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

Jiancheng Lyu [†], Shuai Zhang [*], Yingyong Qi [*], Jack Xin [†]

Abstract

ShuffleNet is a state-of-the-art light weight convolutional neural network architecture. Its basic operations include group, channel-wise convolution and channel shuffling. However, channel shuffling is manually designed empirically. Mathematically, shuffling is a multiplication by a permutation matrix. In this paper, we propose to automate channel shuffling by learning permutation matrices in network training. We introduce an exact Lipschitz continuous non-convex penalty so that it can be incorporated in the stochastic gradient descent to approximate permutation at high precision. Exact permutations are obtained by simple rounding at the end of training and are used in inference. The resulting network, referred to as AutoShuffleNet, achieved improved classification accuracies on CIFAR-10 and ImageNet data sets. In addition, we found experimentally that the standard convex relaxation of permutation matrices into stochastic matrices leads to poor performance. We prove theoretically the exactness (error bounds) in recovering permutation matrices when our penalty function is zero (very small). We present examples of permutation optimization through graph matching and two-layer neural network models where the loss functions are calculated in closed analytical form. In the examples, convex relaxation failed to capture permutations whereas our penalty succeeded.

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 ([20], [13]), IGC (Interleaved Group Convolutions, [19]) and IGCV3 (Interleaved Low-Rank Group Convolutions,[15]). 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. 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 ([20], [13]). 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 [7, 3]. Well-known examples are linear and quadratic assignment problems [16]. Graph matching is a special case of quadratic assignment problem, and can be formulated over $N \times N$ permutation matrices $\mathcal{P}^N$ as: $\min_{\pi \in \mathcal{P}^N} \|A - \pi B \pi^T\|_F^2$, 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 problem may still be non-convex due to the objective function. The explicit realization of $\mathcal{D}^N$ is the set of doubly stochastic matrices $\mathcal{D}^N = \{M \in \mathbf{R}^{N\times N} : M\mathbf{1} = \mathbf{1}, M^T\mathbf{1} = \mathbf{1}, M \geq 0\}$, where $\mathbf{1} = (1, 1, , \cdots, 1)^T \in \mathbf{R}^N$. An approximate yet simpler way to treat

[*]Qualcomm AI Research, San Diego, CA 92121. Email: (shuazhan,yingyong)@qti.qualcomm.com.
[†]Department of Mathematics, UC Irvine, Irvine, CA 92697. Email: (jianchel, jack.xin)@uci.edu.

$\mathcal{D}^N$ is through the classical first order matrix scaling algorithm e.g. the Sinkhorn method [14]. Though in principle such algorithm converges, the cost can be quite high when iterating many times, which causes a bottleneck effect [11]. A non-convex and more compact relaxation of $\mathcal{P}^N$ is by a sorting network [11] 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 [18] which seeks solutions under concave relaxations of $\mathcal{P}^N$ by solving a linear interpolation of convex-concave problems (starting from the convex relaxation). None of the existing relaxations are exact.

Contribution. Our non-convex relaxation is a combination of matrix ℓ_{1-2} penalty function and $\mathcal{D}^N$. The ℓ_{1-2} (the difference of ℓ_1 and ℓ_2 norms) has been proposed and found effective in selecting sparse vectors under nearly degenerate linear constraints [5, 17]. The matrix version is simply a sum of ℓ_{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 and Imagenet data sets, the closeness to $\mathcal{P}^N$ 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 found that convex relaxation by $\mathcal{D}^N$ fails to capture good permutations for LCNNs. To our best knowledge, this is *the first time permutations have been successfully learned on deep CNNs to improve hand-crafted permutations.*

Outline. In section 2, we introduce our exact permutation penalty, and prove the 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 $\mathcal{P}^N$ for SGD. In section 3, we analyze three permutation optimization problems to show the necessity of our penalty. In a 2-layer neural network regression model with short cut (identify 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 both CIFAR-10 and Imagenet data sets. Conclusion is in section 5.

2 Permutation, Matrix ℓ_{1-2} Penalty and Exact Relaxation

The channel shuffle operation in ShuffleNet [20, 13] 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 [20, 13], 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 ℓ_{1-2} penalty [5, 17] to the network loss function in the stochastic gradient descent (SGD) based training.

