AutoShuffleNet: Learning Permutation Matrices via an Exact Lipschitz Continuous Penalty in Deep Convolutional Neural Networks

Jiancheng Lyu, Shuai Zhang, Yingyong Qi
Qualcomm AI Research
jianlyu,shuazhan,yingyong@qti.qualcomm.com

Jack Xin
University of California, Irvine
jack.xin@uci.edu

1 INTRODUCTION

Light convolutional deep neural networks (LCNN) are attractive in resource limited conditions for delivering high performance at low costs. Some of the state-of-the-art LCNNs in computer vision are ShuffleNet [14, 23], IGC (Interleaved Group Convolutions, [22]) and IGCV3 (Interleaved Low-Rank Group Convolutions, [17]). A noticeable feature in the design is the presence of permutations for channel shuffling in between separable convolutions. The permutations are hand-crafted by designers outside of network training however. A natural question is whether the permutations can be learned like network weights so that they are optimized based on training data. An immediate difficulty is that unlike weights, permutations are highly discrete and incompatible with the stochastic gradient descent (SGD) methodology that is continuous in nature. To overcome this challenge, we introduce an exact Lipschitz continuous non-convex penalty so that it can be incorporated in SGD to approximate permutation at high precision and low overhead. Consequently, exact permutations are obtained by simple rounding at the end of network training with negligible drop of classification accuracy. To be specific, we shall work with ShuffleNet architecture [14, 23]. Our approach extends readily to other LCNNs.

Related Work. Permutation optimization is a long standing problem arising in operations research, graph matching among other applications [3, 8]. Well-known examples are linear and quadratic assignment problems [18]. Graph matching is a special case of quadratic assignment problem, and can be formulated over $N \times N$ permutation matrices $\mathcal{P}^N$ as:

$$\min_{\pi \in \mathcal{P}^N} \| A - \pi B \pi^T \|_F,$$

where $A$ and $B$ are the adjacency matrices of graphs with $N$ vertices, and $\| \cdot \|_F$ is the Frobenius norm. A popular way to handle $\mathcal{P}^N$ is to relax it to the Birkhoff polytope $\mathcal{D}_N$, the convex hull of $\mathcal{P}^N$, leading to a convex relaxation. The explicit realization of $\mathcal{D}_N$ is the set of doubly stochastic matrices

$$\mathcal{D}_N = \{ M \in \mathbb{R}^{N \times N} : M1 = 1, M^T 1 = 1, M \succeq 0 \},$$

where $1 = (1, 1, \cdots, 1)^T \in \mathbb{R}^N$. An approximate yet simpler way to treat $\mathcal{D}_N$ is through the classical first order matrix scaling algorithm, e.g. the Sinkhorn method, see [16] and its recent applications [7, 15]. Though in principle such algorithm converges, the cost can be quite high when iterating many times, which causes a bottleneck effect [12]. A non-convex and more compact relaxation of $\mathcal{P}^N$ is by a sorting network [12] which maps the box $[0, 1]^N$ into a manifold that sits inside $\mathcal{D}_N$ and contains $\mathcal{P}^N$. Yet another method is path following algorithm [21] which seeks solutions under concave relaxations of $\mathcal{P}^N$ by solving a linear interpolation.
of convex-concave problems (starting from the convex relaxation). Permutation learning via continuous approximation has been studied in visual data recovery [4]. None of the existing relaxations are exact. In the context of improving ShuffleNet, HadaNet [24] uses Hadamard matrices \( H \) to define a class of structured convolution as the product \( H^T \times \) group convolution \( \times H \) and generalize shuffled group convolution of ShuffleNet. However, the inference cost of HadaNet is much higher than that of ShuffleNet, and relies on special hardware for speedup. Hadamard matrices are constructed to date for certain special orders such as powers of 2, and conjectured to exist for multiples of 4. In particular, they are not applicable to odd channel/group numbers.

