Phase Selective Convolution

Jamie Menjay Lin[1] Parham Noorzad[1] Yang Yang[1] Nojun Kwak[2] Fatih Porikli[1]

Qualcomm AI Research[1]* Seoul National University[2]

{jmlin,pnoorzad,yyangy,fporikli}@qti.qualcomm.com nojunk@snu.ac.kr

Abstract

This paper introduces Phase Selective Convolution (PSC), an enhanced convolution for more deliberate utilization of activations in convolutional networks. Unlike conventional use of convolutions with activation functions, PSC preserves the full space of activations while supporting desirable model nonlinearity. Similar to several other network operations, e.g., the ReLU operation, at the time of their introduction, PSC may not execute as efficiently on platforms without hardware specialization support. As a first step in addressing the need for optimization, we propose a hardware acceleration scheme to enable the intended efficiency for PSC execution. Moreover, we propose a PSC deployment strategy, with which PSC is applied only to selected layers of the networks, to avoid excessive increase in the total model size. To evaluate the results, we apply PSC as a drop-in replacement for selected convolution layers in several networks without affecting their macro network architectures. In particular, PSC-enhanced ResNets achieve higher accuracies by 1.0-2.0% and 0.7-1.0% on CIFAR-100 and ImageNet, respectively, in Pareto efficiency. PSC-enhanced MobileNets (V2 and V3 Large) and MobileNetV3 (Small) achieve 0.9-1.0% and 1.8% accuracy gains, respectively, on ImageNet at little (0.2-0.7%) total model size increase.

1. Introduction

The activation function of convolutional neural networks (CNNs) plays mixed roles. On the one hand, it enhances the expressive power of CNNs over otherwise an interconnection of linear operations. On the other hand, it is also the reason why characterization for the internal processes regarding the preservability and invertibility of information can be difficult [29, 33]. While it is true that not all the information from the input data needs to be preserved in order for the network to express the desirable functions, disposing too much information through the activation functions in certain portions of the network could create undesirable information loss [16, 33].

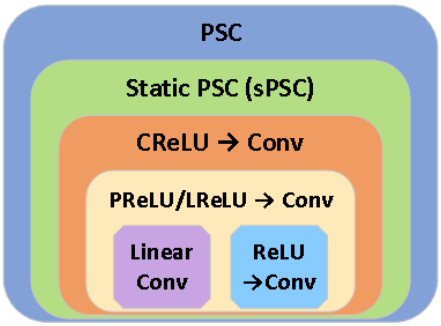


Figure 1. PSC expressive power in comparison to other convolution block types. (Definitions and details are given in 3.1 and 3.2.)

For example, the ReLU [31, 20, 13] activation function, by definition, disposes all negative pre-activations, encouraging the network to turn useful information into the positive space, yet risking variable degrees of information loss. Prior work has studied this issue to some extent. For example, MobileNetV2 [33] deals with the bottleneck architecture and attempts to attain invertability of a convolution layer with ReLU by allocating significantly wider output channels than the input ones. CReLU [34] addresses invertibility by applying ReLU to pre-activations and their negations followed by concatenation of activations. Other works [15, 27, 7, 22, 32, 5, 30, 38] also attempt to address similar problems by introducing new parameters or involving certain portion of negative space for activation functions.

In this paper, we introduce *Phase Selective Convolution* (PSC), a generalization for several classic activation functions, including ReLU [31], PReLU [15], LReLU [27], and CReLU [34], along with the convolution operation. PSC interestingly works as a "plug-and-play" replacement for standard [14, 16] and depthwise [6, 18, 21, 33, 17, 37, 42, 26] convolutions, especially in a building block form, e.g., a

*Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

residual [14] or bottleneck [1] block. To address problems of increased computation and parameters with naive PSC application, we also propose several techniques, as detailed in later sections, to make PSC efficient and practical.

Our contributions in this paper are summarized as follows. (1) We introduce the concepts of static PSC (sPSC) and PSC, as motivated by generalization of several existing activation functions. (2) We discuss activation effects of PSC, for how full space of activations can be leveraged and how network nonlinearity can be increased. (3) We present a PSC hardware acceleration scheme with zero MAC (multiply-accumulate) count increase. (4) We propose weight initialization and deployment strategies for PSC, using examples of ResNets and MobileNets, to avoid excessive weight increase from a naive deployment of PSC.

2. Related Work

The information preservability of ReLU, in a general case, leads to the design of the "inverted residual block" [33]. While such design addresses the information preservability problem to some extent, the problem is not solved completely. Specifically, Sandler *et al.* [33] show that even though a theoretical condition for information preservability is satisfied with random initialization, the problem may persist for a trained network depending on the point of convergence.

Another line of work in addressing this deficiency of ReLU relies on replacing ReLU with other activation functions [27, 15, 34, 7, 22]. For example, Leaky ReLU [27] differs from ReLU by preserving the negative inputs at a small rate. Parametric ReLU [15] further generalizes leaky ReLU by making the preserve rate for negative inputs learnable. Concatenated ReLU (CReLU) [34] applies ReLU to both positive and negative pre-activations, namely

$$x \mapsto \big(\mathrm{ReLU}(x), \mathrm{ReLU}(-x)\big). \tag{1}$$

Note that CReLU preserves information fully, which naturally comes at the cost of doubling complexity. To address this issue, the authors of [34] make changes to their baseline network when applying CReLU; for example, they may add CReLU in only certain layers, and for such layers, they also reduce the number of convolutional filters by half.

