Generalized Depthwise-Separable Convolutions for Adversarially Robust and Efficient Neural Networks

Hassan Dbouk & Naresh R. Shanbhag
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{hdbouk2,shanbhag}@illinois.edu

Abstract

Despite their tremendous successes, convolutional neural networks (CNNs) incur high computational/storage costs and are vulnerable to adversarial perturbations. Recent works on robust model compression address these challenges by combining model compression techniques with adversarial training. But these methods are unable to improve throughput (frames-per-second) on real-life hardware while simultaneously preserving robustness to adversarial perturbations. To overcome this problem, we propose the method of Generalized Depthwise-Separable (GDWS) convolution – an efficient, universal, post-training approximation of a standard 2D convolution. GDWS dramatically improves the throughput of a standard pre-trained network on real-life hardware while preserving its robustness. Lastly, GDWS is scalable to large problem sizes since it operates on pre-trained models and doesn’t require any additional training. We establish the optimality of GDWS as a 2D convolution approximator and present exact algorithms for constructing optimal GDWS convolutions under complexity and error constraints. We demonstrate the effectiveness of GDWS via extensive experiments on CIFAR-10, SVHN, and ImageNet datasets. Our code can be found at https://github.com/hsndbk4/GDWS.

1 Introduction

Nearly a decade of research after the release of AlexNet [18] in 2012, convolutional neural networks (CNNs) have unequivocally established themselves as the de facto classification algorithm for various machine learning tasks [11, 38, 4]. The tremendous success of CNNs is often attributed to their unrivaled ability to extract correlations from large volumes of data, allowing them to surpass human level accuracy on some tasks such as image classification [11].

Today, the deployment of CNNs in safety-critical Edge applications is hindered due to their high computational costs [11, 30, 31] and their vulnerability to adversarial samples [37, 5, 16]. Traditionally, those two problems have been addressed in isolation. Recently, very few bodies of works [19, 35, 42, 6, 34, 7] have addressed the daunting task of designing both efficient and robust CNNs. A majority of these methods focus on model compression, i.e. reducing the storage requirements of CNNs. None have demonstrated their real-time benefits in hardware. For instance, Fig. 1a shows recent robust pruning works HYDRA [34] and ADMM [42] achieve high compression ratios (up to 97×) but either fail to achieve high throughput measured in frames-per-second (FPS) or compromise significantly on robustness. Furthermore, the overreliance of current robust complexity reduction techniques on adversarial training (AT) [45, 21] increases their training time significantly (Fig. 1b). This prohibits their application to complex ImageNet scale problems with stronger attack models, such as union of norm-bounded perturbations [22]. Thus, there is critical need for methods to
design deep nets that are both adversarially robust and achieve high throughput when mapped to real hardware.

To address this need, we propose Generalized Depthwise-Separable (GDWS) convolutions, a universal post-training approximation of a standard 2D convolution that dramatically improves the real hardware FPS of pre-trained networks (Fig. 1a) while preserving their robust accuracy. Interestingly, we find GDWS applied to un-pruned robust networks simultaneously achieves higher FPS and higher robustness than robust pruned models obtained from current methods. This in spite of GDWS’s compression ratio being smaller than those obtained from robust pruning methods. Furthermore, GDWS easily scales to large problem sizes since it operates on pre-trained models and doesn’t require any additional training.

Our contributions:

1. We propose GDWS, a novel convolutional structure that can be seamlessly mapped onto off-the-shelf hardware and accelerate pre-trained CNNs significantly while maintaining robust accuracy.
2. We show that the error-optimal and complexity-optimal GDWS approximations of any pre-trained standard 2D convolution can be obtained via greedy polynomial time algorithms, thus eliminating the need for any expensive training.
3. We apply GDWS to a variety of networks on CIFAR-10, SVHN, and ImageNet to simultaneously achieve higher robustness and higher FPS than existing robust complexity reduction techniques, while incurring no extra training cost.
4. We demonstrate the versatility of GDWS by using it to design efficient CNNs that are robust to union of \( \ell_\infty, \ell_2, \ell_1 \) perturbation models. To the best of our knowledge, this is the first work that proposes efficient and robust networks to the union of norm-bounded perturbation models.

2 Background and Related Work

The problem of designing efficient and robust CNNs, though crucial for safety-critical Edge applications, is not yet well understood. Very few recent works have addressed this problem [19, 35, 42, 6, 34, 7]. We cluster prior works into the following categories:
Quantization Reducing the complexity of CNNs via model quantization in the absence of any adversary is a well studied problem in the deep learning literature [30, 31, 2, 44, 14, 26, 3]. The role of quantization on adversarial robustness was studied in Defensive Quantization (DQ) [19] where it was observed that conventional post-training fixed-point quantization makes networks more vulnerable to adversarial perturbations than their full-precision counterparts. EMPIR [35] also leverages extreme model quantization (up to 2-bits) to build an ensemble of efficient and robust networks. However, [40] broke EMPIR by constructing attacks that fully leverage the model structure, i.e., adaptive attacks. In contrast, GDWS is an orthogonal complexity reduction technique that preserves the base model’s adversarial robustness and can be applied in conjunction with model quantization.