**Contribution.** Our non-convex relaxation is a combination of matrix \( \ell_{1-2} \) penalty function and \( \mathcal{D}^N \). The \( \ell_{1-2} \) (the difference of \( \ell_1 \) and \( \ell_2 \) norms) has been proposed and found effective in selecting sparse vectors under nearly degenerate linear constraints [6, 20]. The matrix version is simply a sum of \( \ell_{1-2} \) over all row and column vectors. We prove that the penalty is zero when applied to a matrix in \( \mathcal{D}^N \) if and only if the matrix is a permutation matrix. Thanks to the \( \mathcal{D}^N \) constraint, the penalty function is Lipschitz continuous (almost everywhere differentiable). This allows the penalty to be integrated directly into SGD for learning permutation in LCNNs. As shown in our experiments on CIFAR-10, CIFAR-100 and ImageNet data sets, the closeness to \( \mathcal{D}^N \) turns out to be remarkably small at the end of network training so that a simple rounding has negligible effect on the validation accuracy. We also observed that convex relaxation by \( \mathcal{D}^N \) fails to capture good permutations for LCNNs. We observed experimentally that a random shuffle could perform better than manual shuffle, but the learned shuffle consistently achieved the best results. To our best knowledge, this is the first time permutations have been successfully learned for the architecture selection of deep CNNs to improve hand-crafted permutations. Moreover, our AutoShuffleNet preserves the inference cost of ShuffleNet for any channel/group numbers.

**Outline.** In section 2, we introduce exact permutation penalty, and prove its closeness to permutation matrices when the penalty values are small, as observed in the experiments. We also present the training algorithm combining thresholding and matrix scaling to approximate projection onto \( \mathcal{D}^N \) for SGD. In section 3, we analyze two permutation optimization problems to show the utility of our penalty. In a 2-layer neural network regression model with short cut (identity map), convex relaxation does not give the optimal permutation even with additional rounding while our penalty can. In section 4, we show experimental results on consistent improvement of auto-shuffle over hand-crafted shuffle on data from CIFAR-10, CIFAR-100 and ImageNet. Conclusion is in section 5.

## 2. Permutation, matrix \( \ell_{1-2} \) penalty and exact relaxation

The channel shuffle operation in ShuffleNet [14, 23] can be represented as multiplying the feature map in the channel dimension by a permutation matrix \( M \). The permutation matrix \( M \) is a square binary matrix with exactly one entry of one in each row and each column and zeros elsewhere. In the ShuffleNet architecture [14, 23], \( M \) is preset by the designers and will be called "manual". In this work, we propose to learn an automated permutation matrix \( M \) through network training, hence removing the human factor in its selection towards a more optimized shuffle. Since permutation is discrete in nature and too costly to enumerate, we propose to approach it by adding a matrix generalization of the \( \ell_{1-2} \) penalty [6, 20] to the network loss function in the stochastic gradient descent based training.

Specifically for \( M = (m_{ij}) \in \mathbb{R}^{N \times N} \), the proposed continuous matrix penalty function is

\[
P(M) := \sum_{i=1}^{N} \left( m_{ij} - \left( \sum_{j=1}^{N} m_{ij}^2 \right)^{1/2} \right) + \sum_{j=1}^{N} \left( \sum_{i=1}^{N} |m_{ij}| - \left( \sum_{i=1}^{N} m_{ij}^2 \right)^{1/2} \right),
\]

in conjunction with the doubly stochastic constraint:

\[
m_{ij} \geq 0, \quad \forall (i,j); \quad \sum_{i=1}^{N} m_{ij} = 1, \quad \forall j; \quad \sum_{j=1}^{N} m_{ij} = 1, \quad \forall i.
\]

**Remark 1.** When the constraints in (2) hold, \( \sum_{j=1}^{N} |m_{ij}| \) and \( \sum_{i=1}^{N} |m_{ij}| \) in \( P(M) \) can be removed. However, in actual computation, the two equality constraints of (2) only hold approximately, so the full expression in (1) is necessary.

**Remark 2.** Thanks to (2), we see that the penalty function \( P(M) \) is actually Lipschitz continuous in \( M \) as \( \sum_{j=1}^{N} m_{ij}^2 \neq 0, \forall i, \text{ and } \sum_{i=1}^{N} m_{ij}^2 \neq 0, \forall j. \) Although there are alternative penalties, we choose \( P(M) \) because it is simple, effective, and integrated well with SGD.

**Theorem 1.** A square matrix \( M \) is a permutation matrix if and only if \( P(M) = 0 \), and the doubly stochastic constraint (2) holds.

