What Happens on the Edge, Stays on the Edge: Toward Compressive Deep Learning

Yang Li and Thomas Strohmer
Department of Mathematics
University of California, Davis
Davis, CA 95616, USA
{ly,strohmer}@math.ucdavis.edu

Abstract

Machine learning at the edge offers great benefits such as increased privacy and security, low latency, and more autonomy. However, a major challenge is that many devices, in particular edge devices, have very limited memory, weak processors, and scarce energy supply. We propose a hybrid hardware-software framework that has the potential to significantly reduce the computational complexity and memory requirements of on-device machine learning. In the first step, inspired by compressive sensing, data is collected in compressed form simultaneously with the sensing process. Thus this compression happens already at the hardware level during data acquisition. But unlike in compressive sensing, this compression is achieved via a projection operator that is specifically tailored to the desired machine learning task. The second step consists of a specially designed and trained deep network. As concrete example we consider the task of image classification, although the proposed framework is more widely applicable. An additional benefit of our approach is that it can be easily combined with existing on-device techniques. Numerical simulations illustrate the viability of our method.

1 Introduction

1.1 Machine Learning on Constrained Devices

While the leitmotif *"local sensing and remote inference"*[1] has been at the foundation of a range of successful AI applications, there exist many scenarios in which it is highly preferable to run machine learning algorithms either directly on the device or at least locally on the edge instead of remotely. As tracking and selling our digital data has become a booming business model [Zub19], there is an increasing and urgent need to preserve privacy in digital devices. Privacy and security are much harder to compromise if data is processed locally instead of being sent to the cloud. Also, some AI applications may be deployed "off the grid", in regions far away from any mobile or internet coverage. In addition, the cost of communication may become prohibitive and scaling to millions of devices may be impractical [DP17]. Moreover, some applications cannot afford latency issues that might result from remote processing.

Hence, there is a strong incentive for local inference "at the edge", *if* it can be accomplished with little sacrifice in accuracy and within the resource constraints of the edge device. Yet, the difficulty in "pushing AI to the edge" lies exactly in these resource constraints of many edge devices regarding computing power, energy supply, and memory.

[1]We borrowed this slogan from [DP17]

In this paper we focus on such *constrained devices* and propose a hybrid hardware-software framework that has the potential to significantly reduce the computational complexity and memory requirements of on-device machine learning.

1.2 Outline of our approach

When we deploy an AI-equipped device in practice, we know a priori what kind of task this device is supposed to carry out. The key idea of our approach can thus be summarized as follows: We can take this knowledge into account already in the data acquisition step itself and try to measure only the task-relevant information, thereby we significantly lower the size of the data that enter the device and thus reduce the computational burden and memory requirements of that device.

To that end we propose *compressive deep learning*, a hybrid hardware-software approach that can be summarized by two steps, see also Figure 1. First, we construct a projection operator that is specifically tailored to the desired machine learning task and which is determined by the entire training set (or a subset of the training set). This projection operator compresses the data simultaneously with the sensing process, like in standard compressive sensing [FR13]. But unlike compressive sensing, our projection operator is tailored specifically to the intended machine learning task, which therefore allows for a much more "aggressive" compression. The construction of the projection operator is of course critical and various techniques are described in Section 2. This projection will be implemented in hardware, thus the data enter the software layer of the device already in compressed form. The data acquisition/compression step is followed by a deep network that processes the compressed data and carries out the intended task.

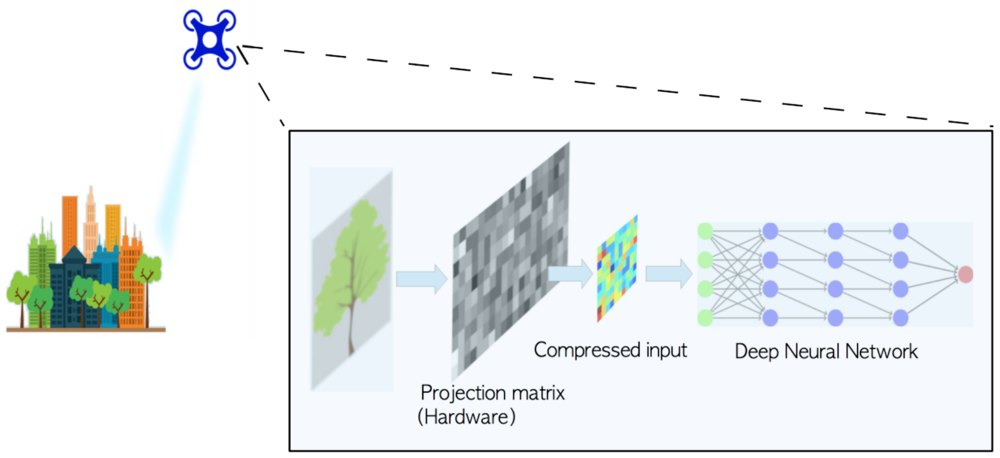

Figure 1: Schematic depiction of compressive deep learning: Data acquisition and compression are carried out simultaneously. Compression is achieved at the hardware level via a projection operator that is specifically tailored to the desired machine learning task. The so compressed data are then fed into a specially trained deep network that performs the intended task.

While our approach is applicable to a wide range of machine learning tasks, in this paper we will mainly focus on image classification. We emphasize that the projection/compression we carry out at the data acquisiton step is completely different from standard (jpeg-type) image compression. Firstly, standard impage compression happens at the software layer and is completely independend from the data acquisition step, while our proposed compression-while-sensing scheme is an intrinsic part of the data acquisition step. Secondly, standard image compression is designed to work for a vast range of images and independent of the task we later perform, while our compression scheme is inherently tied to the image classification task we intend to carry out.

1.3 Compressive sensing and beyond

As mentioned above, our approach is in part inspired by ideas from compressive sensing. The compressive sensing paradigm uses simultaneous sensing and compression. At the core of compressed sensing lies the discovery that it is possible to reconstruct a sparse signal exactly from an underdetermined linear system of equations and that this can be done in a computationally efficient manner via ℓ_1-minimization, cf. [FR13]. Compressive sensing consists of two parts: (i) The sensing step, which simultaneously compresses the signal. This step is usually implemented (mostly) in hardware. (ii) The signal reconstruction step via carefully designed algorithms (thus this is done by software).

Compressed sensing aligns with a few of our objectives, however, they also differ in the following crucial ways:

1. Adaptivity: Standard compressive sensing is not adaptive. It considers all possible sparse signals under certain representations. For different data sets, the fundamental assumptions are the same. This assumption is likely too weak for a specific image data set, where only images with certain characteristics are included. For image classification, this is usually the case. Statistical information in the data set may be exploited to achieve better results.
2. Exact reconstruction: Compressive sensing aims for exact signal reconstruction. That means enough measurements must be taken to ensure all information needed for exact recovery. Clearly, this is too stringent a constraint for image classification where only label recovery is required instead of full signal reconstruction.
3. Storage and processing cost: It is cumbersome to implement random matrices often proposed in compressive sensing in hardware; only certain structured random matrices can be implemented efficiently.