Pruning The goal of pruning is to compress neural networks by zeroing out unimportant weights [10, 8, 46, 41]. The structured pruning method in [42] combines the alternating direction method of multipliers (ADMM) [46] for parameter pruning within the AT framework [21] to design pruned and robust networks. The flexibility of ADMM enables it to achieve a high FPS on Jetson (as seen in Fig. 1a) but suffers from a significant drop in robustness. ATMC [6] augments the ADMM framework [42] with model quantization and matrix factorization to further boost the compression ratio. On the other hand, unstructured pruning methods such as HYDRA [34] prunes models via important score optimization [25]. However, HYDRA’s high pruning ratios (> 90%) doesn’t translate into real-time FPS improvements on off-the-shelf hardware and often requires custom hardware design to fully leverage their capabilities [9]. GDWS is complementary to unstructured pruning methods, e.g., when applied to HYDRA, GDWS boosts the achievable FPS and achieves much higher robustness at iso-FPS when compared to structured (filter) pruning ADMM.

Neural Architecture Search Resource-efficient CNNs can be designed by exploiting design intuitions such as depthwise separable (DWS) convolutions [12, 32, 13, 47, 15, 38]. While neural architecture search (NAS) [48, 27] automates the process, it requires massive compute resources, e.g., thousands of GPU hours for a single network. Differentiable NAS [20] and one-shot NAS [1] drastically reduce the cost of this search. In [7], a one-shot NAS framework [1] is combined with the AT framework [21] to search for robust network architectures, called RobNets. RobNets achieve slightly higher robustness than existing networks with less storage requirements. In this work, we show that applying GDWS to existing architectures, e.g., WideResNet-28-4, achieves significantly higher FPS than RobNet, at iso-robustness and model size.

3 Generalized Depthwise-Separable Convolutions

In this section, we introduce GDWS convolutions and develop error-optimal and complexity-optimal GDWS approximations of standard 2D convolution. These optimal approximations are then employed to construct GDWS networks from any pre-trained robust CNN built from standard 2D convolutions.

Notation: A (C, K, M) standard 2D convolution operates on an input feature map \( X \in \mathbb{R}^{C \times H \times W} \) via \( M \) filters (also referred to as kernels or output channels) each consisting of \( C' \) channels each of dimension \( K \times K \) to generate an output feature map \( Y \in \mathbb{R}^{M \times H' \times W'} \).

2D Convolution as Matrix Multiplication: The \( M \) filters can be viewed as vectors \( \{w_i\}_{i=1}^{M} \in \mathbb{R}^{C K^2} \) obtained by vectorizing the \( K^2 \) elements within a channel and then across the \( C \) channels. The resulting weight matrix \( W \in \mathbb{R}^{M \times CK^2} \) is constructed by stacking these filter vectors, i.e., \( W = [w_1 | w_2 | ... | w_M]^T \).

From an operational viewpoint, the matrix \( W \) can be used to compute the 2D convolution via Matrix Multiplication (MM) with the input matrix \( X = \Psi(X) \in \mathbb{R}^{CK^2 \times H'W'} \):

\[
Y = WX = W\Psi(X) \tag{1}
\]

where \( \Psi \) is an unrolling operator that generates all \( H'W' \) input feature map slices and stacks them in matrix format. The resultant output matrix \( Y \in \mathbb{R}^{M \times H'W'} \) can be reshaped via the operator \( \Phi \) to retrieve \( Y = \Phi(Y) \). The computational complexity of \( (1) \) in terms of multiply-accumulate (MAC) operations is given by:

\[
H'W'MCK^2 \tag{2}
\]

The reshaping operators \( \Phi \) and \( \Psi \) are only for notational convenience and are computation-free.
3.1 GDWS Formulation

**Definition:** A \((C, K, g, M)\) GDWS convolution is parameterized by the channel distribution vector \(g \in \mathbb{Z}_+^C\) in addition to the parameters \((C, K, M)\) of a standard 2D convolution. A GDWS convolution is composed of a \((C, K, g)\) Generalized Depthwise (GDW) convolution and a \((G, 1, M)\) standard pointwise (PW) convolution where \(G = \sum g_c\) with \(c \in [C]\).

A \((C, K, g)\) GDW convolutional layer (Fig. 2) operates on an input feature map \(X \in \mathbb{R}^{C \times H \times W}\) by convolving the \(c\)th channel with \(g_c \in \mathbb{Z}_+^+\) depthwise \(K \times K\) filters to produce a total of \(G\) intermediate output channels each of size \(H' \times W'\). The \((G, 1, M)\) PW layer operates on the intermediate output feature map by convolving it with \(M\) filters of size \(1 \times 1\), thus producing the output feature map \(Y \in \mathbb{R}^{M \times H' \times W'}\).

