Deploying Image Deblurring across Mobile Devices: A Perspective of Quality and Latency

Cheng-Ming Chiang* Yu Tseng Yu-Syuan Xu Hsien-Kai Kuo Yi-Min Tsai
Guan-Yu Chen Koan-Sin Tan Wei-Ting Wang Yu-Chieh Lin Shou-Yao Roy Tseng
Wei-Shiang Lin Chia-Lin Yu BY Shen Kloze Kao Chia-Ming Cheng
Hung-Jen Chen
MediaTek Inc., Hsinchu, Taiwan

Abstract

Recently, image enhancement and restoration have become important applications on mobile devices, such as super-resolution and image deblurring. However, most state-of-the-art networks present extremely high computational complexity. This makes them difficult to be deployed on mobile devices with acceptable latency. Moreover, when deploying to different mobile devices, there is a large latency variation due to the difference and limitation of deep learning accelerators on mobile devices. In this paper, we conduct a search of portable network architectures for better quality-latency trade-off across mobile devices. We further present the effectiveness of widely used network optimizations for image deblurring task. This paper provides comprehensive experiments and comparisons to uncover the in-depth analysis for both latency and image quality. Through all the above works, we demonstrate the successful deployment of image deblurring application on mobile devices with the acceleration of deep learning accelerators. To the best of our knowledge, this is the first paper that addresses all the deployment issues of image deblurring task across mobile devices. This paper provides practical deployment-guidelines, and is adopted by the championship-winning team in NTIRE 2020 Image Deblurring Challenge on Smartphone Track.

1. Introduction

Deep learning based networks have achieved great successes in image enhancement and restoration tasks [38, 57, 19, 37, 34]. Among these applications, image deblurring have become one of the most important camera features for mobile devices [52]. Due to the large input resolution and the characteristic of pixel-to-pixel mapping nature, these contemporary networks demand extremely high complexity and memory footprint. This makes deploying an image deblurring task on mobile devices a great challenge.

In recent years, deep learning communities have noticed the gap between network design and its deployment on mobile devices. Image enhancement and restoration on smartphone contests have been held to shed a light upon this problem [33, 46]. Meanwhile, deep learning accelerators are also widely adopted in most of mobile devices [12, 10, 9, 15, 5]. Following the trend, some benchmark suites are proposed to evaluate the performance of these mobile devices [32, 49, 2, 4]. To alleviate the burden of network deployment, some papers propose light-weight network architectures to reduce the complexity [31, 18, 21, 25, 30]. Another idea is network optimization, which targets on arbitrary network architectures. Among the existing technologies, quantization [35] and pruning [27] are two of the most popular techniques to optimize network performance.

However, the applicability of these optimization techniques with respect to image deblurring task is rarely discussed. Furthermore, the performance of a network is highly affected by the hardware limitations and preferences. Therefore, network portability is another key factor to deploy across a set of mobile devices. Last but not the least, the existing benckmarking efforts lack of a realistic setting to well reflect the practical use-cases, *e.g.*, 720p High-Definition (HD) input resolution (1280 × 720).

In this paper, we compare both quality and latency index of different image deblurring networks across mobile devices. Practical settings are adopted to reflect real user scenarios. Our contributions are summarized as below:

- **Portable Network Architectures.** We conduct a search of portable network architectures for better quality-latency trade-off across mobile devices. This also includes a set of practical application settings to better reflect real user scenarios.

- **Network Optimization.** For image deblurring task, we further present the effectiveness of popular network optimizations, quantization and pruning. We demonstrate that there exist noticeable quality drops with 8-bit quantization-aware training. With 16-bit post-training quantization, it is capable of achieving the same quality level as floating-point network.

16 bit可无损.

[*] Email: jimmy.chiang@mediatek.com

- **Quality and Latency across Mobile Devices.** In terms of image quality and latency, we evaluate various image deblurring networks across mobile devices. Our paper demonstrates the success deployment of image deblurring application on three mobile devices (with deep learning accelerators).

In Section 2, we describe the related work for this paper. We will introduce detailed flow of deploying image deblurring on mobile devices in Section 3. In Section 4, we show detailed analysis from the aspect of quality and latency. The conclusion and future work are summarized in Section 5.

2. Related Work

X

2.1. Image Enhancement and Restoration

一些网络架构

In recent works, most of the image enhancement and restoration methods share common network architectures. U-Net [50] architecture, which is also known as encoder-decoder structure, is widely used in many image enhancement and restoration tasks [53, 22, 48, 39, 43, 19]. In image denosing, Gu *et al.* [26] propose a top-down architecture, Self-Guided Network (SGN), to better exploit multi-scale information in images. In super resolution, there are also quite a few representative network architectures, such as EDSR [41], RDN [57] and DBPN [28]. Most of these architectures keep the same scale across all operations except the last one, which is responsible for upsampling. In image deblurring, besides U-Net architecture, deformable convolution and self-attention module are proposed to model spatially-varying deblurring process in [47]. Recently, Kupyn *et al.* [37] use FPN architecture [42] in image deblurring with Generative Adversarial Network (GAN) based training methodology. To alleviate the computational complexity of deployment, several light-weight architectures are proposed recently [31, 18, 21, 25, 30].

