AFRL-RY-WP-TP-2020-0031

LOCALIZED COMPRESSION: APPLYING CONVOLUTIONAL NEURAL NETWORKS TO COMPRESSED IMAGES (Preprint)

Christopher A. George and Bradley M. West

Boston Fusion Corp.

APRIL 2020
Final Report

THIS IS A SMALL BUSINESS TECHNOLOGY TRANSFER (STTR) PHASE II PAPER.

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
April 2020	Conference Paper Preprint	19 November 2019 – 19 November 2019

4. TITLE AND SUBTITLE
LOCALIZED COMPRESSION: APPLYING CONVOLUTIONAL NEURAL NETWORKS TO COMPRESSED IMAGES (Preprint)

5a. CONTRACT NUMBER
FA8650-17-C-1154

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
65502F

6. AUTHOR(S)
Christopher A. George and Bradley M. West

5d. PROJECT NUMBER
STTR

5e. TASK NUMBER
N/A

5f. WORK UNIT NUMBER
Y1NY

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Boston Fusion Corp.
70 Westview Street, Suite 100
Lexington, MA 02421

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH 45433-7320
Air Force Materiel Command
United States Air Force

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/RYAA

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-RY-WP-TP-2020-0031

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
PAO Case Number: 88ABW-2019-5613, cleared 19 November 2019. This is a Small Business Technology Transfer (STTR) Phase II paper. STTR data rights waived by contractor. Letter on file. This work was funded in whole or in part by Department of the Air Force contract FA8650-17-C-1154. The U.S. Government has for itself and others acting on its behalf an unlimited, paid-up, nonexclusive, irrevocable worldwide license to use, modify, reproduce, release, perform, display, or disclose the work by or on behalf of the U. S. Government. To be presented at SPIE Applications of Machine Learning 2020, San Diego, CA, 12 February 2020. Document contains color.

14. ABSTRACT

We address the challenge of applying existing convolutional neural network (CNN) architectures to compressed images. Existing CNN architectures represent images as a matrix of pixel intensities with a specified dimension; this desired dimension is achieved by downgrading or cropping. Downgrading and cropping are attractive in that the result is also an image; however, an algorithm producing an alternative "compressed" representation could yield better classification performance. This compression algorithm need not be reversible, but must be compatible with the CNN's operations. This problem is thus the counterpart of the well-studied problem of applying compressed CNNs to uncompressed images, which has attracted great interest as CNNs are deployed to size-, weight-, and power- (SWaP)- limited devices. In this brief, we introduce localized compression, a generalization of downgrading in which the original image is divided into blocks and each block is compressed to a smaller size using either sampling- or random-matrix-based techniques. By aligning the size of the compressed blocks with the size of the CNN's convolutional region, localized compression can be made compatible with any CNN architecture. Our experimental results show that localized compression results in classification accuracy approximately 1 to 2 percent higher than is achieved by downgrading to the equivalent resolution.

15. SUBJECT TERMS
artificial neural networks, computer vision, compression algorithms, image classification

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Stephen Hopp
Unclassified	Unclassified	Unclassified	SAR	8	19b. TELEPHONE NUMBER *(Include Area Code)* N/A

Localized Compression: Applying Convolutional Neural Networks to Compressed Images

Christopher A. George *Member, IEEE* and Bradley M. West

***Abstract*—We address the challenge of applying existing convolutional neural network (CNN) architectures to compressed images. Existing CNN architectures represent images as a matrix of pixel intensities with a specified dimension; this desired dimension is achieved by downgrading or cropping. Downgrading and cropping are attractive in that the result is also an image; however, an algorithm producing an alternative "compressed" representation could yield better classification performance. This compression algorithm need not be reversible, but must be compatible with the CNN's operations. This problem is thus the counterpart of the well-studied problem of applying compressed CNNs to uncompressed images, which has attracted great interest as CNNs are deployed to size-, weight-, and power- (SWaP)-limited devices. In this brief, we introduce *Localized Compression*, a generalization of downgrading in which the original image is divided into blocks and each block is compressed to a smaller size using either sampling- or random-matrix-based techniques. By aligning the size of the compressed blocks with the size of the CNN's convolutional region, localized compression can be made compatible with any CNN architecture. Our experimental results show that Localized Compression results in classification accuracy approximately 1-2% higher than is achieved by downgrading to the equivalent resolution.**

