Perceptual-Centric Image Super-Resolution using Heterogeneous Processors on Mobile Devices

Kai Huang
University of Pittsburgh, USA
k.huang@pitt.edu

Xiangyu Yin
University of Pittsburgh, USA
eric.yin@pitt.edu

Tao Gu
Macquarie University, Australia
tao.gu@mq.edu.au

Wei Gao
University of Pittsburgh, USA
weigao@pitt.edu

ABSTRACT

Image super-resolution (SR) is widely used on mobile devices to enhance user experience. However, neural networks used for SR are computationally expensive, posing challenges for mobile devices with limited computing power. A viable solution is to use heterogeneous processors on mobile devices, especially the specialized hardware AI accelerators, for SR computations, but the reduced arithmetic precision on AI accelerators can lead to degraded perceptual quality in upscaled images. To address this limitation, in this paper we present *SR For Your Eyes (FYE-SR)*, a novel image SR technique that enhances the perceptual quality of upscaled images when using heterogeneous processors for SR computations. FYE-SR strategically splits the SR model and dispatches different layers to heterogeneous processors, to meet the time constraint of SR computations while minimizing the impact of AI accelerators on image quality. Experiment results show that FYE-SR outperforms the best baselines, improving perceptual image quality by up to 2×, or reducing SR computing latency by up to 5.6× with on-par image quality.

CCS CONCEPTS

• **Human-centered computing → Ubiquitous and mobile computing**; • **Computing methodologies → Artificial intelligence**;

KEYWORDS

Image super-resolution, perceptual quality, neural networks, heterogeneous computing, mobile devices

ACM Reference Format:
Kai Huang, Xiangyu Yin, Tao Gu, and Wei Gao. 2024. Perceptual-Centric Image Super-Resolution using Heterogeneous Processors on Mobile Devices. In *The 30th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '24), November 18–22, 2024, Washington D.C., DC, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3636534.3690698

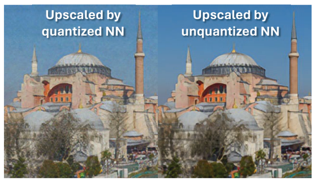
1 INTRODUCTION

Image super-resolution (SR) enhances the image quality by upscaling the image resolution [3, 52], and finds applications in various domains such as gaming [31], video streaming [83] and healthcare [45], when the original image quality from data sources is low. SR has been widely used on mobile devices (e.g., smartphones and tablets) to enhance user experience, due to the increasing complexity of multimedia contents and the need of real-time rendering in gaming and camera preview on these devices [23, 42, 66, 78].

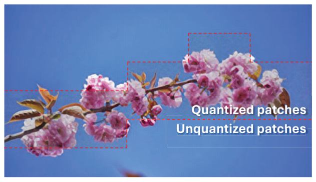
Current SR techniques are mainly based on Neural networks (NNs) that can better capture such non-linearity and hence improve the image quality [12, 33, 40, 73, 82]. However, NN-based SR models[1] are computationally expensive for mobile devices with limited computing power. For example, even when using mobile GPU on flagship smartphones (e.g., a Google Pixel 6 smartphone with a Mali-G78 MP20 GPU that provides 1.94 TFLOPS computing power), it still takes >180ms to upscale a 240p image to 480p, and such high computing latency makes it difficult to apply SR in many interactive applications such as gaming. An intuitive solution is to offload SR computations to the cloud [71, 79], but incurs high communication overhead, especially when the upscaled images have high resolutions [9, 34].

A better alternative is to involve specialized hardware AI accelerators that have been readily available in mobile SoCs, such as Neural Processing Units (NPUs)[2], in addition to traditional processors (e.g., CPU and GPU). These AI accelerators use fixed-point arithmetic and systolic array architecture

[1]Without loss of generality, the terms of "NN model" and "SR model" will be used interchangeably in the rest of this paper.
[2]Typical NPUs include Google Tensor [58] and Qualcomm Hexagon [56].