X

2.2. Network Optimizations

Quantization. Network quantization is one of the most effective methods for deploying networks on mobile devices. Typically, quantization enables efficient integer arithmetics by translating weights and activations of a network into fixed-point (*e.g.*, 8-bit integer) representation. Quantization-aware training and post-training quantization are two well-known techniques supported by TensorFlow [35]. Post-training quantization estimates value ranges for both weights and activations through forward pass of training data while quantization-aware training performs such estimation in both forward and backward pass. In recent works, both techniques demonstrate promising results on image perception tasks [35, 29, 51, 30, 16, 14]. To the best of our knowledge, there are limited works [20, 44] applying quantization on image enhancement problems. To better understand the effectiveness of quantization on image deblurring, this work applies the most widely used quantization techniques and conducts a comprehensive evaluation.

quant的文章较少

Pruning. Network pruning is another widely used optimization for deploying networks on mobile devices. There are two approaches of pruning, unstructured pruning [27, 55] and structured pruning [24, 40, 56]. Unstructured pruning makes the weights of a network sparse instead of changing the network architecture. Structured pruning reduces the number of channels in the network and thus improves latency on general devices. Most of the works focus on image classification or segmentation [27, 55, 24, 40, 56]. Wang *et al.* [54] propose architecture-aware pruning to reduce MAC[1] and memory bandwidth in super resolution [41] and low-light enhancement [19]. In this work, we apply pruning techniques in similar ways and show the effectiveness on image deblurring.

0

2.3. Benchmark Suites & Challenges

not known

Benchmark Suites. *AI Benchmark*, is a comprehensive benchmark suite for mobile devices by Andrey *et al.* [32], which evaluates both latency and accuracy among various tasks. In *AI Benchmark*, the resolution for input images are ranging from 84 × 84 to 512 × 512 (except semantic segmentation which is not the focus of this paper). However, contemporary use-cases of image enhancement typically need larger input resolution, for example, 720p HD or even higher resolution. *MLPerf inference benchmark* [49] is one of the largest benchmark community contributed from both academic and industry. However, image enhancement and restoration tasks are not included for its benchmarking. *AIMark* [2] and *Antutu AI Benchmark* [4] are another two benchmark suites targeting mobile devices. Among these two benchmark suites, platform providers are asked to deploy test applications by using proprietary formats and frameworks. Such benchmarking policy is quiet different from *AI Benchmark* which adopts a unified framework, Android Neural Networks API (NNAPI) [3].

有过两个比赛

X

Challenges. *PIRM 2018 challenge on perceptual image enhancement on smartphone* [33] is the first image enhancement challenge that evaluates latency on mobile devices. Razer phone and Huawei P20 are used as target devices [7], which have their latest generations with higher computation capacity. *NTIRE 2020 image deblurring challenge on smartphone* [46] adopts Google Pixel 4 as its target device. However, the evaluation of latency is conducted on 256 × 256 input resolution, which is far from enough to reflect a real use-case, say HD 720p (1280 × 720).

In this paper, we apply a more realistic setting for image deblurring application and deploy it across a set of mobile

[1] MAC is known as multiply-accumulate. A MAC is roughly two floating-point operations (FLOPs), used in some other papers.

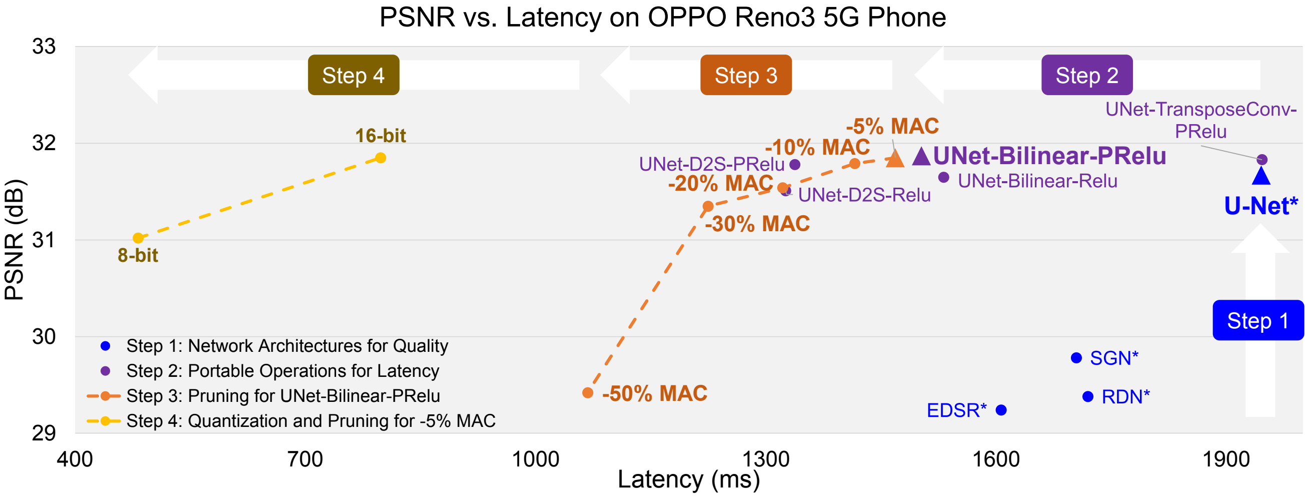


Figure 1. The evolution of trade-off between PSNR and latency on OPPO Reno3 5G.

(*) means the architecture is slightly different from the original paper.
U-Net*: UNet-TransposeConv-Relu.
D2S: abbreviation of DepthToSpace. Bilinear: abbreviation of ResizeBilinear.

大数据集

devices. Differentiation includes (1) We use HD 720p resolution (1280 × 720) to show the capability of mobile devices for real use-cases. (2) We evaluate both quantitative (PSNR[2]) and qualitative (visual) results to better justify the quality measurement. (3) To create a fair comparison across mobile devices, we adopt the unified NNAPI framework [3] for all the evaluations, including both quality and latency.

3. Deploying Image Deblurring across Mobile Devices

In this paper, we take image deblurring as an example for the mobile deployment. We first introduce the problem definition of blind image deblurring. Second, we elaborate the searching procedure of portable network architecture and its interplay between latency portability and PSNR quality. Then, we describe the optimization techniques to further improve performance on mobile devices. Finally, softwares and hardwares for deploying the networks are introduced.