1.4 Prior work

Various approaches have been proposed for improving the computational cost of neural networks and for on-device deep learning. We point out that in the approaches described below, the assumptions made about the properties of these devices and their capabilities may differ for different approaches.

1. Redesigning architecture: Some works proposed new architectures to alleviate the computational burden. This includes MobileNet, LCCL, SqueezeNet, etc. ([IHM+16, HZC+17, DHYY17]).
2. Quantization: It is observed that representing weights in neural networks with lower precision does not lead to serious performance drop in testing accuracy. In this way, the computational cost of basic arithmetic operations is reduced while the model's capability of predicting correct labels is largely preserved.
3. Matrix decomposition: In terms of runtime, convolutional layers are usually the most costly component of a convolutional neural network. However, many of the weights in these layers are redundant and a large amount of insignificant calculations can be avoided. Instead of quantizing the weights, methods from this category exploit the sparsity of the trained convolutional layers and seek to find some low rank representations of these layers.
4. Pruning: To exploit the sparsity in neural networks, besides treating each layer as a whole and find a global replacement, it is also possible to deal with each node or connection locally. For a pre-trained neural network, connections with low importance can be removed, reducing the total number of weights. More advanced pruning techniques have been developed, see e.g. [HMD15, LUW17, AANR17].
5. Hardware: Several special purpose processors have been designed for on-device machine learning, see e.g. [Qua17, Int18, Goo18]. Recently, an all-optical machine learning device has been built, see [LRY+18]. The device can perform image classification at the speed of light without needing external power, and it achieves up to 93.39% classification accuracy for MNIST. Another work describes a hardware implementation of the kernel method, which is potentially a universal pre-processor for any data ([SCC+16]. An image is first encoded into a beam of monochromatic laser, and then projected on a random medium. Because of frequent scattering with the nanoparticles in the medium, the transmitted light can be seen as a signal obtained by applying a kernel operation with a random matrix involved. The resulting signal is then collected by a camera and can be fed into a machine learning model. No energy is consumed except for the incoming laser signal during the process.

Our proposed approach is different from all the methods listed above, but can be combined with any one of them.

Other approaches towards compressing the input in a specific manner before clustering or classifiication can be found e.g. in [DDW$^+$07, HS10, RP12, RRCR13, TPGV16, MMV18]. However, except for [MMV18], these papers do not taylor the compression operator to the classification task, but are rather using compression operators that are in line with classical compressive sensing.

2 Compressive Deep Learning

In a nutshell our framework can be summarized as follows. First, we construct a compression matrix in form of a projection operator, determined by the entire training set (or a subset of the training set). This compression matrix will be implemented in hardware, that automatically feeds the compressed data into the next layer of the system. Following the compression matrix there is a neural network that processes the compressed data and recovers the labels.

There are many possibilities to construct the projection operator. It is important to keep in mind that ultimately the projection/compression step is supposed to be realized in hardware. Therefore it makes sense to impose some structure on the projection operator to make it more amendable to an efficient hardware implementation. Unlike compressive sensing, we will not use a random matrix that samples the image space essentially uniformly at random, but instead we construct a projector that focuses on the regions of interest, i.e., we concentrate our measurement on those regions of the ambient space, in which the images we aim to classify are located, thereby preserving most information with a small number of measurements. To that end, principal component decomposition would suffice to construct the projection matrix. However, a typical PCA matrix is unstructured and is thus hard to implement efficiently, easily and at low cost in hardware.

Hence, we need to impose additional condition on the projection operator to be constructed. There is a range of options, but the most convenient one is arguably to consider projections with convolution structure. Convolutions are ubiquitous in signal- and image processing, they are a main ingredient of many machine learning techniques (deep learning being one of them), and they can be implemented efficiently in hardware [RG75].

We will consider two approaches to construct such a convolution-structured projection:

1. We try to find among all convolution matrices with orthogonal rows the one that is "most similar" to the PCA matrix. While this *matrix nearness problem* is non-convex, we will prove that there is a convex problem "nearby" and that this convex problem has a convenient explicit solution. This construction of the projection matrix is independent of the CNN we use for image classification.
2. We construct the convolution projection by jointly optimizing the projection matrix and the CNN used for image classification. We do this by adding a "zeroth" convolution layer to our image classification CNN. The weights of this zeroth layer will give us the coefficients of the (nonunitary) convolution projection matrix. Of course, for the actual image classification we later remove this zeroth layer, since the whole point is to implement this layer in hardware. In theory this should yield a projection matrix with superior performance, because this approach jointly optimizes the projection and the classification. But due to the non-convex nature of this optimization problem, there is no guarantee that we can actually find the optimal solution.

2.1 Construction of projection with convolution structure via constrained matrix nearness

Given a data set $\mathcal{D} \subseteq \mathbb{R}^M$, one can find all principal components $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$ of this finite point set, which form an orthonormal basis of the space $\mathbb{R}^M$. These m vectors can be grouped into a matrix, with each row being one of the components,

$$\boldsymbol{P} = \left[\boldsymbol{w}_1^\top, \boldsymbol{w}_2^\top, \ldots, \boldsymbol{w}_M^\top\right]^\top \in \mathbb{R}^{M \times M}.$$

The matrix $\boldsymbol{P}$ contains full information about the best approximating affine subspaces of different dimensions to the data set $\mathcal{D}$. Dimension reduction can be achieved by using the first few rows of

$\boldsymbol{P}$. For any $s \leq m$, we can construct

$$\boldsymbol{P}_s = \left[\boldsymbol{w}_1^\top, \boldsymbol{w}_2^\top, \ldots, \boldsymbol{w}_s^\top\right]^\top \in \mathbb{R}^{s \times M},$$

which can serve as a compression matrix that reduces the dimension of the input data from m to s. Here, the ratio $\frac{m}{s}$ represents the compression rate of the input.

Yet, in general $\boldsymbol{P}_s$ is not a structured matrix which is required for efficient hardware implementation. As we mentioned before, we try to find a convolution-type matrix that is in some sense as close as possible to $\boldsymbol{P}_s$. Since we focus on image classification, our input signals are images. Therefore, we consider two-dimensional convolutions. In terms of matrices that means we either deal with block-circulant matrices with circulant blocks (BCCB) or block-Toeplitz matrices with Toeplitz blocks [Dav79]. Here, for convenience we focus on BCCB matrices. Note however that as projection, $\boldsymbol{P}_s$ is naturally a fat matrix (there are more columns than rows), while BCCB are square matrices by definition. Thus, we are concerned with finding the best approximation of $\boldsymbol{P}_s$ by subsampled versions of BCCB matrices.