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

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

in conjunction with the doubly stochastic constraint:

$$m_{ij} \geq 0,\ \forall (i,j);\ \sum_{i=1}^{N} m_{ij} = 1,\ \forall j;\ \sum_{j=1}^{N} m_{ij} = 1,\ \forall i. \tag{2}$$

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 $\sum_{j=1}^{N} m_{ij}^2 \neq 0$, $\forall i$, and $\sum_{i=1}^{N} m_{ij}^2 \neq 0$, $\forall j$.*

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 Cauchy-Schwarz inequality,

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

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

$$|\,\{j : m_{ij} \neq 0\}\,| = 1, \quad \forall i. \tag{3}$$

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

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

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 $m_{ij} \to \max(m_{ij}, 0)$. The normalization conditions in (2) are implemented sequentially once in SGD iteration. Hence they are not strictly enforced. In theory, if the column normalization (divide each column by its sum) and row normalization (divide each row by its sum) are iterated sufficiently many times, the resulting matrices converge to (2). This is known as the Sinkhorn process or RAS method [14], which is a first order method to approximately solve the so called matrix scaling problem (MSP). Simply state, the MSP for a given non-negative real matrix $A \in \mathbf{R}^{N\times N}$ is to scale its rows and columns (i.e. multiply each by a non-negative constant) to realize the prescribed row sums and column sums. The approximate MSP is: given tolerance $\epsilon > 0$, find positive diagonal matrices X and Y such that $|\,XAY\mathbf{1} - \mathbf{1}\,| \leq \epsilon$, $|\,\mathbf{1}^T XAY - \mathbf{1}^T\,| \leq \epsilon$. For a historical account of MSP and a summary of various algorithms to date, see [2] and Table 1 therein. The RAS method is an alternate minimization procedure with convergence guarantees. Each iteration of the RAS method costs complexity $O(m)$, m being the number of non-zero entries in A. If the entries of A are polynomially bounded (which is the case during network training due to the continuous nature of SGD), the RAS method converges in $\tilde{O}(N/\epsilon^2)$ iterations [6], giving total complexity $\tilde{O}(mN/\epsilon^2)$, where tilde hides logarithmic factors in N and ϵ^{-1}. Improvements of complexity bounds via minimizing the log of capacity and higher order methods can be found in [2]. However for our study, the first order method [14] suffices for two reasons. One is that it is computationally low cost, the other is that the error in the matrix scaling step can be compensated in network weight adjustment during SGD. In fact, we did not find much benefit to iterate RAS method more than once in terms of enhancing validation accuracy. This is quite different from solving MAP as a stand alone task.

The multiplication by M can be embedded in the network as a 1×1 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:

$$f = f(w, M) := L(w) + \lambda \sum_{j=1}^{J} P(M_j), \tag{4}$$

where J is the total number of "channel shuffles" M_j's abbreviated as M , w is the network weight, λ a positive parameter. The training algorithm is summarized in Algorithm 1. The ℓ_1 term in the penalty function P has standard sub-gradient, and the ℓ_2 term is differentiable away from zero, which is guaranteed in the algorithm 1 due to continuity of SGD and the normalization in columns and rows. λ is chosen to be 10^{-3} or 2×10^{-3} so as to balance the contributions of the two terms in (4) and drive $\sum_{j=1}^{J} P(M_j)$ close to 0.

We shall see that the penalty P indeed gets smaller and smaller during training. Here we show a theoretical bound on the distance to $\mathcal{P}^N$ when P is small and (2) holds approximately.

Algorithm 1 AutoShuffle Learning.

Input:
mini-batch loss function $f_t(w, M)$, t being the iteration index;
learning rate η^t for (w, M);
penalty parameter λ for P;
total iteration number Tn.
Start:
w: sample from unit Gaussian distribution;
M: sample from unit Gaussian distribution then take absolute value.
WHILE $t < Tn$, **DO**:
(1) Evaluate the mini-batch gradient $(\nabla_w f_t, \nabla_M f_t)$ at (w^t, M^t);
(2) $w^{t+1} = w^t - \eta^t \nabla_w f_t(w^t, M^t)$; // gradient update for weights
(3) $M^{t+1} = M^t - \eta^t \nabla_M f_t(w^t, M^t)$; // 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^{Tn},M^{Tn}; project each matrix M_j^{Tn} inside M^{Tn} to the nearest permutation matrix.

Theorem 2. *Let the dimension N of a non-negative square matrix M be fixed. If $P(M) = O(\epsilon)$, $\epsilon \ll 1$, and the doubly stochastic constraints are satisfied to $O(\epsilon)$, then there exists a permutation matrix P^* such that $\| M - P^* \| = O(\epsilon)$.*

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

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

implying that:

$$| m_{ij} \, m_{ij'} | = O(\epsilon), \quad \forall j \neq j', \quad \forall i. \tag{5}$$

On the other hand for $\forall i$:

$$\sum_{j=1}^{N} m_{ij} = 1 + O(\epsilon). \tag{6}$$

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

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

and from (5) that

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

Hence each row of M is $O(\epsilon)$ close to a unit coordinate vector, with one entry near 1 and the rest near 0. Similarly from $\sum_{i=1}^{N} | m_{ij}| - \left(\sum_{i=1}^{N} m_{ij}^2 \right)^{1/2} = O(\epsilon), \quad \forall j$, and $\sum_{i=1}^{N} m_{ij} = 1 + O(\epsilon)$, we deduce that each column of M is $O(\epsilon)$ 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 $O(\epsilon)$ close to a permutation matrix. □

The learned non-negative matrix M will be called a *relaxed shuffle* and will be rounded to the nearest permutation matrix to produce a final *auto shuffle*. 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 3, the relaxed shuffle comes amazingly close to an exact permutation in network learning. Hence, it turns out unnecessary to solve LAP exactly, indeed a simple rounding will do. AutoShuffleNet units adapted from ShuffleNet v1 [20] and ShuffleNet v2 [13] are illustrated in Figs. 1-2.