***Index Terms*—Artificial neural networks, computer vision, compression algorithms, image classification**

I. Introduction

Convolutional Neural Networks (CNNs) [1] are the state-of-the-art technique for image classification, routinely achieving better-than-human performance. New CNN architectures and applications continue to emerge at a prodigious rate. More recently, substantial interest has arisen in compressing neural networks, including CNNs, to use fewer parameters and to require less memory so as to enable running on devices with limited size, weight, and power (SWaP). Note, "compression" in this context refers to reducing these computational and memory requirements while minimizing the effect on classification accuracy; this does not necessarily require that the compression can be reversed.

Compressing the network, however, addresses only one side of the coin: what about compressing the *images* to which the CNN is applied? Though images are often stored in compressed form, CNN architectures currently uncompress all images prior to classifying them. Being able to compress the images also presents an additional advantage: given a dataset of large images and a network that expects small images, such a compression algorithm may preserve more information than extant techniques such as downgrading (DG) or cropping. Thus, this brief presents a compression algorithm that reduces the images' size on disk and does not require (or even allow) the images to be uncompressed prior to being classified by the CNN.

To understand why extant compression algorithms are inadequate, we we must consider how the CNN ingests the original $l \times w \times c$ image. The first layer of a CNN begins by ingesting a small $r \times r \times c$ "convolutional region" from the top-left of the image (the value of r is set by the CNN architecture). After processing this area, the convolutional region "strides" (is translated) s pixels to the right and the process repeats; in this way, the convolutional region "convolves" left-to-right, top-to-bottom across the matrix of pixel intensities (see Figure 1a). Thus, any effective compression scheme must preserve the localization, such that nearby pixels generally correspond to semantically coherent information. It is this requirement that existing techniques, such as JPEG compression, fail to meet.

In response, we propose "Localized Compression" (LC). Rather than compressing the image as a whole, we divide the original image into $m \times m$ blocks and compress each block to $n \times n$ (with $n < m$). This reduces the number of pixels in the compressed image by a factor of n^2/m^2. While there are no restrictions on m, we require n to be chosen such that s is divisible by n; this ensures that each convolutional region receives the compresed pixels in the same relative order. This is illustrated in Figure 1b.

In principal, we could use standard compression techniques like JPEG compression or Principal Component Analysis (PCA) to compress these $m \times m$ blocks to $n \times n$ blocks. In practice, however, most modern CNN architectures have very small values of s, such as 4 (AlexNet), or even 2 (DenseNet). We must therefore compress already small blocks (e.g., $m = 7$) into much smaller blocks (e.g., $n = s = 2$). This rules out many well-established (reversible) compression techniques, including PCA and JPEG compression.

Instead, we consider two solutions that are compatible with such small sizes: random matrix multiplication (RMM) and percentile-based sampling. RMM entails multiplying the original $m \times m$ matrix by random matrices of the appropriate dimensions; this has proven effective on related problems [2], [3], [4]. Percentile-based sampling techniques entail retaining the maximum, minimum, and other values from the original matrix.

We evaluate LC and compare these different options using to two standard datasets, ImageNet [5] and the German Traffic Sign Recognition Benchmark [6], and two different CNN architectures, AlexNet [7] and DenseNet [8]. Both LC and DG produce images of the same size and therefore offer the same

C. A. George and B. West are with Boston Fusion Corp., 70 Westview Street, Suite 100, Lexington, MA, 02421 (e-mail: alex.george@bostonfusion.com; bradley.west@bostonfusion.com)