Assume samples in the data set $\mathcal{D}$ are all two-dimensional signals of size $m \times n$; vectorizing these signals yields vectors of length mn. The space of subsampled matrices with structure is the space of sub-block-circulant matrices, denoted by $\mathcal{S}_{s,m,n}$, whose elements are matrices with the same dimensions as $\boldsymbol{P}_s$. See the appendix for an exact definition of sub-block-circulant matrices and $\mathcal{S}_{s,m,n}$. We formulate the problem of constructing a convolution-structured projection matrix close to the PCA matrix as follows: find the sub-block-circulant matrix $\boldsymbol{C}$, an element of $\mathcal{S}_{s,m,n}$, that is most similar to $\boldsymbol{P}_s$.

How do we measure similarity between matrices? One way to think about this is to identify each matrix with its row space. In our example, the row space of any $\boldsymbol{C} \in \mathcal{S}_{s,m,n}$ is an s-dimensional subspace of $\mathbb{R}^{mn}$, the mn-dimensional Euclidean space. In other words, we identify each row space with a point in the Grassmannian manifold $G(s, mn)$ and measure the angle $\alpha(\boldsymbol{P}_s, \boldsymbol{C})$ between the row spaces of $\boldsymbol{P}_s$ and $\boldsymbol{C}$. There are various ways to do so. We follow [CHS96] and use the chordal distance for this purpose. Besides its geometric definition, the chordal distance between the row space of $\boldsymbol{P}_s$ and that of $\boldsymbol{C}$ can be computed via

$$\alpha(\boldsymbol{P}_s, \boldsymbol{C}) = \frac{1}{2}\|\boldsymbol{P}_s^*\boldsymbol{P}_s - \boldsymbol{C}^*\boldsymbol{C}\|_F.$$

Hence our goal is to find the sub-block-circulant matrix $\boldsymbol{C} \in \mathcal{S}_{s,m,n}$ that minimizes its chordal distance to $\boldsymbol{P}_s$.

$$\min\ \boldsymbol{\alpha}(\boldsymbol{P}_s, \boldsymbol{C}) \qquad \text{s.t. } \boldsymbol{C} \in \mathcal{S}_{s,m,n} \tag{1}$$

This *structured matrix nearness problem* does not have an easy solution. Therefore we consider a "nearby" problem that does have a nice explicit solution. We compute

$$\begin{aligned}
\alpha(\boldsymbol{P}_s, \boldsymbol{C}) &= \frac{1}{2}\|\boldsymbol{P}_s^*\boldsymbol{P}_s - \boldsymbol{C}^*\boldsymbol{C}\|_F \\
&= \frac{1}{2}\|\boldsymbol{P}_s^*(\boldsymbol{P}_s - \boldsymbol{C}) - (\boldsymbol{P}_s^* - \boldsymbol{C}^*)\boldsymbol{C}\|_F \\
&\leq \frac{1}{2}(\|\boldsymbol{P}_s\|_F + \|\boldsymbol{C}\|_F)\|\boldsymbol{P}_s - \boldsymbol{C}\|_F \\
&= s\|\boldsymbol{P}_s - \boldsymbol{C}\|_F
\end{aligned} \tag{2}$$

We used the fact that both $\boldsymbol{P}_s$ and elements of $\mathcal{S}_{s,m,n}$ have unit vector as rows. The above estimation shows that even though $s\|\boldsymbol{P}_s - \boldsymbol{C}\|_F$ does not determine the optimal chordal distance between two row spaces, it is always an upper bound of the chordal distance. Inspired by this estimation, instead of the original optimization problem (1), we consider the following problem:

$$\min\ \|\boldsymbol{P}_s - \boldsymbol{C}\|_F \quad \text{s.t. } \boldsymbol{C} \in \mathcal{S}_{s,m,n} \tag{3}$$

Note that s here is a constant and is determined by the compression rate. This optimization problem can be solved within a more general setting, namely in the context of complete commuting family of unitary matrices.

To solve (3), we consider the more general problem

$$\min_{\boldsymbol{C} \in \mathcal{A}}\ \|\boldsymbol{W} - \boldsymbol{C}\|_F, \tag{4}$$

where $\boldsymbol{W}$ is a given matrix of dimension $n \times n$ and $\mathcal{A}$ is a commuting family of unitary matrices. The exact definition of $\mathcal{A}$ is given in the appendix. More precisely, we assume $\mathcal{A}$ is a complete commuting family of unitary matrices and $\boldsymbol{U}$ is a unitary matrix that diagonalizes elements of $\mathcal{A}$. Note that unitary transformation preserves the Frobenius norm since for any $\boldsymbol{X}$,

$$\|\boldsymbol{U}^*\boldsymbol{X}\boldsymbol{U}\|_F^2 = \mathrm{Tr}(\boldsymbol{U}^*\boldsymbol{X}\boldsymbol{U}(\boldsymbol{U}^*\boldsymbol{X}\boldsymbol{U})^*) = \mathrm{Tr}(\boldsymbol{X}\boldsymbol{U}\boldsymbol{U}^*\boldsymbol{X}^*\boldsymbol{U}\boldsymbol{U}^*) = \mathrm{Tr}(\boldsymbol{X}\boldsymbol{X}^*) = \|\boldsymbol{X}\|_F^2$$

As a result, the objective function is then

$$\|\boldsymbol{W} - \boldsymbol{C}\|_F = \|\boldsymbol{U}^*\boldsymbol{W}\boldsymbol{U} - \boldsymbol{U}^*\boldsymbol{C}\boldsymbol{U}\|_F,$$

which can be decomposed into its diagonal part and its off-diagonal part. Since the off-diagonal part is independent of $\boldsymbol{C}$, in order to minimize $\|\boldsymbol{W} - \boldsymbol{C}\|_F$, we only need to minimize the diagonal part. Let $\mathrm{diag}(\boldsymbol{c}) = \boldsymbol{U}^*\boldsymbol{C}\boldsymbol{U}$. By definition, $\boldsymbol{c} = \theta_U^{-1}(\boldsymbol{C}) \in T^n$. Hence, minimization of the diagonal part becomes

$$\min_{\boldsymbol{c} \in T^n} \|\,\mathrm{diag}(\boldsymbol{U}^*\boldsymbol{W}\boldsymbol{U}) - \boldsymbol{c}\|,$$

where the norm is simply the Euclidean norm in $\mathbb{C}^n$. The minimizer may not be unique, but one minimizer $\boldsymbol{c}_0$ is given by

$$\begin{cases} (\boldsymbol{c}_o)_i = 1, & y_i = 0, \\ (\boldsymbol{c}_o)_i = y_i/|y_i|, & \text{otherwise}, \end{cases} \tag{5}$$

where $y_i = (\boldsymbol{U}^*\boldsymbol{W}\boldsymbol{U})_{ii}$. Applying the parametrization map, we get

$$\boldsymbol{C}_o = \theta_U(\boldsymbol{c}_o) = \boldsymbol{U}\,\mathrm{diag}^{-1}(\boldsymbol{c}_o)\boldsymbol{U}^* \in \mathcal{A}, \tag{6}$$

due to the completeness of $\mathcal{A}$, and $\boldsymbol{C}_o$ is a minimizer of (4). Thus, we have proved the following theorem.