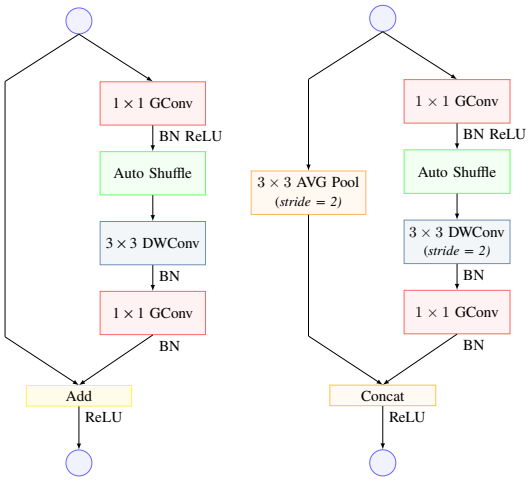

Figure 1: AutoShuffleNet units based on ShuffleNet v1.

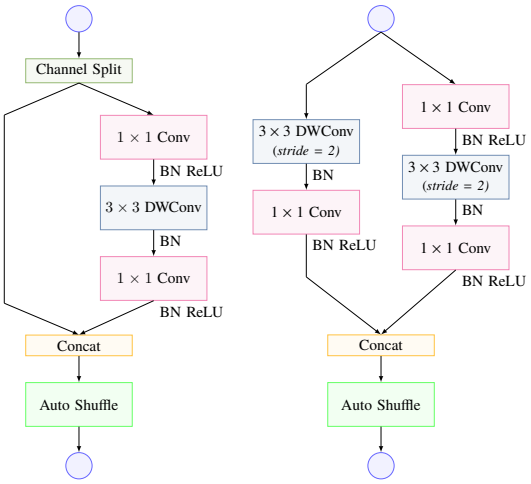

Figure 2: AutoShuffleNet units based on ShuffleNet v2.

3 Permutation Problems Unsolvable by Convex Relaxation

The doubly stochastic matrix condition (2) is a popular convex relaxation of permutation. However, it is not powerful enough to enable auto-shuffle learning as we shall see later. In this section, we present examples from permutation optimization to show the limitation of convex relaxation (2), and how our proposed penalty (1) can strength (2) to retrieve permutation matrices.

Let us recall the graph matching (GM) problem, see [16, 11, 1, 12, 18] and references therein. The goal is to align the vertices of two graphs to minimizes 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 $\|AQ - QB\|_F^2$. Let Π be the set of all permutation matrices, solve

$$Q^* := \text{argmin}_{Q \in \Pi} \|A\,Q - Q\,B\|_F^2. \tag{7}$$

By algebraic identity:

$$\begin{aligned} \|A\,Q - Q\,B\|_F^2 &= \text{trace}\{(A\,Q - Q\,B)^T (A\,Q - Q\,B)\} \\ &= \text{trace}(A^T\,A) + \text{trace}(B^T\,B) - 2\text{trace}(A\,Q\,B^T\,Q^T), \end{aligned}$$

the GM problem (7) is same as:

$$Q^* = \text{argmin}_{Q \in \Pi} \text{trace}((-A)\,Q\,B^T\,Q^T), \tag{8}$$

a quadratic assignment problem (QAP). The general QAP for two real square matrices A and B is [16, 11]:

$$Q^* = \text{argmin}_{Q \in \Pi}\ \text{trace}(A\,Q\,B^T\,Q^T). \tag{9}$$

The convex relaxed GM is:

$$Q_* := \text{argmin}_{Q \in D^N} \|A\,Q - Q\,B\|_F^2. \tag{10}$$

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 $AQ - QB$ equals ($q' := 1 - q$):

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

Example 1: We show that $Q_* \neq Q^*$ by selecting:

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

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

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

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

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

however, the permutation matrix solution Q^* to GM problem (7) is the 2×2 identity matrix at $q = 1$.