One important point to consider is the location where CReLU is applied, not only within the network, but also within a given layer. In [34], a convolution, followed by a CReLU is defined as a layer, whereas in our work, the reverse – an activation function followed by a convolution – is considered. This simple change of viewpoint results in a number of fundamental insights in terms of both architecture design and hardware implementation. While one variation of our PSC module, called "static PSC," is equivalent to applying CReLU prior to a convolutional layer, our main PSC module is more general and captures CReLU as a special case. We describe these ideas in Section 3 in detail.

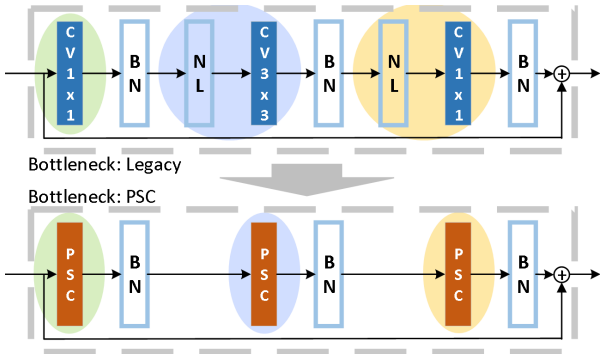


Figure 2. PSC enhances both nonlinear and linear types of layers. Top: A bottleneck block example. Bottom: Drop-in replaceable PSC for all layers to reclaim lost activations and increase nonlinearity. Inclusion of batch normalization in PSC module is optional.

The discussion on activation functions would be incomplete without mentioning swish [32]. Unlike most other activation functions, swish is discovered via automated search. In [32], the authors show that a parameterized version of swish outperforms both LReLU and PReLU on ImageNet. The performance benefits of swish are further demonstrated in the design of MobileNetV3 [17] and EfficientNet [37] models. While the parameterized version of swish provides state-of-the-art results [32], its hardware implementation is costly. Because of this, [17] introduces a "hard" version of swish called "h-swish." While h-swish is more hardware-friendly than swish, it still runs slower than ReLU. In fact, in the same paper, it is demonstrated that using h-swish instead of ReLU in all layers of MobileNetV3-Large adds 20 percent in latency. In Sections 3 and 4, we show the benefit of using PSC with ReLU activation as an even simpler alternative to h-swish.

As a final example on the importance of ReLU, an "IoT-friendly" version of EfficientNet – EfficientNet-EdgeTPU [11], uses ReLU instead of swish. To address performance loss with ReLU, the authors rely on neural architecture search, which results in a network with significantly more MACs but lower latency and improved accuracy. Therefore, the simplicity of ReLU remains an attractive option in certain designs with hardware-awareness.

3. Proposed Method

In this section, we provide the motivation, definition, and analysis for Phase Selective Convolution (PSC). In 3.1, we start with the intuition of simple examples leading to the benefits of PSC. In 3.2, we define PSC and establish that the PSC incurs zero MAC count increase. In 3.3, we provide our perspectives on the activation effects of PSC in nonlinear and linear layers. In 3.4, we propose an acceleration design that ensures not only zero MAC count increase for PSC but also low overhead in other non-MAC operations.

In 3.5, we discuss efficient PSC weight initialization methods for fine-tuning from pre-trained networks.

3.1. Motivation

We first define *Static* PSC (sPSC), a special case of PSC, starting with a simple baseline example as follows

$$[y_1, y_2] = [w_1, w_2] \times \mathrm{ReLU}(x)$$
$$[z_1, z_2] = [y_1, y_2] + [b_1, b_2], \tag{2}$$

where all variables are scalars for simplicity, with x's, w's, y's, b's and z's being the pre-activations, weights, activations, biases, and final activations, respectively. With slight modification from the baseline, now we define sPSC as

$$[y_1, y_2] = [w_1, w_2] \odot [\mathrm{ReLU}(x), -\mathrm{ReLU}(-x)]$$
$$[z_1, z_2] = [y_1, y_2] + [b_1, b_2]. \tag{3}$$

Given the sPSC definition above with $\odot$ denoting the Hadamard product, it is straightforward to see that sPSC preserves both the number of parameters and MAC count. In this case since we have only one scalar input and two scalar outputs, we use a single weight w_1 when $x > 0$ and a single scalar weight w_2 when x is negative. In the more general case of a higher dimensional output, an equal split between positive and negative pre-activations might no longer be optimal. For example, it might be optimal to allocate 60 percent of the weights to positive inputs in one layer, while only 25 percent in another. In our Supplementary Material, we introduce a systematic method to determine how to best allocate the numbers of weights between positive and negative inputs. This ability to support unequal weight allocation allows sPSC to be more general than CReLU [34].

We next demonstrate the power of sPSC compared to our baseline module (2) in a simple regression task. Say we seek to find the function $g\colon [-1, 1] \to \mathbb{R}$ that best approximates the function $f\colon [-1, 1] \to \mathbb{R}$, where $f(x) = |x|$ for each x with respect to the L_2-loss

$$\mathcal{L}(f, g) \triangleq \int_{-1}^{1} \left| f(x) - g(x) \right|^2 dx.$$

Let g be limited to functions which consist of the baseline computational module, as described by (2), followed by a pooling unit. The pooling unit takes the vector input $[z_1, z_2]$ and outputs the scalar $z = z_1 + z_2$. Then it is possible to show that for any such g,

$$\mathcal{L}(f, g) > 0.$$

That is, the L_2-loss is bounded above zero. (In Supplementary Materials we provide the derivation.) However, if we let g be an sPSC module, it is quite straightforward that simply setting $w_1 = -w_2 = 1$ and $b_1 = b_2 = 0$ gives

