Neural Architecture Search for Lightweight Non-Local Networks

Yingwei Li[1*] Xiaojie Jin[2] Jieru Mei[1*] Xiaochen Lian[2] Linjie Yang[2]
Cihang Xie[1] Qihang Yu[1] Yuyin Zhou[1] Song Bai[3] Alan Yuille[1]
[1]Johns Hopkins University [2]ByteDance AI Lab [3]University of Oxford

Abstract

Non-Local (NL) blocks have been widely studied in various vision tasks. However, it has been rarely explored to embed the NL blocks in mobile neural networks, mainly due to the following challenges: 1) NL blocks generally have heavy computation cost which makes it difficult to be applied in applications where computational resources are limited, and 2) it is an open problem to discover an optimal configuration to embed NL blocks into mobile neural networks. We propose ***AutoNL*** *to overcome the above two obstacles. Firstly, we propose a Lightweight Non-Local (LightNL) block by squeezing the transformation operations and incorporating compact features. With the novel design choices, the proposed LightNL block is* ***400*** $\times$ ***computationally cheaper*** *than its conventional counterpart without sacrificing the performance. Secondly, by relaxing the structure of the LightNL block to be differentiable during training, we propose an efficient neural architecture search algorithm to learn an optimal configuration of LightNL blocks in an end-to-end manner. Notably, using only* 32 *GPU hours, the searched AutoNL model achieves 77.7% top-1 accuracy on ImageNet under a typical mobile setting (350M FLOPs), significantly outperforming previous mobile models including MobileNetV2 (+5.7%), FBNet (+2.8%) and MnasNet (+2.1%). Code and models are available at* `https://github.com/LiYingwei/AutoNL`.

1. Introduction

Non-Local (NL) block [2, 40] aims to capture long-range dependencies in deep neural networks, which have been used in a variety of vision tasks such as video classification [40], object detection [40], semantic segmentation [47, 49], image classification [2], image captioning [23] and adversarial robustness [42]. Despite the remarkable progress, the general utilization of non-local modules under resource-constrained scenarios such as mobile devices remains underexplored. This may be due to two factors.

*Work done during an internship at Bytedance AI Lab.

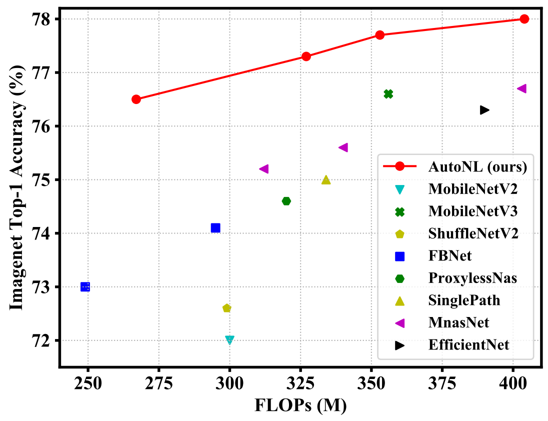

Figure 1. **ImageNet Accuracy *vs.* Computation Cost.** Details can be found in Table 3.

First, NL blocks compute the response at each position by attending to all other positions and computing a weighted average of the features in all positions, which incurs a large computation burden. Several efforts have been explored to reduce the computation overhead. For instance, [7, 22] use associative law to reduce the memory and computation cost of matrix multiplication; Yue *et al.* [45] use Taylor expansion to optimize the non-local module; Cao *et al.* [4] compute the affinity matrix via a convolutional layer; Bello *et al.* [2] design a novel attention-augmented convolution. However, these methods either still lead to relatively large computation overhead (via using heavy operators, such as large matrix multiplications) or result in a less accurate outcome (*e.g.*, simplified NL blocks [4]), making these methods undesirable for mobile-level vision systems.

Second, NL blocks are usually implemented as individual modules which can be plugged into a few manually selected layers (usually relatively deep layers). While it is intractable to densely embed it into a deep network due to the high computational complexity, it remains unclear where to insert those modules economically. Existing methods have not fully exploited the capacity of NL blocks in relational modeling under mobile settings.

Taking the two factors aforementioned into account, we aim to answer the following questions in this work: *is it pos-*

sible to develop an efficient NL block for mobile networks? What is the optimal configuration to embed those modules into mobile neural networks? We propose **AutoNL** to address these two questions. First, we design a Lightweight Non-Local (LightNL) block, which is the first work to apply non-local techniques to mobile networks to our best knowledge. We achieve this with two critical design choices 1) lighten the transformation operators (*e.g.*, 1×1 convolutions) and 2) utilize compact features. As a result, the proposed LightNL blocks are usually **400× computationally cheaper** than conventional NL blocks [40], which is favorable to be applied to mobile deep learning systems. Second, we propose a novel neural architecture search algorithm. Specifically, we relax the structure of LightNL blocks to be differentiable so that our search algorithm can simultaneously determine the compactness of the features and the locations for LightNL blocks during the end-to-end training. We also reuse intermediate search results by acquiring various affinity matrices in one shot to reduce the redundant computation cost, which speeds up the search process.

Our proposed searching algorithm is fast and delivers high-performance lightweight models. As shown in Figure 1, our searched small AutoNL model achieves 76.5% ImageNet top-1 accuracy with 267M FLOPs, which is faster than MobileNetV3 [16] with comparable performance (76.6% top-1 accuracy with 356M FLOPs). Also, our searched large AutoNL model achieves 77.7% ImageNet top-1 accuracy with 353M FLOPs, which has similar computation cost as MobileNetV3 but improves the top-1 accuracy by 1.1%.

To summarize, our contributions are three-fold: (1) We design a lightweight and search compatible NL block for visual recognition models on mobile devices and resource-constrained platforms; (2) We propose an efficient neural architecture search algorithm to automatically learn an optimal configuration of the proposed LightNL blocks; 3) Our model achieves state-of-the-art performance on the ImageNet classification task under mobile settings.

2. Related Work

Attention mechanism. The attention mechanism has been successfully applied to neural language processing in recent years [1, 37, 11]. Wang *et al.* [40] bridge attention mechanism and non-local operator, and use it to model long-range relationships in computer vision applications. Attention mechanisms can be applied along two orthogonal directions: channel attention and spatial attention. Channel attention [18, 38, 28] aims to model the relationships between different channels with different semantic concepts. By focusing on a part of the channels of the input feature and deactivating non-related concepts, the models can focus on the concepts of interest. Due to its simplicity and effectiveness [18], it is widely used in neural architecture search [34, 36, 16, 9].