**Relation to DWS:** Setting \(g_c = 1 \forall c \in [C]\) reduces the GDWS convolution to the standard DWS convolution popularized by [12]. Thus, GDWS generalizes DWS by allowing for more than one \((g_c \geq 1)\) depthwise filters per channel. This simple generalization relaxes DWS’s highly constrained structure enabling accurate approximations of the 2D convolution. Thus, GDWS when applied to pre-trained models preserves its original behavior and therefore its natural and robust accuracy. Furthermore, GDWS achieves high throughput since it exploits the same hardware features that enable networks with DWS to be implemented efficiently. One might ask: why not use DWS on pre-trained models? Doing so will result in very high approximation errors. In fact, in Section 4.2, we show that applying GDWS to a pre-trained complex network such as ResNet-18 achieves better robust accuracy than MobileNet trained from scratch, while achieving similar FPS.

**GDWS Complexity:** The total number of MAC operations required by GDWS convolutions is:

\[
H'W'( \sum_{c=1}^{C} g_c(K^2 + M) ) = H'W'G(K^2 + M)
\] (3)

Thus, replacing standard 2D convolutions with GDWS convolutions results in a complexity reduction by a factor of \(\frac{G(K^2 + M)}{CK^2M}\).

3.2 Properties of GDWS Convolutions

We present properties of the GDWS weight matrix \(W\) that will be vital for developing the optimal approximation procedures.

**Property 1.** The weight matrix \(W \in \mathbb{R}^{M \times CK^2}\) of a \((C, K, g, M)\) GDWS convolution can be expressed as:

\[
W = W_P W_D
\] (4)

where \(W_P \in \mathbb{R}^{M \times G}\) and \(W_D \in \mathbb{R}^{G \times CK^2}\) are the weight matrices of the PW and GDW convolutions, respectively.

Property 1 implies that any GDWS convolution has an equivalent 2D convolution whose weight matrix is the product of \(W_P\) and \(W_D\), where \(W_P\) is a regular convolution weight matrix with \(K = 1\) and \(W_D\) has the following property:

\[^1\text{we use the notation } [C] = \{1, 2, ..., C\} \text{ for brevity.}\]
Property 2. The weight matrix $W_D \in \mathbb{R}^{G \times CK^2}$ of a $(C, K, g)$ GDW convolution has a block-diagonal structure. Specifically, $W_D$ is a concatenation of $C$ sub-matrices where each sub-matrix $W_{D,c} \in \mathbb{R}^{G \times K^2}$ has at most $g_c$ non-zero consecutive rows, starting at row index $1 + \sum_{k=1}^{c-1} g_k$ as shown in Fig. 3.

This structure is due to the fact that input channels are convolved independently with at most $g_c$ depthwise filters per channel. Finally, let $W = [W_1 | W_2 | ... | W_C]$ be represented as the concatenation of $C$ sub-matrices, then combining Properties 1 & 2 establishes the following lemma:

Lemma 1. The weight matrix of a $(C, K, g, M)$ GDWS convolution can be expressed as the concatenation of $C$ sub-matrices $W_c$ where $\text{rank}(W_c) \leq \min(g_c, K^2)$ for all $c \in [C]$.

The reason for this is that each sub-matrix $W_c$ can be expressed as the sum of $g_c$ rank 1 matrices of size $M \times K^2$. A detailed proof of Lemma 1 can be found in the Appendix. A major implication of Lemma 1 is that any 2D standard convolution is equivalent to a GDWS convolution with $g_c = K^2$ for all $c$. Hence, in the rest of this paper we will assume $g_c \leq K^2$ when we approximate 2D convolutions with GDWS.

3.3 Optimal GDWS Approximation Methods

We wish to approximate a standard 2D convolution with weight matrix $W \in \mathbb{R}^{M \times CK^2}$ with a GDWS convolution with weight matrix $Q \in \mathbb{R}^{M \times CK^2}$ to minimize the weighted approximation error defined as:

$$e(W, Q, \alpha) = \sqrt{\sum_{c=1}^{C} \alpha_c ||W_c - Q_c||_F^2}$$

(5)

where $||.||_F$ denotes the Frobenius norm of a matrix and $\alpha \in \mathbb{R}_+^C$ is a vector of positive weights. Setting $\alpha_c = 1 \forall c$ simplifies (5) to the Frobenius norm of the error matrix $W - Q$. Furthermore, from (3), one can upper bound the complexity of the GDWS approximation $Q$ via an upper bound on $G = \sum g_c$ where the $g_c$'s are obtained from Lemma 1.

Based on the GDWS properties and Lemma 1, we state the following error-optimal approximation theorem:

Theorem 1. Given a $(C, K, M)$ standard 2D convolution with weight matrix $W$, the $(C, K, g, M)$ GDWS approximation with weight matrix $\hat{W}$ that minimizes the error in (5) subject to $\sum g_c = G \leq \gamma$ (for some $\gamma \in \mathbb{Z}_+$), can be obtained in polynomial time via Algorithm 1.