$$\mathcal{L}(f, g) = 0.$$

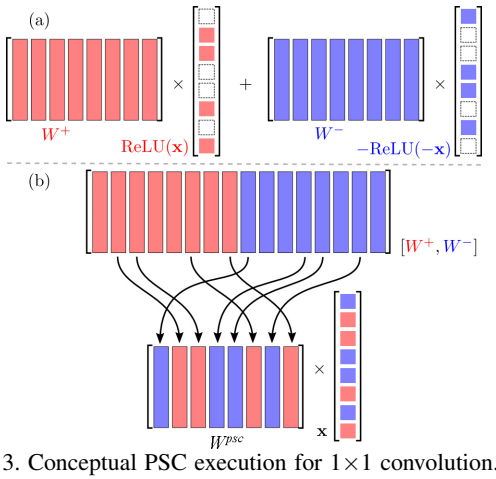


Figure 3. Conceptual PSC execution for 1×1 convolution. (a) A naive approach with straight sum of separate convolutions by signs of activations. (b) Arithmetically equivalent to (a), PSC for one consolidated convolution with the *dynamic kernel* (W^{psc}), keeping total MACs unchanged from the conventional convolution.

The sPSC module slightly modifies the network by splitting a layer into two parts – one for positive and the other for negative inputs. One question that arises is whether it is possible to benefit from negative inputs without modifying the network. To answer this question, we introduce PSC, which has a *dynamic* nature, in contrast to sPSC, in its treatment of the inputs. For the baseline defined by (2), the PSC counterpart is given by

$$[y_{11}, y_{12}] = [w_{11}, w_{12}] \times \mathrm{ReLU}(x)$$
$$[y_{21}, y_{22}] = [w_{21}, w_{22}] \times \big(-\mathrm{ReLU}(-x) \big) \tag{4}$$
$$[z_1, z_2] = [y_{11} + y_{21}, y_{12} + y_{22}] + [b_1, b_2].$$

Equation (4) describes PSC during training, as it connects the input via the weights to the output in a differentiable manner. During inference, PSC may be simplified as

$$[y_1, y_2] = [w_1, w_2] \times x$$
$$[z_1, z_2] = [y_1, y_2] + [b_1, b_2], \tag{5}$$

where

$$[w_1, w_2] = \begin{cases} [w_{11}, w_{12}] & \text{if } x \geq 0, \\ [w_{21}, w_{22}] & \text{else.} \end{cases}$$

Due to the fact that PSC, as defined by (5), preserves the structure of the baseline module, it can serve as a *drop-in replacement* to the baseline network to improve performance. Even though the network is split into two branches during training, the network structure remains unchanged from the baseline during inference. As a result, replacing a layer with PSC preserves the MAC count.

The second benefit of PSC is *generality*. In addition to sPSC, PSC also generalizes modules based on LReLU or

PReLU. To see this, first consider sPSC. In (4), if we set

$$w_{12} = w_{21} = 0, \tag{6}$$

then PSC specializes into sPSC with weights (w_{11}, w_{22}). If, in addition to (6), we further set

$$w_{22} = a \times w_{11} \tag{7}$$

for some scalar a, then PSC specializes into the baseline module (2) with the ReLU replaced by a LReLU or PReLU with parameter a. Furthermore, we remark that, similar to the above example demonstrating the strict benefit of sPSC over the baseline module, it is possible to construct similar regression tasks in which PSC has a strict advantage over sPSC due to its enhanced expressive power. That is, one may set up regression tasks in which PSC achieves zero loss while the counterpart loss of sPSC is lower bounded by a positive number. Figure 1 summarizes our discussion.

3.2. Definition

In this subsection, we define PSC in the (general) tensor input case for various types of convolutions.

Consider first a matrix-vector product for 1×1 convolution after a ReLU. Denoting the weight matrix as W and the input vector as $\mathbf{x}$, the output $\mathbf{y}$ may be expressed as

$$\mathbf{y} = W \times \text{ReLU}(\mathbf{x}). \tag{8}$$

Note that ReLU discards some information by zeroing all negative entries of $\mathbf{x}$ in (8). Denoting $\mathbf{x}^+ \triangleq \text{ReLU}(\mathbf{x})$ and $\mathbf{x}^- \triangleq -\text{ReLU}(-\mathbf{x})$, we may express PSC as

$$\mathbf{y} = W^+\mathbf{x}^+ + W^-\mathbf{x}^-, \tag{9}$$

where W^+ and W^- denote the weight matrix associated with positive and negative activations, respectively. The computation is illustrated in Figure 3(a).

A salient observation reminds that, since $\mathbf{x}^+$ and $\mathbf{x}^-$ must *complement* each other by definition, the convolution operation using a merged *dynamic kernel* is possible, i.e.,

$$\mathbf{y} = W^+\mathbf{x}^+ + W^-\mathbf{x}^- \triangleq W^{\text{PSC}}\mathbf{x}, \tag{10}$$
$$W^{\text{PSC}}_{:j} = \begin{cases} W^+_{:j} & \mathbf{x}_j \geq 0 \\ W^-_{:j} & \mathbf{x}_j < 0 \end{cases},$$

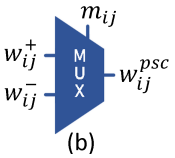
where the columns of matrix W^{PSC} are loaded from either W^+ or W^- according to the signs of corresponding entries in $\mathbf{x}$, and $:j$ denotes the j^{th} column of the associated matrix.