Theorem 1. *For any complete commuting family of unitary matrices $\mathcal{A}$, the optimization problem* (4) *has a solution given by* (6).

We can readily show that the family of BCCB matrices, denoted by $\mathcal{N}_{m,n}$, is a special case.

Corollary 1. *The collection of all unitary block circulant matrices with circulant blocks $\mathcal{N}_{m,n}$ is a complete commuting family of unitary matrices, and a solution to the optimization problem* (4) *is given by*

$$\boldsymbol{C}_o = \theta_U(\boldsymbol{c}_o) = \boldsymbol{F}_{m,n}\,\mathrm{diag}^{-1}(\boldsymbol{c}_o)\boldsymbol{F}_{m,n}^* \in \mathcal{A}, \tag{7}$$

where $\boldsymbol{c}_o$ is determined by (5). *Moreover, if $\boldsymbol{W}$ in* (4) *is a real matrix, then $\boldsymbol{C}_o$ given by* (7) *is also a real matrix.*

See the appendix for the proof.

Definition 1 (Downsampling Operator). *For any $1 \leq k \leq n$, a downsampling operator is 1-1 map ψ from $\mathbb{Z}_k$ to $\mathbb{Z}_n$, where $\mathbb{Z}_p = \{1, 2, \ldots, p\}$.*

The downsampling operator ψ chooses k elements from $\mathbb{Z}_n$ in a certain order without replacement. We will use ψ to sample the rows of a matrix from $\mathcal{N}_{m,n}$.

Definition 2 (Subsampled Unitary BCCB). *The Subsampled Unitary BCCB $\boldsymbol{B}$ formed by downsampling $\boldsymbol{C} \in \mathcal{N}_{m,n}$ via the downsampling operator $\psi : \mathbb{Z}_s \to \mathbb{Z}_{mn}$ is an $s \times mn$ matrix given by*

$$(\boldsymbol{B})_{i,j} = (\boldsymbol{C})_{\psi(i),j}, \quad \forall i \in \mathbb{Z}_s, j \in \mathbb{Z}_{mn}.$$

We denote $\psi(\boldsymbol{C}) = \boldsymbol{B}$. The collection of all such matrices is denoted $\mathcal{S}_{s,m,n;\psi}$, which we will simply refer to as $\mathcal{S}_{s,m,n}$.

A subsampled unitary BCCB is formed by certain rows of some unitary block circulant matrix with circulant blocks whose indices are determined by a fixed downsampling operator.

Consider a downsampling operator $\psi : \mathbb{Z}_s \to \mathbb{Z}_{mn}$. Let $\mathcal{A}$ be a complete commuting family of unitary matrices. If we subsample every element of $\mathcal{A}$ using ψ, we end up getting a set of subsampled unitary matrices,

$$\mathcal{B} = \{\psi(\boldsymbol{C}) : \boldsymbol{C} \in \mathcal{A}\}.$$

Now, consider the following optimization problem,

$$\min_{\boldsymbol{B}\in\mathcal{B}} \quad \|\boldsymbol{W}-\boldsymbol{B}\|_F, \tag{8}$$

where $\boldsymbol{W}$ is a given matrix with dimension of $s\times mn$. Define the *zero-padding map* $\rho_\psi : \mathbb{C}^{s\times mn} \to \mathbb{C}^{mn\times mn}$ as follows:

$$(\rho_\psi(\boldsymbol{W}))_{i,j} = \begin{cases} w_{i,j}, & i \in \operatorname{Im}\psi, \\ 0, & i \notin \operatorname{Im}\psi, \end{cases}$$

for any $i, j \in \mathbb{Z}_{mn}$. Observe that any solution $\boldsymbol{C}_0$ to

$$\min_{\boldsymbol{C}\in\mathcal{A}} \|\rho(\boldsymbol{W})-\boldsymbol{C}\|_F$$

gives a solution $\psi(\boldsymbol{C}_0)$ to (8). This is because for any $\boldsymbol{C}\in\mathcal{A}$,

$$\begin{aligned}\|\rho(\boldsymbol{W})-\boldsymbol{C}\|_F^2 &= \|\boldsymbol{W}-\psi(\boldsymbol{C})\|^2 + \sum_{i\notin\operatorname{Im}\psi}\|\boldsymbol{c}_i\|^2 \\ &= \|\boldsymbol{W}-\psi(\boldsymbol{C})\|^2 + mn - k,\end{aligned}$$

where $\boldsymbol{c}_i$ is the i-th row of $\boldsymbol{C}$ and is a unit vector.

Theorem 2. *If $\mathcal{B}$ is a family of subsampled commuting unitary matrices, then an optimization problem of the form* (8) *can be converted into a problem of commuting family of unitary matrices of the form* (4) *using the zero-padding operator.*

Downsampled from $\mathcal{N}_{m,n}$ by some downsampling operator ψ, the collection of subsampled unitary BCCB $\mathcal{S}_{s,m,n}$ is obviously a special case since $\mathcal{S}_{s,m,n} = \psi(\mathcal{N}_{m,n})$. Therefore, the optimization problem

$$\begin{aligned} \min \quad & \|\boldsymbol{P}_s-\boldsymbol{C}\|_F \\ \text{s.t.} \quad & \boldsymbol{C}\in\mathcal{S}_{s,m,n}\end{aligned}$$

can be solved by following Theorem 2 and Theorem 1. The procedure of solving this problem is summarized in Algorithm 1.

Algorithm 1: Construction of deep learning compression matrix

```
Input  : Data set $\mathcal{D} \subseteq \mathbb{R}^{m\times n}$, dimension $s$ of the compressed data, downsampling operator $\psi$
Output: Compression matrix with structure $\psi(\boldsymbol{C}_0)$
1 Find the first $s$ PCA-components of $\mathcal{D}$ and use them as rows to form an $s \times mn$ matrix $\boldsymbol{P}_s$
2 Form an $mn \times mn$ matrix $\rho_\psi(\boldsymbol{P}_s)$ using the zero-padding operator $\rho_\psi$
3 Apply the 2D DFT matrix $\boldsymbol{F}_{m,n}$ and extract the diagonal
      $\boldsymbol{c} = \operatorname{diag}(\boldsymbol{F}_{m,n}^*\rho_\psi(\boldsymbol{P}_s)\boldsymbol{F}_{m,n})$
4 Form a new vector $\boldsymbol{c}_0$ by normalizing components of $\boldsymbol{c}$. Replace by 1 if the entry is 0
5 Form a diagonal matrix and apply the 2D DFT matrix. The result is a unitary block circulant
    matrix with circulant blocks
      $\boldsymbol{C}_0 = \boldsymbol{F}_{m,n}\operatorname{diag}^{-1}(\boldsymbol{c}_0)\boldsymbol{F}_{m,n}^*.$
6 Subsample the matrix and get $\psi(\boldsymbol{C}_0)$. Use $\psi(\boldsymbol{C}_0)$ to compress the data.
```

2.2 Construction of convolution projection via joint non-convex optimization

In this section we construct a projection matrix by setting up a joint optimization problem that involves both solving the image classification and the finding the optimal projection matrix. To solve this non-convex optimization problem we employ deep learning. More precisely, we construct a CNN to which we add an additional layer as zeroth layer. This zero-th layer will consist of one 2-D convolution followed by downsampling. The overall architecture is illustrated in Figure 2.

