the convolutional layers.

Table 2 shows the original top-1 error (ERRTOP-1) and the number of parameters for all datasets. We used different number of parameters in FC$_1$ for different image sizes to ensure that the corresponding trained networks converge.

CIFAR10	STL10	SVHN	ImageNet10
25.7 (147K)	40.5 (1008K)	8.2 (147K)	27.0 (1561K)

Table 2: Top-1 error of the original TestNet on different datasets. In bracket, we show the number of parameters in each of these networks.

Parameter Sensitivity. For understanding the role of parameters k and ℓ in our tensor sketching approach, we train a number of networks derived from the TestNet architecture for several combinations of these parameters. For the convolutional layer, we construct different networks each of which is obtained by replacing the CONV_2 layer of TestNet with a SK-CONV layer for different values of k and ℓ. We vary $\ell \in \{1, 2, 3\}$ and $k \in \{2, 5, 10\}$, independently, giving rise to 9 different networks. Similarly, we also construct new networks by replacing the FC_1 layer of TestNet with a SK-FC layer with $\ell \in \{1, 2, 3\}$ and $k \in \{5, 10, 25\}$ for smaller images (like CIFAR10) and $k \in \{15, 25, 50\}$ for larger images (like STL10). Figure 7 shows the results for the CIFAR10 and STL10 datasets (results on other two datasets are similar and omitted here). For each compressed model, we show its top-1 error (plots in the top row). The plots in the bottom row present the corresponding compression rate for each network. Note that if the parameters k and ℓ are set too high then the compression rate can be > 1. In this case, we have an expansion over the original network. If a point is missing from a line then the corresponding network failed in the training. As an example, consider the network obtained by replacing FC_1 layer with a SK-FC layer using $k = 5$ and $\ell = 2$. From the plot in Figure 7, the model obtained by training this network on CIFAR10 has ERRTOP-1 $\approx 30\%$. We also see that this network has a compression rate of ≈ 0.5, i.e., the size of the original TestNet has been reduced by a factor of 2. Recall that by design TestNet has much more parameters in the fully connected layers than the convolutional layers, hence compressing the FC_1 layer leads to smaller compression rates than compressing the CONV_2 layer (as observed in Figure 7).

First, from Figure 7, we observe that the accuracy of the compressed models improve as we increase k or ℓ. This is expected because, as we discuss in Section 3, by increasing k or ℓ we are increasing the effective size of the constructed sketches. For example, on the STL10 dataset with $\ell = 1$ and the fully connected layer compression, as we increase k from 15 to 50, the top-1 error goes down from around 62% to 45% (which is comparable to the 40.5% top-1 error of the original (uncompressed) model from Table 1). However, with increasing k or ℓ the compression rate goes up (implying lower overall compression).

Second, due to the averaging effect, increasing ℓ increases the stability in the sketching process. For example, consider CIFAR10 where FC_1 layer is replaced with a SK-FC layer using $k = 10$ and $\ell \in \{1, 2, 3\}$. We trained each resulting network architecture 10 different times each time initializing the SK-FC layer with a different set of random matrices $U_{1_1}, \ldots, U_{1_\ell}, U_{2_1}, \ldots, U_{2_\ell}$ and measured the variance in the top-1 error across different runs. Not surprisingly, increasing ℓ decreases the variance of top-1 error with respect to the randomization that arises from the use of random matrices. For example, with $\ell = 1$ we get an average (over these runs) top-1 error of 30.1 with variance of 0.44, for $\ell = 2$ we get an average top-1 error of 29.1 with variance of 0.29, and for $\ell = 3$ we get an average top-1 error of 28.6 with variance of 0.23.

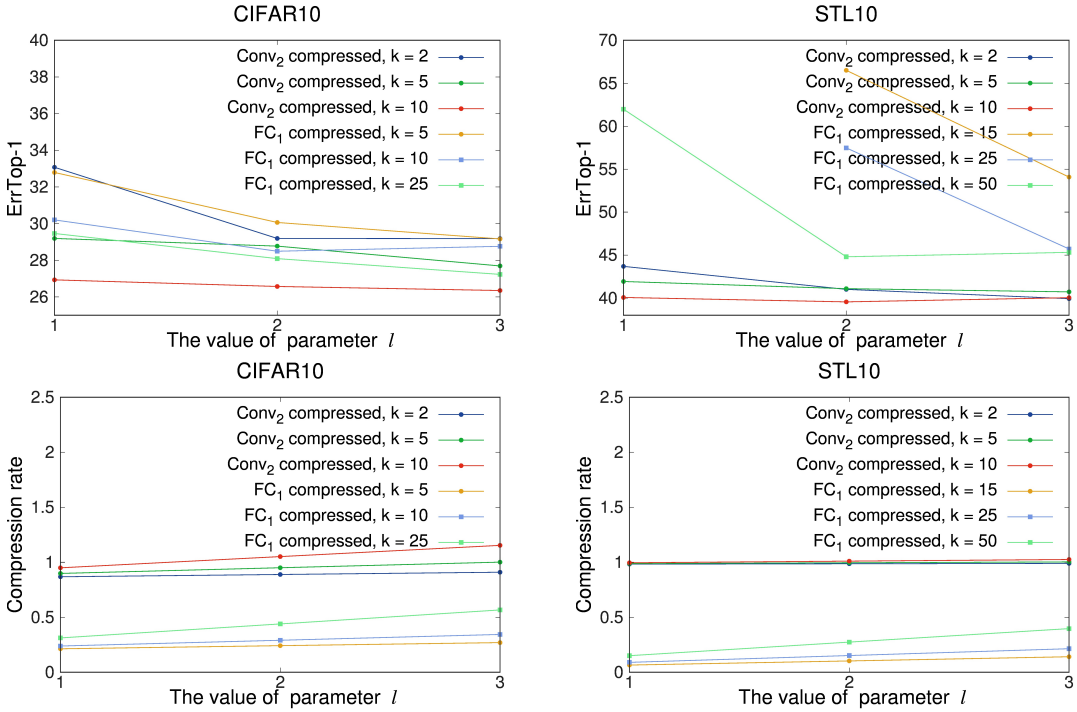

Figure 7: The plots on the left show the top-1 error and the compression rate for the CIFAR10 dataset obtained by varying k and ℓ in our tensor sketching approach on TestNet. The plots on the right show the same for the STL10 dataset.