The PSC expression for 1×1 convolution in (10) also applies to fully connected (linear) operations. For other types of convolutions, including full convolution and depthwise (e.g., 3×3) convolution, PSC may be similarly expressed as

$$\begin{aligned} Y &= W^+ \odot X^+ + W^- \odot X^- \\ &\triangleq W^{\text{PSC}} \odot X, \end{aligned} \tag{11}$$
$$w^{\text{PSC}}_{ij} = \begin{cases} w^+_{ij}, & x_{ij} \geq 0 \\ w^-_{ij}, & x_{ij} < 0 \end{cases},$$

Input Select	Input Line 0	Input Line 1	Output
$m_{ij} = 0$	w^+_{ij}	w^-_{ij}	$w^{psc}_{ij} = w^+_{ij}$
$m_{ij} = 1$			$w^{psc}_{ij} = w^-_{ij}$

(a)



(b)

Figure 4. Weight Selection Unit (WSU) for PSC acceleration. (a) Truth table for WSU. (b) Multiplexer operation serving the WSU.

where $X^+ = [x^+_{ij}]$, $X^- = [x^-_{ij}]$, $W^+ = [w^+_{ij}]$, $W^- = [w^-_{ij}]$, $W^{\text{PSC}} = [w^{\text{PSC}}_{ij}]$, and X, X^+, X^-, W^{PSC}, W^+, and W^- are all in same shape.

3.3. Activation Effects

In this subsection, we provide our perspectives on the activation effects of PSC. We also give an example of how PSC is applied as a drop-in replaceable module.

We consider restrictions associated with two types of convolution layers – a nonlinear layer, which follows a nonlinear (activation) function, and a linear layer otherwise. For discussion simplicity, we limit the types of nonlinear functions to ReLU. For a nonlinear layer, information is restricted (and lost) in the input, as ReLU zeros out all negative activations. For a linear layer, model nonlinearity is also restricted, as the linear layer by definition cannot accommodate a prior nonlinear operation.[1]

PSC helps both types of layers. For a nonlinear layer, PSC takes both signs of activations to reclaim otherwise the lost information, which is empirically shown useful (as discussed in 4.3.) For a linear layer, PSC takes full space of the input and divides activations by their signs to perform selective convolution, therefore increasing model nonlinearity. Figure 2 shows an MobileNet bottleneck block example, in which both nonlinear and linear types of layers can be replaced with PSC for improved model accuracy.

3.4. Hardware Acceleration

In 3.2, we have established that PSC can be applied in a way to incur zero extra MAC from conventional convolution. In this subsection, we delve into some more execution details and propose a hardware acceleration design, which ensures low overhead in these non-MAC operations.

In the hardware acceleration design, we propose three sub-operations to collectively perform PSC. We remark that the first two are parallel or batch execution of non-MAC sub-operations, and the third operation is exact same convolution as in the conventional case.

1. **Sign Masking:** Parallel zero-MAC sub-operation that batch extracts[2] sign bits of input X into a mask M (similar to batch execution of ReLU on tensor signs).

$$M = [m_{ij}] = \text{sign\_ext}(X). \tag{12}$$

[1] Lower nonlinearity in model leads to lower model expressive power.
[2] We define $y = sign\_ext(x) = 1$ for $x < 0$ and $y = 0$ otherwise.

2. **Weight Selection:** Parallel zero-MAC sub-operation that batch selects weights by the sign mask, as facilitated by vectorized *Weight Selection Units* (WSUs) hardware. Figure 4 shows the design for one single unit of the WSU hardware.[3]

$$w_{ij}^{\mathrm{PSC}} = \begin{cases} w_{ij}^{+}, & m_{ij} = 0 \\ w_{ij}^{-}, & m_{ij} = 1 \end{cases}, \quad W^{\mathrm{PSC}} = [w_{ij}^{\mathrm{PSC}}], \tag{13}$$

where $W^{+} = [w_{ij}^{+}]$, and $W^{-} = [w_{ij}^{-}]$.

3. **Convolution:** Convolution (same operation as in conventional case) between the weight W^{PSC} and input X.

$$Y = W^{\mathrm{PSC}} \odot X, \tag{14}$$

where we assume the same type of convolution as in (11). In the case of pointwise convolution, we simply replace the "$\odot$" operation with the matrix-vector multiplication and replace X with $\mathbf{x}$, similar to (10).

3.5. Weight Initialization

In this subsection, we introduce PSC weight initialization methods for fine tuning from a pre-trained network.

Similar to [15, 10], we seek to find suitable initialization methods for the PSC module. While networks with PSC can certainly be trained from randomly initialized weight just like for other networks, such initialization may not always produce the best accuracy. Furthermore, when a baseline network is already well trained, it may be desirable to take such pre-trained weights to initialize our target PSC network for fine tuning to save training time. We propose two PSC weight initialization methods that work very well empirically, as inspired from Net2Net [4], where a larger network (e.g., wider or deeper) is initialized from a smaller network such that the function computed by the larger network initially is arithmetically identical to the smaller network. The principle is that if in every step of gradient descent the loss function is reduced by a non-negative amount, then the performance of the larger network would be at least equal to the smaller network.