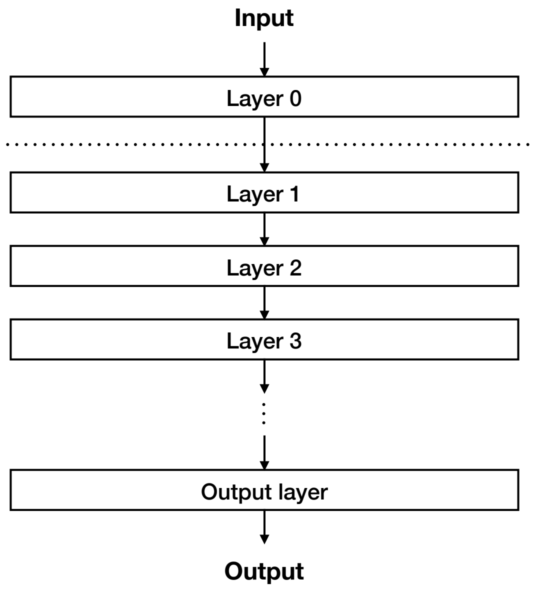

Figure 2: The architecture of a CNN with an additional layer. Layers below the zero-th layer form a CNN themselves. The zero-th layer is a convolutional layer with only one filter and a stride greater than one. Thus, the zero-th layer applies a single convolution filter on the input images and then downsamples the results. The output of the zero-th layer is then passed to the next layer. All layers will be trained jointly on a training set.

3 Numerical experiments

To demonstrate the capabilities of our methods, we test both the PNN (name may be changed) and the circulant approximation against a few other baseline methods. We test these methods on two standard test sets, the MNIST dataset consisting of images depicting hand-written digits, and the Fashion-MNIST dataset, consisting of images of fashion products. The general workflow consists of two steps. In the preprocssing step we subject the images to the projection operator to simulate the compressive image acquistion via hardware. In the second step we feed these images into a convolutional neural network for classification. We will describe the preprocessing methods, the architecture of the network and training details in the rest of the section.

3.1 Preprocessing step: Projection operator

In order to put the PNN and the circulant approximation method in context, we conducted a few other methods in the experiments and compare their results against each other. The list below is a brief summary of all these results, and Table 1 gives the relation between the stride and the compression rate.

1. **Downsampling**: Downsample the images using a certain stride. When the stride is equal to 1, the original dataset is used. Downsampling, i.e. just taking low-resolution images, is the easiest (and least sophisticated) way to reduce dimensionality of the images.
2. **Random Convolution**: A random convolutional filter of size 5×5 is generated and then applied to a few locations in the image, determined by the stride. The size of the resulted image is determined by the stride of the convolution applied to the image. The image size is unchanged when the stride is equal to 1. There is no constraints on the filter size, but we fixed the filter size in our experiments for the sake of simplicity. Projection matrices of this type have been proposed in the context of compressive sensing (e.g. see [Rom09]).
3. **PCA**: Compute the PCA components for the entire training set. Given the compression rate and a raw image, the compressed data is the coefficients of a few leading PCA component

for the image. When uncompressed (the compression rate is equal to 1), all coefficients are used.

4. **Circulant Approximation**: This is the construction outlined in Subsection 2.1. First compute a matrix with structure that is most similar to the PCA matrix of the training set and then compress images using this matrix. The dimension of the matrix with structure is determined by the compression rate. It is a square matrix if the data is uncompressed.
5. **PNN**: This is the construction presented in Subsection 2.2. Add a convolutional layer with a certain stride and one single feature map on the top of the architecture. Train the resulting network on the original dataset. The first convolutional layer serves as a compressor, and it is optimized along with the rest of the network.

Stride	Dimension	Compression
1	28×28	1.00
2	14×14	4.00
3	10×10	7.84
4	7×7	16.00
5	6×6	21.78
6	5×5	31.36

Table 1: Relation between the stride and the compression rate. The dimension of the raw inputs is 28×28. After applying one of the preprocessing method with a certain stride, the dimension of the data becomes smaller. The compression is the ratio between the number of pixels in the the raw data and that of the processed data.

3.2 Architecture and training

The neural network for the raw data has the following architecture. The first weighted layer is a convolution layer with 32 filters of the size 5×5 and followed by ReLU nonlinearity and a maxpooling layer with stride 2. The second weighted layer is the same as the first one except that it has 64 filters, also followed by ReLU and a maxpooling layer with stride 2. The third weighted layer is a fully connected layer with 256 units and followed by a dropout layer. The last layer is a softmax layer with 10 channels, corresponding to the 10 classes in the dataset. This network has 857,738 weights in total. Architectures of neural networks dealing with compressed data are listed in Table 2.

The table also lists the total number of weights and the number of floating point operations for each forward pass. In general, both numbers are decreased when we use smaller inputs. Since we are using very small images in the first place, some networks have two pooling layers and some do not have any. This explains why some networks with smaller inputs have more weights and flops than networks with larger inputs. For larger images, we will be able to apply same number of pooling operations both for the raw data and the compressed data. In this way, the decrease in the number of weights and flops will be even more significant.

For PNN, it starts with a convolutional layer with a single filter of the size 5×5 and a certain stride. The output is then fed into the network described above.

We trained the networks using ADAM. We also did a grid search for hyper-parameters. In general, we found that with learning rate $= 0.01$, dropout rate $= 0.4$ generated the best results. Each network was trained for 10 epochs with mini-batches of the size 32.

3.3 Results

As shown in Figure 3, for both datasets, it is evident that both PNN and the circulant approximation achieve higher accuracy rate than downsampling and random convolution, especially when the data are heavily compressed. In most cases, the PNN method works slightly better than the circulant approximation. However, the circulant approximation method exhibits its ability to retain high accuracy when pushing to more extreme compression rate on MNIST. Since PNN is a relaxation of the circulant approximation method, the global optimum of the former is always no worse than the

Stride	1	2	3	4	5	6
Input	28×28	14×14	10×10	7×7	6×6	5×5
First convolution	conv32					
First pooling	maxpool2			-		
Second convolution	conv32					
Second pooling	maxpool2	-				
Fully connected	FC256					
Output	softmax10					
Weights	857,738	857,738	464,522	857,738	644,746	464,522
MegaFlops-sec	22.93	6.94	3.54	6.70	4.92	3.42

Table 2: Architectures for different input sizes. conv32 means a convolutional layer with 32 filters. All filters have size 5×5. maxpool2 means a maxpooling layer with stride 2×2. FC256 represents a fully connected layer with 256 units and softmax10 is the 10-way softmax layer. The second from the last row are the total numbers of weights in these networks. The last row lists the number of floating point operations for each forward pass. The softmax layer is not included since the cost of the exponential function is hard to estimate. Nonetheless, the last layers of all these networks are the same. Refer to Figure 3 for the performance of these network architectures.