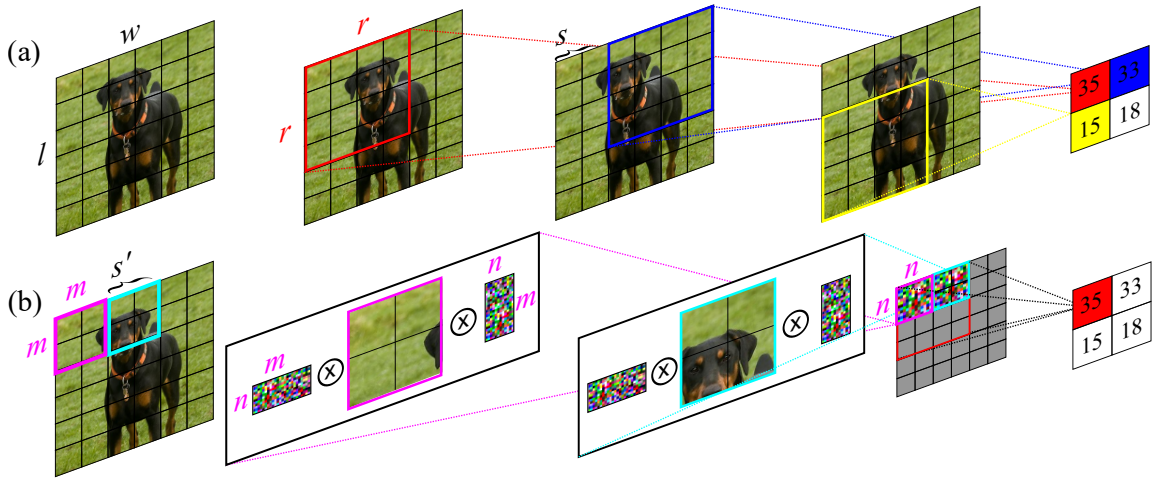

Fig. 1. Illustration of Localized Compression performed with Matrix Sketching

reduction in storage and processing requirements; therefore, we will compare them in terms of classification accuracy. Our results show that LC with percentile-based sampling is approximately 2% more accurate than DG when $m \gg n$.

The remainder of this brief is organized as follows. Section II describes related work. Section III provides more detail on random-matrix-based, sampling-based, and other techniques to represent an $m \times m$ matrix with an $n \times n$ matrix, while Section IV formally defines using these techniques for LC. Section V shows numerical results from applying LC to standard datasets with existing network architectures. Section VI contains a brief digression in which we apply random matrices to the related problem of compressing fully-connected layers. Finally, we draw conclusions in Section VII.

II. Related Work

To our knowledge, there is no past work that addresses applying CNNs to compressed images (i.e., without immediately uncompressing each image). There is, however, much related work about compressing the CNN itself, and about compressing the inputs to other types of classifiers.

With respect to compressing the CNN itself, research has focused on four key areas: (1) reducing the number of network parameters by pruning and sharing, (2) using low-rank factorization to compress the network weights, (3) representing convolutional filters as transformations of a small number of base filters, and (4) transferring the essential knowledge from a deep network to a shallower network (knowledge distillation) [9], [10]. Of these, using low-rank factorization to compress the network weights is most similar to our paradigm. In particular, Denton *et al.* [11] showed that performing tensor decompositions (based on the singular value decomposition) on trained convolutional layers can significantly accelerate CNNs with minimal loss in classification accuracy. While we also consider extensions of the singular value decomposition (Section III), our work is different in that we compress the inputs to the CNN (images) rather than the CNN itself.

Compressing inputs to simple classifiers (single-layer perceptrons) has also been thoroughly studied. In particular, Wimalajeewa and Varshney [2] recently considered sparse random matrices to compress the input to simple classifiers (nearest neighbor classifiers, random forests, and support vector machines), and Wójcik *et al.* [4] studied compressing high-dimensional vector input (e.g., telemetry data) to deep, fully-connected neural networks. While we also consider random matrices, we apply them to CNNs rather than simple classifiers or fully-connected neural networks.