Our work explores in both directions of spatial/channel attention. Although existing works [7, 45, 4, 2, 39] exploit various techniques to improve efficiency, they are still too computationally heavy under mobile settings. To alleviate this problem, we design a lightweight spatial attention module with low computational cost and it can be easily integrated into mobile neural networks.

Efficient mobile architectures. There are a lot of handcrafted neural network architectures [19, 43, 17, 31, 46, 26] for mobile applications. Among them, the family of MobileNet [17, 31] and the family of ShuffleNet [46, 26] stand out due to their superior efficiency and performance. MobileNetV2 [31] proposes the inverted residual block to improve both efficiency and performance over MobileNetV1 [17]. ShuffleNet [26] proposes to use efficient shuffle operations along with group convolutions to design efficient networks. Above methods are usually subject to trial-and-errors by experts in the model design process.

Neural Architecture Search. Recently, it has received much attention to use neural architecture search (NAS) to design efficient network architectures for various applications [35, 13, 24, 44, 21]. A critical part of NAS is to design proper search spaces. Guided by a meta-controller, early NAS methods either use reinforcement learning [51] or evolution algorithm [30] to discover better architectures. These methods are computationally inefficient, requiring thousands of GPU days to search. ENAS [29] shares parameters across sampled architectures to reduce the search cost. DARTS [25] proposes a continuous relaxation of the architecture parameters and conducts one-shot search and evaluation. These methods all adopt a NASNet [51] like search space. Recently, more expert knowledge in handcrafting network architectures are introduced in NAS. Using MobileNetV2 basic blocks in search space [3, 41, 34, 36, 32, 14, 27] significantly improves the performance of searched architectures. [3, 14] reduce the GPU memory consumption by executing only part of the super-net in each forward pass during training. [27] proposes an ensemble perspective of the basic block and simultaneously searches and trains the target architecture in the fine-grained search space. [32] proposes a super-kernel representation to incorporate all architectural hyper-parameters (*e.g.*, kernel sizes, expansion rations in MobileNetV2 blocks) in a unified search framework to reuse model parameters and computations. In our proposed searching algorithm, we focus on seeking an optimal configuration of LightNL blocks in low-cost neural networks which brought significant performance gains.

3. AutoNL

In this section, we present AutoNL: we first elaborate on how to design a Lightweight Non-Local (LightNL) block in

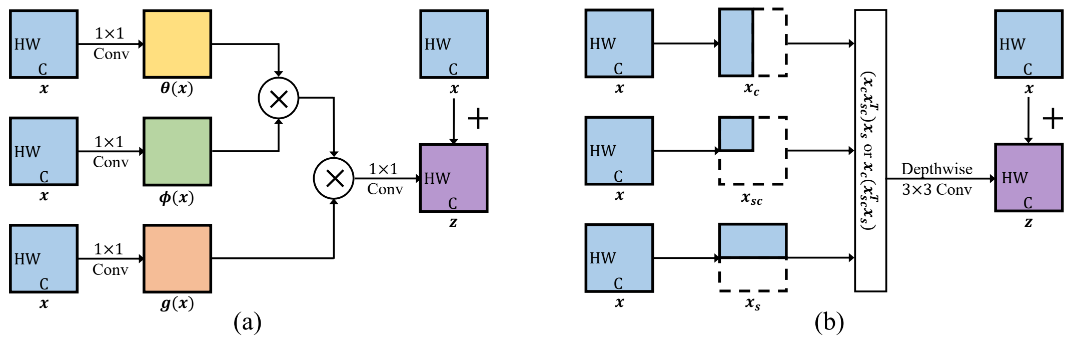

Figure 2. **Original NL *vs.* LightNL Block.** (a) A typical architecture of the NL block contains several heavy operators, such as 1×1 convolution ops and large matrix multiplications. (b) The proposed LightNL block contains much more lightweight operators, such as depthwise convolution ops and small matrix multiplications.

Section 3.1; then we introduce a novel neural architecture search algorithm in Section 3.2 to automatically search for an optimal configuration of LightNL blocks.

3.1. Lightweight Non-Local Blocks

In this section, we first revisit the NL blocks, then we introduce our proposed Lightweight Non-Local (LightNL) block in detail.

Revisit NL blocks. The core component in the NL blocks is the non-local operation. Following [40], a generic non-local operation can be formulated as

$$\mathrm{y}_i = \frac{1}{\mathcal{C}(\mathrm{x})} \sum_{\forall j} f(\mathrm{x}_i, \mathrm{x}_j) g(\mathrm{x}_j), \tag{1}$$

where i indexes the position of input feature x whose response is to be computed, j enumerates all possible positions in x, $f(\mathrm{x}_i, \mathrm{x}_j)$ outputs the affinity matrix between x_i and its context features x_j, $g(\mathrm{x}_j)$ computes an embedding of the input feature at the position j, and $C(\mathrm{x})$ is the normalization term. Following [40], the non-local operation in Eqn. (1) is wrapped into a NL block with a residual connection from the input feature x. The mathematical formulation is given as

$$\mathrm{z}_i = W_z \mathrm{y}_i + \mathrm{x}_i, \tag{2}$$

where W_z denotes a learnable feature transformation.

Instantiation. Dot product is used as the function form of $f(x_i, x_j)$ due to its simplicity in computing the correlation between features. Eqn. (1) thus becomes

$$\mathrm{y} = \frac{1}{\mathcal{C}(\mathrm{x})} \theta(\mathrm{x}) \theta(\mathrm{x})^{\mathrm{T}} g(\mathrm{x}). \tag{3}$$

Here the shape of x is denoted as (H, W, C) where H, W and C are the height, width and number of channels, respectively. $\theta(\cdot)$ and $g(\cdot)$ are 1×1 convolutional layers with C filters. Before matrix multiplications, the outputs of 1×1 convolution are reshaped to $(H \times W, C)$.