That is:

$$\hat{W} = \arg \min_{Q: G \leq \gamma} e(W, Q, \alpha)$$

(6)

can be solved for any weight error vector $\alpha \in \mathbb{R}_+^C$ in polynomial time. While Theorem 1 shows that the optimal GDWS approximation under a complexity constraint can be solved efficiently, a similar result can be obtained for the reverse setting shown next.

Typical CNNs satisfy $K^2 < M$, e.g., $K = 3$ and $M \geq 16$.
Theorem 2. Given a \((C, K, M)\) standard 2D convolution with weight matrix \(W\), the \((C, K, g, M)\) GDWS approximation with weight matrix \(\hat{W}\) that minimizes the complexity in (3) subject to 
\[ e(W, Q, \alpha) \leq \beta \] (for some \(\beta \geq 0\)), can be constructed in polynomial time via Algorithm 2.

That is:
\[
\hat{W} = \arg \min_{Q, \alpha} \frac{C}{e(W, Q, \alpha) \leq \beta} \sum_{c=1}^{C} g_c
\]

can be solved for any weight error vector \(\alpha \in \mathbb{R}_C^r\) in polynomial time. Proofs of Theorems 1 & 2 can be found in the Appendix.

Algorithm 1: (MEGO) Minimum Error Complexity-constrained GDWS Optimal Approximation

**Input:** A \((C, K, M)\) convolution \(W\), weight error vector \(\alpha\), and constraint \(\gamma \in \mathbb{Z}_+\).

**Output:** A \((C, K, g, M)\) GDWS convolution \(W\), satisfying \(\sum g_c \leq \gamma\).

1. Compute SVDs of \(W_c = \sum_{i=1}^{r_c} \sigma_i e_i u_i v_i^T\)
2. Initialize \(g = 0\)
3. while \(\sum g_c < \gamma\) do
   4. \(c' = \arg \max_c \alpha_c \sigma_{g_c+1} \quad \text{// } g_c < r_c\)
   5. \(g_c \leftarrow g_c + 1\)
6. Compute \(W_c\) via truncated SVD of \(W_c\) with rank \(g_c\):
   \(W_c = \sum_{i=1}^{g_c} \sigma_i e_i u_i v_i^T\)
7. Construct \(W = [W_1|\ldots|W_C]\)

Algorithm 2: (LEGO) Least Complex Error-constrained GDWS Optimal Approximation

**Input:** A \((C, K, M)\) convolution \(W\), weight error vector \(\alpha\), and constraint \(\beta \geq 0\).

**Output:** A \((C, K, g, M)\) GDWS convolution \(W\), satisfying \(e \leq \beta\).

1. Compute SVDs of \(W_c = \sum_{i=1}^{r_c} \sigma_i e_i u_i v_i^T\)
2. Initialize \(g_c = r_c\), \(b = 0\),
   \(c' = \arg \min \alpha_c \sigma_c^2, \quad h = \alpha_c \sigma_c^2\)
3. while \(b + h < \beta\) do
   4. \(b \leftarrow b + h\) and \(g_c' \leftarrow g_c' - 1\)
   5. \(c' = \arg \min \alpha_c \sigma_c^2, \quad h = \alpha_c \sigma_c^2\)
6. Compute \(\hat{W}_c\) via truncated SVD of \(W_c\) with rank \(g_c\):
   \(\hat{W}_c = \sum_{i=1}^{g_c} \sigma_i e_i u_i v_i^T\)
7. Construct \(\hat{W} = [\hat{W}_1|\ldots|\hat{W}_C]\)

3.4 Constructing GDWS Networks

When confronted with a CNN with \(D\) convolutional layers, the question arises: How to assign resources (in terms of complexity) amongst the \(L\) layers such that the robustness of the CNN is minimally compromised? To answer this question, we compute per-layer weight error vectors \(\alpha_l\), such that the computed error in (5) weighs how different sub-matrices affect the final output of the CNN.

Let \(f: \mathbb{R}^D \rightarrow \mathbb{R}^N\) be a pre-trained CNN for an \(N\)-way classification problem with \(L\) convolutional layers parameterized by weight matrices \(W^{(l)} \in \mathbb{R}^{M_l \times C_l K_l^2}\). The CNN \(f\) operates on a \(D\)-dimensional input vector \(x\) to produce a vector \(z = f(x)\) of soft outputs or logits. Denote by \(n_x \in [N]\) the predicted class label associated with \(x\), and define \(\delta_{x,j} = z_j - z_{n_x}\) to be the soft output differences \(\forall j \in [N] \setminus \{n_x\}\).