(a) Image quality drop



(b) Visual inconsistency

Figure 1: Reduced arithmetic precision affects the quality and visual consistency of upscaled SR images

[10, 35] to achieve higher speed and energy efficiency in NN computations [35, 49, 65]. However, the use of fixed-point arithmetic could result in low quality in upscaled images [23, 24] when being applied to regression-based SR tasks as shown in Figure 1(a), due to the required quantization and reduced arithmetic precision in NNs [38, 55]. While certain NPU designs (e.g., Qualcomm Snapdragon 8 Gen 3) offer limited support for floating-point computations (e.g., FP16) [27] and such capability of mixed-precision computations have been utilized for SR computations [69], the availability of these NPUs is still limited on commodity mobile devices.

To mitigate such image quality drop, existing schemes split input images into small patches and dispatch these patches to traditional processors and AI accelerators (Figure 2 bottom-left), based on the varying difficulty levels of upscaling each patch [37, 69]. However, when upscaled patches are re-stitched to form a complete image, such image-based split of SR computations often leads to color mismatch and visual inconsistency across image patches, as shown in Figure 1(b). This inconsistency may not impact the structural image quality with a small portion of mismatching patches (e.g., <15% according to [37]), but can largely affect the human perception of images. Providing extra information about images alleviates the difficulty of SR and mitigates such inconsistency [42, 78], but adds significant communication and storage overhead among heterogeneous processors.

To address this limitation, in this paper we present *SR For Your Eyes (FYE-SR)*, a novel image SR technique that aims to enhance the perceptual quality of upscaled images on mobile devices. FYE-SR addresses the visual inconsistency in upscaled images by introducing a new procedure-based approach to splitting SR computations among heterogeneous processors, as opposed to the traditional image-based splitting. We split the SR model (Figure 2 bottom-right) and adaptively dispatch different NN layers of the SR model to heterogeneous processors, according to the computing complexity of these NN layers and how SR computations in these layers are affected by the reduced arithmetic precision. Our goal is to maximize the utilization of AI accelerators within the given time constraints on SR computations, while minimizing their impact on perceptual image quality.

The major challenge is how to ensure proper split of the SR model. The reduced arithmetic precision impacts different

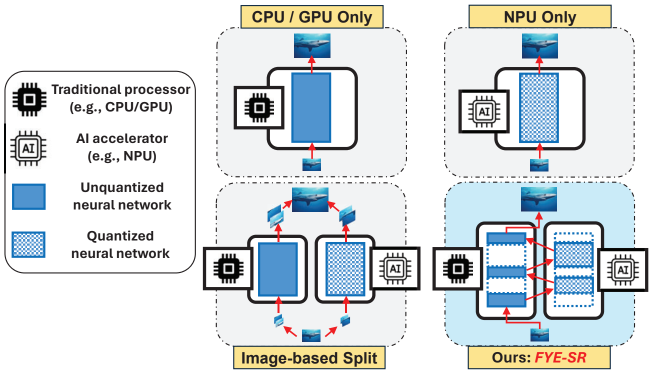

Traditional processor (e.g., CPU/GPU)
AI accelerator (e.g., NPU)
Unquantized neural network
Quantized neural network
CPU / GPU Only
NPU Only
Image-based Split
Ours: FYE-SR

Figure 2: Existing work vs. FYE-SR

NN layers in various ways, but such impact should be cumulatively evaluated for the optimal SR model split. Traditional approaches separately evaluate such impacts of individual layers and aggregate them additively [14, 51, 64], but ignore the interdependency among layers. Instead, our approach is to evaluate such cumulative impact through an end-to-end learning process, by designating the SR model split as a set of trainable parameters and constructing differentiable training losses with respect to perceptual image quality. These parameters are then transformed into vectorized formulations to ensure the stability and computational efficiency of training.

On the other hand, precise measurement of the computing times for different NN layers across heterogeneous processors is essential for a proper split of the SR model. In particular, since such split results in switches between processors during SR computations, the switching costs should also be properly measured, minimized, and considered when deciding the SR model split. To ensure precise profiling of computing times, we developed new software techniques that efficiently utilize the interrupt information from Android OS kernel to extract timestamps of relevant hardware access events. To minimize the runtime computing costs of switching, we made custom modifications to the OpenCL library in Android OS and utilize mobile GPU for parallel conversion of intermediate data formats, a significant aspect of the switching process with substantial overhead.