**Proof.** \((\Rightarrow)\) Since a permutation matrix consists of columns (rows) with exactly one entry of 1 and the rest being zeros, each term inside the outer sum of \( P(M) \) equals zero, and clearly (2) holds. \((\Leftarrow)\) By the elementary inequality,

\[
\left( \sum_{j=1}^{N} |m_{ij}| - \left( \sum_{j=1}^{N} m_{ij}^2 \right)^{1/2} \right) \geq 0, \quad \forall i,
\]

with equality if and only if the row-wise cardinality is 1:

\[
|\{j : m_{ij} \neq 0\}| = 1, \quad \forall i.
\]

This is because the mixed product terms like \( 2|m_{ij} m_{ij'}| (j \neq j') \) in \( \left( \sum_{j=1}^{N} |m_{ij}| \right)^2 \) must be all zero to match \( \sum_{j=1}^{N} m_{ij}^2 \). It only happens when equation (3) is true. Likewise,

\[
\left( \sum_{i=1}^{N} |m_{ij}| - \left( \sum_{i=1}^{N} m_{ij}^2 \right)^{1/2} \right) \geq 0, \quad \forall j,
\]

with equality if and only if

\[
|\{i : m_{ij} \neq 0\}| = 1, \quad \forall j.
\]

In view of (2), \( M \) is a permutation matrix.
The non-negative constraint in (2) is maintained throughout SGD by thresholding $m_{ij} \rightarrow \max(m_{ij}, 0)$. The normalization conditions in (2) are implemented sequentially once in one SGD iteration. Hence they are not strictly enforced. In theory, if the column/row normalization (divide each column/row by its sum) repeats sufficiently many times, the resulting matrices converge to (2), known as the Sinkhorn process [16]. We did not find much benefit to iterate more than once in terms of enhancing validation accuracy since the error in matrix scaling can be compensated in network weight adjustment during SGD.

The multiplication by $M$ can be embedded in the network as a $1 \times 1$ convolution layer with $M$ initialized as absolute value of a random Gaussian matrix. After each weight update, we threshold the weights to $\{0, \infty\}$, normalize rows to unit lengths, then repeat on columns. Let $L$ be the network loss function. The training minimizes the objective function:

$$ f = f(w, M) := L(w) + \lambda \sum_{j=1}^{J} P(M_j), $$

where $J$ is the total number of "channel shuffle", $M_j$'s abbreviated as $M$, $w$ is the network weight, $\lambda$ a positive parameter. The training algorithm is summarized in Alg. 1. Introducing those as $M$ and from (5) that $O$ is the network weight, $\lambda$ is differentiable away from zero, which is maintained in the Alg. 1

Proof. It follows from $P(M) = O(e)$ that

$$ \left( \sum_{j=1}^{N} |m_{ij}| \right)/\left( \sum_{j=1}^{N} m_{ij}^2 \right)^{1/2} = O(e), \forall i, $$

implying that:

$$ |m_{ij}| = O(e), \forall j \neq j', \forall i. $$

On the other hand for $\forall i$:

$$ \sum_{j=1}^{N} m_{ij} = 1 + O(e). $$

Let $j^* = \arg\max_{j'} |m_{ij'}|$, at any $i$. It follows from (6) that

$$ |m_{ij^*}| \geq 1/N + O(e), $$

and from (5) that

$$ m_{ij} = O(e), \forall j \neq j^*. $$

Hence each row of $M$ is $O(e)$ close to a unit coordinate vector, with one entry near 1 and the rest near 0. Similarly from

$$ \sum_{i=1}^{N} |m_{ij}| = \sum_{i=1}^{N} m_{ij}^{1/2} = O(e), \forall j, $$

and $\sum_{i=1}^{N} m_{ij} = 1 + O(e)$, we deduce that each column of $M$ is $O(e)$ close to a unit coordinate vector, with one entry near 1 and the rest near 0. Combining the two pieces of information above, we conclude that $M$ is $O(e)$ close to a permutation matrix. $\square$

The learned non-negative matrix $M$ will be called a relaxed shuffle and rounded to the nearest permutation matrix to produce a final auto shuffle. Relaxed shuffle usually has better performance before rounding but the auto shuffle is desirable since it preserves the shuffle structure of the original ShuffleNet without incurring extra computation. Strictly speaking, this "rounding" involves finding the orthogonal projection to the set of permutation matrices, a problem called the linear assignment problem (LAP), see [1] and references therein. The LAP can be formulated as a linear program over the doubly stochastic matrices or constraints (2), and is solvable in polynomial time [1]. As we shall see later in Table 5, the relaxed shuffle comes amazingly close to an exact permutation in network learning. It is unnecessary to solve LAP exactly, indeed a simple rounding will do. AutoShuffleNet units adapted from ShuffleNet v1 [23] and ShuffleNet v2 [14] are illustrated in Figs. 1-2.

---

**Algorithm 1 AutoShuffle Learning.**

**Input:**
- mini-batch loss function $f_\ell(w, M)$, $t$ being the iteration index;
- learning rate $\eta_t$ for $(w,M)$;
- penalty parameter $\lambda$ for $P$;
- total iteration number $T_n$.

