MobiSR: Efficient On-Device Super-Resolution through Heterogeneous Mobile Processors

Royson Lee[†]*, Stylianos I. Venieris[†]*
Łukasz Dudziak[†], Sourav Bhattacharya[†], Nicholas D. Lane[†,‡]

[†]Samsung AI Center, Cambridge [‡]University of Oxford

* *Indicates equal contribution.*

ABSTRACT

In recent years, convolutional networks have demonstrated unprecedented performance in the image restoration task of super-resolution (SR). SR entails the upscaling of a single low-resolution image in order to meet application-specific image quality demands and plays a key role in mobile devices. To comply with privacy regulations and reduce the overhead of cloud computing, executing SR models locally on-device constitutes a key alternative approach. Nevertheless, the excessive compute and memory requirements of SR workloads pose a challenge in mapping SR networks on resource-constrained mobile platforms. This work presents MobiSR, a novel framework for performing efficient super-resolution on-device. Given a target mobile platform, the proposed framework considers popular model compression techniques and traverses the design space to reach the highest performing trade-off between image quality and processing speed. At run time, a novel scheduler dispatches incoming image patches to the appropriate model-engine pair based on the patch's estimated upscaling difficulty in order to meet the required image quality with minimum processing latency. Quantitative evaluation shows that the proposed framework yields on-device SR designs that achieve an average speedup of 2.13× over highly-optimized parallel difficulty-unaware mappings and 4.79× over highly-optimized single compute engine implementations.

MobiCom '19, October 21–25, 2019, Los Cabos, Mexico

ACM ISBN 978-1-4503-6169-9/19/10…$15.00
https://doi.org/10.1145/3300061.3345455

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing**; • **Computing methodologies** → *Computer vision tasks*;

KEYWORDS

Super-resolution, deep neural networks, mobile computing, heterogeneous computing, scheduling

ACM Reference Format:
Royson Lee, Stylianos I. Venieris, Łukasz Dudziak, Sourav Bhattacharya, Nicholas D. Lane. 2019. MobiSR: Efficient On-Device Super-Resolution through Heterogeneous Mobile Processors. In *The 25th Annual International Conference on Mobile Computing and Networking (MobiCom '19), October 21–25, 2019, Los Cabos, Mexico.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3300061.3345455

1 INTRODUCTION

The rapid progress of convolutional neural networks (CNNs) has led to substantial performance improvements in the computer vision task of super-resolution (SR). SR networks are capable of processing a low-resolution image and producing an output with a significant increase in resolution [6]. This property has made CNN-powered SR an enabling technology for building novel applications on mobile and home devices, including mobile phones, electronic photograph frames and televisions.

Despite their unparalleled performance, state-of-the-art SR networks [23, 37, 62, 63] pose significant deployment challenges. To upscale low-resolution images, SR models often propagate feature maps of large spatial dimensions across their layers, leading to an excessive number of operations and run-time storage requirements.

At the moment, to alleviate this computational barrier, service providers commonly employ cloud-computing solutions. Under this setup, an application collects frames and transmits them to a base server where powerful server-grade machines perform SR. However, in latency- and privacy-sensitive applications, the high response time and security risks of cloud

computing may not be tolerable. Furthermore, the need for constant Internet connectivity and the power consumption overhead of exchanging data with the cloud together with the cost of hosting a data center often prohibits the offloading of computations. As a result, there is an emerging need to develop methods and systems that alleviate the limitations of cloud-based computing by executing SR networks using local on-device processing [13, 36, 52].

However, as SR networks are computationally expensive, achieving 30 fps using on-device resources is impractical for upscaling to large image resolutions. For instance, given that mobile digital cameras, such as Pixel 3's, are able to capture and stream in extremely high image resolutions, achieving such resolutions in real-time by running SR networks locally is currently unrealistic. Therefore, common realistic applications of SR on mobile, such as zoom, are image-centric, rather than video-focused. Another practical application of mobile SR involves saving data. Popular social media networks such as Facebook, Instagram and Reddit and messaging applications such as Snapchat are image-heavy applications which constantly use data as the user scrolls his feed or sends a message. Given the popularity of data-saving alternatives such as Facebook Lite, features that enable devices to download low-resolution images of a user's feed and/or messages and upscale them locally would be not only feasible, but also well-received. Moreover, minimizing the network bandwidth needed to load an image feed would allow the app to work more responsively under harsh network conditions and operate in areas with poor cloud connectivity.

In this paper, we propose MobiSR, a novel automated framework that pushes the performance of on-device SR networks. Drawing from the fact that not all inputs have the same upscaling difficulty, MobiSR introduces model compression as a design dimension for the local processing of SR models and introduces a hardware-aware scheduling scheme for allocating inputs to model-compute engine pairs. To explore the model space, the proposed framework starts from a user-supplied SR network and employs a set of compression techniques in order to generate multiple SR networks with varying accuracy-workload characteristics. Upon deployment, a difficulty evaluation unit estimates the upscaling difficulty of incoming samples. Based on the observation that some image patches are shown to be more difficult to upscale for both large and compact models, while some patches are handled better by larger models, the framework schedules inputs accordingly to strike an optimal balance between image quality and speed. Specifically, the inputs that are classified as difficult are computed using a less accurate, but compact model to obtain a rapid upscaling, while easier inputs are assigned to a larger, but more accurate model. Overall, MobiSR considers the error tolerance of the target application in order to perform model selection and tailors its scheduling policy to both the selected SR models and the available compute engines. The key contributions of this paper are the following:

- The introduction of a two-model super-resolution system that exploits the upscaling difficulty of incoming patches to boost the performance of on-device SR. A novel tunable difficulty evaluation unit is presented that estimates the upscaling difficulty of incoming image patches and schedules them across different model-compute engine pairs at run time.
- A design space exploration methodology that considers the user-supplied SR model and the target mobile platform together with a user-specified error tolerance and generates an optimized SR system. By treating model compression as a design dimension and employing a hardware-aware scheduling policy, the proposed methodology explores candidate designs at both the model and scheduling level and generates an SR system tailored to meet the user-specified error tolerance at the minimum latency.

2 BACKGROUND

Since the introduction of using CNNs for SR tasks in [6], there has been a surge in SR models that utilized popular techniques such as attention [3], residual blocks [15], and generative adversarial networks [10]. These models aim to either map low-resolution images closer to their high-resolution ground truth or make SR images look more naturally pleasing. The former, which are usually trained on either the L1 or Mean Square Error (MSE) loss, favour pixel-to-pixel comparisons and are evaluated on image distortion metrics such as MSE, Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index (SSIM) [54]. The latter, on the other hand, are usually trained using a combination of different loss functions, including perceptual [27] and adversarial losses [10]. These models focus on the perceptual quality of the image and are evaluated on no-reference metrics such as Natural Image Quality Evaluator [41] and perceptual score [38]. In this work, we focus on the former, *i.e.* mapping low-resolution images closer to their high-resolution ground truth.

Unlike models that are optimized for discriminative tasks, SR models are resource-intensive networks as each layer needs to maintain or upscale the spatial dimensions of its feature maps. As a result, the number of multiply-add operations are typically counted in the billions as opposed to millions in discriminative networks [2]. Although the research community has made a few steps towards constructing efficient SR models that are optimized for mobile platforms, (1) running these models on-device is still costly and (2) popular compression techniques have not yet been utilized to derive lightweight, mobile-friendly variants. For instance, in

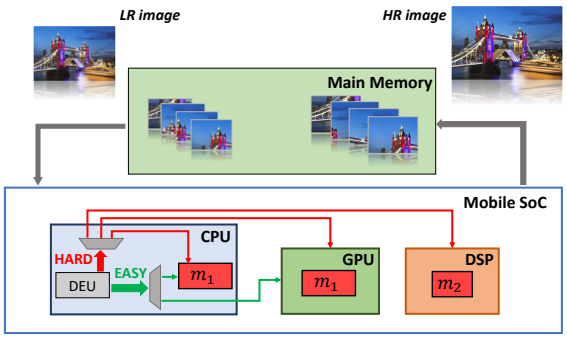


Figure 1: MobiSR's system architecture.

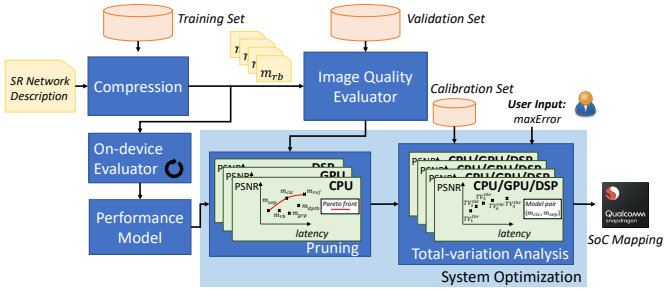


Figure 2: MobiSR's processing flow.

the experiments presented in Section 4, the winning model [53] of the recent 2018 PIRM Challenge on perceptual SR on mobile [25] requires more than 1.4 s to ×4 upscale an image to 720p on the Hexagon DSP of Qualcomm Snapdragon 845.