3.1. Image Deblurring

In this paper, we adopt the same problem formulation, blind image deblurring task, as used in *NTIRE 2020 image deblurring challenge on smartphone* [46].

Dataset. For the training dataset, we use REDS [45] image deblurring dataset which is also used in image deblurring challenges of NTIRE 2020 [46]. In REDS dataset, there are 300 videos divided into 240 sequences for training, 30 sequences for validation, and 30 sequences for testing. Each sequence contains 100 frames of 1280 × 720 resolution. For each frame, blurry image and sharp image are given as a pair. In this paper, we treat each frame as independent and conduct all the experiments with this setting.

3.2. Searching Portable Network Architectures

这里的search真就
是从不同model中挑

In this section, we introduce the searching of a high PSNR quality yet portable network across mobile devices. This includes Step 1 and Step 2 shown in Figure 1. First of all, a set of state-of-the-art network architectures is listed in Figure 1. For fair comparison, networks are slightly adjusted to match a baseline computational complexity (refer to Section 4 for more detail). In Step 1, the goal is to search for the architecture with highest quality. Hence, the network of the highest PSNR, U-Net [50], is selected for the next step. In Step 2, the objective turns to increase the portability across difference mobile devices. In accelerator hardware, optimization usually focuses on limited operations, such as convolution, pooling, activation and so on. Therefore, a network with high quantitative or qualitative quality can have very limited portability since its operations are not optimized on another devices. Thankful to the strong function approximating nature of neural network, it is possible to replace these operations by other semantically similar and optimized ones. According to the above discussion, we derive a set of architectures from the previous step. These architectures are marked as purple in Figure 1 (refer to Section 4 for more detailed discussions). To this end, one is free to choose any of the networks according to the quality or latency requirement.

只优化一些 op

替换 op
(de conv → others)

[2] Peak Signal to Noise Ratio

Table 1. Hardware specification and AI-Scores [1] of the mobile devices.

	Huawei Mate30 Pro 5G	**OPPO Reno3 5G**	**Google Pixel 4**
Chipset	HiSilicon Kirin 990 5G [9]	MediaTek Dimensity 1000L [10]	Qualcomm Snapdragon 855 [12]
CPU	2 Cortex-A76, 2.86 GHz 2 Cortex-A76, 2.36 GHz 4 Cortex-A55, 1.95 GHz	4 Cortex-A77, 2.20 GHz 4 Cortex-A55, 2.00 GHz	1 Kryo 485 Gold Prime, 2.84 GHz 3 Kryo 485 Gold, 2.42 GHz 4 Kryo 485 Silver, 1.80 GHz
GPU	Mali-G76	Mali-G77	Adreno 640
AI Engine	2 Big-Core DaVinci NPU 1 Tiny-Core DaVinci NPU	APU 3.0 (2 Big Cores, 3 Small Cores, 1 Tiny Core)	AIE CPU, AIE GPU, AIE DSP
NNAPI Runtime	nnapi-reference, armnn, liteadaptor	nnapi-reference, neuron-ann	nnapi-reference, google-edgutpu, qti-default, qti-dsp, qti-gpu, qti-hta
AI-Score [1]	76,206	58,628	33,289

3.3. Optimizing Network for Deployment

As discussed in Section 3.2, the procedure searches for a set of portable network architectures with the best quality-latency trade-off. After that, several network optimization techniques can be applied to further boost the performance on mobile devices. The following paragraphs introduce how the widely used pruning and quantization are applied to the portable network architectures.

Pruning. We use structured pruning technique to optimize networks (Step 3 in Figure 1). A structured pruning technique similar to one mentioned in [54] is used. The numbers of channels are adaptively pruned with respect to the given MAC reduction target. With different level of pruning targets, a set of pruned networks are generated and marked orange as in Figure 1. Any choice is a trade-off between quality and latency.

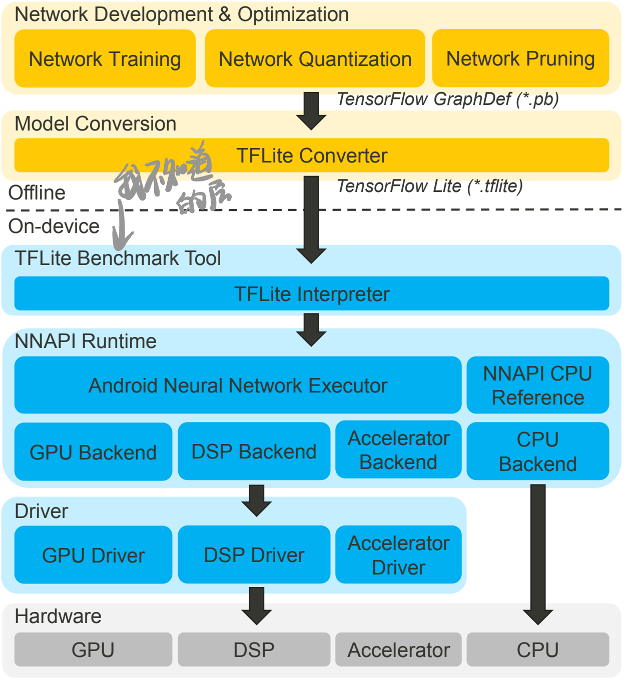
Quantization. We apply quantization-aware training and post-training quantization techniques to demonstrate their applicability on the image deblurring task. We extend the evaluation to both 8-bit and 16-bit quantization, which will be detailed in Section 4.4. Similarly, in Step 4, the quantized networks provide another opportunity for trade-off.

3.4. Deploying Network: Softwares and Hardwares

As shown in Figure 2, this paper adopts TFLite format (.tflite) and *TFLite Benchmark Tool* [17] to evaluate the latency on various mobile devices. NNAPI, underneath TFLite, is a unified inference framework widely supported by various platforms [9, 10, 12]. For fair comparison across mobile devices as in [32], we adopt the unified NNAPI framework to deploy the networks on mobile devices. Last, we deploy the optimized portable networks across several mobile devices and conduct the experimental analysis.