**Start:**
- $w$: sample from unit Gaussian distribution;
- $M$: sample from unit Gaussian distribution then take absolute value.

**WHILE** $t < T_n$, **DO**:

1. Evaluate the mini-batch gradient $(\nabla_{w,f_\ell}, \nabla_{M,f_\ell})$ at $(w^t, M^t)$;
2. $w^{t+1} = w^t - \eta_t \nabla_{w,f_\ell}(w^t, M^t)$; // gradient update for weights
3. $M^{t+1} = M^t - \eta_t \nabla_{M,f_\ell}(w^t, M^t)$; // gradient update for $M$
4. $M^{t+1} \leftarrow \max(M^{t+1}, 0)$; // thresholding to enforce non-negativity constraint
5. normalize each column of $M^{t+1}$ by dividing the sum of entries in the column;
6. normalize each row of $M^{t+1}$ by dividing the sum of entries in the row.

**END WHILE**

**Output:** $w^{T_n}, M^{T_n}$; project each matrix $M^{T_n}$ inside $M^{T_n}$ to the nearest permutation matrix.
we present examples from permutation optimization to show the (1) can strengthen (2) to retrieve permutation matrices.

Let us recall the graph matching (GM) problem, see [1, 12, 13, 18, 21] and references therein. The goal is to align the vertices of two graphs to minimize the number of edge disagreements. Given a pair of $n$-vertex graphs $G_A$ and $G_B$, with respective adjacency $n \times n$ matrices $A$ and $B$, the GM problem is to find a permutation matrix $Q$ to minimize $\|AQ - QB\|^2_F$. Let $\Pi$ be the set of all permutation matrices, solve

$$Q^* := \arg\min_{Q \in \Pi} \|AQ - QB\|^2_F. \quad (7)$$

By algebraic identity

$$\|AQ - QB\|^2_F = \text{trace}((AQ - QB)^T(AQ - QB)) = \text{trace}(A^T A) + \text{trace}(B^T B) - 2\text{trace}(AQB^TQ^T),$$

the GM problem (7) is same as

$$Q^* = \arg\min_{Q \in \Pi} \text{trace}((-A)QB^TQ^T),$$

a quadratic assignment problem (QAP). The general QAP for two real square matrices $A$ and $B$ is [12, 18]:

$$Q^* = \arg\min_{Q \in \Pi} \text{trace}(AQB^TQ^T).$$

The convex relaxed GM is:

$$Q_s := \arg\min_{Q \in \mathcal{D}^N} \|AQ - QB\|^2_F.$$  

As an instance of general QAP, let us consider problem (7) in case $n = 2$ for two real matrices:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad B = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix}. $$

If $Q \in \mathcal{D}^2$, then:

$$Q = \begin{bmatrix} q & 1 - q \\ 1 - q & q \end{bmatrix}, \quad q \in [0, 1];$$

and

$$AQ - QB = \begin{bmatrix} (a - a')q + (b - c')q' & (b - b')q + (a - d')q' \\ (c - c')q + (d - a')q' & (d - d')q + (c - b')q' \end{bmatrix}.$$  

where $q^2 = 1 - q$.

**Example 1:** Let

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix},$$

$$AQ - QB = \begin{bmatrix} 2q - 1 & 0 \\ 1 - q & 1 - q \end{bmatrix},$$

$$\|AQ - QB\|^2_F = (2q - 1)^2 + 2(1 - q)^2 = 6q^2 - 8q + 3,$$

which is convex on $[0, 1]$ and has minimum at $q_s = 2/3$. The convex relaxed matrix solution is:

$$Q_s = \begin{bmatrix} 2/3 & 1/3 \\ 1/3 & 2/3 \end{bmatrix},$$

however, the permutation matrix solution $Q^*$ is the $2 \times 2$ identity matrix at $q = 1$.

In the spirit of objective function (4), let us minimize

$$\|AQ - QB\|^2_F + \lambda P(Q),$$

or equivalently minimize (after skipping additive constants in $P$)

$$F = F(q) := 6q^2 - 8q + 2 - 4\lambda(q^2 + (1 - q)^2)^{1/2}.$$

An illustration of $F$ is shown in Fig. 3. The minimal point moves from the interior of the interval $[0, 1]$ when $\lambda = 0.25$ (dashed line, top curve) to the end point 1 as $\lambda$ increases to 1 (line-star, middle curve) and remains there as $\lambda$ further increases to 2 (line-circle,
with a loop adds 2 to an adjacent matrix.