Challenges of on-device SR. As the size of the image increases, running SR models on a single compute engine is difficult to scale or even impractical; upscaling a large image may lead to a memory overload. Cloud-based solutions [5, 8, 14] can be deployed to offload the expensive computation. However, such solutions rely on a fast and stable communication channel and the need to maintain privacy, assumptions that are difficult to achieve in practice. Another solution would be to load-balance the computation of upscaling across the available on-device compute engines of the target mobile System-on-Chip (SoC). However, naively load-balancing SR models on multiple compute engines fails to utilize hardware-specific optimizations; different compute engines are optimized for different types of layers of a network. Furthermore, using reduced-precision compute engines in an uninformed manner can substantially affect the application-level quality of result (QoR). Therefore, there is a need to better utilize on-device resources to improve both the efficiency and scalability of running SR models locally.

SR model compression. So far, substantial effort has been invested on developing network compression techniques, such as pruning [12, 57], quantization [11], and knowledge distillation [18], for building efficient neural networks. In particular, a number of convolution approximations, such as low-rank tensor decomposition [47], have been successfully employed as a primary component in building fast and accurate discriminative vision models [9, 19, 58, 61]. These techniques typically aim to express a convolution as a sequence of simpler tensor operations, reducing in this manner the storage and computation cost of the network. With current SR models being excessively large, exploiting the potential of existing compression techniques can lead to significant gains in efficiency. Nevertheless, each technique provides varying gains depending on the target hardware optimizations, but also on the model and the level of quantization involved. Therefore, a key challenge in accelerating SR models is selecting appropriate convolution-approximation techniques based on both their impact on the accuracy of the given model and their efficient mapping on the available compute engines.

3 MobiSR

In this section, we present the high-level flow of MobiSR followed by a detailed description of its internal components.

3.1 Overview

Given a particular SR task, MobiSR searches the space of candidate on-device designs and generates a two-model system optimized for the target mobile platform. Upon deployment, the generated super-resolution system (Fig. 1) consists of:

- A compact network mapped on a high-performance compute engine that trades-off QoR with low processing latency; this can be an aggressively compressed model running on the DSP of the target mobile SoC.
- A large network which guarantees the user-specified QoR at the cost of a larger workload; this can be a lightly compressed model or a user-defined reference model running on the CPU and GPU of the target SoC.
- A tunable difficulty-aware scheduler that parallelizes the incoming low-resolution image by dispatching each image patch to the appropriate model-compute engine pair based on its estimated *upscaling difficulty*.

The key idea behind the proposed approach is that, instead of processing the full set of patches using the large and expensive network, inputs that are classified as hard-to-upscale for both networks are rapidly processed by the compact network, with only a fraction of the inputs processed by the expensive large network, reducing in this way the overall latency of the system. Furthermore, the distortion that is induced due to the network compression of the compact model is restored by tuning the portion of images processed by each network based on the user-specified error threshold.

A high-level overview of MobiSR's flow is presented in Fig. 2. The framework is supplied with a high-level description of an SR network (*i.e.* PyTorch[1] model), the specifications of the target mobile platform and an error tolerance in an image reconstruction quality metric (*e.g.* PSNR). As a first step, the *Compression* module applies a set of transformations over the supplied network in order to modify its topology and generate a number of compressed variants. To characterize their latency-QoR trade-off, each model is evaluated with respect to both its SR performance and on-device processing latency by the *Image Quality* and *On-device Evaluator* respectively. The *On-device Evaluator* performs a number of runs on the compute engines of the target mobile platform and measures the average latency of each (model, compute engine) pair.

Given the latency measurements, an analytical performance model is populated which enables the rapid estimation of the attainable latency for different scheduling schemes across the available devices. Next, the *Pruning* module takes as input the (PSNR, latency) of each (model, compute engine) pair as generated by the *Image Quality* and *On-device Evaluators*. By examining the PSNR-latency space of each compute engine, only the models that lie on the Pareto front are kept, with the rest of the dominated models discarded as inefficient, reducing in this manner the space of candidate models. After the pruning step, the *Total-variation Analysis* module is responsible for both tuning the difficulty-aware scheduler and selecting the models to be mapped on the target platform. Overall, given the user-specifed error tolerance, MobiSR generates a two-model system together with an associated scheduler tailored for the target mobile platform.

3.2 Model Space

In MobiSR, the user-supplied SR model comprises the starting point for model selection. In this setting, the space of candidate models is determined by the techniques employed by our framework in order to modify the topology of the reference network. The complete model space is formed by defining a set of model transformations to change the complexity-QoR characteristics of the reference model. Given the computation cost of a standard $K_h \times K_w$ convolution

$$S \cdot D \cdot K_h \cdot K_w \cdot F_h \cdot F_w \tag{1}$$

where S is the number of input channels, D is the number of output channels and $F_h \times F_w$ is the feature map size, MobiSR employs the following set of transformations:

Residual Bottleneck Block $rb(r)$**:** First introduced in the ResNet model [15], the residual bottleneck design substitutes a conventional convolutional layer with a 1×1 convolutional layer, used to compress the number of channels by a reduction factor r, followed by a $K_h \times K_w$ convolutional layer. Then, another 1×1 convolutional layer along with a skip connection are employed to recover the number of output channels. The reduction in computation cost over a standard $K_h \times K_w$ convolutional layer is therefore

$$\frac{S}{D \cdot K_h \cdot K_w \cdot r} + \frac{S}{r^2 \cdot D} + \frac{1}{K_h \cdot K_w \cdot r} \tag{2}$$

Group Convolutions $grp(g)$**:** The use of group convolutions [34] was introduced as a method of reducing the number of both parameters and operations with minimal impact on task-level performance [55]. This is achieved by splitting the convolutions channel-wise and computing them separately. In other words, the input feature maps are grouped and convolution is performed independently in each group. This leads to a computation cost reduction of $\frac{1}{g}$ as compared to a standard convolutional layer.

Depthwise Separable Convolutions $dpth$**:** Depthwise convolutions are referred to as group convolutions in which the number of input channels is equal to the number of groups, $S = g$. In order for information to flow among groups, depthwise convolutions are usually paired with a 1×1 convolution and the combination is known as depthwise separable convolution, which was first introduced in [47] and termed in [19]. From a workload perspective, depthwise separable convolutions yield a computation cost reduction of

$$\frac{1}{D} + \frac{1}{(K_h \cdot K_w)^2} \tag{3}$$

Separable Convolutions sep**:** This technique substitutes each $K_h \times K_w$ convolutional layer with a $1 \times K_h$ followed by a $K_w \times 1$ convolution, separating the convolution dimension-wise and resulting in a computation cost reduction of

$$\frac{1}{K_h} + \frac{1}{K_w} \tag{4}$$

Inverted Residual Blocks $invr(e)$**:** Inverted residual blocks expand the number of channels by an expansion factor of e by means of a 1×1 convolution, followed by a $K_h \times K_w$ convolution and another 1×1 convolution to recover the initial number of channels. This technique enables the use of skip connections directly on the bottleneck layers, resulting in an increase in computation cost, which is equal to that of Eq. (2) with $r = \frac{1}{e}$, but also in performance. Due to the increase in workload, inverted residual blocks were used together with depthwise convolutions when first introduced in [45].

Channel Shuffle $chlshf$**:** Channel shuffling was introduced in [61] to improve representational capability by changing the order of the channels, allowing information flow among channel groups. Specifically, an output of a grouped convolutional layer, which has g groups of $\frac{D}{g}$ channels each,

[1] https://pytorch.org/

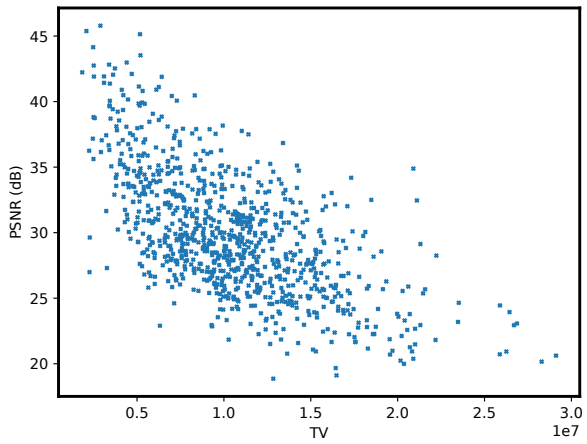


Figure 3: Low-resolution images along with their TV and the PSNR achieved after ×4 upscaling using our reference model, m_{ref}, for images in the DIV2K training and validation dataset.

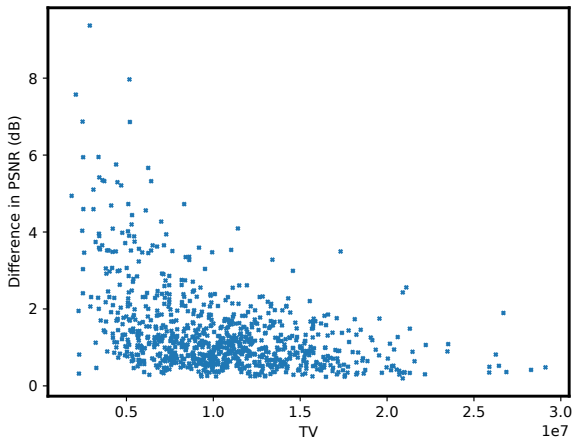


Figure 4: PSNR difference of ×4 upscaling between our reference model, m_{ref}, and a more compact model, m_{s2}.