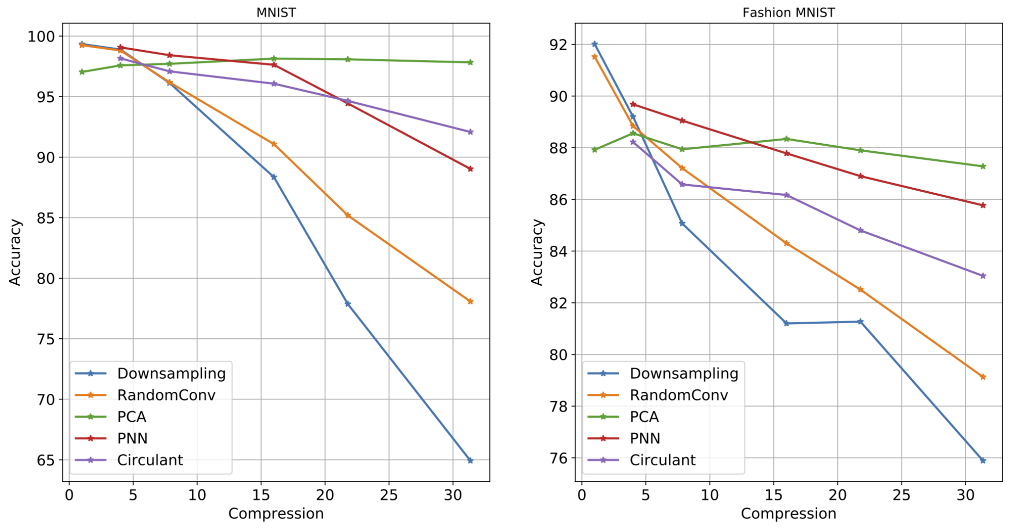

Figure 3: Accuracy rates of compressive deep learning with various choices for the projection matrix, tested on MNIST and on Fashion-MNIST. The x-axis represents the compression rate of the input (this is determined by the strides). The relation between the strides and the compression is given in Table 1.

latter. What we observe in MNIST is a result of the training process, which has no guarantee for global optimum of the neural network.

Another interesting result is that the PCA method seems not to be affected by extreme compression rates but rather benefits from them. This is probably because only the coefficients of the leading PCA components have high signal-to-noise ratio, and the rest are mostly noise. Therefore, the PCA method performs better simply by discarding the noisy coefficients. For the purpose of our work, the PCA method cannot be compared with the other methods directly since *the PCA matrix is not a convolution and cannot be implemented by hardware*.

3.4 Reducing the number of filters

As we have seen in Table 2, because of the small image size, some networks with smaller inputs have more weights and flops than networks with larger inputs. To further reduce the number of weights and flops for networks with small inputs, we explore the compressibility of these networks

Index	1	2	3	4	5
Input	5×5				
First convolution	conv32	conv16	conv8	conv4	conv2
Second convolution	conv64	conv32	conv16	conv8	conv4
Fully connected	FC256				
Output	softmax10				
Weights	464,522	220,874	108,650	54,938	28,682
MegaFlops-sec	3.42	1.07	0.37	0.15	0.06

Table 3: Neural networks architectures with fewer filters in their convolutional layers. The number of weights and consequently also the number of flops are reduced.

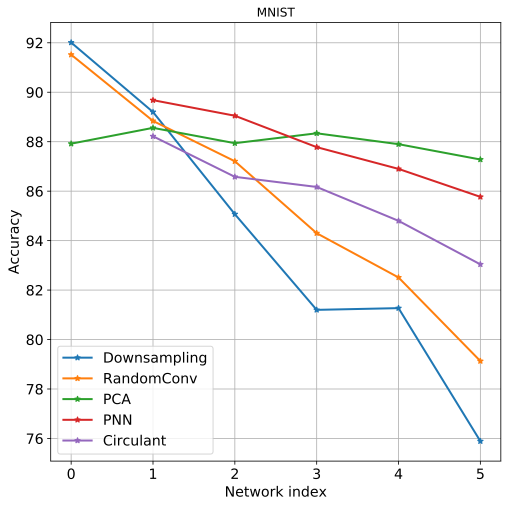

Figure 4: Accuracy rates of compressive deep learning with various choices for the projection matrix, tested on MNIST. The x-axis represents different network architectures defined in Table 3. Networks with larger indices have less weights and smaller number of flops.

by reducing the number of filters. For the network for the case of stride 6, the network given in Table 2 has 32 filters in the first convolutional layer and 64 filters in the second one. Here, we consider using networks with much less filters. The architectures of these networks are listed below. The network with index 1 is the same as the network with stride 6 in Table 2.

The results of these neural networks for MNIST are shown in Figure 4. Both the PNN and the circulant approximation have relatively high accuracy rates even with very small number of filters.

4 Appendix

4.1 Matrices with circulant strcuture

Circulant matrices can be thought as discrete convolutions applicable to one dimensional signals. They have the nice property that circulant matrices can be diagonalized by the Discrete Fourier Matrix. For two-dimensional signals, the corresponding matrix is a block circulant matrix with circulant blocks (BCCB).

Definition 3 (Block Circulant Matrix with Circulant Blocks)**.** *An $mn \times mn$ block circulant matrix with circulant blocks $\boldsymbol{C}$ has an $m \times m$ circulant block structure and each of its block $\boldsymbol{C}_i \in \mathcal{A}_n$ is* *an $n \times n$ circulant matrix itself. The matrix $\boldsymbol{C}$ can be written as*

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{C}_0 & \boldsymbol{C}_{m-1} & \cdots & \boldsymbol{C}_2 & \boldsymbol{C}_1 \\ \boldsymbol{C}_1 & \boldsymbol{C}_0 & \boldsymbol{C}_{m-1} & & \boldsymbol{C}_2 \\ \vdots & \boldsymbol{C}_1 & \boldsymbol{C}_0 & \ddots & \vdots \\ \boldsymbol{C}_{m-2} & & \ddots & \ddots & \boldsymbol{C}_{m-1} \\ \boldsymbol{C}_{m-1} & \boldsymbol{C}_{m-2} & \cdots & \boldsymbol{C}_1 & \boldsymbol{C}_0 \end{bmatrix}.$$

The lemma below from [Dav79] (Theorem 5.8.1) shows that BCCBs can be diagonalized by the 2D DFT.