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

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

or equivalently minimize (after skipping some additive constants in P):

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

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 λ increases to 1 (line-star, middle curve) and remains there as λ further increases to 2 (line-circle, bottom curve). So for $\lambda \in [1, 2]$, Q^* is recovered with our proposed penalty. □

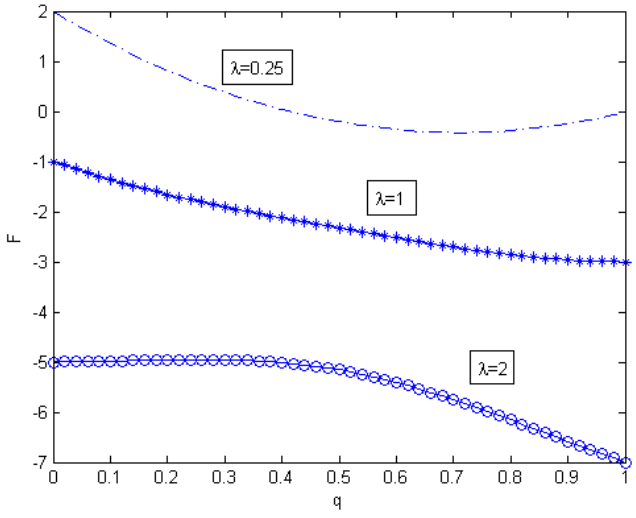

Figure 3: The function $F(q)$ as penalty parameter λ varies from 0.25 (interior minimal point, dashed line, top) to 1 (line-star, middle) and 2 (line-circle, bottom). Minimal point occurs at $q = 1$ in the latter two curves.

Example 2: Consider the adjacent matrix B (A) of an un-directed graph of 2 nodes and 1 edge (with a loop at node 1). An edge adds 1 and a loop adds 2 to an adjacent matrix.

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

Then:

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

$$\|A\,Q - Q\,B\|_F^2 = 4[q^2 + (1-q)^2].$$

So $Q_* = Q(1/2) \neq Q^* = Q(0) = Q(1)$. The P regularized objective function (modulo additive constants) is:

$$F = 4[q^2 + (1-q)^2] - 4\lambda(q^2 + (1-q)^2)^{1/2},$$

with $F(0) = F(1) = 4 - 4\lambda$. In view of:

$$F'/4 = (2q-1)[2 - \lambda/(q^2 + (1-q)^2)^{1/2}],$$

two possible interior critical points are:

$$q = 1/2 \;\text{ or }\; q^2 + (1-q)^2 = \lambda^2/4. \tag{11}$$

Since $\max_{q\in[0,1]}\{q^2 + (1-q)^2\} = 1$, if $\lambda > 2$, the second equality in (11) is ruled out. Comparing $F(1/2) = 2 - 4\lambda 2^{-1/2} = 2(1-\sqrt{2}\lambda)$ with $F(0)$, we see that the global minimal point does not occur at $q = 1/2$ if

$$1 - \sqrt{2}\lambda > 2 - 2\lambda \;\text{ or }\; \lambda > 1/(2-\sqrt{2}) \approx 1.7071.$$

Hence if $\lambda > 2$, minimizing F recovers Q^*. □

In Fig. 4, we show that two minimal points of F occur in the interior of $(0,1)$ when $\lambda = 1.8, 1.9$, and transition to $q = 0, 1$, at $\lambda = 2$. When $\lambda^2/4 < \min_{q\in[0,1]}\{q^2 + (1-q)^2\} = 1/2$, or $\lambda < \sqrt{2}$, the second equality in (11) cannot hold, F becomes convex with a unique minimal point at $q = 1/2$.

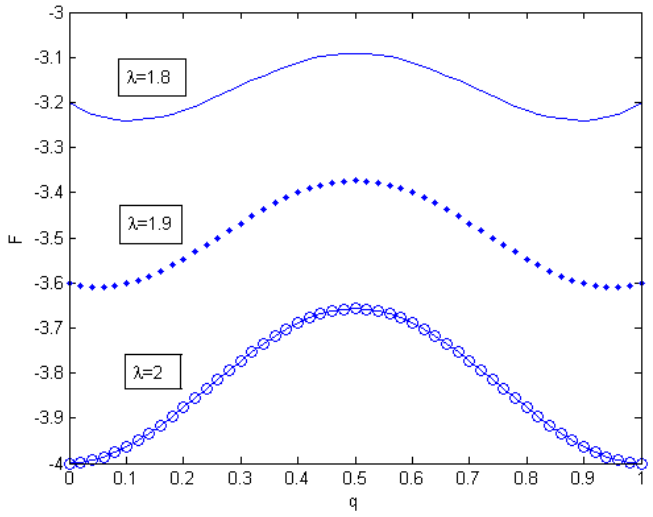

Figure 4: The function $F(q)$ as penalty parameter λ varies from 1.8 (solid line, top) to 1.9 (dot, middle) and 2 (line-circle, bottom) where minimal points occur at $q = 0, 1$. Interior minimal points occur on $[0,1]$ when $\lambda = 1, 8, 1.9$.