(a) 0301.png
PSNR: 45.38 / TV: 0.2e7

(b) 0063.png
PSNR: 20.62 / TV: 2.9e7

Figure 5: TV of low-resolution images from the DIV2K dataset and the PSNR achieved after ×4 upscaling using our reference model, m_{ref}.

is reshaped into $\left(g, \frac{D}{g}\right)$, transposed into $\left(\frac{D}{g}, g\right)$, and flattened back to the number of output channels, D.

Channel Split *chlsplt*: The splitting of feature channels into branches is termed as a "channel split" in [39] and was introduced to improve processing speed. For instance, [39] uses channel splitting to split the number of channels into two branches. Convolutions are performed only on a single branch before both branches are concatenated, resulting in a reduction in workload.

Given these compression methods, we define the transformations set T as follows:

$$T = \{rb(r), grp(g), dpth, sep, invr(e), chlshf, chlsplt\} \quad (5)$$

To generate a new candidate model, we apply one transformation from the transformations set over the reference model:

$$m \xleftarrow{t} m_{\text{ref}}, \; t \in T \quad (6)$$

Formally, we capture the configuration of a model by defining a tuple representation of m and the overall model space by means of a model set $\mathcal{M}$ (Eq. (7)) that contains all reachable candidate models.

$$\mathcal{M} = \{m \mid m = \langle m_{\text{ref}}, T^*, \boldsymbol{\theta} \rangle\}, \; T^* \subset T \quad (7)$$

where m_{ref} is the topology of the reference model, T^* is the subset of applied transformations that are applied on m_{ref} to obtain m, and $\boldsymbol{\theta}$ are the learned parameters of m after the training process.

3.3 Difficulty Evaluation Unit

To sustain the QoR within the tolerance bounds of the user while achieving higher processing speed, MobiSR exploits the fact that not all image patches have the same upscaling difficulty. To this end, the Difficulty Evaluation Unit (DEU) is responsible for examining each patch and determining its complexity. To estimate upscaling difficulty, we employ the total variation (TV) metric [44]. Total variation captures the complexity of an image by examining its spatial variation, with its anistropic version for a patch p defined as:

$$TV(p) = \sum_{i,j} |p_{i+1,j} - p_{i,j}| + |p_{i,j+1} - p_{i,j}| \quad (8)$$

Fig. 5 presents a visual comparison between two images with low (Fig. 5a) and high (Fig. 5b) TV values together with the associated PSNR achieved after ×4 upscaling using our reference model, m_{ref}. As illustrated in the figures, an image consisting of unstructured fine details and texture (Fig. 5b), has a higher TV and is harder to upscale compared to a highly structured or smoother image (Fig. 5a).

To investigate the relationship between upscaling difficulty and TV of a given image in super-resolution settings, we examined the TV of each image in the DIV2K training and

validation sets, together with the achieved PSNR obtained by our reference model, m_{ref}, which is described in Section 4. As depicted in Fig. 3, images with higher TV values tend to yield lower PSNR and hence are harder to upscale, while lower-TV images tend to reach higher PSNR and thus are upscaled with higher quality.

Following these observations, we define an image patch as hard-to-upscale based on the following criterion:

$$TV(p) > TV^{\text{thr}} \tag{9}$$

where the TV threshold TV^{thr} is a tunable parameter whose value is automatically configured by MobiSR as discussed in Section 3.5.

Upscaling-Difficulty-aware Scheduling. After computing the TV of an incoming patch, the DEU is responsible for dispatching it to the suitable model between m_1 and m_2. The goal is to employ a scheduling strategy that will not exceed the user-specified error tolerance and will yield the lowest latency. To this end, we explore the behavior of the model pair (m_{ref}, m_{s2}) on patches with varying TV values. Fig. 4 shows the PSNR difference between models m_{ref} and m_{s2} as a function of the value of TV for the DIV2K training and validation set. As observed from the figure, patches that are harder to upscale based on the TV criterion (*i.e.* towards the right in Fig. 4) are almost equally hard for the two models. On the other hand, on easier-to-upscale patches, the larger model is able to achieve significantly higher PSNR. To exploit this property, an upscaling-difficulty-aware scheduling policy is proposed which directs easy-to-upscale patches to the larger model and hard-to-upscale patches to the more compact model. In this manner, higher-TV patches that are almost equally hard for both models are processed rapidly using the more compact model, with easier patches processed by the larger model to sustain the PSNR within the specified bounds.

Algorithm 1 presents the overall scheduling scheme. Model m_1 is mapped on the CPU and GPU engines with model m_2 mapped on the available DSP. Instead of solely using the per-patch upscaling difficulty as a scheduling criterion, load balancing is also employed to sustain the utilization of the available compute engines high. In this setting, Algorithm 1 takes as inputs the SR model pair (m_1, m_2), the estimated execution time t_{ce} for processing a patch with model m on compute engine ce and the selected TV threshold TV^{thr}. For each patch, the DEU first computes the associated TV value (line 3) and then dispatches the patch to the appropriate model-compute engine pair based on the total-variation criterion (line 4). In the case of an easy-to-upscale patch, the patch is allowed to be processed only by m_1 and thus the DEU dispatches the patch to either the CPU or the GPU, aiming to balance the load of the two engines (lines 5-7). In the case of hard-to-scale patches, the DEU allows the patch

Algorithm 1: Upscaling-difficulty-aware scheduling for parallel load-balanced on-device super-resolution

Input: Image I
SR model m
Execution time per compute engine $\boldsymbol{t}^{\text{CE}}_{\{C\mathcal{E}\}} \in \mathbb{R}_0^{+|C\mathcal{E}|}$
Total-variation threshold TV^{thr}

1 $\boldsymbol{t}^{\text{end}}_{\{C\mathcal{E}\}} = \mathbb{0}^{|C\mathcal{E}|}$ // *End time per compute engine*
2 **foreach** *patch* $p \in I$ **do**
3 $\quad TV \leftarrow \text{CalcTV}(p)$
4 $\quad$ **if** $TV \leq TV^{thr}$ **then**
5 $\quad\quad i \leftarrow \text{argmin}\left(\boldsymbol{t}^{\text{end}}_{\{\text{CPU,GPU}\}}\right.$
6
7 $\quad\quad\quad\quad \left. + \boldsymbol{t}^{\text{CE}}_{\{\text{CPU,GPU}\}}\right)$
8 $\quad$ **else**
9 $\quad\quad i \leftarrow \text{argmin}\left(\boldsymbol{t}^{\text{end}} + \boldsymbol{t}^{\text{CE}}\right)$
10
11 $\quad$ **end**
12 $\quad \boldsymbol{t}^{\text{end}}_i \leftarrow \boldsymbol{t}^{\text{end}}_i + \boldsymbol{t}^{\text{CE}}_i$
13 **end**

PSNR-preserving engines (*e.g.* CPU, GPU) w/ restricted load balancing.

Low-precision engines (*e.g.* DSP) w/ load balancing across $C\mathcal{E}$.

to be directed to m_2, but also includes m_1 as a candidate in order to avoid oversubscription of m_2's compute engine. Since processing a patch with m_1 does not degrade the resulting PSNR, hard patches are also allowed to be processed by m_1 in case the DSP is overloaded (lines 8-10). On the other hand, easy patches are restricted to run using m_1 in order to avoid a significant quality loss due to m_2's compression.

The range of values of total variation tends to vary between different domains. To estimate the dynamic range of TV on the target domain, MobiSR employs a user-supplied calibration set consisting of a small number of input samples. Given a few patches, the dynamic range of TV for a given dataset is estimated in order to tune the domain-specific total-variation threshold, TV^{thr}.

3.4 Performance Model

To efficiently explore different candidate designs without the need for implementations, a performance model is constructed that rapidly estimates a design's latency. To formally capture the processing resources of the target mobile platform, we define a compute engine set, $C\mathcal{E}$, which includes the compute engines that are available on the target chipset. In general, $C\mathcal{E}$ can represent a diversity of mobile SoCs hosting heterogeneous compute engines, ranging from the ubiquitous mobile CPUs and GPUs to the newer emerging NPUs [26]. For instance, Qualcomm Snapdragon 845 SoC (SDM845) is represented as $C\mathcal{E}_{\text{SDM845}} = \{CPU, GPU, DSP\}$. With this formulation, given an SR model m and a single compute engine $ce \in C\mathcal{E}$, the execution time of upscaling an image I using the (m, ce) pair is estimated as:

$$t^{\text{total}}_{ce}(I, m) = \sum_{\text{patch } p \in I} t_{ce}(p, m) \tag{10}$$

where $t_{ce}(p, m)$ is the execution time for a single patch p when model m is mapped on compute engine ce. The per-patch execution time $t_{ce}(p, m)$ is measured by the *On-device Evaluator* by means of a number of benchmark runs.