Inspired by [30, 31], we propose a simple yet effective method for computing the per-layer weight error vectors as follows:

\[
\alpha_{c,l} = \frac{1}{M_l K_l^2} \mathbb{E} \sum_{j=1}^{N} \frac{||D^{(c,l)}_{x,j}||^2}{2 \sigma_{x,j}^2} \quad \forall l \in [L], \forall c \in [C_l]
\]

(8)

where \(D^{(c,l)}_{x,j} \in \mathbb{R}^{M_l \times K_l^2}\) is the derivative of \(\delta_{x,j}\) w.r.t. the sub-matrix \(W^{(l)}_{c,j}\). The expectation is taken over the input vector \(x\). Equation (8) can be thought of as the expected noise gain from a particular channel in a particular layer to the network output required to flip its decision. The Appendix provides a detailed rationale underlying (8).

Computation of (8) can be simplified by obtaining an estimate of the mean over a small batch of inputs sampled from the training set and by leveraging software frameworks such as PyTorch.
Algorithm 3: Constructing GDWS networks

**Input:** CNN $f$ with convolutional layers $\{W^{(l)}\}$, $\{\alpha_l\}$ computed via (8), and constraint $\beta \geq 0$.

**Output:** CNN $\hat{f}$ with GDWS convolutions $\{\hat{W}^{(l)}\}$

1. $\hat{f} = f$ // Initialize $\hat{f}$
2. for $l \in \{1, \ldots, L\}$ do
   3. $\hat{W}^{(l)} = \text{LEGO}(W^{(l)}, \alpha_l, \beta)$ // solve via Algorithm 2
   4. Decompose $\hat{W}^{(l)}$ into GDW and PW convolutions via Property 1 and Lemma 1
   5. Replace the $l$th convolution layer in $\hat{f}$ with GDW and PW convolutions

4 Experiments

4.1 Evaluation Setup

We measure the throughput in FPS by mapping the networks onto an NVIDIA Jetson Xavier via native PyTorch [24] commands. We experiment with VGG-16 [36], ResNet-18 [11], ResNet-50, and WideResNet-28-4 [43] network architectures, and report both natural accuracy ($A_{\text{nat}}$) and robust accuracy ($A_{\text{rob}}$). Following standard procedure, we report $A_{\text{rob}}$ against $\ell_\infty$ bounded perturbations generated via PGD [21] with standard attack strengths: $\epsilon = 8/255$ with PGD-100 for both CIFAR-10 [17] and SVHN [23] datasets, and $\epsilon = 4/255$ with PGD-50 for the ImageNet [29] dataset. Section 4.3 studies union of multiple perturbation models ($\ell_\infty, \ell_2, \ell_1$). In the absence of publicly released pre-trained models, we establish strong baselines using AT [21] following the approach of [28] which utilizes early stopping to avoid robust over-fitting. Details on the training/evaluation setup can be found in the Appendix.

Table 1: Comparison between RobNet [7] and GDWS on the CIFAR-10 dataset. GDWS is applied to standard pre-trained models.

<table>
<thead>
<tr>
<th>Models</th>
<th>$A_{\text{nat}}$ [%]</th>
<th>$A_{\text{rob}}$ [%]</th>
<th>Size [MB]</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>RobNet [7]</td>
<td>82.72</td>
<td>52.23</td>
<td>20.8</td>
<td>5</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>84.21</td>
<td>53.05</td>
<td>89.7</td>
<td>16</td>
</tr>
<tr>
<td>+ GDWS ($\beta = 0.001$)</td>
<td>83.72</td>
<td>52.94</td>
<td>81.9</td>
<td>37</td>
</tr>
<tr>
<td>WRN-28-4</td>
<td>84.00</td>
<td>51.80</td>
<td>22.3</td>
<td>17</td>
</tr>
<tr>
<td>+ GDWS ($\beta = 1 \times 10^{-5}$)</td>
<td>83.37</td>
<td>51.70</td>
<td>18.9</td>
<td>65</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>82.41</td>
<td>51.55</td>
<td>42.6</td>
<td>28</td>
</tr>
<tr>
<td>+ GDWS ($\beta = 0.005$)</td>
<td>81.17</td>
<td>50.98</td>
<td>29.1</td>
<td>104</td>
</tr>
<tr>
<td>VGG-16</td>
<td>77.49</td>
<td>48.92</td>
<td>56.2</td>
<td>36</td>
</tr>
<tr>
<td>+ GDWS ($\beta = 0.25$)</td>
<td>77.17</td>
<td>49.56</td>
<td>28.7</td>
<td>129</td>
</tr>
</tbody>
</table>

4.2 Results

Ablation Study: We first show the effectiveness of GDWS on the CIFAR-10 datasets using four network architectures. Table 1 summarizes $A_{\text{nat}}$ and $A_{\text{rob}}$ as well as FPS and model size. It is clear that GDWS networks preserve robustness as both $A_{\text{nat}}$ and $A_{\text{rob}}$ are always within $\sim 1\%$ of their respective baselines. The striking conclusion is that in spite of GDWS offering modest reductions in model size, it drastically improves the FPS of the base network across diverse architectures. For instance, a ResNet-18 utilizing GDWS convolutions is able to run at 104 FPS compared to the baseline’s 28 FPS ($>250\%$ improvement) without additional training and without compromising on robust accuracy. In the Appendix, we explore the benefits of applying GDWS using both Algorithms 1 & 2, providing more detailed results on CIFAR-10 and show that similar gains are observed with SVHN dataset.