Then:

$$Q = \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. $$

So

$$Q = Q(1/2) \neq Q^* = Q(0) = Q(1). $$

Remark 3. We refer to [13] on certain correlated random Bernoulli graphs where $Q^* \neq Q$. On the other hand, there is a class of friendly
graphs where $Q^* = Q$. Existing techniques to improve convex relaxation on GM and QAP include approximate quadratic programming, sorting networks and path following based homotopy methods
[12, 18, 21]. Our proposed penalty (1)-(2) appears more direct and generic. A detailed comparison will be left for a future study.

Remark 4. In Example 1, if the convex relaxed $q_2 = 2/3$ is rounded up to 1, then $Q_s = Q^*$. In Example 2 (Fig. 4), the two interior minimal points at $\lambda = 1.8, 1.9$, after rounding down (up), become zero or one. So convex relaxation with the help of rounding happens to recover the exact permutation. We show in Example 3 below that convex relaxation still fails after rounding (to 1 if the number is above 1/2, to 0 if the number is below 1/2).

Example 3: We consider the two-layer neural network model with one hidden layer [11]. Given $m \geq 0$, the forward model is the following function:

$$f_m(x, W) = \| \phi((ml + W)x) \|_1,$$

where $\phi(x) = \max(x, 0)$ is the ReLU activation function, $x = (x_1, x_2) \in \mathbb{R}^2$ is the input random vector drawn from a probability distribution, $W \in \mathbb{R}^{2 \times 2}$ is the weight matrix, $I$ is the identity matrix. Consider a two-layer teacher network with $2 \times 2$ weight matrix

$$W^* = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad a, b, c, d \geq 0.$$
We train the student network with doubly stochastic constraint on $W$ using the $l_2$ loss:

$$L(W) = E_x \left[ f_m(x, W) - f_m(x, W^*) \right]^2.$$ 

Let $p \in [0, 1]$.

$$W = \begin{bmatrix} p & 1 - p \\ 1 - p & p \end{bmatrix}.$$ 

We write the loss function as

$$s, t \in \{=\} 2 \, E_G (\theta_1 (\lambda)) \, \cdot \, E_m (\phi (x_1 + d x_2))^2$$

$$\, \cdot \, \cdot \phi ( (m + p) x_1 + (1 - p) x_2)$$

$$+ \phi ((1 - p) x_1 + (m + p) x_2)$$

$$- \phi (a x_1 + b x_2) - \phi (c x_1 + d x_2)$$

$$= E_x \phi ((m + p) x_1 + (1 - p) x_2)^2$$

$$+ E_x \phi ((1 - p) x_1 + (m + p) x_2)^2$$

$$+ 2 E_x \phi ((m + p) x_1 + (1 - p) x_2)$$

$$- \phi ((1 - p) x_1 + (m + p) x_2)$$

$$- 2 G_m (p, a, b) - 2 G_m (p, c, d)$$

$$+ E_x \phi (a x_1 + b x_2) + \phi (c x_1 + d x_2)$$

$$\, (9)$$

where for $s, t \geq 0$, $G_m (p, s, t)$ is defined as

$$E_x \phi ((m + p) x_1 + (1 - p) x_2)$$

$$\, \cdot \, \phi (a x_1 + b x_2)$$

$$\, \cdot \, \phi (c x_1 + d x_2)$$

$$\, (9)$$

Define

$$I (q, r, s, t) := E_x [\phi (q x_1 + r x_2) \, \phi (s x_1 + t x_2)]$$

then

$$I (q, r, s, t) = I (s, t, q, r)$$

and

$$G_m (p, s, t) = I (m + p, 1 - p, s, t) + I (1 - p, m + p, s, t)$$

For simplicity, let $x$ obey uniform distribution on $[-1, 1]^2$. For

$$q t \geq r, \quad q + r > 0, \quad s + t > 0,$$

$$I (q, r, s, t)$$

$$\, (10)$$

We have

$$E_x \phi ((m + p) x_1 + (1 - p) x_2)^2$$

$$= E_x \phi ((1 - p) x_1 + (m + p) x_2)^2$$

$$= \frac{2}{3} \left[ (m + p)^2 + (1 - p)^2 \right]$$

$$E_x [\phi ((m + p) x_1 + (1 - p) x_2) \, \phi ((1 - p) x_1 + (m + p) x_2)]$$

$$= (m + p)^3 + (m + p)^4$$

$$3 \theta_m (p) + 12 \theta_m (p)^2.$$ 

$$\, (12)$$

where $\theta_m (p) := \max (m + p, 1 - p)$. The last term in (9) is a constant:

$$E_x \left(\phi (ax_1 + bx_2) + \phi (cx_1 + dx_2)\right)^2$$

$$= \frac{2}{3} \left( a^2 + b^2 + c^2 + d^2 \right) + 2 I (a, b, c, d).$$

(13)