Section VI discusses using random matrices to compress a CNN's fully-connected layers. Here, there is substantial related work: in particular, Cheng *et al.* [12] has shown that circulant projection matrices (a subset of random projection matrices) efficiently compress fully-connected layers in CNNs, and Wójcik *et al.* [4] considers random projection matrices for the same purpose in fully-connected (non-convolutional) neural networks. Our work on the fully-connected layers fills in the gap, using random, non-circulant matrices to compress fully-connected layers in CNNs.

III. Theory

We begin by considering how to compress two-dimensional $m \times m$ blocks to $n \times n$ with $n < m$ (in this work, we treat each channel separately). We refer to the $m \times m$ block as b_i. Here, we consider five options.

Downgrading entails taking a weighted average or interpolation of neighboring pixels. This technique is already widely used: raw images are typically down- or up-sampled (as well as reshaped or cropped) to a standard size prior to applying the CNN. We do not consider DG as a form of LC since downgraded images are $l \times w \times c$ matrices of uncompressed pixels just like the uncompressed images. Rather, in this work, we use DG (as implemented in OpenCV's INTER_AREA algorithm [13]) as the baseline against which LC is compared.

Principal Component Analysis [14] is a widely-used compression procedure in which an image is approximated as a linear combination of its P principal eigenvectors. We apply PCA to b_i and store the resulting parameters in an $n \times n$ matrix, padding with zeros as needed. We must therefore choose P

such that the number of PCA parameters does not exceed n^2. Concretely, we must require that:

$$n \geq \sqrt{2mP + m} \tag{1}$$

Though PCA has been widely studied, it offers two disadvantages. First, its computational complexity is very high, as eigenvectors must be calculated for each block. Second, Equation 1 implies that PCA is simply incompatible with some dimensionalities. For example, it is impossible to represent even a single principal component of any $m \times m$ block in an $n \times n$ block if $m > 4$ and $n \leq 4$. For these reasons, we do not consider PCA further in this work.

Percentiles. With percentile-based sampling, we sort the m^2 uncompressed points by their intensity and then sample from this distribution at pre-determined percentile values (e.g., the minimum, 33rd percentile, 67th percentile, and maximum). The computational complexity of this compression technique is somewhat high, as each block's m^2 values must be sorted.

Random Matrix Multiplication (RMM). Some recent work in compressive sensing [2] has looked at performing dimensionality reduction as a precursor to classification by multiplying the original features on the left by a (sparse) random matrix. In our case, we define $\vec{b_i}$ as the vectorized form of b_i. We then fill an $n^2 \times m^2$ matrix, $\mathcal{M}$, with values randomly drawn according to:

$$\mathcal{M}_{ij} \sim \begin{cases} 0 & \text{with probability } 1-\gamma \\ \mathcal{N}(0, 1/|\mathcal{M}|) & \text{with probability } \gamma \end{cases} \tag{2}$$

where $0 \leq \gamma \leq 1$ (lower values of γ are more efficient; in this work, we set $\gamma = 1$). We then perform dimensionality reduction according to $\vec{b_i} \rightarrow \mathcal{M}\vec{b_i}$. We then reshape the resulting vector to $n \times n$.

Random Matrix Sketching (MS). Similar to RMM, MS fills an $n \times m$ matrix according to equation (2) (we again set $\gamma = 1$) and then compresses b_i according to $b_i \rightarrow \mathcal{M} b_i \mathcal{M}^T$. This smaller dimensionality further reduces the compression technique's computational complexity.

IV. Method

Localized Compression entails using the techniques in Section III to compress entire $l \times w \times c$ images. We refer to the entire uncompressed image as x_i. We begin by defining $m \times m$ blocks over x_i (with $m \ll l, w$). We compress each channel separately and so consider only two dimensions here. In principal, these $m \times m$ blocks can be offset from one another by any number of pixels (i.e., an uncompressed pixel can be in zero, one, or multiple $m \times m$ blocks), but for simplicity, we consider only blocks that completely tile the image with no overlap (i.e., each uncompressed pixel is in exactly one $m \times m$ block).