☆ Latency 可用这个?

TFLite Benchmark Tool. *TFLite Benchmark Tool* [17] can be used to evaluate the latency of a TFLite model on both desktops and Android devices. It provides several accelerations on mobile devices, *e.g.*, XNNPACK delegate is optimized for floating-point inference on ARM CPU, GPU delegate for floating-point inference on mobile GPU, NNAPI delegate for both floating-point and 8-bit fixed-point inference on Android devices, and Hexagon delegate is optimized for 8-bit fixed-point inference on Qualcomm DSP. With these tools, comprehensive latency evaluation can be conducted on different mobile devices.



我不知道的层

Figure 2. Software stack for developing networks, optimizing networks, and inference on mobile devices.

Android NNAPI. NNAPI [3] is designed for accelerating deep learning operations on Android devices. It provides base operators of functionality for higher-level machine learning frameworks, such as TensorFlow Lite (TFLite) and Caffe2. With NNAPI, platform providers can have specific acceleration of frequently used operations for both IEEE 754 16-bit floating-point and 8-bit fixed-point data type.

Table 2. Quality, complexity, and latency of different network architectures.

Network	PSNR / SSIM	MAC	Latency (ms)		
	(floating-point)	($\times 10^9$)	Huawei Mate30 Pro	OPPO Reno3 5G	Google Pixel 4
U-Net [50]*	**31.67 / 0.899**	238	54206[1]	**1946**	31870[2]
EDSR [41]*	29.24 / 0.836	249	**518**	1607	3468
RDN [57]*	29.40 / 0.839	243	7367	**1720**	Failed[3]
DBPN [28]*	31.23 / 0.886	242	Failed[3]	Failed[3]	Failed[3]
Inception-ResNetV2-FPN [37]*	28.63 / 0.830	250	22164[4]	**3245**	20150[5]
SGN [26]*	29.78 / 0.858	247	**931**	1705	3004[6]

(*) means the architecture is slightly different from the original paper.
[1] TRANSPOSE_CONV_2D operations fall back to CPU.
[2] TRANSPOSE_CONV_2D, CONCATENATION and CONV_2D operations after CONCATENATION fall back to CPU.
[3] ERROR: NN API returned error ANEURALNETWORKS_OP_FAILED.
[4] MUL and RESIZE_NEAREST_NEIGHBOR operations fall back to CPU.
[5] MUL, RESIZE_NEAREST_NEIGHBOR, some CONV_2D and MAP_POOL_2D operations fall back to CPU.
[6] SPACE_TO_DEPTH and DEPTH_TO_SPACE operations fall back to CPU.

Hardware Acceleration. This paper focuses on three mobile devices (with deep learning accelerators) as in *AI-Score* [1], including Huawei Mate30 Pro 5G, OPPO Reno3 5G, and Google Pixel 4 [6]. Table 1 summarizes the hardware specification of these mobile devices. Table 1 also lists the details of NNAPI runtime library for each platform.

4. Experiment Results

In this section, we discuss the experiment results and its implementation details. Section 4.1 elaborates the details of dataset, training setups and evaluation methods. The proposed architecture search is discussed in Section 4.2. Section 4.3 covers the portability discussion at operation level. The interplay between networks and optimization methods are discussed in Section 4.4. Last but not the least, we discuss the quality considerations in Section 4.5.

4.1. Implementation Details

In this paper, we implement and train all the networks by TensorFlow. We crop each training image in REDS dataset [45] into 15 patches, with 256 × 256 resolution for each patch. A total of 360,000 pairs of training patches are prepared from 240 training sequences (100 frames for each). We follow the *NTIRE 2020 image deblurring challenge on smartphone* for frame selection (1 out of every 10 frames) [46]. As a result, a total of 300 frames in validation set are used to calculate PSNR for quality assessment.

All networks are trained for 1M steps on a single RTX-2080 Ti GPU with batch size 16, L_1 loss, Adam optimizer [36] and exponential decay for learning rate. We set initial learning rate as 2×10^{-4}, decay rate as 0.98, and 5K decay steps for exponential decay. All convolutional operations are initialized with Xavier initialization [23].

To measure the latency on mobile devices, we use *TFLite Benchmark Tool* [17] with arguments $use\_nnapi = true$, $allow\_fp16 = true$[3], $num\_runs = 10$, and $num\_threads = 4$. We also use *taskset*[4] command to reduce the variation of CPU time between different runs.

4.2. Network Architectures for Quality

To search across different network architectures, we compare some widely used networks in image enhancement domain, including U-Net [50], Inception-ResNetV2-FPN [37], EDSR [41], RDN [57], DBPN [28], and SGN [26]. For fair comparison, a network's operations and channels are slightly adjusted to match a baseline computational complexity, roughly 250×10^9 MAC. We remove upsampling in EDSR and RDN, since these operations were designed for super resolution. Likewise, in DBPN, we remove the first up-projection unit to keep the same resolution for input and output tensors. We exclude deformable convolution and self-attention based networks [47] since no mobile device supports these types of operations. Knowing that this paper focuses on deploying the architectures on mobile devices, any training methodology can also be applied to the architectures of interest, *e.g.*, GAN-based training. The detail training settings are summarized in Section 4.1.

比其他base好?

Table 2 summarizes PSNR and MAC of the candidates of network architectures. We also list the measured latency of these networks on all the three target mobile devices. According to the quality index in Table 2, U-Net outperforms all the counterparts by its highest PSNR. However, an interesting finding is that, an unsupported operation by accelerator[5], *e.g.*, *TRANSPOSE_CONV_2D*, will cause a fallback to NNAPI CPU-reference-implementation. This prevents the execution from being accelerated and results in unreasonable high latency as shown in Table 2.