Consider a special case when $a = 1/3$, $b = 2/3$, $c = 1/4$ and $d = 3/4$. By (11)-(13), the loss function $l_m (p)$ equals

$$\frac{2}{3} \left[(m + p)^2 + (1 - p)^2\right] + \frac{2}{3} \left[(m + 1)^3 + (m + 1)^4\right]$$

$$- 2 G_m (p, 1/3, 2/3) - 8113.2184.$$

Let

$$F_m (p) := l_m (p) - 4 \lambda \sqrt{p^2 + ((1 - p)^2}.$$ 

When $m = 0$, $\lambda = 0$, Fig. 5 (top) shows $l_0 (p)$ has minimal points in the interior of $(0, 1)$. A permutation matrix $W$ that minimizes $L (W)$ can be achieved by rounding the minimal points. However, when $m = 1$, $\lambda = 0$, (Fig. 5, bottom), rounding the interior minimal point of $l_1 (p)$ gives the wrong permutation matrix at $p = 1$. At $\lambda = 0.4$, the $P$ regularization selects the correct permutation matrix.

**Figure 5:** $F_m (p)$ $(m = 0 \, \text{top}, \, m = 1 \, \text{bottom})$ as penalty parameter $\lambda$ varies for the uniformly distributed input data on $[-1, 1]^2$.

**Remark 5.** If $x$ obeys the unit Gaussian distribution as in [11], the $F_m (p)$ functions are more complicated analytically, however their plots resemble those for uniformly distributed $x$, see Fig. 6.
4 EXPERIMENTS

We relax the shuffle units in ShuffleNet v1 [23] and ShuffleNet v2 [14] and perform experiments on CIFAR-10, CIFAR-100 [9] and a subset of 20 classes (Tab. 3) from ImageNet [5, 10] classification datasets. The 20-class data set consists of most common objects from ImageNet, and forms a typical setting for LCNN application. The accuracy results of auto shuffles are evaluated after the relaxed shuffles are rounded. There is no finetuning of weights after the rounding.

On CIFAR-10 and CIFAR-100 datasets, we set the $\ell_1 - \ell_2$ penalty parameter $\lambda = 10^{-3}$. All experiments are randomly initialized with learning rate linearly decaying from 0.2. We train each network for 200 epochs on CIFAR-10 and 300 epochs on CIFAR-100. We set weight-decay $10^{-4}$, momentum 0.95 and batch size 128. With $w$ and $M$ initialized from unit Gaussian distribution in Alg. 1, we never run into zero rows and columns in $M$. An explanation is that those degenerate cases are not generic to cause problems for SGD based training. In Tab. 1-2, we see that auto shuffle consistently improves on manual shuffle in v1 and v2 models, by as much as 2% on v1 (g=3). In ShuffleNet v2, $r$ is the fraction of channels that are fed into the right branch of the shuffle unit at Channel Split (Fig. 2). The smaller the $r$, the lighter the model, the more the auto shuffle improvement.

Table 1: CIFAR-10 validation accuracies.

<table>
<thead>
<tr>
<th>Network</th>
<th>v1 (g=8)</th>
<th>v1 (g=3)</th>
<th>v2 1x</th>
<th>v2 1.5x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual</td>
<td>90.06</td>
<td>90.55</td>
<td>91.90</td>
<td>92.56</td>
</tr>
<tr>
<td>Auto</td>
<td>91.26</td>
<td>91.76</td>
<td>92.81</td>
<td>93.22</td>
</tr>
</tbody>
</table>

Table 2: CIFAR-100 validation accuracies.

<table>
<thead>
<tr>
<th>Network</th>
<th>v1 (g=8)</th>
<th>v1 (g=3)</th>
<th>v2 1x</th>
<th>v2 1.5x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual</td>
<td>69.65</td>
<td>70.16</td>
<td>72.75</td>
<td>73.51</td>
</tr>
<tr>
<td>Auto</td>
<td>70.89</td>
<td>71.89</td>
<td>73.40</td>
<td>74.26</td>
</tr>
</tbody>
</table>

Table 3: 20-class subset (A=Architectures, L=Landscapes).