Levi *et al.* [22] discover that for NL blocks instantiated in the form of Eqn. (3), employing the associative law of matrix multiplication can largely reduce the computation overhead. Based on the associative rules, Eqn. (3) can be written in two equivalent forms:

$$\mathrm{y} = \frac{1}{\mathcal{C}(\mathrm{x})} \left(\theta(\mathrm{x})\theta(\mathrm{x})^{\mathrm{T}}\right) g(\mathrm{x}) = \frac{1}{\mathcal{C}(\mathrm{x})} \theta(\mathrm{x}) \left(\theta(\mathrm{x})^{\mathrm{T}} g(\mathrm{x})\right). \tag{4}$$

Although the two forms produce the same numerical results, they have different computational complexity [22]. Therefore in computing Eqn. (3), one can always choose the form with smaller computation cost for better efficiency.

Design principles. The following part introduces two key principles to reduce the computation cost of Eqn. (3).

Design principle 1: Share and lighten the feature transformations. Instead of using two different transformations (θ and g) on the same input feature x in Eqn. (3), we use a shared transformation in the non-local operation. In this way, the computation cost of Eqn. (3) is significantly reduced by reusing the result of $g(\mathrm{x})$ in computing the affinity matrix. The simplified non-local operation is

$$\mathrm{y} = \frac{1}{\mathcal{C}(\mathrm{x})} g(\mathrm{x}) g(\mathrm{x})^{\mathrm{T}} g(\mathrm{x}). \tag{5}$$

The input feature x (output of hidden layer) can be seen as the transformation of input data x_0 through a feature transformer $F(\cdot)$. Therefore Eqn. (5) can be written as

$$\mathrm{y} = \frac{1}{\mathcal{C}(F(\mathrm{x}_0))} g(F(\mathrm{x}_0)) g(F(\mathrm{x}_0))^{\mathrm{T}} g(F(\mathrm{x}_0)). \tag{6}$$

In the scenario of using NL blocks in neural networks, $F(\cdot)$ is represented by a parameterized deep neural network. In contrast, $g(\cdot)$ is a single convolution operation. To further simplify Eqn. (6), we integrate the learning process of $g(\cdot)$ into that of $F(\cdot)$. Taking advantage of the strong capability of deep neural networks on approximating functions [15], we remove $g(\cdot)$ and Eqn. (6) is simplified as

$$\mathrm{y} = \frac{1}{\mathcal{C}(\mathrm{x})} \mathrm{x}\mathrm{x}^{\mathrm{T}} \mathrm{x}. \tag{7}$$

At last, we introduce our method to simplify "W_z", another heavy transformation function in Eqn. (2). Recent works [40] instantiate it as a 1×1 convolutional layer. To further reduce the computation cost of NL blocks, we propose to replace the 1×1 convolution with a 3×3 depthwise convolution [17] since the latter is more efficient. Eqn. (2) is then modified to be

$$\mathrm{z} = \mathrm{DepthwiseConv}(\mathrm{y}, W_d) + \mathrm{x}, \tag{8}$$

where W_d denotes the depthwise convolution kernel.

Design principle 2: Use compact features for computing affinity matrices. Since x is a high-dimensional feature, directly performing matrix multiplication using the full-sized x per Eqn. (7) leads to large computation overhead. To solve this problem, we propose to downsample x first to obtain a more compact feature which replaces x in Eqn. (7). Since x is a three-dimensional feature with depth (channels), width and height, we propose to downsample x along either **channel dimension**, **spatial dimension** or **both dimensions** to obtain compact features $\mathrm{x_c}$, $\mathrm{x_s}$ and $\mathrm{x_{sc}}$ respectively. Consequently, the computation cost of Eqn. (7) is reduced.

Therefore, based on Eqn. (7), we can simply apply the compact features $\{\mathrm{x_c}, \mathrm{x_{sc}}, \mathrm{x_s}\}$ in the NL block to compute $\mathrm{x_c}\mathrm{x_{sc}^T}$ and $\mathrm{x_{sc}^T}\mathrm{x_s}$ as

$$y = \frac{1}{\mathcal{C}(\mathrm{x})} \mathrm{x}_c \mathrm{x}_{sc}^{\mathrm{T}} \mathrm{x}_s. \tag{9}$$

Note that there is a trade-off between the computation cost and the representation capacity of the output (*i.e.*, y) of the non-local operation: using more compact features (with a lower downsampling ratio) reduces the computation cost but the output fails to capture the informative context information in those discarded features; on the other hand, using denser features (with a higher downsampling ratio) helps the output capture richer contexts, but it is more computationally demanding. Manually setting the downsampling ratios requires trial-and-errors. To solve this issue, we propose a novel neural network architecture search (NAS) method in Section 3.2 to efficiently search for the configuration of NL blocks that achieve descent performance under specific resource constraints.

Before introduce our NAS method, let's briefly summarize the advantages of the proposed LightNL blocks. Thanks to the aforementioned two design principles, our proposed LightNL block is empirically demonstrated to be much more efficient (refer to Section 4) than the conventional NL block [40], making it favorable to be deployed in mobile devices with limited computational budgets. In addition, since the computational complexity of the blocks can be easily adjusted by the downsampling ratios, the proposed LightNL blocks can provide better support on deep learning models at different scales. We illustrate the structure of the conventional NL block and that of the proposed block in Figure 2.

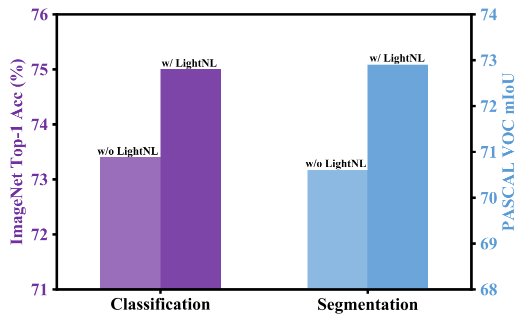

Figure 3. **MobileNetV2 *vs.* MobileNetV2 + LightNL.** The proposed LightNL block improves the baseline by 1.6% in ImageNet top-1 accuracy and 2.3% in PASCAL VOC 2012 mIoU.

3.2. Neural Architecture Search for LightNL

To validate the efficacy and generalization of the proposed LightNL block for deep networks, we perform a proof test by applying it to every MobileNetV2 block. As shown in Figure 3, such a simple way of using the proposed LightNL blocks can already significantly boost the performance on both image classification and semantic segmentation. This observation motivates us to search for a better configuration of the proposed LightNL blocks in neural networks to fully utilize its representation learning capacity. As can be seen in Section 3.1, except for the insert locations in a neural network, the downsampling scheme that controls the complexity of the LightNL blocks is another important factor to be determined. We note that both insert locations and downsampling schedule of LightNL blocks are critical to the performance and computational cost of models. To automate the process of model design and find an optimal configuration of the proposed LightNL blocks, we propose an efficient Neural Architecture Search (NAS) method. Concretely, we propose to jointly search the configurations of LightNL blocks and the basic neural network architectural parameters (*e.g.*, kernel size, number of channels) using a cost-aware loss function.