Following our difficulty-aware scheduling presented in Section 3.3, each model-compute engine pair processes only the samples that lie within its total-variation threshold, TV^{thr}. To capture this strategy the execution time model is modified as follows:

$$t_{ce}^{\text{total}}(I, m, TV^{\text{thr}}) = \begin{cases} \sum_{p \in I} t_{ce}(p, m) \mathbb{1}(TV(p) \leq TV^{\text{thr}}), & m \text{ is } m_1 \\ \sum_{p \in I} t_{ce}(p, m) \mathbb{1}(TV(p) > TV^{\text{thr}}), & m \text{ is } m_2 \end{cases}$$

where $\mathbb{1}(\cdot)$ is the unity function that evaluates to 1 when its bracketed condition is true. MobiSR distributes patches across the available engines in order to maximize the utilization of the on-chip compute resources and exploit the inherent parallelism across independent patches. Under this scheme, the overall latency of upscaling image I using model m on the target SoC is estimated as in Eq. (11).

$$L(I, m, TV^{\text{thr}}, \mathcal{CE}) = \max\left(\left\{t_{ce}^{\text{total}}(I, m, TV^{\text{thr}}) \mid ce \in \mathcal{CE}\right\}\right) + t^{\text{stitch}} \quad (11)$$

where the first term captures the parallel execution of patches across engines and t^{stitch} represents the overhead of assembling together the partial results of all patches to form the final high-resolution image.

3.5 System Optimization

The developed framework aims to determine a pair of models together with a total-variation threshold that minimize the processing latency of performing on-device SR on the target mobile platform, given a user-supplied error tolerance. In this context, we pose the following optimization problem:

$$\min_{(m_1, m_2), TV^{\text{thr}}} L\left(I, (m_1, m_2), TV^{\text{thr}}, \mathcal{CE}\right) \quad (12)$$

$$\text{s.t. } \text{PSNR}(I, m_{\text{ref}}, 0) - \text{PSNR}\left(I, (m_1, m_2), TV^{\text{thr}}\right) \leq \epsilon_{\max}$$

where L, TV^{thr} and $\epsilon_{\max}$ are the latency in s/input, the total-variation threshold and the user-specified error tolerance respectively. Under this formulation, the objective function aims to find the tuple $\langle (m_1, m_2), TV^{\text{thr}} \rangle$ that minimizes latency with a constraint on the degradation of QoR as captured by PSNR.

Given a reference SR model m_{ref}, the optimization problem in Eq. (12) is defined over all candidate model pairs in the model space $\mathcal{M}$ presented in Section 3.2. Formally, we express this as the product $\mathcal{M} \times \mathcal{M}$. Furthermore, each pair can be deployed with a different TV threshold and therefore, given N_{TV} discrete candidate total-variation thresholds, the total number of alternative designs to be explored can be calculated as follows:

$$|\mathcal{M}|^2 \cdot N_{\text{TV}} \quad (13)$$

In this setup, the objective function $L : \{\mathcal{M} \times \mathcal{M}, TV^{\text{thr}}\} \rightarrow \mathbb{R}^+$ can be evaluated for all $(m_1, m_2) \in \mathcal{M} \times \mathcal{M}$ by means of the performance model of Section 3.4. In theory, the optimal design could be obtained by means of an exhaustive search with complete enumeration of all possible designs.

With latency and PSNR being a function of the TV of each patch of the processed image, evaluating $L(\cdot)$ and $\text{PSNR}(\cdot)$ is data-dependent and hence requires running each possible design $\langle (m_1, m_2), TV^{\text{thr}} \rangle$ over a task-specific dataset to assess its attainable PSNR and latency. To avoid the overhead of an excessive number of evaluation runs, MobiSR employs two strategies for pruning the design space: 1) for each compute engine, we keep only the models that lie on the Pareto front of the PSNR-latency space. In this manner, models that are dominated with respect to their PSNR-latency balance on a given compute engine are discarded as inefficient; and 2) we impose the constraint that m_2 is equally or more compact than m_1. In this manner, we guide MobiSR to select two models with different PSNR-latency characteristics, in order to combine the high PSNR of m_1 with the fast processing of m_2.

After the pruning stage, MobiSR searches the remaining design space to determine the highest performing configuration of the tuple $\langle (m_1, m_2), TV^{\text{thr}} \rangle$. To enable fast and exhaustive exploration, the developed performance model of Section 3.4 is employed. For each (m_1, m_2) pair, an analysis is initially performed over the user-supplied calibration set, yielding the achieved PSNR and latency for different values of TV^{thr}. As a final step, MobiSR selects the fastest design that lies within the tolerated error of the target application.

4 EVALUATION

This section presents the effectiveness of MobiSR in significantly improving the performance of on-device super-resolution by examining its core components and comparing with the currently standard implementations and highly optimized difficulty-unaware designs.

4.1 Experimental Setup

In our experiments, we target Intrinsyc's Open-Q 845 board mounting the Qualcomm Snapdragon 845 SoC (SDM845). SDM845 integrates an octa-core Kryo 385 CPU alongside an Adreno 630 mobile GPU and a Hexagon 685 DSP on the same chip. [2] All SR models were developed and trained using PyTorch (v1.0) and run on the Open-Q 845 board using the

[2] MobiSR can also target modern mobile chipsets equipped with CPU, GPU and NPU/DSP engines such as Samsung Exynos 9820, Qualcomm Snapdragon 855 and Huawei Kirin 810 SoCs.

Model		Params (K)	Latency (ms)			Average PSNR/SSIM†			
			CPU	GPU	DSP	Set5	Set14	B100	Urban100
SRCNN	[6]	**57**	9742.97	**584.83**	**656.44**	30.47/0.8610	27.57/0.7528	26.89/0.7108	24.51/0.7232
VDSR	[30]	665	198027.52	7164.60	2623.61	31.53/0.8840	**28.42/0.7830**	27.29/0.7262	25.18/0.7534
FEQE-P	[53]	96	**2996.92**	911.61	1475.45	31.53/0.8824	28.21/0.7714	27.32/0.7273	25.32/0.7583
m_{ref}		152	4570.08	2792.43	1220.00	**31.73/0.8873**	28.24/0.7729	**27.33/0.7283**	**25.34/0.761**

†Calculated using full 32-bit floating-point precision (FP32).

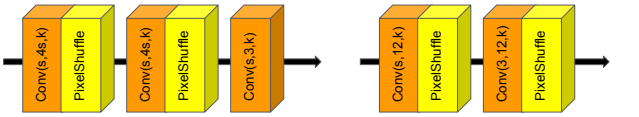
Table 1: Comparison of reference model with state-of-the-art efficient SR models (×4 upscaling).

Snapdragon Neural Processing Engine (SNPE)[3] SDK (v1.21). SNPE allows targeting all three CPU, GPU and DSP engines of the SDM845 platform with highly optimized execution of CNN layers. The three compute engines employ different precision for data representation; namely the CPU, GPU and DSP use single-precision floating-point (FP32), half-precision floating-point (FP16) and 8-bit fixed-point (INT8) respectively for both storage and computation. All models that were run on the Hexagon DSP were first quantized offline to INT8 using linear quantization, with the per-layer scaling factors tuned based on the dynamic range of weights and activations on the DIV2K validation set.

Datasets and Training Scheme. Following the common practice of the super-resolution community [37, 60, 63], all SR models were trained on the training set of the DIV2K dataset [50] and validated on its validation set, comprising 800 and 100 images of 2K resolution with diverse contents respectively. For the evaluation, four benchmark datasets were used which constitute the standard for assessing SR models in the super-resolution literature [50]: Set5 [4] and Set14 [56] comprising five and fourteen images respectively that are commonly used across the image processing community, B100 [40] with 100 images of real-life scenes and Urban100 [22] consisting of 100 images depicting urban environments.

For the training of the SR models, we employ a similar scheme to the one used by [62] and [37]. First, data augmentation was applied on the DIV2K training set by randomly flipping horizontally and rotating by 90°, and all images were normalized by subtracting the training set's mean. Next, training was performed in 300 epochs using an Adam optimizer [32] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ and L1 loss. Each mini-batch consists of 16 RGB patches with input size of 96×96 for both ×2 and ×4 upscaling. The starting learning rate was set to 10^{-4} and was halved after 200 epochs. Lastly, we train ×2 models from scratch and use them as pre-trained models to train ×4 models, confirming the findings of [37] that using the weights of the ×2 models as initial weight values for ×4 models leads to faster training convergence.

[3]https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk



(a) RCAN ×4 Upsampling Module **(b) m_{ref} ×4 Upsampling Module**

Figure 6: We reduce the number of feature maps in RCAN to improve processing speed. $Conv(S, D, K)$ is a K×K convolution with S input and D output feature maps.

Implementation Details. All SR designs presented in this section were run on SDM845 using the high-performance profile of the SNPE SDK that configures the hardware for maximum processing speed. All inputs to the SR models during on-device inference are partitioned into overlapping patches of size 90×160, with partial results stitched together at the end. The reported latency in all experiments is based on the average across 100 runs, with the latency measurements conducted using the SNPE's timing utilities. The images in the aforementioned SR datasets have different sizes and therefore we report the average latency taken to upscale an image in the given dataset. For all other experiments, we assume a target high-resolution image with 720p resolution (1280×720).

4.2 Evaluation of Model Transformations

In MobiSR, the user supplies a reference model and the framework applies a series of model transformations to generate a set of compressed models. This set is then automatically pruned to remove suboptimal models, resulting in a list of Pareto-optimal candidate models to select from in order to produce the resulting two-model SR system. In our experiments, we exemplify this process by selecting a reference model, m_{ref}, that is comparable to the state-of-the-art models in the existing literature for mobile SR and then pass it through MobiSR.