To our best knowledge, our work is the first that aims to explicitly enhance the perceptual quality of upscaled images on mobile devices, and our key contributions are as follows:

- Our end-to-end learning approach ensures proper split of SR model between heterogeneous processors, by applying computing times and perceptual image quality of SR into the loss with differentiable forms.
- We propose system techniques to ensure accurate profiling of SR computing times on different processors, by retrieving timestamps of hardware access events from Android OS kernel.
- Our custom OpenCL modification supports computationally efficient conversion of data formats and hence

minimizes the runtime switching costs between heterogeneous processors.

We implemented FYE-SR on the Google Pixel smartphones, and utilize their on-board GPU and Tensor NPU for SR computations by converting intermediate data in SR models between INT8 and FP32. We evaluated the performance of FYE-SR on multiple commonly used image datasets with different SR configurations. From our experiment results, we have the following conclusions:

- FYE-SR is *highly effective*. Compared to the best baselines, FYE-SR reaches the similar SR computing latency but improves the upscaled image quality by up to 50%, or retains on-par image quality but reduces the SR computing latency by up to 5.6×.
- FYE-SR is *adaptive*. It can retain high perceptual quality of upscaled images with different SR configurations and datasets. When the timing constraint of SR computation varies, FYE-SR can adaptively adjust the SR model split to achieve the optimal tradeoff between image quality and SR computing latency.
- FYE-SR is *lightweight*. It consumes <20% of battery power after 1-hr use on various smartphone models, and hence ensures continuous SR computations in practical applications.

2 BACKGROUND & MOTIVATION

We first provide backgrounds about image SR and different metrics to measure image quality. We further highlight the possibility of speeding up NN-based SR computations using AI accelerators, and the necessity of splitting SR computations between traditional processors and AI accelerators.

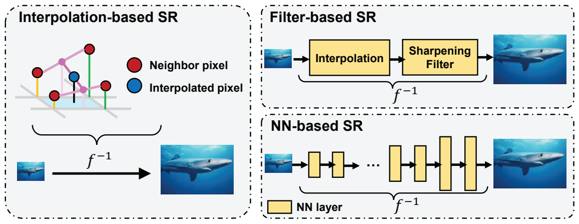


Figure 3: Different image SR approaches that approximate the reverse form (f^{-1}) of degradation function

2.1 Overview of Image Super-Resolution

Image super-resolution (SR) recovers high-resolution (HR) images from low-resolution (LR) images, by approximating the reverse form f^{-1} of the degradation function $f : HR \rightarrow LR$. In practice, since a LR image may correspond to multiple possibilities of HR images, f is irreversible unless providing assumptions about f^{-1} [3]. As shown in Figure 3, basic interpolation-based SR assumes full linearity of f^{-1} and thus creates new pixels via simple arithmetic (e.g., weighted average) over existing nearby pixels [13, 67], but the restored HR images usually lack high-frequency graphics information (e.g., edge of objects) due to over-simplified f^{-1}. Filter-based SR [17, 18, 52, 74] incorporates a moderate level of non-linearity into f^{-1} with sharpening filters (e.g., Lanzcos kernel [15]), but the HR image quality is still limited.

In recent years, convolutional NNs [12, 40] and transformers [8, 43] have been adopted for SR to learn a generic non-linear approximation of f^{-1} via a regression task. Advanced deep learning techniques, such as adversarial training [33], can be further used to enhance the training feedback and improve the training quality. In the rest of this paper, we will focus on NN-based SR for the maximum image quality.

SR Model	CPU	GPU	**Tensor NPU**
SRCNN [11]	0.075s	0.022s	**0.006s**
EDSR [40]	1.91s	0.21s	**0.034s**
ESRGAN [73]	3.24s	0.46s	**0.074s**

Table 1: SR computing latency using different processors on a Google Pixel 6 smartphone. 4× SR is applied to an input image with size of 640×480.

However, most NN-based SR models are computationally expensive. Being different from NNs in other tasks (e.g., image classification), the intermediate data size in SR models increases as the layer goes deeper, and SR models hence incur much more computations with the same parameter size. For example, both SwinIR [39] (SR model) and ResNet18 [19] (classification model) contain 11M parameters, but the former incurs 202 GFLOPs in inference and the latter only incurs 2 GFLOPs. Recent popular SR models, such as EDSR [40] and ESRGAN [73], have even larger parameter sizes and their inferences can only reach 2.2-4.8 FPS on flagship smartphones as shown in Table 1. Such high computing costs motivate us to explore more opportunities of speedup using heterogeneous on-device processors.