<table>
<thead>
<tr>
<th>Cats</th>
<th>Dogs</th>
<th>Vehicles</th>
<th>A</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Egyptian</td>
<td>Sheepdog</td>
<td>Bike</td>
<td>Bridge</td>
<td>Valley</td>
</tr>
<tr>
<td>Persian</td>
<td>Bulldog</td>
<td>Sports car</td>
<td>Dam</td>
<td>Sandbar</td>
</tr>
<tr>
<td>Tiger</td>
<td>Mountain</td>
<td>Scooter</td>
<td>Castle</td>
<td>Cliff</td>
</tr>
<tr>
<td>Siamese</td>
<td>Maltese</td>
<td>Cab</td>
<td>Fence</td>
<td>Volcano</td>
</tr>
</tbody>
</table>

Table 4: Validation accuracies on 20-class in Table 3.

<table>
<thead>
<tr>
<th>Network</th>
<th>v1 (g=8)</th>
<th>v1 (g=3)</th>
<th>v2 (r=0.3)</th>
<th>v2 (r=0.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual</td>
<td>82.84</td>
<td>82.00</td>
<td>84.63</td>
<td>86.11</td>
</tr>
<tr>
<td>Auto</td>
<td>83.68</td>
<td>84.00</td>
<td>85.58</td>
<td>86.84</td>
</tr>
</tbody>
</table>

The permutation matrix of the first shuffle unit in ShuffleNet v1 (g=3) is a matrix of size $60 \times 60$, which can be visualized in Fig. 7 (manual, left) along with an auto shuffle (right). The dots (blanks) denote locations of 1’s (0’s). The auto shuffle looks disordered while the manual shuffle is ordered. However, the inference cost of auto shuffle is same as manual shuffle since the shuffle is fixed and stored after training.

The accuracy drop due to rounding to produce auto shuffle from relaxed shuffle is indicated by relative change in Tab. 5. On CIFAR-10 dataset, negligible drop is observed for ShuffleNet v1 (g=3) is a matrix of size $60 \times 60$, which can be visualized in Fig. 7 (manual, left) along with an auto shuffle (right). The dots (blanks) denote locations of 1’s (0’s). The auto shuffle looks disordered while the manual shuffle is ordered. However, the inference cost of auto shuffle is same as manual shuffle since the shuffle is fixed and stored after training.

The accuracy drop due to rounding to produce auto shuffle from relaxed shuffle is indicated by relative change in Tab. 5. On CIFAR-10 dataset, negligible drop is observed for ShuffleNet v1 (g=3) is a matrix of size $60 \times 60$, which can be visualized in Fig. 7 (manual, left) along with an auto shuffle (right). The dots (blanks) denote locations of 1’s (0’s). The auto shuffle looks disordered while the manual shuffle is ordered. However, the inference cost of auto shuffle is same as manual shuffle since the shuffle is fixed and stored after training.

The accuracy drop due to rounding to produce auto shuffle from relaxed shuffle is indicated by relative change in Tab. 5. On CIFAR-10 dataset, negligible drop is observed for ShuffleNet v1 (g=3) is a matrix of size $60 \times 60$, which can be visualized in Fig. 7 (manual, left) along with an auto shuffle (right). The dots (blanks) denote locations of 1’s (0’s). The auto shuffle looks disordered while the manual shuffle is ordered. However, the inference cost of auto shuffle is same as manual shuffle since the shuffle is fixed and stored after training.

The accuracy drop due to rounding to produce auto shuffle from relaxed shuffle is indicated by relative change in Tab. 5. On CIFAR-10 dataset, negligible drop is observed for ShuffleNet v1 (g=3) is a matrix of size $60 \times 60$, which can be visualized in Fig. 7 (manual, left) along with an auto shuffle (right). The dots (blanks) denote locations of 1’s (0’s). The auto shuffle looks disordered while the manual shuffle is ordered. However, the inference cost of auto shuffle is same as manual shuffle since the shuffle is fixed and stored after training.

The accuracy drop due to rounding to produce auto shuffle from relaxed shuffle is indicated by relative change in Tab. 5. On CIFAR-10 dataset, negligible drop is observed for ShuffleNet v1 (g=3) is a matrix of size $60 \times 60$, which can be visualized in Fig. 7 (manual, left) along with an auto shuffle (right). The dots (blanks) denote locations of 1’s (0’s). The auto shuffle looks disordered while the manual shuffle is ordered. However, the inference cost of auto shuffle is same as manual shuffle since the shuffle is fixed and stored after training.
Figure 7: Permutation matrices of the first shuffle unit in ShuffleNet v1 (g=3) of manual shuffle (top) and auto shuffle (bottom). The auto shuffle is trained on CIFAR-10 dataset. The dots (blanks) indicate locations of 1’s (0’s). The auto shuffle looks disordered while the manual shuffle is ordered. The inference cost of auto shuffle is comparable to manual shuffle in inference.