Insert location. Motivated by [32], we select several candidate locations for inserting LightNL blocks throughout the network and decide whether a LightNL block should be used by comparing the L_2 *norm* of the depthwise convolution kernel W_d to a trainable latent variable t:

$$\hat{W}_d = \mathbb{1}\left(\|W_d\|^2 > t\right) \cdot W_d, \tag{10}$$

where $\hat{W}_d$ replaces W_d to be used in Eqn. (8), and $\mathbb{1}(\cdot)$ is an indicator function. $\mathbb{1}(\cdot) = 1$ indicates that a LightNL block will be used with $\hat{W}_d = W_d$ being the depthwise convolution kernel. Otherwise, $\hat{W}_d = 0$ when $\mathbb{1}(\cdot) = 0$ and thus Eqn. (8) is degenerated to $z = x$ meaning no lightweight non-local block will be inserted.

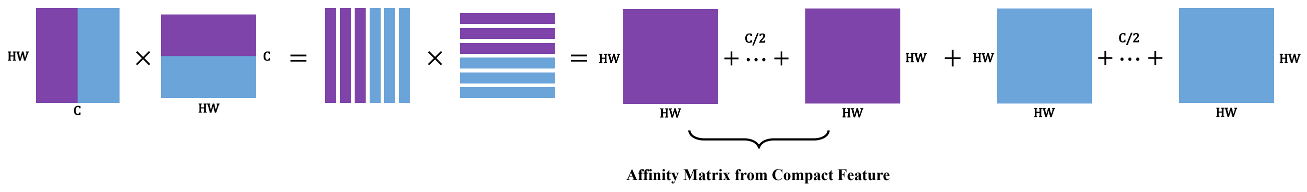

Figure 4. Illustrate the feature reuse paradigm along channel dimension.

Instead of manually selecting the value of threshold t, we set it to be a trainable parameter, which is jointly optimized with other parameters via gradient decent. To compute the gradient of t, we relax the indicator function $I(x,t) = \mathbb{1}(x > t)$ to a differentiable sigmoid function $\sigma(\cdot)$ during the back-propagation process.

Module compactness. As can be seen from Eqn. (7), the computational cost of LightNL block when performing the matrix multiplication is determined by the compactness of downsampled features. Given a search space R which contains n candidate downsampling ratios, *i.e.*, $R = \{r_1, r_2, r_3, ..., r_n\}$ where $0 \le r_1 < r_2 < r_3 < ... < r_n \le 1$, our goal is to search for an optimal downsampling ratio r^* for each LightNL block. For the sake of clarity, here we use the case of searching downsampling ratios along the channel dimension to illustrate our method. Note that searching downsampling ratios along other dimensions can be performed in the same manner.

Different from searching for the insert locations through Eqn. (10), we encode the choice of downsampling ratios in the process of computing affinity matrix:

$$\begin{aligned} \mathrm{x_{att}} =& \mathbb{1}(r_1) \cdot \mathrm{x_{r_1}} \mathrm{x_{r_1}^T} + \mathbb{1}(r_2) \cdot \mathrm{x_{r_2}} \mathrm{x_{r_2}^T} + ... \\ &+ \mathbb{1}(r_{n-1}) \cdot \mathrm{x_{r_{n-1}}} \mathrm{x_{r_{n-1}}^T} + \mathbb{1}(r_n) \cdot \mathrm{x_{r_n}} \mathrm{x_{r_n}^T}, \end{aligned} \tag{11}$$

where $\mathrm{x_{att}}$ denotes the computed affinity matrix, x_r denotes the downsampled feature with downsampling ratio r, and $\mathbb{1}(r)$ is an indicator which holds true when r is selected. By setting the constraint that only one downsampling ratio is used, Eqn. (11) can be simplified as $\mathrm{x_{att}} = \mathrm{x_{r_i}} \mathrm{x_{r_i}^T}$ when r_i is selected as the downsampling ratio.

A critical step is how to formulate the condition of $\mathbb{1}(\cdot)$ for deciding which downsampling ratio to use. A reasonable intuition is that the criteria should be able to determine whether the downsampled feature can be used to compute an accurate affinity matrix. Thus, our goal is to define a "similarity" signal that models whether the affinity matrix from the downsampled feature is close to the "ground-truth" affinity matrix, denotes as $\mathrm{x_{gt}} \mathrm{x_{gt}^T}$. Specifically, we write the indicator as

$$\begin{aligned} \mathbb{1}(r_1) =& \mathbb{1}\left(\|\mathrm{x_{r_1}} \mathrm{x_{r_1}^T} - \mathrm{x_{r_{gt}}} \mathrm{x_{gt}^T}\|^2 < t\right), \\ \mathbb{1}(r_2) =& \mathbb{1}\left(\|\mathrm{x_{r_2}} \mathrm{x_{r_2}^T} - \mathrm{x_{gt}} \mathrm{x_{gt}^T}\|^2 < t\right) \wedge \neg \mathbb{1}(r_1), \\ & ... \\ \mathbb{1}(r_n) =& \mathbb{1}\left(\|\mathrm{x_{r_n}} \mathrm{x_{r_n}^T} - \mathrm{x_{gt}} \mathrm{x_{gt}^T}\|^2 < t\right) \\ & \wedge \neg \mathbb{1}(r_1) \wedge \neg \mathbb{1}(r_2) \wedge ... \wedge \neg \mathbb{1}(r_{n-1}), \end{aligned} \tag{12}$$

where $\wedge$ denotes the logical operator AND. An intuitive explanation to the rational of Eqn. (12) is the algorithm always selects the smallest r with which the Euclidean distance between $\mathrm{x_r} \mathrm{x_r}^T$ and $\mathrm{x_{gt}} \mathrm{x_{gt}}^T$ is lower than threshold t. To ensure $\mathbb{1}(r_n) = 1$ when all other indicators are zeros, we set $\mathrm{x_{gt}} = \mathrm{x_{r_n}}$ so that $\mathbb{1}\left(\|\mathrm{x_{r_n}} \mathrm{x_{r_n}^T} - \mathrm{x_{gt}} \mathrm{x_{gt}^T}\|^2 < t\right) \equiv 1$. Meanwhile, we relax the indicator function to sigmoid when computing gradients and update the threshold t via gradient descent. Since the output of indicator changes with different input feature x, for better training convergence, we get inspired from batch normalization [20] and use the exponential moving average of affinity matrices in computing Eqn. (12). After the searching stage, the downsampling ratio is determined by evaluating the following indicators:

$$\begin{aligned} \mathbb{1}(r_1) =& \mathbb{1}\left(\mathrm{EMA}(\|\mathrm{x_{r_1}} \mathrm{x_{r_1}^T} - \mathrm{x_{r_{gt}}} \mathrm{x_{gt}^T}\|^2) < t\right), \\ & ... \\ \mathbb{1}(r_n) =& \mathbb{1}\left(\mathrm{EMA}(\|\mathrm{x_{r_n}} \mathrm{x_{r_n}^T} - \mathrm{x_{gt}} \mathrm{x_{gt}^T}\|^2) < t\right) \\ & \wedge \neg \mathbb{1}(r_1) \wedge \neg \mathbb{1}(r_2) \wedge ... \wedge \neg \mathbb{1}(r_{n-1}). \end{aligned} \tag{13}$$

where $\mathrm{EMA}(\mathrm{x})$ denotes the exponential moving averaged value of x.

From Eqn. (12), one can observe that the output of indicator depends on indicators with smaller downsampling ratio. Based on this finding, we propose to reuse the affinity matrix computed with low-dimensional features (generated with lower downsampling ratios) when computing affinity matrix with high-dimensional features (generated with higher downsampling ratios). Concretely, $\mathrm{x_{r_i}}$ can be partitioned into $[\mathrm{x_{r_{i-1}}}, \mathrm{x_{r_i \backslash r_{i-1}}}]$, $i > 1$. The calculation of affinity matrix using $\mathrm{x_{r_i}}$ can be decomposed as

$$\begin{aligned} \mathrm{x_{r_i}} \mathrm{x_{r_i}^T} &= \begin{bmatrix} \mathrm{x_{r_{i-1}}} & \mathrm{x_{r_i \backslash r_{i-1}}} \end{bmatrix} \begin{bmatrix} \mathrm{x_{r_{i-1}}^T} \\ \mathrm{x_{r_i \backslash r_{i-1}}^T} \end{bmatrix} \\ &= \mathrm{x_{r_{i-1}}} \mathrm{x_{r_{i-1}}^T} + \mathrm{x_{r_i \backslash r_{i-1}}} \mathrm{x_{r_i \backslash r_{i-1}}^T}, \end{aligned} \tag{14}$$

where $\mathrm{x_{r_{i-1}}} \mathrm{x_{r_{i-1}}^T}$ is the reusable affinity matrix computed with a smaller downsampling ratio (recall that $r_{i-1} < r_i$). This feature reusing paradigm can largely reduce the search overhead since computing affinity matrices with more choices of downsampling ratios does not incur any additional computation cost. The process of feature reusing is illustrated in Figure 4.

Searching process. We integrate our proposed search algorithm with Single-path NAS [32] and jointly search basic architectural parameters (following MNasNet [34]) along with the insert locations and downsampling schemes of LightNL blocks. We search downsampling ratios along both spatial and channel dimensions to achieve better compactness. To learn efficient deep learning models, the overall objective function is to minimize both standard classification loss and the model's computation complexity which is related to both the insert locations and the compactness of LightNL blocks:

$$\min \mathcal{L}(\mathbf{w}, \mathbf{t}) = CE(\mathbf{w}, \mathbf{t}) + \lambda \cdot \log(CC(\mathbf{t})), \qquad (15)$$

where $\mathbf{w}$ denotes model weights and $\mathbf{t}$ denotes architectural parameters which can be grouped in two categories: one is from LightNL block including the insert positions and downsampling ratios while the other follows MNasNet [34] including kernel size, number of channels, *etc.* $CE(\cdot)$ is the cross-entropy loss and $CC(\cdot)$ is the computation (*i.e.*, FLOPs) cost. We use gradient descent to optimize the above objective function in an end-to-end manner.

4. Experiments

We first demonstrate the efficacy and efficiency of LightNL by manually inserting it into lightweight models in Section 4.1. Then we apply the proposed search algorithm to the LightNL blocks in Section 4.2. The evaluation and comparison with state-of-the-art methods are done on ImageNet classification [10].

4.1. Manually Designed LightNL Networks

Models. Our experiments are based on MobileNetV2 1.0 [31]. We insert LightNL blocks after the second 1×1 point-wise convolution layer in every MobileNetV2 block. We use 25% channels to compute the affinity matrix for the sake of low computation cost. Also, if the feature map is larger than 14×14, we downsample it along the spatial axis with a stride of 2. We call the transformed model MobileNetV2-LightNL for short. We compare the two models with different depth multipliers, including 0.5, 0.75, 1.0 and 1.4.

Training setup. Following the training schedule in MNasNet [34], we train the models using the synchronous training setup on 32 Tesla-V100-SXM2-16GB GPUs. We use an initial learning rate of 0.016, and a batch size of 4096 (128 images per GPU). The learning rate linearly increases to 0.256 in the first 5 epochs and then is decayed by 0.97 every 2.4 epochs. We use a dropout of 0.2, a weight decay of $1e{-5}$ and Inception image preprocessing [33] of size 224×224. Finally, we use exponential moving average on model weights with a momentum of 0.9999. All batch normalization layers use a momentum of 0.99.

ImageNet classification results. We compare the results between the original MobileNetV2 and MobileNetV2-LightNL in Figure 5. We observe consistent performance gain even without tuning the hyper-parameters of LightNL blocks for models with different depth multipliers. For example, when the depth multiplier is 1, the original MobileNetV2 model achieves a top-1 accuracy of 73.4% with 301M FLOPs, while our MobileNetV2-LightNL achieves 75.0% with 316M FLOPs. According to Figure 5, it is unlikely to boost the performance of the MobileNetV2 model to the comparable performance by simply increasing the width to get a 316M FLOPs model. When the depth multiplier is 0.5, LightNL blocks bring a performance gain of 2.2% with a marginal increase in FLOPs (6M).