[3] To inference floating-point networks with 16-bit floating-point data type

所以不支持16bit case、不能用自己的量化方法?

[4] We use "taskset f0" to specify using 4 big cores of CPU

[5] We use "adb shell setprop debug.nn.vlog 1" to open debug option and use "adb shell logcat — grep -e findBestDeviceForEachOperation" to check whether an operation is executed on CPU or accelerator

Table 3. Quality, complexity, and latency of different up-sampling and activation operations for U-Net.

Network	PSNR / SSIM	MAC	Latency (ms)		
	(floating-point)	($\times 10^9$)	Huawei Mate30 Pro	OPPO Reno3 5G	Google Pixel 4
UNet-TransposeConv-Relu†	31.67 / 0.899	238	54206[1]	**1946**	31870[2]
UNet-TransposeConv-PRelu	31.83 / 0.903		78402[1]	**1947**	32390[2]
UNet-DepthToSpace-Relu	31.51 / 0.895	222	**805**	1326	2697[3]
UNet-DepthToSpace-PRelu	31.78 / 0.900		**908**	1338	3060[3]
UNet-ResizeBilinear-Relu	31.65 / 0.898	256	**1184**	1532	8425[4]
UNet-ResizeBilinear-PRelu	**31.87 / 0.903**		**1281**	1503	9770[4]

† UNet-TransposeConv-Relu is the same as U-Net [50]* in Table 2
[1] TRANSPOSE_CONV_2D operations fall back to CPU.
[2] TRANSPOSE_CONV_2D, CONCATENATION and CONV_2D operations after CONCATENATION fall back to CPU.
[3] DEPTH_TO_SPACE operations fall back to CPU.
[4] CONCATENATION and CONV_2D after CONCATENATION operations fall back to CPU.

4.3. Portable Operations for Latency

换叩

As discussed in Section 4.2, an unsupported operation across devices can result in unreasonable high latency. The major functionality of *TRANSPOSE_CONV_2D* operation is for up-sampling. Hence, in order to deploy the network across all the three mobile devices, an alternative solution is to replace such operations by other operations (with similar functionality). In this paper, we replace *TRANSPOSE_CONV_2D* by *DEPTH_TO_SPACE*[6] and *RESIZE_BILINEAR*. The replacement is also evaluated on both *RELU* and *PRELU* activations to show its effectiveness.

As shown in Table 3, such replacement avoids most of the cases in which a fallback to NNAPI CPU-reference-implementation happens. Thus, a network with better trade-off between quality and latency can be conducted in this way. One is free to choose any of the networks according to the quality or latency. In this paper, *UNet-ResizeBilinear-PRelu* is selected for the following experiments.

4.4. Network Optimization

According to the discussion in Section 4.3, this paper selects *UNet-ResizeBilinear-PRelu* and applies network optimizations to further boost its performance. Experiment results of network optimization are summarized in Table 4.

4.4.1 Quantization.

For 8-bit quantization, post-training quantization suffers a destructive 2 dB PSNR drop. Even with quantization-aware training, there exists at least noticeable 0.8 dB PSNR drop. In contrast, 16-bit post-training quantization, is capable to preserve almost the same quality as floating-point network. In our experiments, most devices have latency improvement with quantized networks except for Huawei Mate30 Pro. This is due to the lack of support for quantized *RESIZE_BILINEAR* operation in its accelerator. We suggest future works to consider quantization configuration during the stage of architecture search. Note that NNAPI does not support 16-bit fixed-point inference. Hence, the evaluation requires proprietary SDK provided by platform providers. Qualcomm's SNPE [13] supports 16-bit fixed-point inference with HTA hardware. However, the corresponding software (HTA runtime library) is not available in Google Pixel 4. For Huawei's HiAI SDK [8], we cannot find appropriate information for its support of 16-bit fixed-point inference. Therefore, we only report the results of 16-bit fixed-point inference for MediaTek's NeuroPilot SDK [11] in Table 4.

4.4.2 Pruning.

We apply five different settings of MAC reduction targets. Most devices have latency improvement except Huawei Mate30 Pro. Surprisingly, we observe over 60% latency improvement with roughly 0.5 db PSNR drop on Google Pixel 4 (in 30% MAC reduction setting). Hence, as a future direction, such hardware limitations and preferences should also be considered when searching network architectures.

Last, we combine both network pruning and quantization for further optimization. Based on the network pruned with 5% MAC reduction, we quantize the network with 8-bit quantization-aware training and 16-bit post-training quantization. As shown in Table 4, the latency could be further reduced when compared with quantization only.

4.5. Ablation Study of Quality

In this section, we show the impact of different network optimization by examining visual results. The quantitative results are also illustrated to uncover the computation errors across different hardware implementations.

4.5.1 Visual Quality on Optimized Networks

Figure 3 shows visual results of quantization and pruning. 16-bit post-training quantization (PTQ) perfectly preserve

[6] DEPTH_TO_SPACE is also known as pixel shuffle in some papers or frameworks

Table 4. Quality, complexity, and latency of different optimization techniques.

Network	Optimization Type	Setting	PSNR / SSIM	MAC ($\times 10^9$)	Latency (ms)		
					Huawei Mate30 Pro	OPPO Reno3 5G	Google Pixel 4†
UNet-ResizeBilinear-PRelu	None	Float	31.87 / 0.903	256	**1281**	1503	9770[1]
	Quantization (fixed-point)	8-bit PTQ	29.66 / 0.835	256	31220[2]	**504**	2175[3]
		8-bit QAT	31.03 / 0.873		33490[2]	**488**	2128[3]
		16-bit PTQ‡	31.87 / 0.903		–	**825**[4]	–
	Pruning (floating-point)	-5% MAC	31.85 / 0.903	243	1693	**1469**	8419[1]
		-10% MAC	31.79 / 0.902	230	1854	**1416**	8189[1]
		-20% MAC	31.54 / 0.896	202	1853	**1322**	7313[1]
		-30% MAC	31.35 / 0.893	179	1690	**1225**	3756[1]
		-50% MAC	29.81 / 0.854	127	1477	1068	**934**
	Pruning + Quantization (fixed-point)	-5% MAC + 8-bit QAT	31.02 / 0.872	243	28521[2]	**482**	2098[3]
		-5% MAC + 16-bit PTQ	31.85 / 0.903	243	–	**798**[4]	–

PTQ, abbreviation of Post-Training Quantization; QAT, abbreviation of Quantization-Aware Training.