Algorithms 1 and 2 formally define LC for single-channel images (multi-channel images simply apply the compression operation to each channel separately). Algorithm 1 ("inline mode") is a proof-of-concept in which the compression is performed at runtime: that is, we simply run the CNN as normal, inserting a step wherein we apply the compression operation to each $m \times m$ block and then apply the normal convolutional operation to the resulting $n \times n$ block. This is conceptually straightforward, but offers little or no savings in terms of storage efficiency (the uncompressed images must be stored) or computational efficiency (the reduction in learned convolutional parameters is roughly offset by the addition of compression operations). Algorithm 2 ("default mode") makes some adjustments such that the compression is performed prior to runtime. Default mode achieves the same storage and computational efficiency as DG; however, this introduces some complications with respect to data augmentation (described below).

Algorithm 1 ("inline mode") begins by resizing ($\mathcal{R}$) each image to $\ell \times w$ and writing these images to disk. We then cycle through the images as normal. For each image, we use the data augmentation operations ($\mathcal{A}$) with randomly-drawn parameters to modify each image: these operations may include cropping to $M \times M$, taking a left-right flip, or any other data augmentation strategy. We then locally-compress each image ($\mathcal{C}$) and classify it using the CNN ($\mathcal{N}$). Note, when compressing with random matrices, we use the same random matrix for each image.

Algorithm 2 ("default mode") differs in that it writes to disk *after* performing the data augmentation and localized compression. It also requires that s be divisible by n so that the convolutional region will always stride over an integer number of compressed blocks. In this way, only the compressed images are written to disk (reducing the storage requirement), and the compression operations must only be performed once (reducing the computational requirement). The challenge with this ordering is that after compression, the full suite of data augmentation techniques can no longer be used; instead, only some limited set of data augmentation techniques ($\mathcal{A}_{lim}$) can be applied. In particular:

- Crops. It is customary to take the final $M \times M$ crop during data augmentation (i.e., after resizing the image to $\ell \times w$). In default mode, this is still possible, however, the crops must not be allowed to sub-divide the $n \times n$ blocks.
- Flips. It is customary to take left-right flips of the image during data augmentation. In default mode, it is still possible to reverse the ordering of the $n \times n$ blocks; however, the internal structure of each block must not be changed.

Other data augmentation schemes may or may not be applicable post-compression.

We therefore expect that networks trained in default mode will be somewhat less accurate than networks trained in inline mode. To bridge this gap, we allow default mode to produce c copies of each image. These c copies are produced using the full suite of data augmentation techniques ($\mathcal{A}$); at runtime, we randomly select one of these c images and then apply the limited set of data augmentation techniques ($\mathcal{A}_{lim}$) to achieve further augmentation. We therefore expect that increasing c will increase our classification accuracy, but will also increase our storage requirements.

Algorithm 1 Classification with LC (inline mode)

Input: Image set X, network architecture $\mathcal{N}$
Output: Label set $L = \ell_1, \ldots, \ell_Z$
Hyperparameters: l, w, m, n, $nEpochs$, M

for $x_i \in X$ **do**
  $x_i \leftarrow \mathcal{R}(l \times w)x_i$.
end for
Write X.
for $e \in nEpochs$ **do**
  for $x_i \in X$ **do**
    $x_i \leftarrow \mathcal{A}(M \times M)x_i$
    for $i \in (0, l/m)$ **do**
      for $j \in (0, w/m)$ **do**
        $x_c[i \cdot n : (i+1) \cdot n, j \cdot n : (j+1) \cdot n]$ $\mathcal{C}x_i[i \cdot m : (i+1) \cdot m, j \cdot m : (j+1) \cdot m]$
      end for
    end for
    $\ell \leftarrow \mathcal{N}x_c$
  end for
end for
return L

Algorithm 2 Classification with LC (default mode)