2.2 Image Quality Metrics

The quality of upscaled images produced by SR can be evaluated with different metrics. The commonly used structural metrics, such as PSNR and SSIM, measure the structural similarity of target images with their corresponding reference images. However, they cannot correctly depict humans' perception on images, because they tend to underestimate humans' sensitivity to visual artifacts, especially distortions at hard edges and in high-intensity regions [21, 50]. Subjective metrics, such as Mean Opinion Score [3], use human ratings to ensure correct alignments with human perception but are time-consuming and expensive.

Instead, objective Image Quality Assessment (IQA) metrics, including LPIPS [81], NIQE [47] and PIQE [70], are more commonly used to measure images' perceptual quality, as the target images' statistical deviation from natural images that

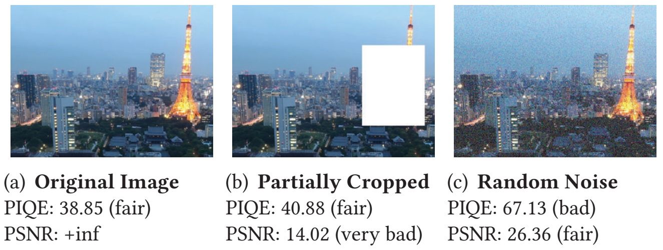

(a) **Original Image** PIQE: 38.85 (fair) PSNR: +inf
(b) **Partially Cropped** PIQE: 40.88 (fair) PSNR: 14.02 (very bad)
(c) **Random Noise** PIQE: 67.13 (bad) PSNR: 26.36 (fair)

Figure 4: Image quality evaluated with PSNR (structural) and PIQE (perceptual) metrics

are known to be perceptually photo-realistic. These perceptual metrics, hence, reflect different aspects of image quality and are usually inconsistent with traditional structural quality metrics [6]. For example, Figure 4 shows that the same image distortion could result in large difference in PIQE and PSNR quality scores. Most existing work in SR, however, are limited to measuring the quality of upscaled images using structural metrics and ignore the images' perceptual quality. In our work, we explicitly use perceptual quality metrics as the feedback when training SR models, hence ensuring the maximum perceptual quality of upscaled images.

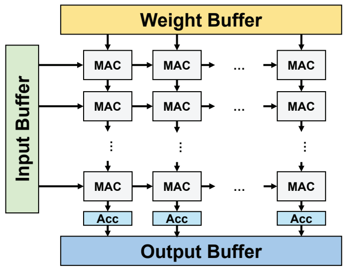


Figure 5: The micro-architecture of a typical systolic array (MAC: multiply-and-add, Acc: accumulator)

2.3 Speedup with AI Accelerators

As shown in Figure 5, current designs of AI accelerators use systolic arrays to allow 2D multiplication and summation to be performed concurrently, hence accelerating NN operations based on matrix multiplications [10]. Table 1 shows that such concurrency can reduce SR computing latency by up to 6.3× from GPU execution and 55.8× from CPU execution.

On the other hand, such speedup requires a sufficient amount of multiply-and-add (MAC) units in systolic arrays. Most AI accelerators, hence, only support integer computations with fixed-point arithmetic (e.g., INT8) [35] to save chip space, because floating-point MAC units (e.g., FP32) require 25× more space [27] and 3× more transistors. The reduced arithmetic precision could result in significant drop of upscaled images' quality, even if measured with perceptual metrics, as shown in Figure 6. Some recent designs of AI accelerators (e.g., Qualcomm Snapdragon 8 Gen 3) provide limited support on floating-point computations (e.g., FP16) [27], but their floating-point performance is much lower. For example, as shown in Table 2, the FP16 performance of Qualcomm Hexagon NPU on Samsung Galaxy S24 Ultra smartphone is only 11.6% of its INT8 performance and is even lower than the smartphone's GPU performance.