† In Google Pixel 4, all operations of quantized networks are executed on *qti-default* runtime unless fallbacks on CPU are specified.

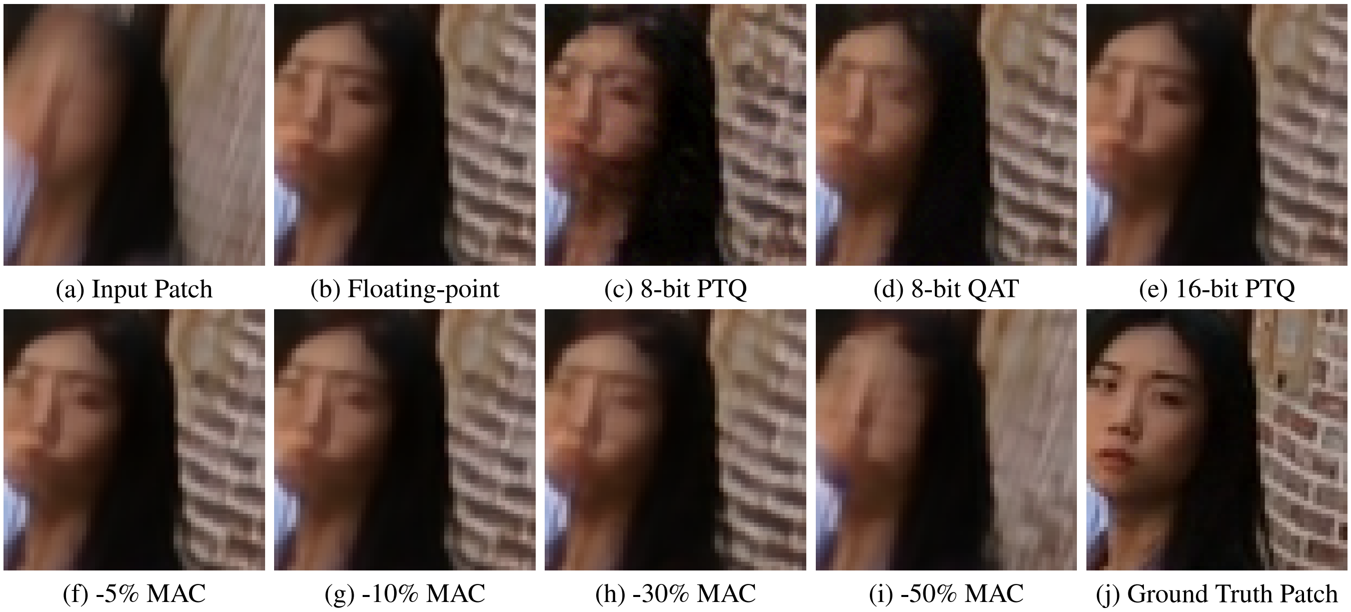
‡ For 16-bit fixed-point inference on mobile devices, please refer to Section 4.4 for more details.

[1] CONCATENATION and CONV_2D after CONCATENATION operations fall back to CPU.

[2] PRELU and RESIZE_BILINEAR operations fall back to CPU.

[3] CONCATENATION operations fall back to CPU.

[4] Latency evaluated with MediaTek NeuroPilot SDK [11].


(a) Input Patch (b) Floating-point (c) 8-bit PTQ (d) 8-bit QAT (e) 16-bit PTQ

(f) -5% MAC (g) -10% MAC (h) -30% MAC (i) -50% MAC (j) Ground Truth Patch

Figure 3. Visual results of UNet-ResizeBilinear-PRelu with network quantization and pruning. (c)(d)(e) represent different quantization settings as in Table 4. (e)(f)(g)(h) show the results of pruning given different MAC reduction targets. The patch is cropped from *000/00000039.png* in REDS validation set

the visual quality of floating-point network. However, 8-bit post-training quantization (PTQ) and quantization-aware training (QAT) show different levels of quantization errors. For pruning results, the visual quality degrades with the increasing of MAC reduction. When pruning the network by 50% MAC, a noticeable blurry result appears.

4.5.2 Quality Index on Mobile Devices

Table 5 shows the PSNR and per-pixel L2 error. Such calculations are between the results of TensorFlow (*checkpoint* format) on desktops and the results of TFLite on mobile devices. In floating-point data type, 32-bit data are used in

Table 5. Error measurement on mobile devices with various data types. Evaluated on UNet-ResizeBilinear-Relu network†

Data Type	PSNR	Comparison between results on mobile devices (TFLite) and results on Desktop (TensorFlow)					
		Huawei Mate30 Pro		OPPO Reno3 5G		Google Pixel 4	
		PSNR	L2 Error	PSNR	L2 Error	PSNR	L2 Error
Float[1]	31.65	50.82 ± 0.29	1.95×10^{-7}	**50.85 ± 0.19**	$\mathbf{1.94\times10^{-7}}$	50.59 ± 0.73	2.09×10^{-7}
16-bit	31.65	–	–	**65.37 ± 0.89**	$\mathbf{6.99\times10^{-9}}$	–	–
8-bit	31.36	43.07 ± 1.52	1.25×10^{-6}	**43.33 ± 1.39**	$\mathbf{1.16\times10^{-6}}$	41.95 ± 1.24	1.57×10^{-6}

Standard deviation of PSNR is calculated with 300 validation images.