Lemma 1 (Diagonalization by 2D DFT). *Let $\boldsymbol{F}_{m,n}$ be the two-dimensional unitary Discrete Fourier Transform matrix. Then, a matrix $\boldsymbol{C}$ is a block circulant matrix with circulant blocks if and only if $\boldsymbol{F}_{m,n}^*\boldsymbol{C}\boldsymbol{F}_{m,n}$ is a diagonal matrix. In particular, if $\boldsymbol{C}$ is a block circulant matrix with circulant blocks and $\boldsymbol{c}$ is its first column, then $\boldsymbol{C}$ can be diagonalized by $\boldsymbol{F}_{m,n}$ as*

$$\boldsymbol{F}_{m,n}^*\boldsymbol{C}\boldsymbol{F}_{m,n} = \sqrt{mn}\,\mathrm{diag}(\boldsymbol{F}_{m,n}\boldsymbol{c}).$$

4.2 Complete commuting family of unitary matrices

Definition 4 (Commuting Family of Unitary Matrices). *$\mathcal{A}$ is called a commuting family of unitary matrices if $\mathcal{A}$ is consisted of unitary matrices of the same dimension that commute with each other.*

Properties of commuting families of unitary matrices are presented below.

Definition 5 (Parametrization Map). *Let $\boldsymbol{U}$ be a unitary matrix, then the parametrization map θ_U induced by $\boldsymbol{U}$ is defined by*

$$\theta_U : \mathbb{C}^n \to \mathbb{C}^{n\times n}, \boldsymbol{c} \to \boldsymbol{U}\,\mathrm{diag}(\boldsymbol{c})\boldsymbol{U}^*.$$

Lemma 2. *Let $\mathcal{A}$ be a commuting family of unitary matrices. There exists a unitary matrix $\boldsymbol{U}$ that diagonalizes any $\boldsymbol{C} \in \mathcal{A}$. Furthermore, the preimage of $\mathcal{A}$ under θ_U is a subset of the n-dimensional torus T^n, where $T^n = S^1 \times S^1 \times \cdots \times S^1 \subseteq \mathbb{C}^n$.*

Proof. A commuting family of matrices can be simultaneously diagonalized, and unitary matrices are normal matrices and hence can be diagonalized by some unitary matrix. Therefore, there exists a unitary matrix $\boldsymbol{U}$ that diagonalizes any $\boldsymbol{C} \in \mathcal{A}$. For any $\boldsymbol{C} \in \mathcal{A}$, $\boldsymbol{U}^*\boldsymbol{C}\boldsymbol{U} = \mathrm{diag}(\boldsymbol{c})$ for some $\boldsymbol{c} \in \mathbb{C}^n$. Note that entries of $\boldsymbol{c}$ are eigenvalues of $\boldsymbol{C}$ since $\boldsymbol{U}$ is unitary. Hence, $\boldsymbol{c} \in T^n$ since $\boldsymbol{C}$ is a unitary matrix itself and eigenvalues of a unitary matrix are complex numbers with magnitude 1. Finally, θ_U is a one-to-one map and $\theta_U^{-1}(\boldsymbol{C}) = \boldsymbol{c}$ since $\boldsymbol{C} = \boldsymbol{U}\,\mathrm{diag}(\boldsymbol{c})\boldsymbol{U}^*$. □

To ensure solvability of (4), we will need completeness for $\mathcal{A}$.

Definition 6 (Complete Commuting Family of Unitary Matrices). *A commuting family of unitary matrices $\mathcal{A}$ is called complete if its preimage under the parametrization map is the entire torus. In other words, $\theta_U^{-1}(\mathcal{A}) = T^n$.*

Proof of Corollary 1. For any $\boldsymbol{C} \in \mathcal{N}_{m,n}$, $\boldsymbol{F}_{m,n}^*\boldsymbol{C}\boldsymbol{F}_{m,n}$ is a diagonal matrix. This shows $\mathcal{N}_{m,n}$ is a commuting family, and the unitary matrix $\boldsymbol{U}$ that diagonalizes all elements of the family is $\boldsymbol{F}_{m,n}$. By its definition, $\mathcal{N}_{m,n}$ consists of unitary matrices. To prove it is complete, we need to show $\theta_{F_{m,n}}(\boldsymbol{c}) \in \mathcal{N}_{m,n}$, for any $\boldsymbol{c} \in T^n$. Denote $\boldsymbol{C}_0 = \theta_{F_{m,n}}(\boldsymbol{c}) = \boldsymbol{F}_{m,n}\,\mathrm{diag}(\boldsymbol{c})\boldsymbol{F}_{m,n}^*$. On the one hand, thanks to Lemma 1, $\boldsymbol{C}_0 \in \mathcal{BC}_{m,n}$ since $\boldsymbol{F}_{m,n}^*\boldsymbol{C}_0\boldsymbol{F}_{m,n}$ is diagonal. On the other hand, $\boldsymbol{C}_0$ is unitary since

$$\begin{aligned} \boldsymbol{C}_0\boldsymbol{C}_0^* &= \boldsymbol{F}_{m,n}\,\mathrm{diag}(\boldsymbol{c})\boldsymbol{F}_{m,n}^*\boldsymbol{F}_{m,n}\,\mathrm{diag}(\boldsymbol{c}^*)\boldsymbol{F}_{m,n}^* \\ &= \boldsymbol{F}_{m,n}\,\mathrm{diag}(\boldsymbol{c}\odot\boldsymbol{c}^*)\boldsymbol{F}_{m,n}^* \\ &= \boldsymbol{F}_{m,n}\boldsymbol{F}_{m,n}^* \\ &= \boldsymbol{I}_{mn}. \end{aligned}$$

Here, $\boldsymbol{c} \odot \boldsymbol{c}^* = \boldsymbol{1}$ is the entrywise product and we used the fact that $\boldsymbol{c} \in T^n$. This shows $\theta_{F_{m,n}}(\boldsymbol{c}) = \boldsymbol{C}_0 \in \mathcal{N}_{m,n}$.