Ablation study. To diagnose the proposed LightNL block, we present a step-by-step ablation study in Table 1. As shown in the table, every modification preserves the model performance but reduces the computation cost. By comparing with the baseline model, the proposed LightNL block

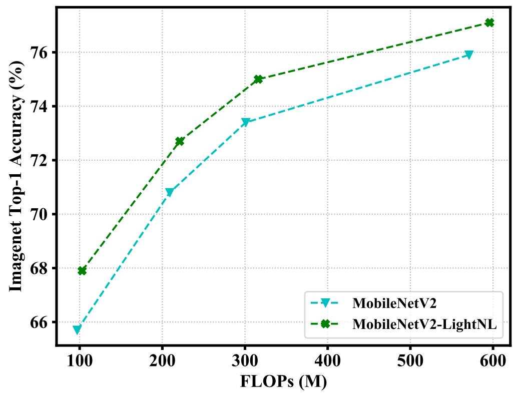

Figure 5. **MobileNetV2 *vs.* MobileNetV2-LightNL.** We apply LightNL blocks to MobileNetV2 with different depth multipliers, *i.e.*, 0.5, 0.75, 1.0, 1.4, from left to right respectively. Despite inserting LightNL blocks manually, consistent performance gains can be observed for different MobileNetV2 base models.

Non-local Module		FLOPs /	Acc (%)
Operator	Wrapper	ΔFLOPs	
-	-	301M	73.4
Wang *et al.* [40]	Wang *et al.* [40]	+6.2G	75.2
Levi *et al.* [22]		+146M	75.2
Zhu *et al.* [50]		+107M	-
Eqn. (3)	Wang *et al.* [40]	+119M	75.2
Eqn. (5)		+93M	75.1
Eqn. (7)		+66M	75.0
Eqn. (9)		+38M	75.0
Eqn. (9)	Eqn. (8)	**+15M**	**75.0**

Table 1. **Ablation Analysis.** We present the comparison of different NL blocks and different variants in our design. The base model is MobileNetV2, which achieves a top-1 accuracy of 73.4 with 301M FLOPs.

Method	FLOPs (M)	mIoU
MobileNetV2	301	70.6
MobileNetV2-LightNL (**ours**)	**316**	**72.9**

Table 2. Comparison of FLOPs and mIoU on PASCAL VOC 2012.

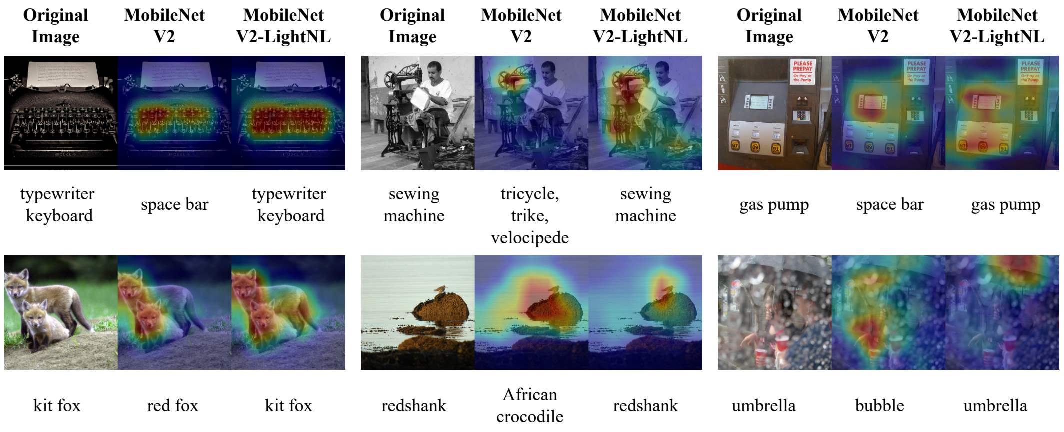

Figure 6. Class Activation Map (CAM) [48] for MobileNetV2 and MobileNetV2-LightNL. The three columns correspond to the ground truth, predictions by MobileNetV2 and predictions by MobileNetV2-LightNL respectively. The proposed LightNL block helps the model attend to image regions with more class-specific discriminative features.

improves ImageNet top-1 accuracy by 1.6% (from 73.4% to 75.0%), but only increases 15M FLOPs, which is only 5% of the total FLOPs on MobileNetV2. Comparing with the standard NL block, the proposed LightNL block is about 400× computationally cheaper (6.2G *vs.* 15M) with comparable performance (75.2% *vs.* 75.0%). Comparing with Levi *et al.* [22] which optimized the matrix multiplication with the associative law, the proposed LightNL block is still 10× computationally cheaper. Compared with a very recent work proposed by Zhu *et al.* [50] which leverages the pyramid pooling to reduce the complexity, LightNL is around 7× computationally cheaper.

CAM visualization. In order to illustrate the efficacy of our LightNL, Figure 6 compares the class activation map [48] for the original MobileNetV2 and MobileNetV2-LightNL. We see that LightNL is capable of helping the model to focus on more relevant regions while it is much computationally cheaper than the conventional counterparts as analyzed above. For example, at the middle top of Figure 6, the model without the LightNL blocks focus on only a part of the sewing machine. When LightNL is applied, the model can "see" the whole machine, leading to more accurate and robust predictions.

PASCAL VOC segmentation results. To demonstrate the generalization ability of our method, we compare the performance of MobileNetV2 and MobileNetV2-LightNL on the PASCAL VOC 2012 semantic segmentation dataset [12]. Following Chen *et al.* [8], we use the classification model as a drop-in replacement for the backbone feature extractor in the Deeplabv3 [6]. It is cascaded by an Atrous Spatial Pyramid Pooling module (ASPP) [5] with three 3×3 convolutions with different atrous rates. The modified architectures share the same computation costs as the backbone models due to the low computation cost of LightNL blocks. All models are initialized with ImageNet pre-trained weights and then fine-tuned with the same training protocol in [5]. It should be emphasized here that the focus of this part is to assess the efficacy of the proposed LightNL while keeping other factors fixed. It is notable that we do not adopt complex training techniques such as multi-scale and left-right flipped inputs, which may lead to better performance. The results are shown in Table 2, LightNL blocks bring a performance gain of 2.3 in mIoU with a minor increase in FLOPs. The results indicate the proposed LightNL blocks are well suitable for other tasks such as semantic segmentation.