Reference Model Selection. We adopted the residual channel attention network (RCAN) [62] as our reference

Model	Params (K)	Latency (ms)			Speedup			Error	
		CPU	GPU	DSP	CPU	GPU	DSP	CPU/GPU†	DSP
m_{ref}	152	4570.08	2792.43	1220.0	1.00	1.00	1.00	0.00%	0.00%
$m_{\text{rn(r=2)}}$	58	2694.94	2626.78	1508.56	1.69	1.06	0.80	2.84%	14.27%
$m_{\text{rn(r=4)}}$	22	1434.42	2657.49	1561.97	3.18	1.05	0.78	3.94%	22.39%
m_{rxn}	30	1850.82	10692.13	2508.63	2.46	0.26	0.48	3.70%	19.27%
m_{m1}	30	1969.74	2723.66	1080.59	2.32	1.02	1.12	2.48%	4.15%
m_{eff}	24	1284.21	2700.24	1398.407	3.55	1.03	0.87	3.19%	5.71%
m_{m2}	88	4045.80	2846.70	1327.65	1.12	0.98	0.91	2.32%	2.11%
m_{clc}	30	1910.74	2722.86	1061.38	2.39	1.02	1.14	**1.93%**	**0.87%**
m_{s1}	13	1263.90	12367.24	3060.82	3.61	0.22	0.39	4.69%	26.1%
m_{s2}	17	1023.26	2595.59	973.07	**4.46**	**1.07**	**1.25**	3.03%	3.07%

† We obtained similar results on CPU with FP32 and GPU with FP16.

Table 2: Performance of our explored model space for ×4 upscaling. Error drop is based on PSNR on Urban100.

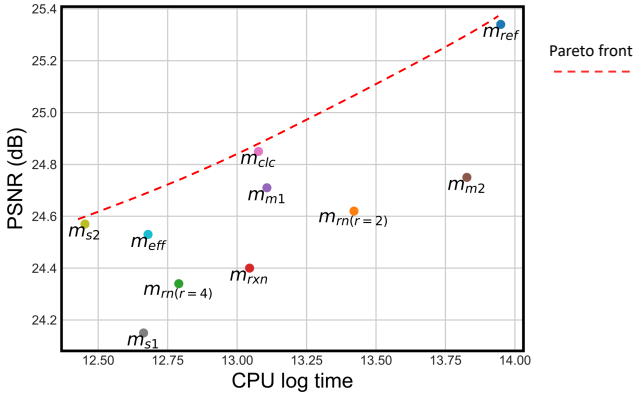


Figure 7: PSNR vs CPU latency of MobiSR-generated models on SDM845 (×4 upscaling on Urban100).

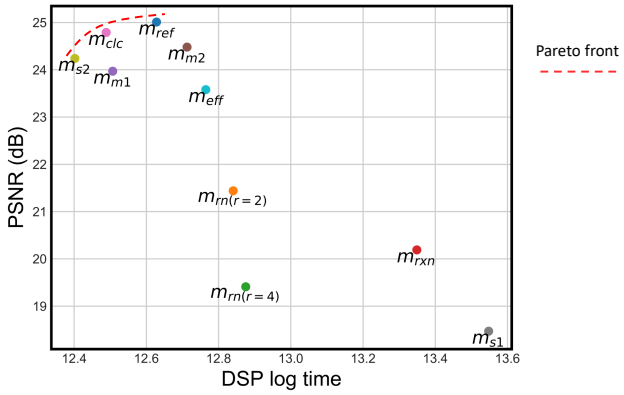


Figure 8: PSNR vs DSP latency of MobiSR-generated models on SDM845 (×4 upscaling on Urban100).

model, m_{ref}, as RCAN yields the state-of-the-art performance based on PSNR/SSIM among large-scale SR models. In order for RCAN to be comparable to existing state-of-the-art mobile SR models, its architecture was modified by reducing the number of residual groups to 3, the number of residual channel attention blocks to 10, and the number of feature maps to 16. Additionally, to further reduce the computational cost of the reference model, the number of feature maps in the upscaling module was reduced by a factor of 5 and the last convolutional layer was removed. Fig. 6 shows the slight change in the upsampling module between RCAN and m_{ref}.

As shown on Table 1, by constructing a shallower variant of RCAN, we are able to achieve comparable results with state-of-the-art SR models that are hand-optimized for increased efficiency. Notably, our reference model manages to outperform the winning model of the 2018 PIRM Challenge [25] on perceptual SR on mobile, FEQE, by 20% when run on the Hexagon DSP and achieves higher PSNR across all four SR datasets. Furthermore, m_{ref} yields an average speedup of 16.01× (6.2× geo. mean) over VDSR with 4.3× fewer parameters and achieves an average PSNR improvement of 0.8 dB over the lightweight SRCNN. Regardless, MobiSR accepts any starting reference model and searches for a set of model transformations that will work best for that reference model on the given compute engines.

Explored Model Space. Based on the findings of recently proposed high- and low-level vision models, we examined specific transformation combinations from the transformation set T (detailed in Section 3.2) on our reference model by focusing on the ones that have demonstrated the highest effectiveness in the deep learning literature. Table 4 details the topologies of the MobiSR-generated compressed models together with the associated transformations that were applied over our reference model. Specifically, given the reference model, MobiSR replaces all 3×3 convolutional layers that lie in the core of the network, excluding the first layer

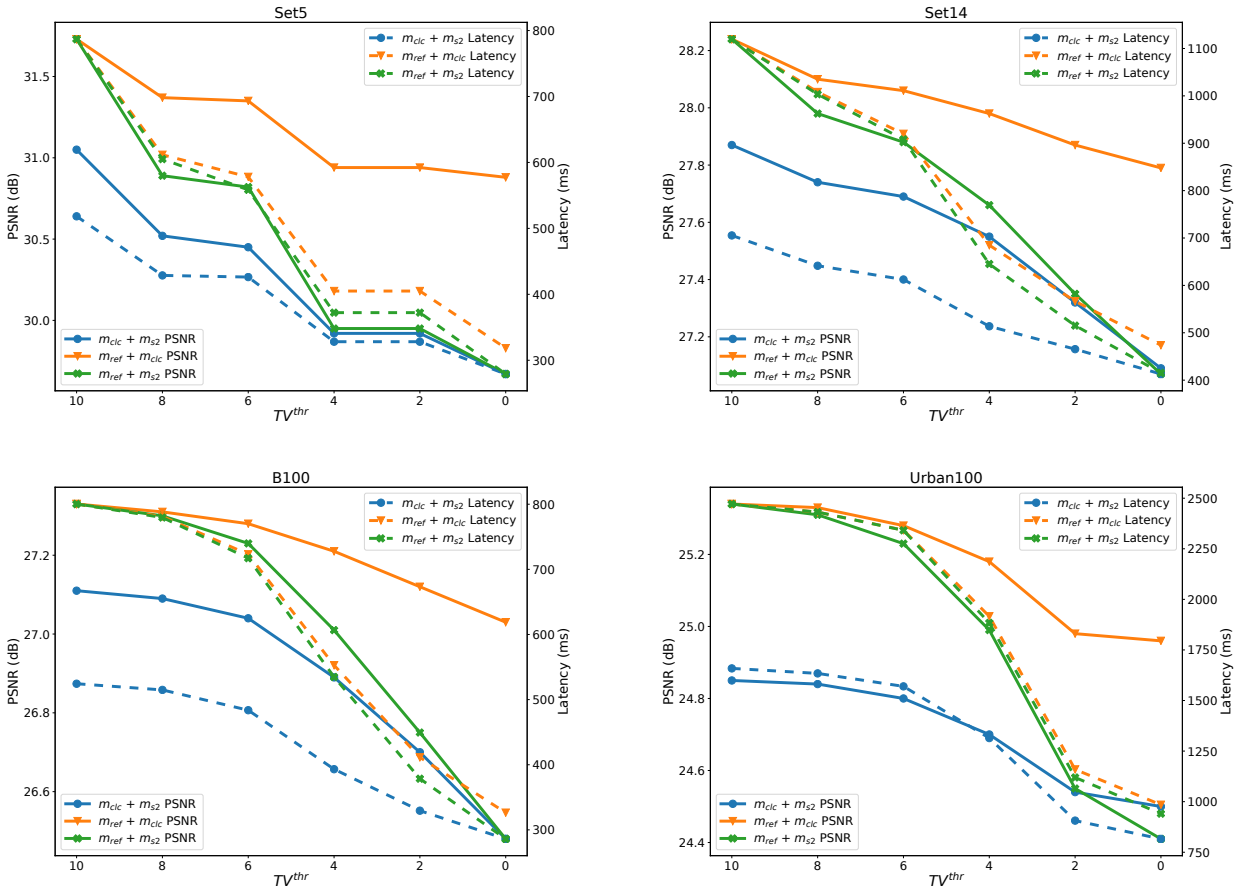


Figure 9: Achieved PSNR and measured performance as a function of TV on SDM854.

and those within the upsampling block, using a subset of transformations. After this step, the obtained compressed models are retrained from scratch following the training scheme of Section 4.1.