Remark 3. *We refer to [12] on certain correlated random Bernoulli graphs where $Q^* \neq Q_*$. On the other hand, there is a class of friendly graphs [1] 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 [16, 11, 18]. 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 above, if the convex relaxed $q_* = 2/3$ is rounded up to 1, then $Q_* = Q^*$. 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 still recovers 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 [10]. Given $m \geq 0$, the forward model is the following function:

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

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

$$L(W) = \mathbf{E}_x \left[f_m(x, W) - f_m(x, W^*) \right]^2.$$

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

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

We write the loss function as

$$\begin{aligned} l_m(p) :=& L(W) \\ =& \mathbf{E}_x \left[\phi((m+p)x_1 + (1-p)x_2) + \phi((1-p)x_1 + (m+p)px_2) \right. \\ & \left. -\phi(ax_1 + bx_2) - \phi(cx_1 + dx_2) \right]^2 \\ =& \mathbf{E}_x \phi((m+p)x_1 + (1-p)x_2)^2 + \mathbf{E}_x \phi((1-p)x_1 + (m+p)x_2)^2 \\ & + 2\mathbf{E}_x \left[\phi((m+p)x_1 + (1-p)x_2) \phi((1-p)x_1 + (m+p)x_2) \right] \\ & - 2G_m(p, a, b) - 2G_m(p, c, d) + \mathbf{E}_x \left[\phi(ax_1 + bx_2) + \phi(cx_1 + dx_2) \right]^2, \end{aligned} \tag{12}$$

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

$$\mathbf{E}_x \left[\phi((m+p)x_1 + (1-p)x_2) \phi(sx_1 + tx_2) + \phi((1-p)x_1 + (m+p)x_2) \phi(sx_1 + tx_2) \right].$$

Define $\mathcal{I}(q, r, s, t) := \mathbf{E}_x \left[\phi(qx_1 + rx_2) \phi(sx_1 + tx_2) \right]$, then $\mathcal{I}(q, r, s, t) = \mathcal{I}(s, t, q, r)$ and

$$G_m(p, s, t) = \mathcal{I}(m+p, 1-p, s, t) + \mathcal{I}(1-p, m+p, s, t).$$

For simplicity, let x obey uniform distribution on $[-1, 1]^2$. For $qt \geq rs$, $q + r > 0$, $s + t > 0$, $\mathcal{I}(q, r, s, t)$ equals

$$\begin{cases} \frac{2}{3}(qs + rt) + \frac{q^2(qt - 3rs)}{24r^2} + \frac{s^2(3qt - rs)}{24t^2}, q < r \\ \frac{1}{3}(qs + rt) + \frac{1}{4}(qt + rs) + \frac{1}{24}(\frac{r^2}{q^2} + \frac{s^2}{t^2})(3qt - rs), \ q \geq r \text{ and } t \geq s \\ \frac{2}{3}(qs + rt) + \frac{r^2(3qt - rs)}{24q^2} + \frac{t^2(qt - 3rs)}{24s^2}, t < s. \end{cases} \tag{13}$$

We have

$$\begin{aligned} \mathbf{E}_x \phi((m+p)x_1 + (1-p)x_2)^2 &= \mathbf{E}_x \phi((1-p)x_1 + (m+p)x_2)^2 \\ &= \frac{2}{3} \left[(m+p)^2 + (1-p)^2 \right], \end{aligned} \tag{14}$$

$$\mathbf{E}_x \left[\phi((m+p)x_1 + (1-p)x_2) \phi((1-p)x_1 + (m+p)x_2) \right] = \frac{(m+1)^3}{3\theta_m(p)} + \frac{(m+p)^4}{12\theta_m(p)^2}, \tag{15}$$

where $\theta_m(p) := \max(m + p, 1 - p)$. The last term in (12) is a constant:

$$\mathbf{E}_x \left[\phi(ax_1 + bx_2) + \phi(cx_1 + dx_2) \right]^2 = \frac{2}{3}(a^2 + b^2 + c^2 + d^2) + 2\mathcal{I}(a, b, c, d). \tag{16}$$

Consider a special case when $a = 1/3$, $b = 2/3$, $c = 1/4$ and $d = 3/4$. By (14)-(16), the loss function $l_m(p)$ equals

$$\frac{2}{3}\left[(m+p)^2 + (1-p)^2\right] + \frac{2(m+1)^3}{3\theta_m(p)} + \frac{(m+1)^4}{6\theta_m(p)^2} - 2G_m(p, \frac{1}{3}, \frac{2}{3}) - 2G_m(p, \frac{1}{4}, \frac{3}{4}) + \frac{8113}{5184}.$$

Let $F_m(p) := l_m(p) - 4\lambda\sqrt{p^2 + ((1-p)^2}$. When $m = 0$, $\lambda = 0$, Fig. 5 (top left) 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, top right), 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.