Input: Image set X, network architecture $\mathcal{N}$
Output: Label set $L = \ell_1, \ldots, \ell_Z$
Hyperparameters: l, w, m, n, $nEpochs$, c, M

for $x_i \in X$ **do**
  for $c_i \in c$ **do**
    $x_i \leftarrow \mathcal{R}(l \times w)x_i$.
    $x_i \leftarrow \mathcal{A}(M \times M)x_i$.
    for $i \in (0, l/m)$ **do**
      for $j \in (0, w/m)$ **do**
        $x_c[i \cdot n : (i+1) \cdot n, j \cdot n : (j+1) \cdot n]$ $\mathcal{C}x_i[i \cdot m : (i+1) \cdot m, j \cdot m : (j+1) \cdot m]$
      end for
    end for
  end for
end for
Write X.
for $e \in nEpochs$ **do**
  for $x_i \in X$ **do**
    x_i $\mathcal{A}_{lim}(M \times M)x_i$
    $\ell \leftarrow \mathcal{N}x_i$
  end for
end for
return L

V. Numerical Experiments

We test our procedure on two network architectures and two datasets. Our architectures are AlexNet [7] and DenseNet [8]: AlexNet is a dated architecture that has been widely used to evaluate compression algorithms, while DenseNet is a more modern architecture that achieves considerably higher accuracy. Both architectures require input images of a uniform size; we take 224×224 as the reference size for all images. Our datasets are the German Traffic Signs Recognition Benchmark (GTSRB) [6] (39K training images over 37 classes) and ImageNet [5] 2012 (1.3M training images over 1000 classes). While these are both standard datasets for classification challenges, a key difference is that most ImageNet images are larger than the reference size, whereas most GTSRB images are smaller than the reference size. We expect that LC will be more effective on large images (as there is more information to exploit).

We base our implementation of the networks, including parameters such as weight decay, on those from TensorFlow Slim [15], [16]. In all tests (except where indicated), we begin by resizing and reshaping the images to 256×256 (e.g., $l = w = 256$), cropping a random 224×224 patch from this (i.e., $M = 224$), and then performing a left-right flip at random. All evaluation is performed with a single center crop and no left-right flip. We train all networks with the Momentum Optimizer with momentum 0.9 and a learning rate that begins at 0.01 and is reduced by an order of magnitude every 20 epochs, for a total of 65 epochs. This simple scheme is fully network-agnostic and offers relatively fast training times while giving top-1 accuracies only slightly lower than those reported by the network authors.

Our first test compare the percentile, RMM, and MS compression algorithms (as described in Section III) against the baseline of simply downgrading the images to the equivalent size. We use inline mode to allow all algorithms to use identical, off-the-shelf dataset augmentation techniques (as described in Section IV). We set $m = 7$ and $n = 2$; thus, the final images are 64×64. Note, these small images sizes require removing the last pooling layer from AlexNet. Our results, shown in Table I, illustrate that LC is viable: all methods give results within a few percent of the baseline (DG), and the percentile method gives an accuracy 1-2% higher than DG. We therefore perform LC with the percentile-based compression technique in the remainder of this work.

Our second test validates default mode. In particular, we compare default mode's accuracy with various of c against the accuracy achieved by inline mode. In addition to classification accuracy, Table II also shows the uncompressed-to-compressed storage ratio (SR) and the uncompressed-to-compressed computational ratio (CR). The results show that LC is still viable in default mode: even with $c = 1$, LC remains more accurate than DG. Setting $c = 2$ further increases the classification accuracy; however, continuing to increase c shows only a modest improvement in classification accuracy. In the remainder of this work, we use default mode with $c = 2$.

Third, we test LC for different compression ratios. In particular, we keep $n = 2$ and vary the size of m; larger values of m therefore result in smaller compressed image sizes. Our results for LC and DG are given in Table III, and show that LC consistently outperforms DG for significant compression ratios (i.e., reducing the number of pixels by more than a factor of 4), but advantage is less clear for smaller compression ratios.