Table 2 lists the attainable latency of each model, the speedup over the reference model and the error with respect to PSNR, as obtained by MobiSR's *On-device* and *Image Quality Evaluator* modules. Apart from the latency-PSNR trade-off of the generated compressed models, Table 2 also highlights the compatibility of each compute engine with the various transformations that have been applied. Different compute engines are more highly optimized for a different subset of transformations. For instance, the sole use of bottleneck residual blocks, $m_{\text{rn(r=4)}}$, obtains a greater speedup on the CPU, but lower gains on the DSP as compared to the sole use of depthwise separable convolutions, m_{m1}. Furthermore, models that employ group convolutions, such as m_{s1} and m_{rxn}, yield worse latency when executed on the GPU due to suboptimal mapping. Additionally, the 8-bit quantization of the DSP had a severe impact on the representational capacity of compressed models that utilized the residual bottleneck blocks, such as $m_{\text{rn(r=2)}}$, $m_{\text{rn(r=4)}}$ and m_{rxn}. As a result, selecting the highest performing set of compressed models is dependent on both the provided reference model and the available compute engines.

Overall, Fig. 7 and 8 depict the PSNR-latency space of the generated models on the CPU and DSP of the Qualcomm SDM845 respectively. In this case, the framework picked the same three models, namely m_{ref}, m_{clc} and m_{s2}, that lie on the Pareto fronts of all three compute engines.

4.3 MobiSR PSNR and Performance vs TV

In this section, the PSNR and performance of the MobiSR-generated designs are evaluated as a function of total-variation threshold. Given the three Pareto-optimal models from Section 4.2 and the pruning strategy that dictates that m_2 is more compact than m_1, three model pairs were selected in

Model Pair	Set5 Speedup	Set5 Avg/G. Mean	Set14 Speedup	Set14 Avg/G. Mean	B100 Speedup	B100 Avg/G. Mean	Urban100 Speedup	Urban100 Avg/G. Mean
Running m_{ref} on the CPU								
(m_{ref}, m_{clc})	1.74×-4.31×	2.90×/2.78×	1.96×-4.65×	3.05×/2.90×	1.76×-4.31×	2.63×/2.47×	2.35×-5.91×	3.53×/3.28×
(m_{ref}, m_{s2})	1.74×-4.91×	3.12×/2.94×	1.97×-5.30×	3.26×/3.05×	1.76×-4.91×	2.79×/2.58×	2.35×-6.18×	3.62×/3.34×
(m_{clc}, m_{s2})	2.64×-4.91×	3.72×/3.64×	3.12×-5.34×	4.09×/4.01×	2.68×-4.91×	3.51×/3.42×	3.51×-7.13×	4.79×/4.59×
Running m_{ref} on the GPU								
(m_{ref}, m_{clc})	1.06×-2.63×	1.78×/1.70×	1.20×-2.84×	1.86×/1.77×	1.07×-2.63×	1.61×/1.51×	1.44×-3.61×	2.16×/2.01×
(m_{ref}, m_{s2})	1.06×-3.00×	1.91×/1.80×	1.20×-3.24×	1.99×/1.87×	1.07×-3.00×	1.71×/1.59×	1.44×-3.78×	2.21×/2.04×
(m_{clc}, m_{s2})	1.61×-3.00×	2.27×/2.23×	1.91×-3.26×	2.50×/2.45×	1.64×-3.00×	2.15×/2.09×	2.14×-4.36×	2.93×/2.80×
Running m_{ref} on the CPU & GPU								
(m_{ref}, m_{clc})	1.28×-2.47×	1.80×/1.75×	1.11×-2.36×	1.66×/1.59×	1.02×-2.45×	1.59×/1.51×	1.01×-2.51×	1.60×/1.49×
(m_{ref}, m_{s2})	1.30×-2.82×	1.95×/1.87×	1.11×-2.69×	1.79×/1.69×	1.02×-2.79×	1.71×/1.59×	1.01×-2.62×	1.64×/1.52×
(m_{clc}, m_{s2})	1.52×-2.82×	2.13×/2.09×	1.58×-2.71×	2.08×/2.04×	1.52×-2.79×	2.09×/2.04×	1.49×-3.03×	2.04×/1.95×

Table 3: Performance comparison of Pareto-optimal model pairs with faithful reference model m_{ref}.

the valid design space; namely (m_{ref}, m_{clc}), (m_{ref}, m_{s2}) and (m_{clc}, m_{s2}).

Fig. 9 shows the measured latency on SDM854 and the achieved PSNR across different TV thresholds for the four SR datasets. When TV^{thr} has substantially high values (towards the left hand side of the plots), the majority of incoming samples is processed by m_1 on the CPU and GPU of SDM845. In this manner, PSNR remains high, but at the cost of increased latency due to the underutilization of the DSP. As TV^{thr} decreases from left to right, the three model pairs trade off a decreased PSNR for substantially reduced processing latency. Eventually, as TV^{thr} reaches very low values, the DEU relaxes the constraints and its scheduling policy reduces to a load balancing of the incoming samples across the three compute engines, without the need to exploit the upscaling difficulty of each sample. In this manner, the highest speed up is achieved for low TV^{thr} at the expense of a significant drop in the achieved PSNR.

Table 3 lists the speedup gains of the three model pairs over the execution of m_{ref} on the CPU, GPU, and both CPU and GPU. Since m_{ref} is mapped only on the CPU and/or GPU, no degradation of PSNR is induced due to the 8-bit operations of the DSP. Overall, the parametrization of the DEU based on TV^{thr} allows the tuning of the system at a fine granularity so that even a small increase in the application-level error tolerance can be capitalized as reduced processing latency.

4.4 Evaluation of MobiSR Performance

This section presents the performance gains of MobiSR with respect to processing speed. This is investigated by comparing the generated two-model design for different PSNR drop values with a baseline single-model network. For each interval of PSNR drop, each MobiSR instance is compared with the fastest baseline single-model architecture that achieves the same or higher PSNR as the MobiSR system (shown on top of each plot in Fig. 10). The single-model baselines do not employ MobiSR's TV-based scheduling; instead each model is allowed to run in one of two modes: *i)* either with load balancing across the CPU and GPU and no PSNR degradation or *ii)* with load balancing across CPU, GPU, and DSP with PSNR drop due to the DSP's reduced precision. In this respect, the fastest single model that satisfies the PSNR drop constraint is selected at each PSNR drop interval. The overall measured runtime includes the DEU, processing all patches and the overhead of combining the partial results to construct the final high-resolution image.

Fig. 10 presents the achieved speedup across a wide range of PSNR tolerance values on the SDM845 platform when targeting the four benchmark datasets. When minimal to no PSNR drop is allowed (towards the left of Fig. 10), MobiSR selects a strict scheduling policy for the DEU with high TV values. In this manner, the large majority of patches are processed by m_1 on the CPU and GPU and the DSP remains underutilized, leading to minimal speedup. As more error is allowed, the proposed system outperforms the baseline by up to 47%, 78%, 94% and 29% for the same PSNR drop budget in Set5, Set14, B100 and Urban100 respectively.

Finally, in the case of high error tolerance, the speedup becomes less significant as uninformed load balancing using the fastest compressed model across the CPU, GPU, and DSP becomes the fastest design.

5 RELATED WORK

The emergence of mobile image-centric applications has attracted the attention of the computer vision community, with efforts for alleviating the large compute demands of large-scale SR models. Addressing on-device SR from a model perspective, SRCNN [6] and the second-generation FSRCNN [7]

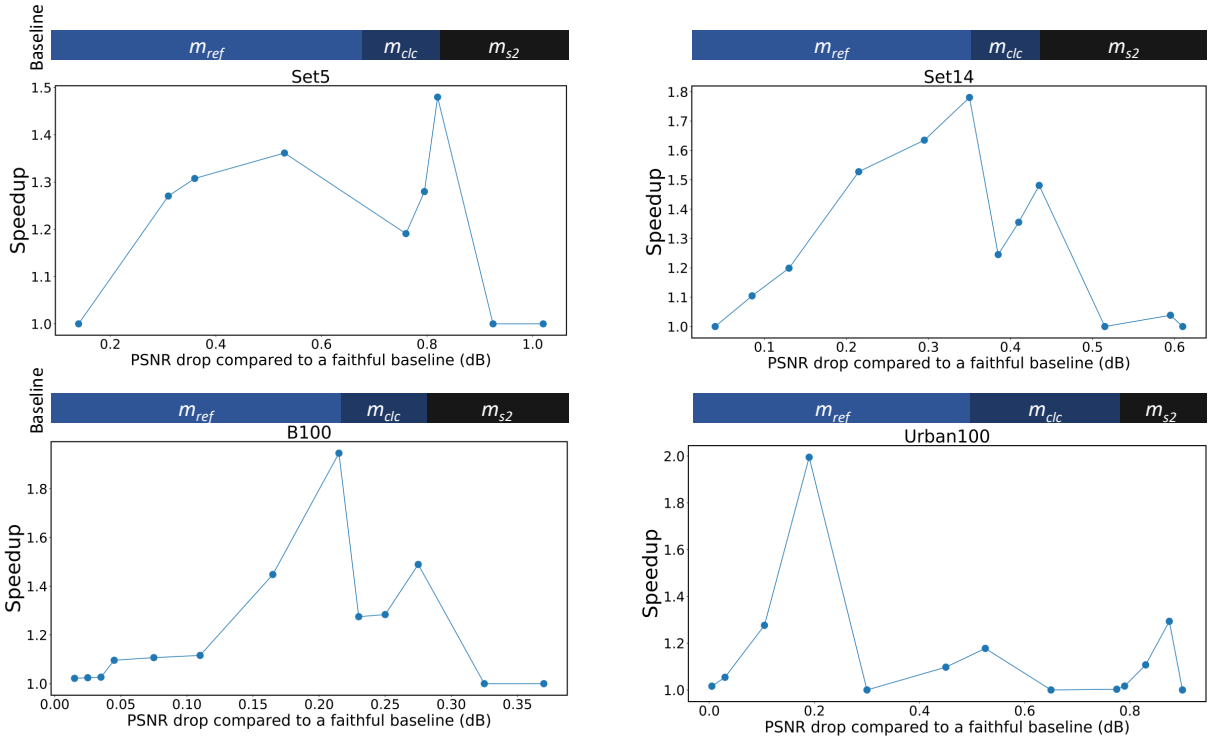


Figure 10: MobiSR's speedup as a function of error degradation.