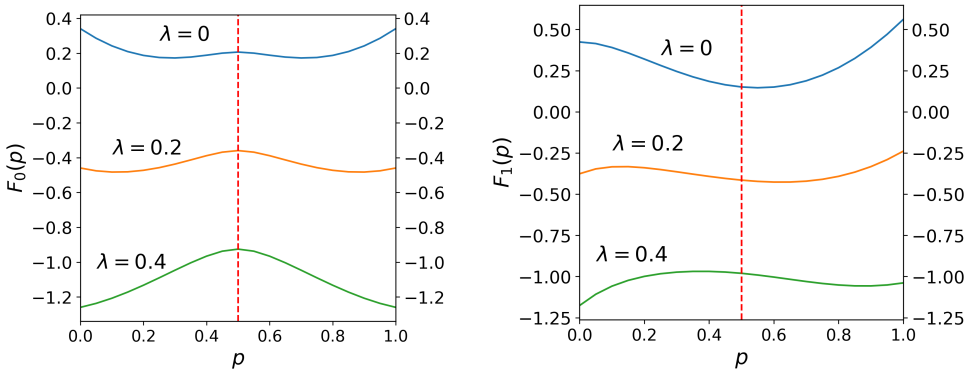

Figure 5: $F_m(p)$ ($m = 0$ left, $m = 1$ right) as penalty parameter λ varies for the uniformly distributed input data on $[-1, 1]^2$.

Remark 5. *If x obeys the unit Gaussian distribution as in [10], the $F_m(p)$ functions are more complicated analytically, however their plots resemble those for uniformly distributed x, see Fig. 6.*

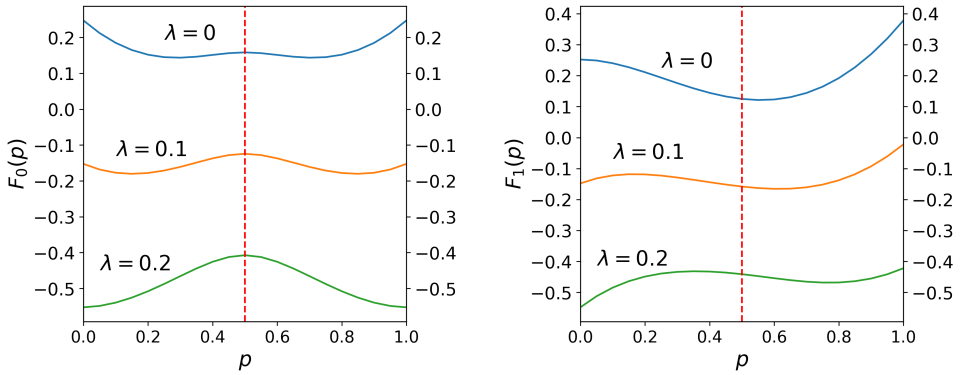

Figure 6: $F_m(p)$ ($m = 0$ left, $m = 1$ right) as penalty parameter λ varies for unit Gaussian input data on $\mathbf{R}^2$.

4 Experiments

We relax the shuffle units in ShuffleNet v1 [20] and ShuffleNet v2 [13] and perform experiments on CIFAR-10 [8] and ImageNet [4, 9] classification datasets. The accuracy results of auto shuffles are evaluated after the relaxed shuffles are rounded.

On CIFAR-10 dataset, we set the ℓ_{1-2} 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. We set weight-decay 10^{-4}, momentum 0.95 and batch size 128. The experiments are

carried out on a machine with single Nvidia GeForce GTX 1080 Ti GPU. In Table 1, we see that auto shuffle consistently improves by as much as 1% on manual shuffle in both v1 and v2 models of ShuffleNet.

Table 1: CIFAR-10 validation accuracies.

Networks	Manual	Auto
ShuffleNet v1 (g=3)	90.55	91.76
ShuffleNet v1 (g=8)	90.06	91.26
ShuffleNet v2 (1×)	91.90	92.81
ShuffleNet v2 (1.5×)	92.56	93.22

On ImageNet dataset, we set the ℓ_{1-2} penalty parameter $\lambda = 2 \times 10^{-3}$. For each experiment, the training process includes two training cycles: the first cycle is randomly initialized with learning rate starting at 0.2 and the second one is resumed from the first one with learning rate starting at 0.1. Each cycle consists of 100 epochs and the learning rate decays linearly. We set weight-decay 4×10^{-5}, momentum 0.9 and batch size 512. The experiments are carried out on a machine with 4 Nvidia GeForce GTX 1080 Ti GPUs. In Table 2, we see that auto shuffle again consistently improves on manual shuffle for both v1 and v2 models.

Table 2: ImageNet top-1 validation accuracies.

Networks	Manual	Auto
ShuffleNet v1 (g=3)	65.50	65.62
ShuffleNet v1 (g=8)	65.18	65.76
ShuffleNet v2 (1×)	67.46	67.69
ShuffleNet v2 (1.5×)	70.58	70.60

The permutation matrix of the first shuffle unit in ShuffleNet v1 (g=3) is a matrix of size 60×60, 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 comparable to manual shuffle since the shuffle is fixed and stored after training.