To show the last statement of the corollary, we will use the fact that the eigenvalues of a real circulant matrix are conjugate symmetric, and vice versa (cf. [Dav79], p. 72-76.). For any $\boldsymbol{C} \in \mathcal{N}_{m,n}$, $\boldsymbol{C}$ can be written as $\boldsymbol{C} = \sum_i \boldsymbol{E}_i \otimes \boldsymbol{C}_i$, where $\boldsymbol{C}_i$ are circulant blocks of $\boldsymbol{C}$ and $\boldsymbol{E}_i = \mathrm{Circ}(\boldsymbol{e}_i)$. Hence, $\boldsymbol{F}_{m,n}^* \boldsymbol{C} \boldsymbol{F}_{m,n} = \sum_i \boldsymbol{F}_m^* \boldsymbol{E}_i \boldsymbol{F}_m \otimes \boldsymbol{F}_n^* \boldsymbol{C}_i \boldsymbol{F}_n$. Since $\boldsymbol{E}_i$ and $\boldsymbol{C}_i$ are real matrices, both $\boldsymbol{F}_m^* \boldsymbol{E}_i \boldsymbol{F}_m$ and $\boldsymbol{F}_n^* \boldsymbol{C}_i \boldsymbol{F}_n$ are diagonal matrices with conjugate symmetric diagonals. Then, $\mathrm{diag}(\boldsymbol{F}_m^* \boldsymbol{E}_i \boldsymbol{F}_m)$ and $\mathrm{diag}(\boldsymbol{F}_n^* \boldsymbol{C}_i \boldsymbol{F}_n)$ are normalized as in (5). The resulting vectors are still conjugate symmetric. Denote these two vectors $\boldsymbol{v}_i^{(1)}$ and $\boldsymbol{v}_i^{(2)}$, both column vectors. Thus, $\boldsymbol{c}_o = \sum_i \boldsymbol{v}_i^{(1)} \otimes \boldsymbol{v}_i^{(2)}$, and $\boldsymbol{C}_o = \boldsymbol{F}_{m,n} \,\mathrm{diag}^{-1}(\boldsymbol{c}_o) \boldsymbol{F}_{m,n}^* = \sum_i \boldsymbol{M}_i^{(1)} \otimes \boldsymbol{M}_i^{(2)}$, where

$$\begin{cases} \boldsymbol{M}_i^{(1)} = \boldsymbol{F}_m \,\mathrm{diag}^{-1}\left(\boldsymbol{v}_i^{(1)}\right) \boldsymbol{F}_m^* \\ \boldsymbol{M}_i^{(2)} = \boldsymbol{F}_n \,\mathrm{diag}^{-1}\left(\boldsymbol{v}_i^{(2)}\right) \boldsymbol{F}_n^* \end{cases}$$

Both $\boldsymbol{M}_i^{(1)}$ and $\boldsymbol{M}_i^{(2)}$ are real, and thus $\boldsymbol{C}_o$ is also real. $\square$

5 Discussion and Conclusion

We have introduced the first steps towards developing a principled hybrid hardware-software framework that has the potential to significantly reduce the computational complexity and memory requirements of on-device machine learning. At the same time, the proposed framework raises several challenging questions. How much can we compress data so that we can still with high accuracy conduct the desired machine learning task? In certain cases literature offers some answers (see e.g. [RRCR13, TPGV16, MMV18]), but for more realistic scenarios research is still in its infancy. Moreover, ideas from transfer learning should help in designing efficient deep networks for compressive input. We hope to address some of these questions in our future work.

Acknowledgments

The authors want to thank Donald Pinckney for discussions and initial simulations related to the topic of this paper. The authors. acknowledge partial support from NSF via grant DMS 1620455 and from NGA and NSF via grant DMS 1737943.

References

[AANR17] Alireza Aghasi, Afshin Abdi, Nam Nguyen, and Justin Romberg. Net-trim: Convex pruning of deep neural networks with performance guarantee. In *Advances in Neural Information Processing Systems*, pages 3177–3186, 2017.

[CHS96] John H. Conway, Ronald H. Hardin, and Neil JA Sloane. Packing lines, planes, etc.: Packings in grassmannian spaces. *Experimental mathematics*, 5(2):139–159, 1996.

[Dav79] Philip J. Davis. *Circulant matrices*. John Wiley & Sons, 1979.

[DDW$^+$07] Mark A Davenport, Marco F Duarte, Michael B Wakin, Jason N Laska, Dharmpal Takhar, Kevin F Kelly, and Richard G Baraniuk. The smashed filter for compressive classification and target recognition. In *Computational Imaging V*, volume 6498, page 64980H. International Society for Optics and Photonics, 2007.

[DHYY17] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[DP17] M. Deisher and A. Polonski. Implementation of efficient, low power deep neural networks on next-generation intel client platforms. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6590–6591, March 2017.

[FR13] S. Foucart and H. Rauhut. *A Mathematical Introduction to Compressive Sensing*. Applied and Numerical Harmonic Analysis. Springer, Basel, 2013.

[Goo18] Google. Edge TPU, 2018. https://cloud.google.com/edge-tpu.

[HMD15] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[HS10] Blake Hunter and Thomas Strohmer. Compressive spectral clustering. In *AIP Conference Proceedings*, volume 1281(1), pages 1720–1722. AIP, 2010.

[HZC$^+$17] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[IHM$^+$16] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[Int18] Intel. Hardware, 2018. https://ai.intel.com/hardware.

[LRY$^+$18] Xing Lin, Yair Rivenson, Nezih T Yardimci, Muhammed Veli, Yi Luo, Mona Jarrahi, and Aydogan Ozcan. All-optical machine learning using diffractive deep neural networks. *Science*, 361(6406):1004–1008, 2018.

[LUW17] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3288–3298, 2017.

[MMV18] Culver McWhirter, Dustin G Mixon, and Soledad Villar. Squeezefit: Label-aware dimensionality reduction by semidefinite programming. *arXiv preprint arXiv:1812.02768*, 2018.

[Qua17] Qualcomm. We are making on-device AI ubiquitous, 2017. https://www.qualcomm.com/news/onq/2017/08/16/we-are-making-device-ai-ubiquitous.

[RG75] Lawrence R Rabiner and Bernard Gold. *Theory and application of digital signal processing*. Englewood Cliffs, NJ, Prentice-Hall, Inc., 1975.

[Rom09] Justin Romberg. Compressive sensing by random convolution. *SIAM Journal on Imaging Sciences*, 2(4):1098–1128, 2009.

[RP12] Andrzej Ruta and Fatih Porikli. Compressive clustering of high-dimensional data. In *2012 11th International Conference on Machine Learning and Applications*, volume 1, pages 380–385. IEEE, 2012.

[RRCR13] Hugo Reboredo, Francesco Renna, Robert Calderbank, and Miguel RD Rodrigues. Compressive classification. In *2013 IEEE International Symposium on Information Theory*, pages 674–678. IEEE, 2013.

[SCC$^+$16] Alaa Saade, Francesco Caltagirone, Igor Carron, Laurent Daudet, Angélique Drémeau, Sylvain Gigan, and Florent Krzakala. Random projections through multiple optical scattering: Approximating kernels at the speed of light. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 6215–6219. IEEE, 2016.

[TPGV16] Nicolas Tremblay, Gilles Puy, Rémi Gribonval, and Pierre Vandergheynst. Compressive spectral clustering. In *International Conference on Machine Learning*, pages 1002–1011, 2016.

[Zub19] Shoshana Zuboff. *The Age of Surveillance Capitalism: The Fight for the Future at the New Frontier of Power*. PublicAffairs, 2019.