were first proposed as efficient neural architectures for SR, consisting of only three convolutional layers. By aiming to improve the image quality, the VDSR [30] network employed a deeper design consisting of twenty layers. With a direction towards mobile settings, CARN-M [1] was proposed as a lightweight variant of the CARN architecture. Inspired by MobileNet [19], CARN-M employs recursive blocks and group convolutions to reduce the storage and compute requirements at inference time. In 2018, FEQE [53] introduced the desubpixel block, enabling a lossless downsampling at the start of the network, while reducing the computation cost throughout the rest of the network. The aforementioned works are primarily hand-tuned architectures with manually selected architectural choices aiming to reach a balance between image quality and computation cost. In this paper, we focus on the largely unexplored space of applying model transformations in a hardware-aware manner with the goal to tailor the generated system to both the application-level image quality requirements and the target mobile platform characteristics.

From a systems perspective, apart from task-agnostic frameworks for executing deep neural networks on mobile platforms [17, 24, 35, 42, 48], research efforts have mainly focused in the direction of 1) cascade systems [13, 20, 28, 33, 46, 59], 2) early-exit classifiers [21, 29, 49] and 3) specialized accelerators for CNNs [43, 51] and SR models [16, 31].

Cascade systems. Cascade systems base their operation on conditionally passing input samples through a pipeline of classifiers based on information obtained at each classification stage. `VideoStorm` [59], `NoScope` [28], `Focus` [20] and Shen *et al.* [46] focus on the task of issuing queries on video databases. A common element between these systems and MobiSR is the use of multiple networks. However, a key differentiating factor in the model generation approach is that, by exploiting video-specific optimization opportunities, these systems train class-specialized models based on the object classes that appear more often in a given video stream. In contrast, the generative nature of the super-resolution task is not amenable to such an approach. Furthermore, at the run-time model selection stage, each stage of the cascade determines whether a particular object class is present or not, and if not, the input sample is propagated to the next classification stage. Contrary to this approach, MobiSR's DEU determines which model to use based on the input image complexity and the current load of the available compute engines, without requiring information to be passed between models.

In a similar manner to MobiSR, MCDNN [13] employs a form of model selection. However, while MCDNN focuses on

Model	T^*	Inspired By	Building Block
m_{rn}	$\{rb(2)\}$ & $\{rb(4)\}$	ResNet [15]	$y \leftarrow Conv(x, s, \frac{S}{r}, (1,1), 1)$ $y \leftarrow ReLU(y)$ $y \leftarrow Conv(y, \frac{S}{r}, \frac{S}{r}, (K_h, K_w), 1)$ $y \leftarrow ReLU(y)$ $y \leftarrow Conv(y, \frac{S}{r}, s, (1,1), 1)$ $y \leftarrow y + x$
m_{rxn}	$\{rb(2), grp(4)\}$	ResNeXt [55]	$y \leftarrow Conv(x, S, \frac{S}{r}, (1,1), 1)$ $y \leftarrow ReLU(y)$ $y \leftarrow Conv(y, \frac{S}{r}, \frac{S}{r}, (K_h, K_w), g)$ $y \leftarrow ReLU(y)$ $y \leftarrow Conv(y, \frac{S}{r}, S, (1,1), 1)$ $y \leftarrow y + x$
m_{m1}	$\{dpth\}$	MobileNet [19]	$y \leftarrow Conv(x, S, S, (K_h, K_w), s)$ $y \leftarrow ReLU(y)$ $y \leftarrow Conv(y, S, D, (1,1), 1)$
m_{eff}	$\{rb(2), dpth, sep\}$	EffNet [9]	$y \leftarrow Conv(x, S, \frac{S}{r}, (1,1), 1)$ $y \leftarrow ReLU(y)$ $y \leftarrow Conv(y, \frac{S}{r}, \frac{S}{r}, (1, K_w), \frac{S}{r})$ $y \leftarrow Conv(y, \frac{S}{r}, \frac{S}{r}, (K_h, 1), \frac{S}{r})$ $y \leftarrow ReLU(y)$ $y \leftarrow Conv(y, \frac{S}{r}, S, (1,1), 1)$ $y \leftarrow y + x$
m_{m2}	$\{dpth, invr(2)\}$	MobileNetV2 [45]	$y \leftarrow Conv(x, S, S \times e, (1,1), 1)$ $y \leftarrow ReLU(y)$ $y \leftarrow Conv(y, S \times e, S \times e, (K_h, K_w), S \times e)$ $y \leftarrow ReLU(y)$ $y \leftarrow Conv(y, s \times e, S, (1,1), 1)$ $y \leftarrow y + x$
m_{clc}	$\{grp(16), chlshf\}$	ClcNet [58]	$y \leftarrow Conv(x, S, S, (K_h, K_w), g)$ $y \leftarrow chlshf(y)$ $y \leftarrow Conv(y, S, D, (1,1), 1)$
m_{s1}	$\{rb(2), grp(4), dpth, chlshf\}$	ShuffleNet [61]	$y \leftarrow Conv(x, S, \frac{S}{r}, (1,1), g)$ $y \leftarrow ReLU(y)$ $y \leftarrow chlshf(y)$ $y \leftarrow Conv(y, \frac{S}{r}, \frac{S}{r}, (K_h, K_w), \frac{S}{r})$ $y \leftarrow Conv(y, \frac{S}{r}, S, (1,1), g)$ $y \leftarrow y + x$
m_{s2}	$\{chlsplt, dpth, chlshf\}$	ShuffleNet V2 [39]	$y_1, y_2 \leftarrow chlsplit(x)$ $y_1 \leftarrow Conv(y_1, \frac{S}{2}, \frac{S}{2}, (1,1), 1)$ $y_1 \leftarrow ReLU(y_1)$ $y_1 \leftarrow Conv(y_1, \frac{S}{2}, \frac{S}{2}, (K_h, K_w), \frac{S}{2})$ $y_1 \leftarrow Conv(y_1, \frac{S}{2}, \frac{S}{2}, (1,1), 1)$ $y \leftarrow [y_1, y_2]$ $y \leftarrow chlshf(y)$

Table 4: All 3×3 convolutional layers in m_{ref} are replaced with the corresponding building blocks where *ReLU* represents the rectifier activation function, and $Conv(x, S, D, (K_h, K_w), g)$ represents a $K_h \times K_w$ convolutional layer with input x, S input channels, D output channels and g groups.

run-time model selection and the partitioning of computation between cloud and device, MobiSR employs a difficulty-aware mechanism to exploit the heterogeneous compute engines that are available on-device and parallelizes both within an image (*i.e.* parallel processing of patches) and across images (*i.e.* pipelined execution as long as images are available). Moreover, while MCDNN aims to maximize the average accuracy of classification tasks, MobiSR sets a constraint on the PSNR drop and guarantees that the average PSNR will not be compromised below user-specified bounds.

From a target platform perspective, the aforementioned systems are optimized for cloud setups that have substantially different characteristics compared to MobiSR's fully on-device system. In our case, the available compute engines share the same main memory, and in turn the same storage and bandwidth. This poses a significant additional challenge and calls for the mobile-specific methodology of MobiSR to develop and implement high-performing mobile designs.

Finally, the cascading approach of CascadeCNN [33] involves a two-model cascade with each classifier quantized at a different precision level. In this case, input samples are first processed rapidly by an aggressively quantized model. If the prediction confidence of a sample is below a tunable threshold, the input sample is passed to a higher-precision model for recomputation. Despite the fact that variable precision quantization could be integrated in the transformations set of MobiSR, CascadeCNN has so far been evaluated on FPGA-based platforms targeting image recognition tasks.

Early-exit classifiers. Designs such as BranchyNet [49], MSDNet [21] and Shallow-Deep Networks [29] approach inference acceleration from an architectural aspect. First, they focus on classification rather than generative tasks. Secondly, they explicitly introduce early-exit outputs on a single model in order to reduce the workload-quality characteristics. Nevertheless, by exploiting the fact that different samples require different amount of computation to yield a correct classification, such designs share a similar philosophy to our upscaling-difficulty-aware scheme. However, the criterion to capture an input sample's difficulty is the prediction confidence at each early exit, which is relevant to classification tasks and differs to the upscaling-difficulty metric used by MobiSR for run-time model selection.

Hardware acceleration. Several works have explored the design of custom hardware architectures for the efficient execution of CNN workloads in resource- and power-constrained settings [43, 51]. With a focus on SR, He *et al.* [16] proposed a highly optimized FPGA-based hardware accelerator tailored to the FSRCNN [7] network. Furthermore, by adopting a hardware-software codesign methodology, Kim *et al.* [31] derived a CNN-based SR model and implemented it on an FPGA-based platform. Our work focuses on programmable mobile platforms which are more flexible and enable the efficient execution of SR models in a network-agnostic manner.