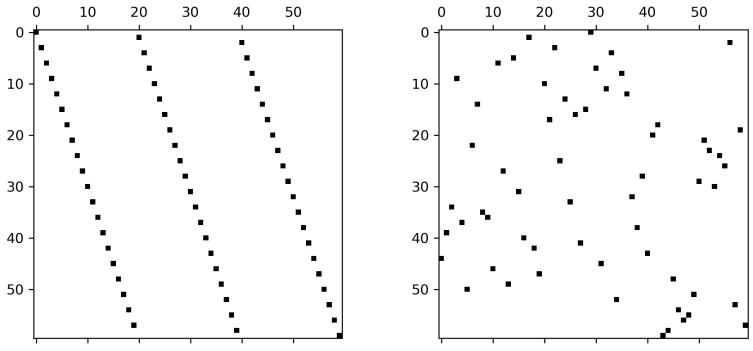

Figure 7: Permutation matrices of the first shuffle unit in ShuffleNet v1 (g=3) of manual shuffle (left) and auto shuffle (right). The auto shuffle is trained on CIFAR-10 dataset. The dots (blanks) indicate locations of 1's (0's). The auto shuffle looks disordered while the manual shuffle is ordered. However, the inference cost of auto shuffle is comparable to manual shuffle in inference.

The accuracy drop due to rounding to produce auto shuffle from relaxed shuffle is indicated

by relative error in Table 3. On CIFAR-10 dataset, negligible drop is observed for ShuffleNet v1. Interestingly, rounding even gained accuracy for ShuffleNet v1 on ImageNet dataset.

Table 3: Relative error of accuracy of rounding relaxed shuffle. The -/+ refer to accuracy drop/gain after rounding to produce auto shuffle from relaxed shuffle.

Dataset	Networks	Relative Error
CIFAR-10	ShuffleNet v1 (g=3)	0
	ShuffleNet v1 (g=8)	0
	ShuffleNet v2 (1×)	-2.15E-4
	ShuffleNet v2 (1.5×)	-1.07E-3
ImageNet	ShuffleNet v1 (g=3)	+6.10E-5
	ShuffleNet v1 (g=8)	+3.04E-5
	ShuffleNet v2 (1×)	0
	ShuffleNet v2 (1.5×)	-2.83E-5

The ℓ_{1-2} penalty of ShuffleNet v1 (g=3) is plotted in Fig. 8. As the penalty decays, the validation accuracy of **auto shuffle** (after rounding) becomes closer to **relaxed shuffle** (before rounding), see Fig. 9.

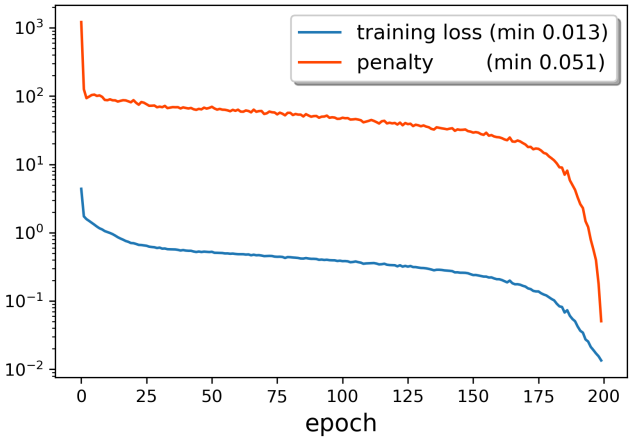

Figure 8: Training loss L and penalty P of ShuffleNet v1 (g=3) with relaxed shuffle on CIFAR-10.

To demonstrate the significance of the ℓ_{1-2} regularization, we also tested auto shuffle with various λ on ShuffleNet v1 (g=3). Table 4 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 λ 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. We showed through a regression problem of a 2-layer neural network with short cut that convex relaxation fails even with additional rounding while our relaxation is precise. We plan to extend our work to auto-shuffling of other LCNNs and hard permutation problems in the future.

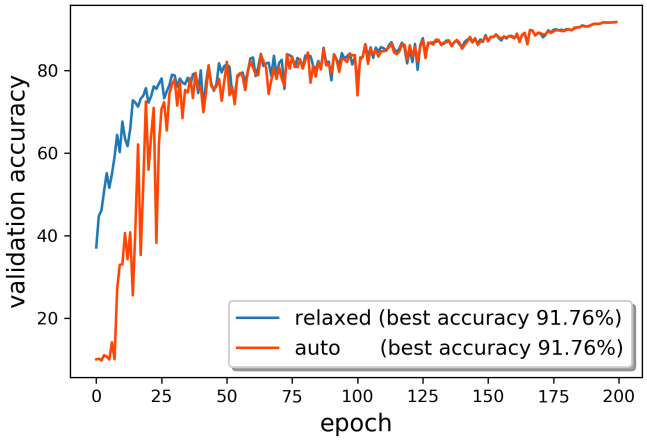

Figure 9: Validation accuracy of ShuffleNet v1 (g=3) with relaxed shuffle (before rounding) and auto shuffle (after rounding) on CIFAR-10. The rounding error becomes smaller during training.