Finally, we consider applying LC to larger images: rather than beginning with 224×224 and compressing, we begin with large 784×784 images and compress to 224×224. To evaluate this, we consider only the ImageNet dataset, and select only those images with more than 784^2 pixels (of which there are

TABLE I
TEST 1 RESULTS: LOCALIZED COMPRESSION (INLINE MODE) ACCURACY VS. COMPRESSION ALGORITHM

		ImageNet		GTSRB	
Method	Size	AlexNet	DenseNet	AlexNet	DenseNet
Uncompressed	224×224	56.8%	68.7%	96.8%	94.9%
Downgraded	64×64	27.0%	47.3%	92.8%	92.9%
Percentiles	64×64	29.4%	47.9%	94.3%	93.8%
RMM	64×64	26.4%	45.7%	93.1%	94.0%
MS	64×64	26.4%	42.0%	93.4%	93.4%

TABLE II
TEST 2 RESULTS: LOCALIZED COMPRESSION (DEFAULT MODE) ACCURACY VS. c

			ImageNet		GTSRB	
c	CR	SR	AlexNet	DenseNet	AlexNet	DenseNet
1	12.25x	12.25x	28.4%	47.4%	94.5%	94.4%
2	12.25x	6.125x	28.9%	47.9%	94.1%	93.5%
4	12.25x	3.06x	29.0%	48.0%	94.4%	93.7%
inline	1x	1x	29.4%	47.9%	94.3%	93.8%

TABLE III
TEST 3 RESULTS: LOCALIZED COMPRESSION (DEFAULT MODE) ACCURACY VS. COMPRESSION RATIO

	ImageNet				GTSRB			
	AlexNet		DenseNet		AlexNet		DenseNet	
Compression Ratio	LC	DG	LC	DG	LC	DG	LC	DG
$8 \times 8 \to 2 \times 2$	27.6%	25.6%	42.7%	42.3%	94.0%	93.2%	93.7%	93.5%
$7 \times 7 \to 2 \times 2$	28.9%	27.0%	47.9%	47.3%	94.1%	92.8%	93.5%	92.9%
$6 \times 6 \to 2 \times 2$	31.8%	30.7%	50.2%	50.2%	94.0%	94.1%	94.0%	93.3%
$5 \times 5 \to 2 \times 2$	37.0%	36.1%	53.0%	53.2%	94.9%	94.4%	94.7%	93.7%
$4 \times 4 \to 2 \times 2$	44.2%	45.4%	58.1%	61.2%	95.5%	95.0%	94.3%	94.2%
$3 \times 3 \to 2 \times 2$	52.5%	52.2%	62.6%	63.4%	96.0%	97.0%	94.3%	94.0%

TABLE IV
TEST 4 RESULTS: LOCALIZED COMPRESSION VS. DOWNSAMPLING ON 784×784 IMAGENET IMAGES

	AlexNet	DenseNet
DG	16.7%	15.5%
LC	17.2%	17.8%

30,192 for training and 1,110 for testing). Our results are given in Table IV. Though the CNNs are clearly data starved, our results suggest that LC gives higher accuracy than DG for larger images just as it did for smaller images.

VI. APPLYING RANDOM MATRICES TO FULLY-CONNECTED LAYERS

We now take a brief digression to consider a related problem: using the techniques of Section III to compress the fully-connected layers inside the CNN itself. As discussed in Section II, substantial work has been put into compressing these fully-connected layers in deep neural networks; our contribution is to extend this work by applying static, non-circulant random matrices to CNNs. Our strategy is to introduce a new deterministic layer immediately prior to each fully-connected layer that compresses the inputs to the following layer. The percentile-based sampling method, though effective for LC, is not an appropriate choice for this layer, as it would continually reorder the features. We therefore select the MS-based technique for this layer; this efficiently and dramatically reduces the number of weights in the hidden layer.

TABLE V
RESULTS: MS COMPRESSION OF EACH FC LAYER IN ALEXNET

CR by Layer						
FC1	FC2	FC3	nNodes	GTSRB	ImageNet	CR
1.00	1.00	1.00	4096	96.8%	58.4%	1.00
0.50	1.00	1.00	4096	96.5%	57.5%	0.59
0.50	0.50	0.50	4096	96.0%	57.8%	0.55
0.50	0.50	0.50	2048	98.2%	54.3%	0.28

As an numerical example, we again consider the AlexNet architecture, which contains three fully-connected layers, the first of which contains 6400 weights. We then reshape these weights to 25×256 and multiply on the left by a 13×25 random matrix. This produces 3328 weights, which we feed into the next fully-connected layer. We repeat this for each fully-connected layers. We can also reduce the number of nodes in each FC layer to compensate for the reduced number of inputs.