† Abnormal PSNR drops (for Floating-point setting) happen to UNet-ResizeBilinear-PRelu network on Google Pixel 4. Since the root cause is not confirmed, this table reports the results of UNet-ResizeBilinear-Relu network for a fair comparison.

[1] Floating-point data type used for mobile and desktop are 16-bit and 32-bit, respectively

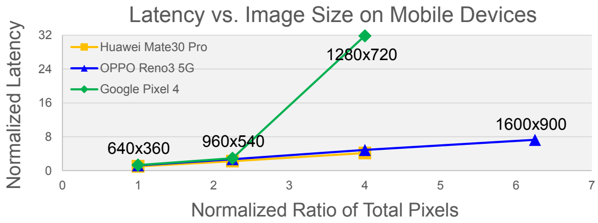


Figure 4. Latency with various input resolution. All the latencies are normalized to Huawei Mate30 Pro with 360p resolution. Note that only OPPO Reno3 5G successfully run with 1600×900 (HD+) input resolution. All devices fail to run with 1920×1080 (Full HD resolution.

TensorFlow inference while mobile devices use 16-bit data for acceleration. PSNR and L2 error between 32-bit and 16-bit floating-point are about 50 dB and 1.94×10^{-7}, respectively. For quantized data type, TensorFlow uses *fake quantitzation* operations [35] to simulate the behavior of quantization. However, the inference is still computed by using floating-point arithmetic, which is different from the fixed-point ones used by TFLite. As shown in the table, the error of 8-bit data type is much larger than 16-bit and floating-point data type. This provides an in-depth quality assessment for deploying quantized networks on mobile devices. In general, the lower error between mobile devices and desktops, the closer result between algorithm development and its deployment on devices.

4.6. Discussions

Considering the differences of software and hardware between all the three platforms, several non-trial deployment issues are reported in the previous sections. This section summarizes all the findings and discussions.

- **First of all,** as listed in Table 2, the latency are highly inconsistent when deploying the out-of-the-box network architectures across platforms. Some of platforms (Huawei Mate30 Pro and Google Pixel 4) present unreasonably high latency. Fortunately, as in Table 3, such pitfall can be partially mitigated by leveraging operations with better portability.

useful.

- **Second,** the network optimization techniques (both quantization and pruning) do not consistently reduce the latency across platforms. As listed in Table 4, network optimizations cause even higher latency in Huawei Mate30 Pro. Meanwhile, the latency of pruned networks do not scale linearly w.r.t. MAC reduction in Google Pixel 4.

- **Last but not the least,** as shown in Figure 4, the latency does not scale linearly with input resolution. In Google Pixel 4, there is a huge latency increment when the input resolution scales to 1280 × 720.

In summary, these non-trivial performance pitfalls make mobile deployment an even challenging work. This urges deployment-guidelines to conduct 1) portable network architectures, 2) network optimization and 3) trade between quality and latency across mobile devices.

5. Conclusion and Future Work

In summary, this paper conducts a search of portable network architectures for better quality-latency trade-off across mobile devices. Besides, we also present the effectiveness of quantization and pruning for image deblurring task. The searched portable networks are evaluated with a set of comprehensive experiments and comparisons. Our experiments and comparisons provide an in-depth analysis for both latency and image quality. In conclusion, we demonstrate a success deployment of image deblurring on three mobile devices. We also suggest two promising directions for future work: [(1) searching portable network architecture while considering more device related factors, *e.g.*, quantization, pruning and/or hardware limitation/preference] and (2) systematic searching methodology for portable network architecture, *e.g.*, Network Architecture Search (NAS) for device portability.

但这是SDK层面的约束吧.

References

[1] AI Benchmark Performance Ranking. http://ai-benchmark.com/ranking.html. 4, 5

[2] AImark of Ludashi. http://www.ludashi.com/page/aimark.php. 1, 2

[3] Android Neural Networkk API (NNAPI). https://developer.android.com/ndk/guides/neuralnetworks. 2, 3, 4

[4] Antutu AI Benchmark,. 1, 2

[5] Apple A13 Bionic Chipset. https://en.wikichip.org/wiki/apple/ax/a13. 1

[6] Comparison of Huawei Mate30 Pro 5G, OPPO Reno3 and Google Pixel 4. https://www.gsmarena.com/compare.php3?&idPhone1=9880&idPhone2=9942&idPhone3=9896. 5

[7] Comparison of Razer Phone and Huawei P20. https://www.gsmarena.com/compare.php3?idPhone1=8923&idPhone2=9107. 2

[8] Huawei HiAI SDK. https://developer.huawei.com/consumer/en/hiai. 6

[9] Huawei Kirin 990 5G Chipset. https://en.wikichip.org/wiki/Kirin_990. 1, 4

[10] MediaTek Dimensity 1000L 5G Chipset. https://en.wikichip.org/wiki/mediatek/dimensity/1000l#Neural_processor. 1, 4

[11] MediaTek NeuroPilot SDK. https://neuropilot.mediatek.com/. 6, 7

[12] Qualcomm Snapdragon 855 Chipset. https://en.wikichip.org/wiki/qualcomm/snapdragon_800/855. 1, 4

[13] Qualcomm Snapdragon Neural Processing Engine SDK. https://developer.qualcomm.com/docs/snpe/overview.html. 6

[14] Quantize DeepLab Model for Faster on-device Inference. https://github.com/tensorflow/models/blob/master/research/deeplab/g3doc/quantize.md. 2

[15] Samsung Exynos 990 Mobile Processor. https://en.wikichip.org/wiki/samsung/exynos/990. 1