---

[24] For CIFAR-10 and SVHN, we use the standard pre-activation version of ResNets.
GDWS vs. RobNet: In Table 1, we also compare GDWS networks (obtained from standard networks) with a publicly available pre-trained RobNet model, the robust network architecture designed via the NAS framework in [7]. Note that RobNet utilizes DWS convolutions which precludes the use of GDWS. However, despite the efficiency of DWS convolutions in RobNet, its irregular cell structure leads to extremely poor mapping on the Jetson as seen by its low 5 FPS. For reference, a standard WideResNet-28-4 (WRN-28-4) runs at 17 FPS with similar robustness and model size. Applying GDWS to the WideResNet-28-4 further increases the throughput to 65 FPS which is a 1200% improvement compared to RobNet while maintaining robustness. This further supports our assertion that model compression alone does not lead to enhanced performance on real hardware.

GDWS vs. Lightweight Networks: A natural question that might arise from this work: why not train lightweight networks utilizing DWS convolutions from scratch instead of approximating pre-trained complex networks with GDWS? In Table 2, we compare the performance of GDWS networks (obtained from Table 1) vs. standard lightweight networks: MobileNetV1 [12], MobileNetV2 [32], and ResNet-20 [11], as well as a DWS-version of the standard ResNet-18 trained from scratch on the CIFAR-10 dataset. We find that applying GDWS to a pre-trained complex network such as ResNet-18 achieves better $A_{nat}$ and $A_{rob}$ than all lightweight networks, while achieving DWS-like FPS and requiring no extra training despite offering modest reductions in model size. The only benefit of using lightweight networks is the much smaller model size compared to GDWS networks.

GDWS vs. Structured Pruning: In Table 3, we compare GDWS with the robust structured pruning method ADMM [42] on CIFAR-10, using two networks: VGG-16 and ResNet-18. Due to the lack of publicly available pre-trained models, we use their released code to reproduce both the AT baselines, and the corresponding pruned models at different pruning ratios. The nature of structured pruning allows ADMM pruned networks ($p \geq 50\%$) to achieve both high compression ratios and significant improvement in FPS over their un-pruned baselines but at the expense of robustness and accuracy. For instance, a ResNet-18 with 75% of its channels pruned results in a massive 7% (4%) drop in $A_{nat}$ ($A_{rob}$) compared to the baseline even though it achieves a 160% improvement in FPS. In contrast, a post-training application of GDWS to the same ResNet-18 baseline results in a massive 275% improvement in FPS while preserving both $A_{nat}$ and $A_{rob}$ within 1% of their baseline values. Thus, despite achieving modest compression ratios compared to ADMM, GDWS achieves comparable improvements in FPS without compromising robustness.

GDWS vs. Unstructured Pruning: We compare GDWS with HYDRA [34] which is an unstructured robust pruning method. On both CIFAR-10 and ImageNet datasets. We use the publicly released HYDRA models as well as their AT baselines, and apply GDWS to both the un-pruned and pruned models. Table 4 summarizes the robustness and FPS of HYDRA and GDWS networks on CIFAR-10. HYDRA pruned models have arbitrarily sparse weight matrices that cannot be leveraged by off-the-shelf hardware platforms immediately. Instead, we rely on the extremely high sparsity (99%) of these matrices to emulate channel pruning whereby channels are discarded only if all filter weights are zero.