Table 4: CIFAR-10 validation accuracies of ShuffleNet v1 (g=3) with relaxed shuffle (before rounding) and auto shuffle (after rounding), and penalty values of relaxed shuffle at various λ's. The penalty and rounding error tends to zero as λ increases.

λ	0	1E-5	1E-4	5E-4	1E-3
relaxed	90.00	90.18	90.48	91.45	91.76
auto	10.00	38.18	11.37	71.50	91.76
penalty	3.37E3	1.59E3	4.95E2	3.13E-1	5.07E-2

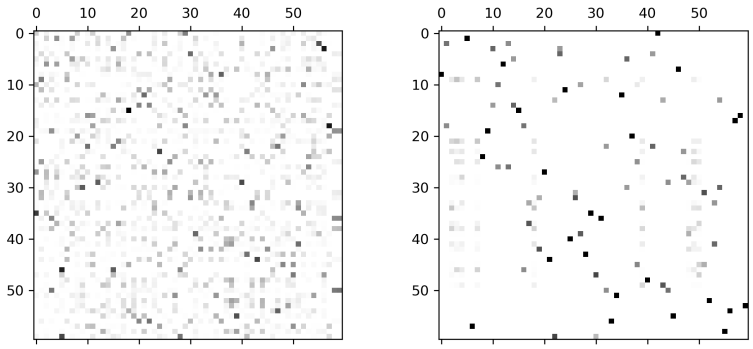

Figure 10: Stochastic matrices of the first shuffle unit in ShuffleNet v1 (g=3) with relaxed shuffle before rounding at $\lambda = 0$ (left) and $\lambda = 10^{-5}$ (right). The relaxed shuffle is trained on CIFAR-10 dataset. The matrices are quite diffusive and not close to optimal permutation matrices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

6 Acknowledgement

The work was partially supported by NSF grant IIS-1632935.

References

[1] Y. Aalo, A. Bronstein, and R. Kimmel. On convex relaxation of graph isomorphism. *Proc. National Academy Sci*, 112(10):2942–2947, 2015.

[2] Zeyuan Allen-Zhu, Yuanzhi Li, Rafael Oliveira, and Avi Wigderson. Much faster algorithms for matrix scaling. *58th Annual IEEE Symposium on Foundations of Computer Science*, pages 890–901, 2017.

[3] R. Burkard. The quadratic assignment problem. *in: Handbook of Combinatorial Optimization*, pages 2741–2814, 2013.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[5] Ernie Esser, Yifei Lou, and Jack Xin. A method for finding structured sparse solutions to non-negative least squares problems with applications. *SIAM J. Imaging Sciences*, 6:2010–2046, 2013.

[6] B. Kalantari, I. Lari, F. Ricca, and B. Simeone. On the complexity of general matrix scaling and entropy minimization via the ras algorithm. *Mathematical Programming*, 112(2):371–401, 2008.

[7] T. Koopmans and M. Beckman. Assignment problems and the location of economic activities. *The Econometric Society*, 25:53–76, 1957.

[8] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Tech Report*, 2009.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Conference on Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

[10] Y. Li and Y. Yuan. Convergence analysis of two-layer neural networks with relu activation. *arXiv preprint 1705.09886v2*, 2017.

[11] C. Lim and S. Wright. A box-constrained approach for hard permutation problems. *Proceedings of the 33rd International Conference on Machine Learning*, page 10 pages, 2016.

[12] V. Lyzinski, D. Fishkind, M. Fiori, J. Vogelstein, C. Priebe, and G. Sapiro. Graph matching: Relax at your own risk. *arXiv preprint 1405.3133*, 2014.

[13] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *arXiv preprint arXiv:1807.11164*, 2018.

[14] Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964.

[15] K. Sun, M. Li, D. Liu, and J. Wang. Igcv3: Interleaved low-rank group convolutions for efficient deep neural networks. *arXiv preprint 1806.00178v1*, 2018.

[16] J. Vogelstein, J. Conroy, V. Lyzinski, L. Podrazik, S. Kratzer, E. Harley, D. Fishkind, R. Vogelstein, and C. Priebe. Fast approximate quadratic programming for graph matching. *PloS one*, 10(4), 2015.

[17] Penghang Yin, Yifei Lou, Qi He, and Jack Xin. Minimization of ℓ_{1-2} for compressed sensing. *SIAM J. Sci. Computing*, 37(1):A536–A563, 2015.

[18] M. Zaslavskiy, F. Bach, and J. Vert. A path following algorithm for the graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:2227–2242, 2009.

[19] T. Zhang, G-J Qi, B. Xiao, and J. Wang. Interleaved group convolutions. *CVPR*, pages 4373–4382, 2017.

[20] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.