- **Incremental Nonlinear Initialization (INI):** Incremental initialization for a nonlinear PSC layer (that follows a ReLU operation) such that the target network, even though larger, is equivalent to the source network.

$$W^{+} = W^{0} \quad \text{and} \quad W^{-} = \mathbf{0}, \tag{15}$$

where W^{+} and W^{-} are PSC weights, and W^{0} is the pre-trained weight of the corresponding non-PSC layer. We refer to this as **INI**, since inserting (15) to (10), we get

$$\begin{aligned} y &= W^{+}\mathbf{x}^{+} + W^{-}\mathbf{x}^{-} = W^{0}\mathbf{x}^{+} + \mathbf{0}\mathbf{x}^{-} \\ &= W^{0}\mathbf{x}^{+} = W^{0} \cdot \mathrm{ReLU}(\mathbf{x}). \end{aligned} \tag{16}$$

- **Incremental Linear Initialization (ILI):** Incremental initialization for a linear PSC layer (that does not follow a nonlinear operation) such that the target network, even though larger, is equivalent to the source network.

$$W^{+} = W^{-} = W^{0}. \tag{17}$$

We refer to this as **ILI**, since inserting (17) to (10), we get

$$\begin{aligned} y &= W^{+}\mathbf{x}^{+} + W^{-}\mathbf{x}^{-} = W^{0}\mathbf{x}^{+} + W^{0}\mathbf{x}^{-} \\ &= W^{0}(\mathbf{x}^{+} + \mathbf{x}^{-}) = W^{0}x. \end{aligned} \tag{18}$$

In (16) and (18), our methods are shown to initialize the target (PSC) network to be equivalent to the pre-trained source network, whether the source layer is linear or not.[4] The elegance of these initialization techniques and their decent performance make us wonder whether further theoretical insight exists. This is subject to future work.

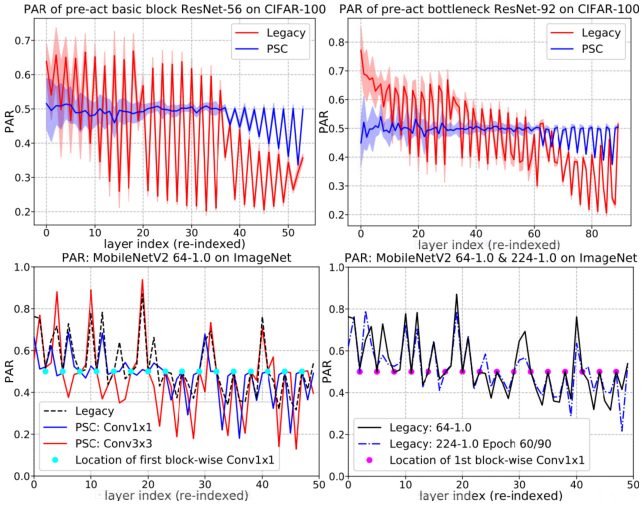


Figure 5. Positive activation ratio (PAR) analysis: In all subfigures shown in various scenarios, ResNet and MobileNetV2 demonstrate similar behavior in preserving information in initial layers and disposing later, as only positive pre-activations stay over ReLU. ResNet with PSC, however, has a PAR of around 50% in most layers for utilization of all activations. For MobileNetV2, when PSC is applied to only one layer (Conv3x3 or Conv1x1 projection) of each bottleneck block, those layers of PSC also tend to keep near-50% PARs. MobileNetV2 for different image resolutions (224x224 and 64x64) shows similar PAR behaviors over most layers of the networks.

[3] One acceleration design example is to co-locate (next to each other) each pair of positive and negative (PN) weights in weight memory, such that weight selection is efficiently executed as *batch binary weight selection* on the PN-paired weight tensor according to the sign mask.

[4] Fine tuning for each increment takes only a smaller number of epochs empirically, as training of the target network starts from a good state.

4. Experiments

In this section, we present experiments by training both baseline and proposed PSC networks. In 4.1, we provide a PSC design examples and study *positive-activation ratio* (PAR). We then discuss our proposed PSC deployment strategy. In 4.2, we show experiments with ResNe, MobileNetV2, and MobileNetV3 on CIFAR-100 and ImageNet. In 4.3, we compare PSC with an interesting dual-positive variant. In 4.4, we discuss execution latency. Results are shown in Table 1 and Figures 5, 6, and 7.

4.1. Implementation and Deployment

4.1.1 PSC Module Design for A Single Layer

In 3.2, we have shown the *complementary* property of the merged PSC *dynamic kernel* with proposed use of the ReLU function. In 3.4, we have also shown our proposed scheme for PSC hardware acceleration, which includes sign masking and weight selection using non-MAC hardware logic units as explained to be similar to the hardware logic design for ReLU and as illustrated in Figure 4. With those, the execution of one single PSC module may be accelerated.

4.1.2 Positive Activation Ratio Analysis

To understand how activations of opposite phases interplay in the network, we define a metric, *Positive Activation Ratio* (PAR), as the ratio between the number of positive activations and that of all activations for a given *PAR observation point* (POP). In this study, we place one POP immediately prior to each convolution layer. Figure 5 shows results for ResNet and MobileNetV2. Our main observations are

- PSC significantly reduces PAR swings over layers.
- PSC tends to stay much closer to the PAR $= 0.5$ line.
- Swings match with the basic/bottleneck block sizes.
- PAR variance significantly reduces in later layers.
- Swing range shifts towards lower PARs in later layers.

In Figures 5, the PAR swings somehow reflect the block structure. For example, we observe that the baseline ResNet tends to permit more activations in initial layers and reject more activations in later layers, leading to more maxima and more minima in the initial and final layers, respectively.

The PAR observation is also interesting on MobileNetV2 for three variant models shown: baseline (i.e., no PSC), PSC in pointwise projection layers, and PSC in depthwise layers. First, the PARs at the expansion layers are always near 0.5 regardless of the models, as those layers have no preceding ReLUs. Second, PARs at the projection layers for PSC also are close to 0.5, while PARs at the depthwise convolutions show no obvious consistency, most likely resulting from the smaller number of kernel entries per channel. Finally, we see downward trend with the swings over layers, somewhat similar to with ResNet.

PAR also suggests the utilization of activations. In contrast, a PSC layer does not suffer from such information loss, as activations in either phase are always convolved by a PSC kernel; that is, PSC has *full* utilization of activations.