Table 5: Relative change (Rel. Ch) of accuracy of rounding relaxed shuffle. The +/- refer to accuracy drop/gain after rounding to produce auto shuffle from relaxed shuffle.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Network</th>
<th>Rel. Ch. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>v1 1x (g=8)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>v1 1x (g=3)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>v2 1x</td>
<td>-0.02</td>
</tr>
<tr>
<td></td>
<td>v2 1.5x</td>
<td>-0.11</td>
</tr>
<tr>
<td>20-class</td>
<td>v1 0.25x (g=8)</td>
<td>+0.25</td>
</tr>
<tr>
<td></td>
<td>v1 0.25x (g=3)</td>
<td>+0.76</td>
</tr>
<tr>
<td></td>
<td>v2 0.5x (r=0.3)</td>
<td>+0.37</td>
</tr>
<tr>
<td></td>
<td>v2 0.5x (r=0.5)</td>
<td>0</td>
</tr>
</tbody>
</table>

rounding) becomes closer to relaxed shuffle (before rounding), see Fig. 9.

Figure 8: Training loss $L$ and penalty $P$ of ShuffleNet v1 (g=3) with relaxed shuffle on CIFAR-10.

Figure 9: Validation accuracy of ShuffleNet v1 (g=3) with relaxed shuffle (before rounding) and auto shuffle (after rounding) on CIFAR-10. The rounding error becomes smaller during training.

To demonstrate the significance of the $\ell_1$−$\ell_2$ regularization, we also tested auto shuffle with various $\lambda$ on ShuffleNet v1 (g=3). Tab. 6 shows that the accuracy drops much after the relaxed shuffle is rounded. We plot the stochastic matrix of the first shuffle unit of the network at $\lambda = 0$ and $\lambda = 10^{-5}$ respectively in Fig. 10. The penalty is large when $\lambda$ is relatively small, indicating that the stochastic matrices learned are not close to optimal permutation matrices.

5 CONCLUSION

We introduced a novel, exact and Lipschitz continuous relaxation for permutation and learning channel shuffling in ShuffleNet. The learned shuffle consistently out-performs manual shuffle on CIFAR-10, CIFAR-100, and 20 sub-class of ImageNet data sets across various
Table 6: CIFAR-10 validation accuracies of ShuffleNet v1 (g=3) with relaxed (R) shuffle before rounding and auto (A) shuffle after rounding, and penalty (P) values of relaxed shuffle at various λ’s. The penalty and rounding error tends to zero as λ increases.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1E-5</th>
<th>1E-4</th>
<th>5E-4</th>
<th>1E-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>90.00</td>
<td>90.18</td>
<td>90.48</td>
<td>91.45</td>
<td>91.76</td>
</tr>
<tr>
<td>A</td>
<td>10.00</td>
<td>38.18</td>
<td>11.37</td>
<td>71.50</td>
<td>91.76</td>
</tr>
<tr>
<td>P</td>
<td>3.37E3</td>
<td>1.59E3</td>
<td>4.95E2</td>
<td>3.13E-1</td>
<td>5.07E-2</td>
</tr>
</tbody>
</table>

![Graph showing stochastic matrices of the first shuffle unit in ShuffleNet v1 (g=3) with relaxed shuffle at various λ’s.](image)

Figure 10: Stochastic matrices of the first shuffle unit in ShuffleNet v1 (g=3) with relaxed shuffle before rounding at λ = 0 (top) and λ = 10^-5 (bottom). The relaxed shuffle is trained on CIFAR-10 dataset. The matrices are quite diffusive, and not close to optimal permutation matrices when λ is relatively small.

The idea of auto-shuffle applies broadly to permutation learning problems in science and engineering such as neuron identification from the worm C. elegans [2], object tracking [19], to name a few. We plan to extend our work to auto-shuffling in other LCNNs and a wide range of permutation optimization problems of data science in the future.

ACKNOWLEDGMENTS

The work was partially supported by NSF grants IIS-1632935, DMS-1854434, a Qualcomm Faculty Award, and Qualcomm AI Research.

REFERENCES