Model	#Params	Flops	Top-1	Top-5
MobileNetV2 [31]	3.4M	300M	72.0	91.0
MBV2 (our impl.)	3.4M	301M	73.4	91.4
ShuffleNetV2 [26]	3.5M	299M	72.6	-
FBNet-A [41]	4.3M	249M	73.0	-
Proxyless [3]	4.1M	320M	74.6	92.2
MnasNet-A1 [34]	3.9M	312M	75.2	92.5
MnasNet-A2	4.8M	340M	75.6	92.7
AA-MnasNet-A1 [2]	4.1M	350M	75.7	92.6
MobileNetV3-L	5.4M	217M	75.2	-
MixNet-S [36]	4.1M	256M	75.8	92.8
AutoNL-S (ours)	4.4M	**267M**	**76.5**	**93.1**
FBNet-C [41]	5.5M	375M	74.9	-
Proxyless (GPU) [3]	-	465M	75.1	92.5
SinglePath [32]	4.4M	334M	75.0	92.2
SinglePath (our impl.)	4.4M	334M	74.7	92.2
FairNAS-A [9]	4.6M	388M	75.3	92.4
EfficientNet-B0 [35]	5.3M	388M	76.3	93.2
SCARLET-A [9]	6.7M	365M	76.9	93.4
MBV3-L (1.25x) [16]	7.5M	356M	76.6	-
MixNet-M [36]	5.0M	360M	77.0	93.3
AutoNL-L (ours)	5.6M	**353M**	**77.7**	**93.7**

Table 3. Comparison with the state-of-the-art models on ImageNet 2012 `Val` set.

4.2. AutoNL

We apply the proposed neural architecture search algorithm to search for an optimal configuration of LightNL blocks. Specifically, we have five LightNL candidates for

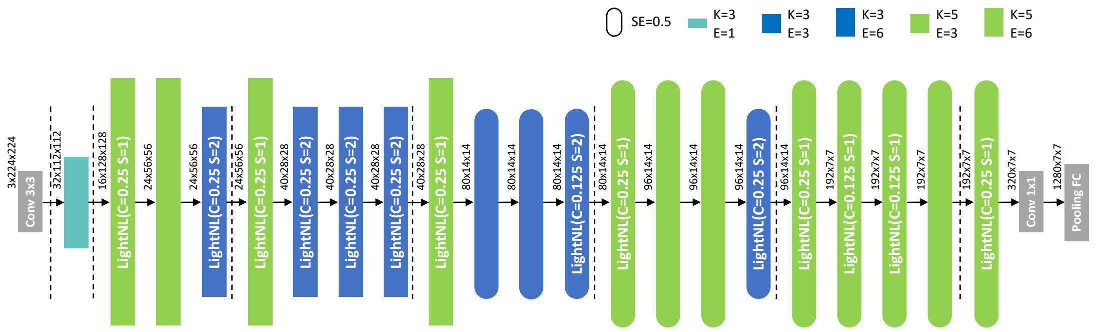

Figure 7. The searched architecture of AutoNL-L. C and S denote channel downsampling ratio and the stride of spatial downsampling respectively. We use different colors to denote the kernel size (K) of the depthwise convolution and use height to denote the expansion rate (E) of the block. We use the round corner to denote adding SE [18] to the MobileNetV2 block.

each potential insert location, *i.e*., sampling 25% or 12.5% channels to compute affinity matrix, sampling along spatial dimensions with stride 1 or 2, inserting a LightNL block at the current position or not. Note that it is easy to enlarge the search space by including other LightNL blocks with more hyper-parameters. In addition, similar to recent work [34, 41, 3, 32], we also search for optimal kernel sizes, optimal expansion ratios and optimal SE ratios with MobileNetV2 block [31] as the building block.

We directly search on the ImageNet training set and use a computation cost loss and the cross-entropy loss as guidance, both of which are differentiable thanks to the relaxations of the indicator functions during the back-propagation process. It takes 8 epochs (about 32 GPU hours) for the search process to converge.

Performance on classification. We obtain two models using the proposed neural architecture search algorithm; we denote the large one as AutoNL-L and the small one as AutoNL-S in Table 3. The architecture of AutoNL-L is presented in Figure 7.

Table 3 shows that AutoNL outperforms all the latest mobile CNNs. Comparing to the handcrafted models, AutoNL-S improves the top-1 accuracy by 4.5% over MobileNetV2 [31] and 3.9% over ShuffleNetV2 [26] while saving about 10% FLOPs. Besides, AutoNL achieves better results than the latest models from NAS approaches. For example, compared to EfficientNet-B0, AutoNL-L improves the top-1 accuracy by 1.4% while saving about 10% FLOPs. Our models also achieve better performance than the latest MobileNetV3 [16], which is developed with several manual optimizations in addition to architecture search.

AutoNL-L also surpasses the state-of-the-art NL method (*i.e*., AA-MnasNet-A1) by 2% with comparable FLOPs. Even AutoNL-S improves accuracy by 0.8% while saving 25% FLOPs. We also compare with MixNet, which is a very recent state-of-the-art model under mobile settings, both AutoNL-L and AutoNL-S achieve 0.7% improvement with comparable FLOPs but with much less search time (32 GPU hours *vs*. 91,000 GPU hours [41], 2,800× faster).

We also search for models under different combinations

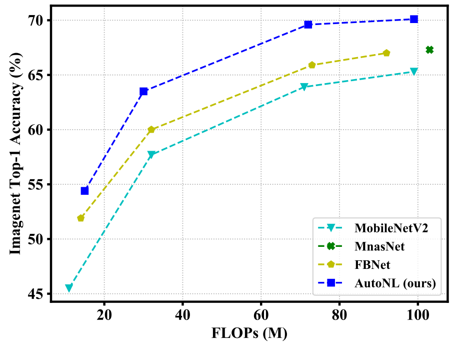

Figure 8. Performance comparison on different input resolutions and depth multipliers under extremely low FLOPs. For MobileNetV2 [31], FBNet [41] and our searched models, the tuples of (input resolution, depth multiplier) are $(96, 0.35)$, $(128, 0.5)$, $(192, 0.5)$ and $(128, 1.0)$ respectively from left to right. For MNasNet [34], we show the result of 128 input resolution with 1.0 depth multiplier.

of input resolutions and channel sizes under extremely low FLOPs. The results are summarized in Figure 8. AutoNL achieves consistent improvement over MobileNetV2, FBNet, and MNasNet. For example, when the input resolution is 192 and the depth multiplier is 0.5, our model achieves 69.6% accuracy, outperforming MobileNetV2 by 5.7% and FBNet by 3.7%.