6 CONCLUSION

The MobiSR framework described in this paper uses several techniques to achieve high performance for fully on-device super-resolution. Through the generation of a two-model processing system tailored to the available compute engines of the target mobile platform, the proposed framework demonstrates significant speedup compared to single-model designs without penalizing the achieved image quality. By considering the user-specified error tolerance in the design space exploration phase and exploiting the heterogeneous compute engines of commodity mobile platforms, MobiSR is able to deliver high-speed SR on-device while meeting the application-level image quality requirements.

Furthermore, as the proposed methodology is parametrized to target any arbitrary mobile SoC with heterogeneous compute engines, using MobiSR to take advantage of newer emerging platforms that consist of neural accelerators can be a key enabler for efficient mobile super-resolution with potentially larger room for performance gains.

REFERENCES

[1] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. 2018. Fast, Accurate, and Lightweight Super-Resolution with Cascading Residual Network. In *ECCV*.

[2] Mario Almeida, Stefanos Laskaridis, Ilias Leontiadis, Stylianos I. Venieris, and Nicholas D. Lane. 2019. EmBench: Quantifying Performance Variations of Deep Neural Networks Across Modern Commodity Devices. In *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications (EMDL '19)*. 1–6.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations (ICLR)*.

[4] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie line Alberi Morel. 2012. Low-Complexity Single-Image Super-Resolution based on Nonnegative Neighbor Embedding. In *British Machine Vision Conference (BMVC)*.

[5] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J. Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger. 2016. A Cloud-Scale Acceleration Architecture. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–13.

[6] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2016. Image Super-Resolution Using Deep Convolutional Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38 (2016), 295–307.

[7] Chao Dong, Chen Change Loy, and Xiaoou Tang. 2016. Accelerating the Super-Resolution Convolutional Neural Network. In *ECCV*.

[8] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger. 2018. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 1–14.

[9] Ido Freeman, Lutz Roese-Koerner, and Anton Kummert. 2018. Effnet: An Efficient Structure for Convolutional Neural Networks. In *IEEE International Conference on Image Processing (ICIP)*. 6–10.

[10] Ian Goodfellow et al. 2014. Generative Adversarial Nets. In *Advances in Neural Processing Systems*.

[11] Song Han, Huizi Mao, and William J Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations (ICLR)*.

[12] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems 28*. 1135–1143.

[13] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '16)*. 123–136.

[14] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 620–629.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[16] Z. He, H. Huang, M. Jiang, Y. Bai, and G. Luo. 2018. FPGA-Based Real-Time Super-Resolution System for Ultra High Definition Videos. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 181–188.

[17] Gopalakrishna Hegde, Siddhartha, and Nachiket Kapre. 2017. CaffePresso: Accelerating Convolutional Networks on Embedded SoCs. *ACM Trans. Embed. Comput. Syst.* 17, 1, Article 15 (Nov. 2017), 26 pages.

[18] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop*.

[19] Andrew G. Howard et al. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR* abs/1704.04861 (2017).

[20] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, and Onur Mutlu. 2018. Focus: Querying Large Video Datasets with Low Latency and Low Cost. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*.

[21] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. 2018. Multi-Scale Dense Networks for Resource Efficient Image Classification. In *International Conference on Learning Representations (ICLR)*.

[22] J. Huang, A. Singh, and N. Ahuja. 2015. Single image super-resolution from transformed self-exemplars. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[23] Zheng Hui, Xiumei Wang, and Xinbo Gao. 2018. Fast and Accurate Single Image Super-Resolution via Information Distillation Network. In *CVPR*. 723–731.

[24] Loc Huynh, Rajesh Krishna Balan, and Youngki Lee. 2016. DeepSense: A GPU-based Deep Convolutional Neural Network Framework on Commodity Mobile Devices. In *WearSys@MobiSys*.

[25] Andrey Ignatov et al. 2018. PIRM Challenge on Perceptual Image Enhancement on Smartphones: Report. In *ECCV Workshops*.

[26] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. 2018. AI Benchmark: Running Deep Neural Networks on Android Smartphones. In *The European Conference on Computer Vision (ECCV) Workshops*.

[27] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In *ECCV*.

[28] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proc. VLDB Endow.* 10, 11 (Aug. 2017), 1586–1597.

[29] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. [n. d.]. Shallow-Deep Networks: Understanding and Mitigating Network Overthinking. In *International Conference on Machine Learning (ICML)*.

[30] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. 2016. Accurate Image Super-Resolution Using Very Deep Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1646–1654.

[31] Y. Kim, J. Choi, and M. Kim. 2018. A Real-Time Convolutional Neural Network for Super-Resolution on FPGA with Applications to 4K UHD 60 fps Video Services. *IEEE Transactions on Circuits and Systems for Video Technology* (2018).

[32] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.

[33] A. Kouris, S. I. Venieris, and C. Bouganis. 2018. CascadeCNN: Pushing the Performance Limits of Quantisation in Convolutional Neural Networks. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. 155–1557. https://doi.org/10.1109/FPL.2018.00034

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*.

[35] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. 2016. DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 1–12.

[36] N. D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi, and F. Kawsar. 2017. Squeezing Deep Learning into Mobile and Embedded Devices. *IEEE Pervasive Computing* 16, 3 (2017), 82–88.

[37] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. 2017. Enhanced Deep Residual Networks for Single Image Super-Resolution. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

[38] Chao Ma, Chih-Yuan Yang, Xiaokang Yang, and Ming-Hsuan Yang. 2017. Learning a No-Reference Quality Metric for Single-Image Super-Resolution. *Computer Vision and Image Understanding* 158 (2017), 1–16.

[39] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *The European Conference on Computer Vision (ECCV)*.

[40] D. Martin, C. Fowlkes, D. Tal, and J. Malik. 2001. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *IEEE International Conference on Computer Vision (ICCV)*.

[41] Anish Mittal, Rajiv Soundararajan, and Alan C. Bovik. 2013. Making a "Completely Blind" Image Quality Analyzer. *IEEE Signal Processing Letters* 20 (2013), 209–212.

[42] Seyyed Salar Latifi Oskouei, Hossein Golestani, Matin Hashemi, and Soheil Ghiasi. 2016. CNNdroid: GPU-Accelerated Execution of Trained Deep Convolutional Neural Networks on Android. In *ACM Multimedia*.

[43] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and

Huazhong Yang. 2016. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '16)*. ACM, 26–35.

[44] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. 1992. Nonlinear Total Variation Based Noise Removal Algorithms. *Phys. D* 60, 1-4 (Nov. 1992), 259–268.

[45] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[46] Haichen Shen, Seungyeop Han, Matthai Philipose, and Arvind Krishnamurthy. 2017. Fast Video Classification via Adaptive Cascading of Deep Models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[47] Laurent Sifre and Stéphane Mallat. 2014. Rigid-motion scattering for image classification. *PhD thesis, Ph. D. thesis* 1 (2014), 3.

[48] Mingcong Song, Kan Zhong, Jiaqi Zhang, Yang Hu, Duo Liu, Weigong Zhang, Jing Wang, and Tao E Li. 2018. In-Situ AI: Towards Autonomous and Incremental Deep Learning for IoT Systems. *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2018), 92–103.

[49] S. Teerapittayanon, B. McDanel, and H. T. Kung. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. 2464–2469.

[50] R. Timofte et al. 2017. NTIRE 2017 Challenge on Single Image Super-Resolution: Methods and Results. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*.

[51] Stylianos I. Venieris and Christos-Savvas Bouganis. 2018. fpgaConvNet: Mapping Regular and Irregular Convolutional Neural Networks on FPGAs. *IEEE Transactions on Neural Networks and Learning Systems* 30 (2018), 326–342.

[52] Stylianos I Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. 2018. Deploying Deep Neural Networks in the Embedded Space. In *2nd International Workshop on Embedded and Mobile Deep Learning (EMDL)*.

[53] Thang Vu, Cao Van Nguyen, Trung Xuan Pham, Tung Minh Luu, and Chang Dong Yoo. 2018. Fast and Efficient Image Quality Enhancement via Desubpixel Convolutional Neural Networks. In *The European Conference on Computer Vision (ECCV) Workshops*.

[54] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13 (2004), 600–612.

[55] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated Residual Transformations for Deep Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[56] Jianchao Yang, John Wright, Thomas S. Huang, and Yi Ma. 2010. Image Super-resolution via Sparse Representation. *Trans. Img. Proc.* 19, 11 (2010), 2861–2873.

[57] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[58] Dong-Qing Zhang. 2018. clcNet: Improving the Efficiency of Convolutional Neural Network Using Channel Local Convolutions. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).

[59] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-tolerance. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation (NSDI'17)*. 377–392.

[60] Kai Zhang, Wangmeng Zuo, and Lei Zhang. 2018. Learning a Single Convolutional Super-Resolution Network for Multiple Degradations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[61] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[62] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. 2018. Image Super-Resolution Using Very Deep Residual Channel Attention Networks. In *European Conference on Computer Vision (ECCV)*.

[63] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. 2018. Residual Dense Network for Image Super-Resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.