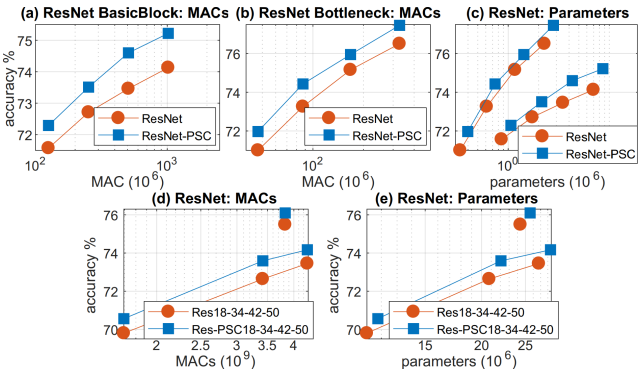


Figure 6. We train and test both baseline and PSC counterpart ResNets from scratch in *all-identical* settings. In (a), (b), and (c) on CIFAR-100, PSC applied to front 2/3 layers of ResNet-29/56/110/218/434 outperforms legacy ResNet of either basic or bottleneck blocks. In (c), PSC is more efficient even when comparing over parameters in bottom and top curves for basic and bottleneck blocks, respectively. In (d) and (e) on ImageNet, PSC applied to front half layers of ResNet also outperforms. Top two points are for ResNet50 and rest are for shallower ResNets.

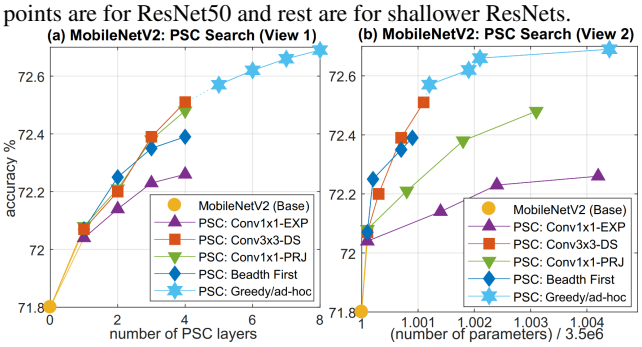


Figure 7. We discover that our proposed MobileNetV2/V3-PSC gain particularly well with suboptimal (restricted) search using incremental initialization and training. (a) shows accuracy over number of (applied) PSC layers, and (b) shows accuracy over number of parameters relative to the baseline (1X). We see that PSC on depthwise convolutions achieves best among all three types of convolutions in a bottleneck block. The "suboptimal" curve shows search results with greedy/ad-hoc layer choices for PSC. Our best MobileNetV2-PSC result from the overall suboptimal search has PSC applied in layers $\{0, 1, 3, 4, 6, 7, 9, 10\}$, which achieves 0.9% accuracy gain at only 0.4% total parameter increase.

4.1.3 PSC Deployment Strategy

The PAR analysis in 4.1.2 suggests that a full-network application for PSC may not be necessary. Even though PSC

in theory (see Figure 1 and 3.1) generalizes several types of classic activation functions and should perform equal to or better than its non-PSC counterpart, applying PSC to more layers than necessary would only increase parameters at diminishing or no gains. We summarize our empirical PSC deployment strategy as follows.

- We prioritize on front layers as possible for PSC application, as a front layer tends to have fewer channels and hence smaller parameter count.
- We prefer depthwise convolution to pointwise one as possible for PSC application, as the former has less model size impact.
- We apply PSC also to the stem layer (namely, the first convolution layer in the network) as possible.

4.2. Image Classification

In this section, we experiment on PSC with several CNN architectures, involving residual and bottleneck blocks as well as standard, pointwise, and depthwise convolutions. To train each instance of networks with PSC (applied only in selected layers) in our experiments with ResNets, we train both the baseline and our proposed PSC networks in *all-identical* settings, including hyperparameters, optimizer, and learning rate schedule. For MobileNets and EfficientNets, we take the pre-trained weights from the baseline and fine tune the PSC layers by 40-50 epochs on a cosine annealing schedule with one-tenth learning rates from the original under SGD optimizer of the unchanged loss function. We discover that our proposed Mobilenets with PSC respond pretty well with non-trivial accuracy gains using such training setup.

4.2.1 ResNet on CIFAR-100 and ImageNet

We evaluate PSC on ResNet [16]. Given our PAR analysis in 4.1.2 and Figure 5, which show a tendency towards higher PARs in front layers of the network, we choose the front portion of the layers in the network to apply (drop-in replaceable) PSC. Figure 6 shows our experiment results, where we observe consistent accuracy gains with PSC in the range of 0.7-1.1% over both computation complexity and model size, we apply PSC to the front $2/3$ layers of ResNet-29/56/110/218/434 on CIFAR-100 [23] and front half layers of ResNet-18/34/42/50 on ImageNet [8].[5]

4.2.2 MobileNetV2 and MobileNetV3 on ImageNet

MobileNetV2 [33] and MobileNetV3 [17] are a family of CNN architectures that employs depthwise separable convolutions and the inverted residual bottleneck structure to handle feature sizes over the pointwise expansion and projection layers according to the expansion ratio. Unlike the other two layers in the bottleneck block, the first convolution layer in the bottleneck block has no preceding ReLU (and hence a linear layer). We can apply (drop-in replaceable) PSC to any of these layers, whether linear or not.