5. Conclusion

As an important building block for various vision applications, NL blocks under mobile settings remain underexplored due to their heavy computation overhead. To our best knowledge, AutoNL is the first method to explore the usage of NL blocks for general mobile networks. Specifically, we design a LightNL block to enable highly efficient context modeling in mobile settings. We then propose a neural architecture search algorithm to optimize the configuration of LightNL blocks. Our method significantly outperforms prior arts with 77.7% top-1 accuracy on ImageNet under a typical mobile setting (350M FLOPs).

Acknowledgements This work was partially supported by ONR N00014-15-1-2356.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015. 2

[2] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V Le. Attention augmented convolutional networks. *arXiv preprint arXiv:1904.09925*, 2019. 1, 2, 7

[3] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. 2, 7, 8

[4] Yue Cao, Jiarui Xu, Stephen Lin, Fangyun Wei, and Han Hu. Gcnet: Non-local networks meet squeeze-excitation networks and beyond. *arXiv preprint arXiv:1904.11492*, 2019. 1, 2

[5] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 40(4):834–848, 2017. 7

[6] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 7

[7] Yunpeng Chen, Yannis Kalantidis, Jianshu Li, Shuicheng Yan, and Jiashi Feng. Aˆ 2-nets: Double attention networks. In *NeurIPS*, pages 352–361, 2018. 1, 2

[8] Yukang Chen, Gaofeng Meng, Qian Zhang, Shiming Xiang, Chang Huang, Lisen Mu, and Xinggang Wang. Renas: Reinforced evolutionary neural architecture search. In *CVPR*, pages 4787–4796, 2019. 7

[9] Xiangxiang Chu, Bo Zhang, Jixiang Li, Qingyuan Li, and Ruijun Xu. Scarletnas: Bridging the gap between scalability and fairness in neural architecture search. *CoRR*, abs/1908.06022, 2019. 2, 7

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 6

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186, 2019. 2

[12] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *IJCV*, 111(1):98–136, 2015. 7

[13] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *CVPR*, pages 7036–7045, 2019. 2

[14] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *CoRR*, abs/1904.00420, 2019. 2

[15] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. 3

[16] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. *arXiv preprint arXiv:1905.02244*, 2019. 2, 7, 8

[17] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2, 4

[18] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, pages 7132–7141, 2018. 2, 8

[19] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 5

[21] Qing Jin, Linjie Yang, and Zhenyu Liao. Adabits: Neural network quantization with adaptive bit-widths. *arXiv preprint arXiv:1912.09666*, 2019. 2

[22] Hila Levi and Shimon Ullman. Efficient coarse-to-fine non-local module for the detection of small objects. *arXiv preprint arXiv:1811.12152*, 2018. 1, 3, 6, 7

[23] Zhuowan Li, Quan Tran, Long Mai, Zhe Lin, and Alan Yuille. Context-aware group captioning via self-attention and contrastive features. In *CVPR*, 2020. 1

[24] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Autodeeplab: Hierarchical neural architecture search for semantic image segmentation. In *CVPR*, pages 82–92, 2019. 2

[25] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019. 2

[26] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018. 2, 7, 8

[27] Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. Atomnas: Fine-grained end-to-end neural architecture search. In *ICLR*, 2020. 2

[28] Jongchan Park, Sanghyun Woo, Joon-Young Lee, and In So Kweon. BAM: bottleneck attention module. In *BMVC*, page 147, 2018. 2

[29] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, pages 4092–4101, 2018. 2

[30] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *AAAI 2019*, pages 4780–4789, 2019. 2

[31] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 2, 6, 7, 8

[32] 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. *arXiv preprint arXiv:1904.02877*, 2019. 2, 4, 5, 7, 8

[33] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017. 6

[34] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019. 2, 6, 7, 8

[35] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, pages 6105–6114, 2019. 2, 7

[36] Mingxing Tan and Quoc V Le. Mixnet: Mixed depthwise convolutional kernels. In *BMVC*, 2019. 2, 7

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017. 2

[38] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification. In *CVPR*, pages 6450–6458, 2017. 2

[39] Huiyu Wang, Yukun Zhu, Bradley Green, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Axial-deeplab: Stand-alone axial-attention for panoptic segmentation. *arXiv preprint arXiv:2003.07853*, 2020. 2

[40] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, pages 7794–7803, 2018. 1, 2, 3, 4, 6

[41] 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. In *CVPR*, pages 10734–10742, 2019. 2, 7, 8

[42] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. In *CVPR*, 2019. 1

[43] Guotian Xie, Jingdong Wang, Ting Zhang, Jianhuang Lai, Richang Hong, and Guo-Jun Qi. IGCV2: interleaved structured sparse convolutional neural networks. In *CVPR*, 2018. 2

[44] Qihang Yu, Dong Yang, Holger Roth, Yutong Bai, Yixiao Zhang, Alan L Yuille, and Daguang Xu. C2fnas: Coarse-to-fine neural architecture search for 3d medical image segmentation. In *CVPR*, 2020. 2

[45] Kaiyu Yue, Ming Sun, Yuchen Yuan, Feng Zhou, Errui Ding, and Fuxin Xu. Compact generalized non-local network. In *NeurIPS*, pages 6510–6519, 2018. 1, 2

[46] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018. 2

[47] Hengshuang Zhao, Yi Zhang, Shu Liu, Jianping Shi, Chen Change Loy, Dahua Lin, and Jiaya Jia. Psanet: Point-wise spatial attention network for scene parsing. In *ECCV*, pages 267–283, 2018. 1

[48] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *CVPR*, pages 2921–2929, 2016. 7

[49] Yuyin Zhou, David Dreizin, Yingwei Li, Zhishuai Zhang, Yan Wang, and Alan Yuille. Multi-scale attentional network for multi-focal segmentation of active bleed after pelvic fractures. In *International Workshop on Machine Learning in Medical Imaging*, pages 461–469. Springer, 2019. 1

[50] Zhen Zhu, Mengde Xu, Song Bai, Tengteng Huang, and Xiang Bai. Asymmetric non-local neural networks for semantic segmentation. In *ICCV*, pages 593–602, 2019. 6, 7

[51] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 2