Table V shows our results applying this to the GTSRB and ImageNet datasets with the AlexNet network architecture (we did not consider DenseNet, as it contains only a single fully-connected layer). Table V reports the compression ratio for each of the three fully-connected layers (FC1, FC2, and FC3), and the number of nodes contained in each FC layer (nNodes). Compared to the uncompressed AlexNet, we observe a 1% increase in accuracy on GSTRB when compressing the network up to a 72% and less than 1% decrease in accuracy when tested using ImageNet compressing the network by 45%. The counter-intuitive increase in accuracy for GTSRB may be

because the original images are so small that the large network has too many parameters relative to the images' information content.

VII. Conclusions

The primary contribution of this work is to introduce Localized Compression (LC), an alternative to downgrading when CNNs require an image size much smaller than the original image's resolution. In some sense, LC is a generalization of downgrading: downgrading always performs some sort of pixel averaging and requires $n = 1$, whereas LC supports different compression techniques and different values of n. The most successful compression technique, percentile-based sampling, could be viewed as applying a generalization of a pooling layer to the original image. By choosing n such that n divides r, LC supports any network architecture.

We also extended previous work [2], [4] on applying sparse random matrices to deep neural networks. Though percentile-based sampling outperformed random-matrix-based techniques on LC, we showed that sparse random matrices are an effective way to compress both convolutional and fully-connected layers in CNNs.

With respect to LC, our results show that when it is used with percentile-based sampling and relatively high compression ratios, LC gives a 1-2% accuracy improvement over downgrading. LC can therefore be useful in applications where the average image size is much larger than a CNN's reference size. This is potentially a useful capability: many modern cameras can produce high-resolution images; LC provides a way to exploit the extra information that such devices provide without increasing the computational or SWaP requirements.

Acknowledgment

The authors would like to acknowledge Profs. Pramod Varshney and Thakshila Wimalajeewa at Syracuse University for sharing their expertise with using random matrices for classification.

This work was supported by the Air Force Research Laboratory under contract FA8650-17-C-1154.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[2] T. Wimalajeewa and P. K. Varshney, "Impact of very sparse random projections on compressed classification," *Publication forthcoming.*, 2018.

[3] T. Wimalajeewa, Y. C. Eldar, and P. K. Varshney, "Recovery of Sparse Matrices via Matrix Sketching," *ArXiv e-prints*, Nov. 2013.

[4] P. I. Wójcik and M. Kurdziel, "Training neural networks on high-dimensional data using random projection," *Pattern Analysis and Applications*, Mar 2018. [Online]. Available: https://doi.org/10.1007/s10044-018-0697-0

[5] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, June 2009, pp. 248–255.

[6] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, no. 0, pp. –, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608012000457

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, p. 2012.

[8] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993

[9] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015. [Online]. Available: http://arxiv.org/abs/1510.00149

[10] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *CoRR*, vol. abs/1710.09282, 2017. [Online]. Available: http://arxiv.org/abs/1710.09282

[11] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation," *ArXiv e-prints*, Apr. 2014.

[12] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S. F. Chang, "An exploration of parameter redundancy in deep networks with circulant projections," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, no. 1, pp. 2857–2865, 2015.

[13] W. Dong. (2018) What is opencv's inter_area actually doing? [Online]. Available: https://medium.com/@wenrudong/what-is-opencvs-inter-area-actually-doing-282a626a09b3

[14] J. Shlens, "A tutorial on principal component analysis," *CoRR*, vol. abs/1404.1100, 2014. [Online]. Available: http://arxiv.org/abs/1404.1100

[15] S. Guadarrama and N. Silberman, "Tensorflow-slim: a lightweight library for defining, training and evaluating complex models in tensorflow," https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/slim, 2016.

[16] pudae. (2018) Tensorflow-densenet. [Online]. Available: https://github.com/pudae/tensorflow-densenet/blob/master/nets/densenet.py