[16] Tensorflow Detection Model Zoo. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. 2

[17] TFLite Model Benchmark Tool. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/tools/benchmark. 4, 5

[18] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. Fast, accurate, and lightweight super-resolution with cascading residual network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 252–268, 2018. 1, 2

[19] Chen Chen, Qifeng Chen, Jia Xu, and Vladlen Koltun. Learning to see in the dark. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3291–3300, 2018. 1, 2

[20] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. Learning low precision deep neural networks through regularization. *arXiv preprint arXiv:1809.00095*, 2018. 2

[21] Xiangxiang Chu, Bo Zhang, Hailong Ma, Ruijun Xu, Jixiang Li, and Qingyuan Li. Fast, accurate and lightweight super-resolution with neural architecture search. *arXiv preprint arXiv:1901.07261*, 2019. 1, 2

[22] Hongyun Gao, Xin Tao, Xiaoyong Shen, and Jiaya Jia. Dynamic scene deblurring with parameter selective sharing and nested skip connections. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3848–3856, 2019. 2

[23] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010. 5

[24] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1586–1595, 2018. 2

[25] Shuhang Gu, Wen Li, Luc Van Gool, and Radu Timofte. Fast image restoration with multi-bin trainable linear units. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4190–4199, 2019. 1, 2

[26] Shuhang Gu, Yawei Li, Luc Van Gool, and Radu Timofte. Self-guided network for fast image denoising. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2511–2520, 2019. 2, 5

[27] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1, 2

[28] Muhammad Haris, Gregory Shakhnarovich, and Norimichi Ukita. Deep back-projection networks for super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1664–1673, 2018. 2, 5

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2

[30] 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. 1, 2

[31] Zheng Hui, Xiumei Wang, and Xinbo Gao. Fast and accurate single image super-resolution via information distillation network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 723–731, 2018. 1, 2

[32] Andrey Ignatov, Radu Timofte, Andrei Kulik, Seungsoo Yang, Ke Wang, Felix Baum, Max Wu, Lirong Xu, and Luc Van Gool. Ai benchmark: All about deep learning on smartphones in 2019. *arXiv preprint arXiv:1910.06663*, 2019. 1, 2, 4

[33] Andrey Ignatov, Radu Timofte, Thang Van Vu, Tung Minh Luu, Trung X Pham, Cao Van Nguyen, Yongwoo Kim,

Jae-Seok Choi, Munchurl Kim, Jie Huang, et al. Pirm challenge on perceptual image enhancement on smartphones: Report. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018. 1, 2

[34] Andrey Ignatov, Luc Van Gool, and Radu Timofte. Replacing Mobile Camera ISP with a Single Deep Learning Model. *arXiv e-prints*, page arXiv:2002.05509, Feb 2020. 1

[35] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018. 1, 2, 8

[36] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[37] Orest Kupyn, Tetiana Martyniuk, Junru Wu, and Zhangyang Wang. Deblurgan-v2: Deblurring (orders-of-magnitude) faster and better. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8878–8887, 2019. 1, 2, 5

[38] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photorealistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017. 1

[39] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise: Learning image restoration without clean data. *arXiv preprint arXiv:1803.04189*, 2018. 2

[40] Jiashi Li, Qi Qi, Jingyu Wang, Ce Ge, Yujian Li, Zhangzhang Yue, and Haifeng Sun. Oicsr: Out-in-channel sparsity regularization for compact deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7046–7055, 2019. 2

[41] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 136–144, 2017. 2, 5

[42] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 2

[43] Jiaming Liu, Chi-Hao Wu, Yuzhi Wang, Qin Xu, Yuqian Zhou, Haibin Huang, Chuan Wang, Shaofan Cai, Yifan Ding, Haoqiang Fan, et al. Learning raw image denoising with bayer pattern unification and bayer preserving augmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019. 2

[44] Yinglan Ma, Hongyu Xiong, Zhe Hu, and Lizhuang Ma. Efficient super resolution using binarized neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019. 2

[45] Seungjun Nah, Sungyong Baik, Seokil Hong, Gyeongsik Moon, Sanghyun Son, Radu Timofte, and Kyoung Mu Lee. Ntire 2019 challenges on video deblurring and super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019. 3, 5

[46] Seungjun Nah, Sanghyun Son, Radu Timofte, and Kyoung Mu Lee. Ntire 2020 challenge on image and video deblurring. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020. 1, 2, 3, 5

[47] Kuldeep Purohit and A. N. Rajagopalan. Region-Adaptive Dense Network for Efficient Motion Deblurring. *arXiv e-prints*, page arXiv:1903.11394, Mar. 2019. 2, 5

[48] Kuldeep Purohit, Anshul Shah, and AN Rajagopalan. Bringing alive blurred moments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6830–6839, 2019. 2

[49] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. Mlperf inference benchmark. *arXiv preprint arXiv:1911.02549*, 2019. 1, 2

[50] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 2, 3, 5, 6

[51] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 2

[52] Jun Tan, Kang Yang, Shiwei Song, Tianzhang Xing, and Dingyi Fang. Mobile-deblur: A clear image will on the smart device. In *2017 3rd International Conference on Big Data Computing and Communications (BIGCOM)*, pages 97–105. IEEE, 2017. 1

[53] Xin Tao, Hongyun Gao, Xiaoyong Shen, Jue Wang, and Jiaya Jia. Scale-recurrent network for deep image deblurring. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8174–8182, 2018. 2

[54] Wei-Ting Wang, Han-Lin Li, Wei-Shiang Lin, Cheng-Ming Chiang, and Yi-Min Tsai. Architecture-aware network pruning for vision quality applications. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2701–2705. IEEE, 2019. 2, 4

[55] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5687–5695, 2017. 2

[56] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 2130–2141, 2019. 2

[57] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2472–2481, 2018. 1, 2, 5