<table>
<thead>
<tr>
<th>Models</th>
<th>$A_{nat}$ [%]</th>
<th>$A_{rob}$ [%]</th>
<th>Size [MB]</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18 + GDWS</td>
<td>81.17</td>
<td>50.98</td>
<td>29.1</td>
<td>104</td>
</tr>
<tr>
<td>VGG-16 + GDWS</td>
<td>77.17</td>
<td>49.56</td>
<td>28.7</td>
<td>129</td>
</tr>
<tr>
<td>MobileNetV1</td>
<td>79.92</td>
<td>49.08</td>
<td>12.3</td>
<td>125</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>79.59</td>
<td>48.55</td>
<td>8.5</td>
<td>70</td>
</tr>
<tr>
<td>ResNet-18 (DWS)</td>
<td>80.12</td>
<td>48.52</td>
<td>5.5</td>
<td>120</td>
</tr>
<tr>
<td>ResNet-20</td>
<td>74.82</td>
<td>47.00</td>
<td>6.4</td>
<td>125</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Models</th>
<th>$A_{nat}$ [%]</th>
<th>$A_{rob}$ [%]</th>
<th>Size [MB]</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16 (AT from [42])</td>
<td>77.45</td>
<td>45.78</td>
<td>56.2</td>
<td>36</td>
</tr>
<tr>
<td>+ GDWS ($\beta = 0.5$)</td>
<td><strong>76.40</strong></td>
<td><strong>46.28</strong></td>
<td>38.8</td>
<td><strong>119</strong></td>
</tr>
<tr>
<td>VGG-16 ($p = 25%$)</td>
<td>77.88</td>
<td>43.80</td>
<td>31.6</td>
<td>26</td>
</tr>
<tr>
<td>VGG-16 ($p = 50%$)</td>
<td>75.33</td>
<td>42.93</td>
<td>14.0</td>
<td>113</td>
</tr>
<tr>
<td>VGG-16 ($p = 75%$)</td>
<td>70.39</td>
<td>41.07</td>
<td>3.5</td>
<td>174</td>
</tr>
<tr>
<td>ResNet-18 (AT from [42])</td>
<td>80.65</td>
<td>47.05</td>
<td>42.6</td>
<td>28</td>
</tr>
<tr>
<td>+ GDWS ($\beta = 0.75$)</td>
<td><strong>79.13</strong></td>
<td><strong>46.15</strong></td>
<td>30.4</td>
<td><strong>105</strong></td>
</tr>
<tr>
<td>ResNet-18 ($p = 25%$)</td>
<td>81.61</td>
<td>42.67</td>
<td>32.1</td>
<td>31</td>
</tr>
<tr>
<td>ResNet-18 ($p = 50%$)</td>
<td>79.42</td>
<td>42.23</td>
<td>21.7</td>
<td>60</td>
</tr>
<tr>
<td>ResNet-18 ($p = 75%$)</td>
<td>74.62</td>
<td>43.23</td>
<td>11.2</td>
<td>74</td>
</tr>
</tbody>
</table>
This explains why, despite their high compression ratios, HYDRA models do not achieve significant improvements in FPS compared to their baselines.

For instance, a 99% HYDRA pruned WideResNet model achieves a massive ~100× compression ratio and improves the FPS from 17 to 28, but suffers from a large ~10% drop in both $A_{\text{nat}}$ and $A_{\text{rob}}$. In contrast, GDWS applied to the same un-pruned baseline preserves robustness and achieves significantly better throughput of 68 FPS, even though the model size reduction is negligible. Interestingly, we find that applying GDWS directly to HYDRA pruned models results in networks with high compression ratios with no robustness degradation and massive improvements in FPS compared to the pruned baseline. For example, applying GDWS to the same 99% HYDRA pruned WideResNet achieves a ~20× compression ratio and improves the throughput from 28 FPS to 68 FPS while preserving $A_{\text{nat}}$ and $A_{\text{rob}}$ of the pruned baseline. This synergy between HYDRA and GDWS is due to the fact that highly sparse convolution weight matrices are more likely to have low-rank and sparse sub-matrices. This implies that, using Lemma 1, sparse convolutions can be transformed to sparse GDWS versions with negligible approximation error. We explore this synergy in detail in the Appendix. Table 5 shows that GDWS benefits also show up in ImageNet using ResNet-50.

<table>
<thead>
<tr>
<th>Models</th>
<th>$\xi_{\text{nat}}$ (%)</th>
<th>$\xi_{\text{rob}}$ (%)</th>
<th>Size [MB]</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16 (AT from [34]) + GDWS ($\beta = 0.5$)</td>
<td>82.72</td>
<td>51.93</td>
<td>58.4</td>
<td>36</td>
</tr>
<tr>
<td>WRN-28-4 (AT from [34]) + GDWS ($\beta = 1$)</td>
<td>85.35</td>
<td>57.23</td>
<td>22.3</td>
<td>17</td>
</tr>
<tr>
<td>VGG-16 (p = 90%) + GDWS ($\beta = 0.1$)</td>
<td>80.54</td>
<td>49.44</td>
<td>5.9</td>
<td>36</td>
</tr>
<tr>
<td>WRN-28-4 (p = 90%) + GDWS ($\beta = 0.125$)</td>
<td>83.69</td>
<td>55.20</td>
<td>2.3</td>
<td>17</td>
</tr>
<tr>
<td>VGG-16 (p = 95%) + GDWS ($\beta = 0.005$)</td>
<td>82.68</td>
<td>54.18</td>
<td>1.1</td>
<td>17</td>
</tr>
<tr>
<td>WRN-28-4 (p = 99%) + GDWS ($\beta = 0.0025$)</td>
<td>75.62</td>
<td>47.21</td>
<td>0.2</td>
<td>28</td>
</tr>
</tbody>
</table>

Table 4: Comparison between HYDRA [34] and GDWS using VGG-16 and WRN-28-4 on CIFAR-10.

Table 5: Comparison between HYDRA [34] and GDWS using ResNet-50 on ImageNet.