Network Architecture	Top-1 Acc	Acc Gain	MACs (mil)	Params (mil)
MobileNetV3-S	66.4%	-	60	**2.94**
MobileNetV3-S-PSC (ours)	**68.2%**	**1.8%**	60	2.96
MobileNetV2	71.8%	-	300	**3.50**
MobileNetV2-PSC (ours)	**72.7%**	**0.9%**	300	3.51
MobileNetV3-L	72.6%	-	220	**5.48**
MobileNetV3-L-PSC (ours)	**73.6%**	**1.0%**	220	5.49
EfficientNet-B0	77.0%	-	402	**5.29**
EfficientNet-B0-PSC (ours)	**77.5%**	**0.5%**	402	5.31
EfficientNet-B1	78.9%	-	713	**7.80**
EfficientNet-B1-PSC (ours)	**79.2%**	**0.3%**	713	7.84

Table 1. MobileNet and Efficient results on ImageNet, with PSC hardware acceleration and selected-layer deployment.

While being able to train MobileNets straight from scratch at higher resource costs, in this experiment we showcase the incremental initialization methods, as proposed in 3.5, to leverage from pre-trained non-PSC networks for resource saving and to study PSC effects on different types of convolutions in the bottleneck blocks. Given the freedom to apply PSC or not for each layer, we phrase this as a search problem for the best combination of layer-wise PSC application choices to maximize network efficiency under costs. Instead of applying exhaustive search over 2^L possible networks, with L denoting the number of PSC-applicable layers, we use our PAR insight in 4.1.2 and proceed with a sub-optimal (restricted) global search along same types of convolution of each bottleneck (conceptually similar to depth-first search) and over the layer sequence (conceptually similar to breadth-first search). Over a few iterations, we then take the winners and continue with *ad hoc* "local search" (conceptually similar to the greedy search) until further improvement in performance efficiency cannot be made[6]. Results are shown in Figure 7, Table 1, and Table 2. Remarkably, PSC works for both types of linear and nonlinear layers and these PSC accuracy gains require only $0.2 - 0.7\%$ parameter increase, as only several front layers in the network are applied with PSC.

4.2.3 EfficientNet on ImageNet

EfficientNet is a later family of networks derived with coefficient model scaling in multiple parameter dimensions [37]. Similar to MobileNetV3, which uses a h-swish activation function [32], EfficientNet uses the swish activation function for a subset of network layers. In our experi-

[5] We keep all hyperparameters unchanged for ResNet-PSC, as parameter increase is small, and train ResNet-PSC from scratch.

[6] Another promising search method is Dynamic Programming (DP) with memorization, which takes incremental PSC layers on top of pre-trained weights for initialization and training. This is a subject of our ongoing work.

ment, when we apply (drop-in replaceable) PSC for a layer, the h-swish or swish function of the same layer is replaced along. We again used our proposed incremental initialization methods to initialize and train the PSC network. Results are shown in Table 1.

4.3. PSC vs. Dual-Positive Convolution

To confirm the significance of dual-phase pre-activations utilized by PSC, we evaluate a PSC variant where the negative phase is substituted with the positive phase, resulting in a *dual-positive* convolution that is same as PSC in complexity and model size. I.e., only the negation operation into the PSC modules is switched off for the dual-positive variant. In our results, we observe that PSC consistently outperforms such dual-positive counterparts with ResNet by a significant margin in the range of $[0.7\%, 1.8\%]$ over a wide range of model sizes and complexities.

4.4. Latency Measurements

As PSC benefits from wider space of the input activations to the convolution, it is different enough from standard convolutions with classic activation functions. To enable best efficiency for PSC, we anticipate hardware acceleration for PSC in an inference engine.

To clarify on the feasibility of our acceleration proposal, here we summarize on some design considerations.

- PSC acceleration in our proposal consists of 3 sub-operations as described in 3.4. The first two operations, Sign Masking (SM) and Weight Selection (WS), involve no MAC operations, and the third is identical to the standard (conventional) convolution.
- The logic for the SM sub-operation is similar to that for the ReLU operation, and therefore it can be optimized (just like for ReLU).
- The logic for the WS sub-operation is achievable with *vectorized* WSU operations as proposed in Figure 4.
- To further facilitate weight memory access, we propose a weight structure with *weight parification*, which allocates weight elements of both phases next to each other, to lighten up with local binary selections.

Before PSC acceleration becomes available for a hardware platform, we provide some latency measurements in Table 2 as examples, based on the *naive* design of PSC (Figure 3(a)) in a current platform. I.e., the *unoptimized "double convolutiones"* for each PSC layer with EfficientNet.

5. Conclusion

We discuss the PSC method and considerations. We further propose initialization and acceleration techniques that help enable higher performance and efficiency of PSC with new hardware acceleration. We demonstrate that PSC is more Pareto efficient with ResNets and MobileNets.

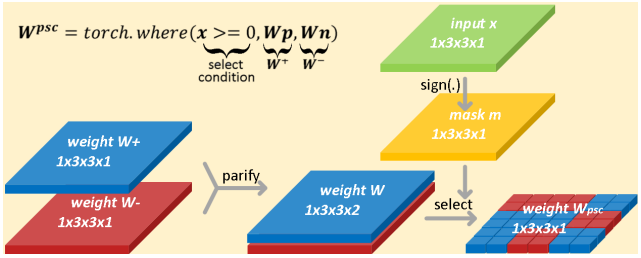


Figure 8. Derivation for W^{psc} with an existing PyTorch function, **torch.where**$(x >= 0, Wp, Wn)$, for non-hardware-acceleration realization to our proposed sub-operations of **sign masking** and **weight selection** in an example of a depthwise 3×3 kernel between $Wp = W^+$ and $Wn = W^-$ in batch processing. We include all required sub-operations and needed sizes of tensor operands for the weights and activations, as described in 3.4, in our PSC acceleration latency estimation. Although these PyTorch commands are not completely identical to our proposed ideal PSC accelertation in the hardware perspective, they are equivalent in number of operations for our latency estimation purpose.