<table>
<thead>
<tr>
<th>Models</th>
<th>top-1 / 5 $A_{\text{nat}}$ (%)</th>
<th>top-1 / 5 $A_{\text{rob}}$ (%)</th>
<th>Size [MB]</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50 (AT from [34]) + GDWS ($\beta = 50$)</td>
<td>60.25 / 82.39</td>
<td>31.94 / 61.13</td>
<td>97.5</td>
<td>15</td>
</tr>
<tr>
<td>ResNet-50 (p = 95%) + GDWS ($\beta = 0.5$)</td>
<td>44.60 / 70.12</td>
<td>19.53 / 44.28</td>
<td>5.1</td>
<td>15</td>
</tr>
<tr>
<td>ResNet-50 (p = 99%) + GDWS ($\beta = 0.5$)</td>
<td>27.68 / 52.55</td>
<td>11.32 / 28.83</td>
<td>1.2</td>
<td>17</td>
</tr>
</tbody>
</table>

4.3 Defending against Union of Perturbation Models

Recent work has shown that adversarial training with a single perturbation model leads to classifiers vulnerable to the union of $(\ell_\infty, \ell_2, \ell_1)$-bounded perturbations [33, 39, 22]. The method of multi-steepest descent (MSD) [22] achieves state-of-the-art union robust accuracy ($A_{\text{rob}}^U$) against the union of $(\ell_\infty, \ell_2, \ell_1)$-bounded perturbations. We demonstrate the versatility of GDWS by applying it to a publicly available [22] robust pre(MSD)-trained ResNet-18 model on CIFAR-10. Following the setup in [22], all attacks were run on a subset of the first 1000 test images with 10 random restarts with the following attack configurations: $\epsilon_{\infty} = 0.03$ with PGD-100, $\epsilon_2 = 0.5$ with PGD-500, and $\epsilon_1 = 12$ with PGD-100. Table 6 shows that applying GDWS with $\beta = 0.01$ to the pre-trained ResNet-18 incurs a negligible ($<1\%$) drop in $A_{\text{nat}}$ and $A_{\text{rob}}^U$ while improving the throughput from 28 FPS to 101 FPS ($>250\%$ improvement).
Table 6: Benefits of GDWS when evaluated against union of perturbation models on CIFAR-10. $A^U_{\text{rob}}$ is the fraction of test images that are simultaneously resistant to all perturbation models.

<table>
<thead>
<tr>
<th>Models</th>
<th>$A_{\text{nat}}$ [%]</th>
<th>$A_{\text{rob}}$ [%]</th>
<th>$A^1_{\text{rob}}$ [%]</th>
<th>$A^2_{\text{rob}}$ [%]</th>
<th>$A^U_{\text{rob}}$ [%]</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18 (AT from [22])</td>
<td>81.74</td>
<td>47.50</td>
<td>53.60</td>
<td>66.10</td>
<td>46.10</td>
<td>28</td>
</tr>
<tr>
<td>+ GDWS ($\beta = 0.0025$)</td>
<td>81.67</td>
<td>47.60</td>
<td>53.60</td>
<td>66.00</td>
<td>46.30</td>
<td>87</td>
</tr>
<tr>
<td>+ GDWS ($\beta = 0.005$)</td>
<td>81.43</td>
<td>47.30</td>
<td>52.60</td>
<td>65.60</td>
<td>45.70</td>
<td>92</td>
</tr>
<tr>
<td>+ GDWS ($\beta = 0.01$)</td>
<td><strong>81.10</strong></td>
<td><strong>47.20</strong></td>
<td><strong>52.20</strong></td>
<td><strong>65.00</strong></td>
<td><strong>45.20</strong></td>
<td><strong>101</strong></td>
</tr>
</tbody>
</table>

5 Discussion

We have established that the proposed GDWS convolutions are universal and efficient approximations of standard 2D convolutions that are able to accelerate any pre-trained CNN utilizing standard 2D convolution while preserving its accuracy and robustness. This facilitates the deployment of CNNs in safety critical edge applications where real-time decision making is crucial and robustness cannot be compromised. One limitation of this work is that GDWS alone does not achieve high compression ratios compared to pruning. Combining unstructured pruning with GDWS alleviates this problem to some extent. Furthermore, GDWS cannot be applied to CNNs utilizing DWS convolutions, such as RobNet for instance. An interesting question is to explore the possibility of training GDWS-structured networks from scratch. Another possible direction is fine-tuning post GDWS approximation to recover robustness, which we explore in the Appendix.

In summary, a GDWS approximated network inherits all the properties, e.g., accuracy, robustness, compression and others, of the baseline CNN while significantly enhancing its throughput (FPS) on real hardware. Therefore, the societal impact of GDWS approximated networks are also inherited from those of the baseline CNNs.

Acknowledgments

This work was supported by the Center for Brain-Inspired Computing (C-BRIC) and the Artificial Intelligence Hardware (AIHW) program funded by the Semiconductor Research Corporation (SRC) and the Defense Advanced Research Projects Agency (DARPA).

References


12