Network Architecture	CPU Latency (ms)	CPU Δ	GPU Latency (ms)	GPU Δ
EfficientNetB0	**49.2**	-	**0.491**	-
EfficientNetB0-PSC (ours)	52.5	+6.7%	0.518	+5.5%
EfficientNetB1	**66.8**	-	**0.675**	-
EfficientNetB1-PSC (ours)	70.7	+5.8%	0.708	+4.9%

Table 2. Estimated inference latency on ImageNet: PyTorch with TeslaV100-SXM2 32GRAM GPU and Intel i7-9700 CPU. Note that this estimation is with a current-generation platform and does ***not*** use our proposed *PSC hardware acceleration*. Please see the "Supplementary Materials" section for more description.

Additionally, we have discussed activation analysis with positive activation ratios (PAR) and selected PSC deployment with method of depth-first search, breadth-first search, and greedy search for suboptimal (restricted) PSC layer search. While in this work we mainly focus on the design and simple restrictive search with the goal of achieving high efficiency, the general problem of where PSC ought to be applied in the network would be best handled by automated neural architecture search (NAS) methods [44, 45, 25, 24]. In particular, hardware-aware NAS algorithms [9, 36, 40, 39, 3, 35, 2, 41] allow one to incorporate metrics in search that may be more important in practice than model size, such as on-device latency or total energy consumption. This is a subject of ongoing work.

While some of these mentioned topics remain open questions, we are pleased to present our recent results and intend to continue with developments in future study.

Acknowledgements

We would like to thank our Qualcomm AI Research colleagues for their support and assistance, in particular that of Kristopher Urquhart, Yash Bhalgat, and Mihir Jain.

References

[1] Rana Ali Amjad and Bernhard C. Geiger. Learning representations for neural network-based classification using the information bottleneck principle. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(9):2225–2239, Sep 2020. 2

[2] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *Int. Conf. Learn. Represent.*, 2020. 8

[3] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. *Int. Conf. Learn. Represent.*, 2019. 8

[4] T. Chen, I. Goodfellow, and J. Shlens. Net2Net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations (ICLR)*, May 2016. 5

[5] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic relu, 2020. 1

[6] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 1

[7] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In *International Conference on Learning Representations (ICLR)*, May 2016. 1, 2

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009. 7

[9] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. DPP-Net: Device-aware progressive search for pareto-optimal neural architectures. *Eur. Conf. Comput. Vis.*, 2018. 8

[10] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256. PMLR, May 2010. 5

[11] S. Gupta and M. Tan. EfficientNet-EdgeTPU: Creating accelerator-optimized neural networks with AutoML. *Google AI Blog*, Aug 2019. 2

[12] Amirhossein Habibian, Ties van Rozendaal, Jakub M. Tomczak, and Taco S. Cohen. Video Compression With Rate-Distortion Autoencoders. *International Conference on Computer Vision*, 2019. 9

[13] Richard H.R. Hahnioser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and Hyunjune Sebastian Seung. Digital selection and analogue amplification coexist in a cortex- inspired silicon circuit. *Nature*, 405(6789):947–951, June 2000. 1

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 1, 2

[15] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *IEEE International Conference on Computer Vision (ICCV)*, December 2015. 1, 2, 5

[16] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *arXiv:1603.05027*, 2016. 1, 7

[17] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam. Searching for MobileNetV3. *arXiv:1905.02244*, May 2019. 1, 2, 7

[18] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, April 2017. 1

[19] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations (ICLR)*, April 2017. 9

[20] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, 2009. 1

[21] L. Kaiser, A. N. Gomez, and F. Chollet. Depthwise separable convolutions for neural machine translation. In *International Conference on Learning Representations (ICLR)*, May 2018. 1

[22] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, December 2017. 1, 2

[23] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 7

[24] Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Blockwisely supervised neural architecture search with knowledge distillation. *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020. 8

[25] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. *Int. Conf. Learn. Represent.*, 2019. 8

[26] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design, 2018. 1

[27] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning (ICML)*, June 2013. 1, 2

[28] C. J. Maddison, A. Mnih, and Y. W. Teh. The Concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, April 2017. 9

[29] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 1

[30] Diganta Misra. Mish: A self regularized non-monotonic activation function, 2020. 1

[31] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010. 1

[32] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv: 1710.05941*, Oct 2017. 1, 2, 7

[33] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 1, 2, 7

[34] W. Shang, K. Sohn, D. Almeida, and H. Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *International Conference on Machine Learning (ICML)*, December 2016. 1, 2, 3

[35] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-Path NAS: Designing hardware-efficient convnets in less than 4 hours. *ECML PKDD*, 2019. 8

[36] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-aware neural architecture search for mobile. *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 8

[37] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning (ICML)*, June 2019. 1, 2, 7

[38] Ludovic Trottier, Philippe Giguère, and Brahim Chaib-draa. Parametric exponential linear unit for deep convolutional neural networks, 2018. 1

[39] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, Peter Vajda, and Joseph E. Gonzalez. FBNetV2: Differentiable neural architecture search for spatial and channel dimensions. *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020. 8

[40] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 8

[41] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. BigNAS: Scaling up neural architecture search with big single-stage models. *Eur. Conf. Comput. Vis.*, 2020. 8

[42] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices, 2017. 1

[43] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating Very Deep Convolutional Networks for Classification and Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):1943–1955, Oct 2016. 9

[44] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *Int. Conf. Learn. Represent.*, 2017. 8

[45] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 8