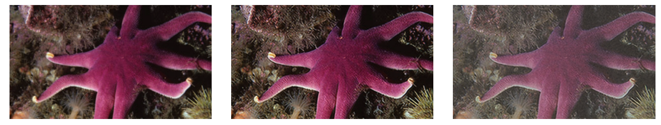

(a) **Original Image** PIQE: 18.03 (excellent)
(b) **After SR** PIQE: 36.73 (fair)
(c) **Quantized SR** PIQE: 53.35 (poor)

Figure 6: Impact of reduced arithmetic precision on perceptual image quality using SRCNN [11]

Device Model	GPU	NPU (FP16)	NPU (INT8)
Google Pixel 6	0.022s	-	0.006s
Galaxy S24 Ultra	0.019s	0.043s	0.005s

Table 2: The SR computing latencies using different arithmetic precisions. 4× SR with SRCNN model [11] is applied to an input image with size of 640×480.

Such limitation of AI accelerators motivates us to split SR computations between traditional processors and AI accelerators, and enhance the perceptual quality of upscaled images by avoiding SR computations that are sensitive to arithmetic precision from being executed on AI accelerators after being quantized. In particular, as shown in Figure 7, different layers in a SR model exhibit highly diverse sensitivity to arithmetic precision. These results verify that we can still execute the majority of SR computations on AI accelerators, to speed up SR computations with the minimum perceptual quality loss.

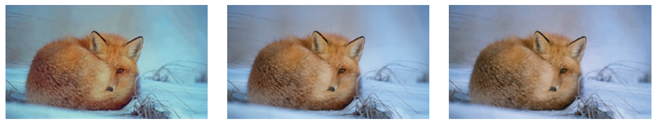

(a) First 3 blocks quantized. PIQE: 40.02
(b) Middle 3 blocks quantized. PIQE: 70.57
(c) Last 3 blocks quantized. PIQE: 13.77

Figure 7: Image quality difference with different layers of the SR model being quantized. 4× SR is applied using EDSR model with 32 residual blocks [40].

In practice, a proper split of SR computations builds on quantitative understanding about SR model's sensitivity to arithmetic precision. Existing work uses layer importance metrics based on accuracy loss [64], weight magnitude [51] or gradient [14] to quantize such sensitivity, but ignores the interdependency among NN layers and cannot correctly reflect the cumulative sensitivity of multiple NN layers. Instead, in FYE-SR we adopt a learning-based approach to implicitly incorporate such interdependency.

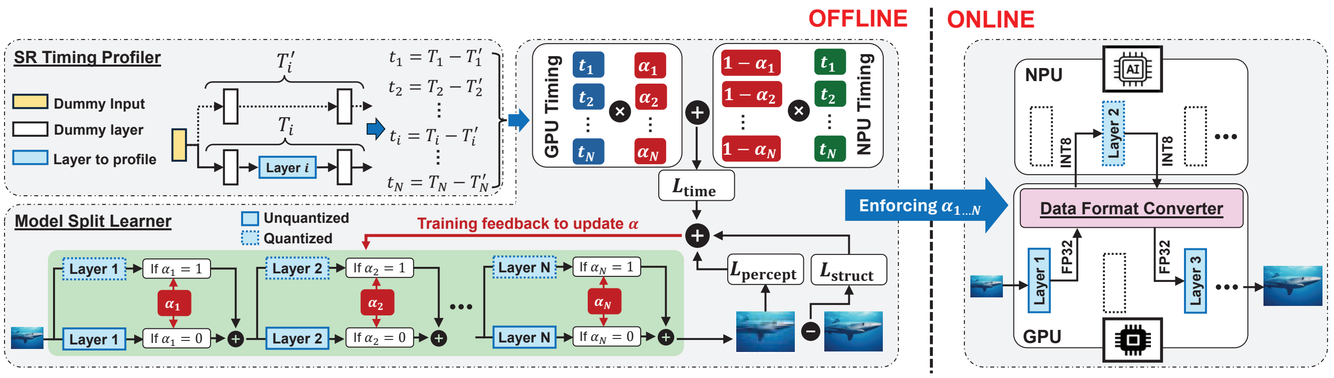


Figure 8: Overview of FYE-SR Design

3 SYSTEM OVERVIEW

Our goal of FYE-SR design is to simultaneously minimize the end-to-end SR computing latency (T) and maximize the perceptual quality of upscaled images (Q), by solving the following multi-objective optimization problem:

$$\min_{\mathcal{P}} \, (T(\mathcal{P}), -Q(\mathcal{P})), \tag{1}$$

where $\mathcal{P}$ indicates the SR computation split between traditional processors and AI accelerators. In most practical scenarios when these two optimization objectives cannot be simultaneously met, we instead look for a Pareto optimal solution, in which either objective cannot be further improved without degrading the other. More specifically, we adopt the ϵ-constraint method [46] where $T(\mathcal{P})$ is considered as minimized if it's lower than a required T_{obj}, and the optimization problem in Eq. (1) is then simplified as

$$\min_{\mathcal{P}} \, (-Q(\mathcal{P})) \quad \text{s.t. } T(\mathcal{P}) \le T_{obj}. \tag{2}$$

Based on this formulation, our design of FYE-SR consists of three main modules as shown in Figure 8. During the offline phase, we first use a *SR Timing Profiler* to measure the computing latencies of SR model's different NN layers on traditional processors (e.g., GPU) and AI accelerators (e.g., NPU), respectively. Then, knowledge about such latencies will be used to train a *Model Split Learner* to solve Eq. (2) for the optimal split of SR model.

During the online phase, FYE-SR enforces such model split, and uses a *Data Format Converter* to convert the intermediate feature maps into the right data formats (e.g., INT8 and FP32) for properly switching SR computations between heterogeneous processors. Note that, FYE-SR splits the SR model by layers, because more fine-grained partitions (e.g., tensor-level split) cannot benefit from AI accelerator's micro-architecture that is specifically designed for fused NN operations (e.g., a complete layer).

3.1 SR Timing Profiler

To measure the computing latency of a NN layer in the SR model, an intuitive approach is to run another small NN model that contains only the target layer. However, the measured latency in this way also includes the overhead of input and output data handling. Instead, as shown in Figure 8, given a target layer i of the SR model, we first measure the end-to-end computing latency of layer i and dummy input/output layers[3], and then measure the computing latency of these dummy layers only. The computing latency of layer i can then be derived as the difference of these two measurements. Results from our preliminary results in Table 3 show that our method can effectively remove the timings of input/output data handling from measurements on the target NN layer.

Method	Layer 1	Layer 2	Layer 3
Small NN model with target layer only	20.5	51.0	58.6
Dummy layer differences	5.2	7.0	9.8

Table 3: Timing measurement (ms) for different layers in the SRCNN model on Google Pixel 6 GPU

When different layers of SR models are executed on heterogeneous processors, FYE-SR needs to switch between these processors during the SR procedure. The timings of such switching between processors will also be profiled and taken into account when making decisions on SR model split. More specifically, we extract and utilize the interrupt information from Android OS kernel and use the timestamps of hardware access events to ensure precise calculation of these timings. More details about such timestamp access are in Section 4.

3.2 Model Split Learner

Due to the nonlinearity of Eq. (2), FYE-SR decides the optimal split of SR model through an iterative learning process. In the offline phase, the *Model Split Learner* creates two replicas of the SR model, i.e., quantized and unquantized, and flexibly alters the data flow to enter layers in each replica to emulate different split options. Such alternation between

[3]In practice, these dummy layers should be at least a few convolutional layers, to avoid being automatically dropped from the model's computing graph in deep learning frameworks (e.g., TFLite).

split options is grounded by a vector of binary variables $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_M]$, where $\alpha_i = 1$ indicates that layer i is executed on AI accelerators (e.g., NPU) and $\alpha_i = 0$ indicates otherwise. FYE-SR learns these binary variables by using 1) the profiled computation timings of different NN layers and 2) a SR image dataset that contains pairs of LR and HR images (e.g., DIV2K [1] and UHDSR [80]) as training data. Such learning is jointly guided by the following loss terms that are aligned with the objective and constraint in Eq. (2):

First, the **Timing loss** ($L_{time} = \max(0, T(\boldsymbol{\alpha}) - T_{obj})$), where $T(\boldsymbol{\alpha})$ indicates the end-to-end SR computing latency when the SR model is split with $\boldsymbol{\alpha}$, enforces that the SR model split should satisfy the time constraint (T_{obj}) of SR computations. The use of $\max(\cdot)$ ensures that $L_{time} \geq 0$ even if the latency constraint T_{obj} cannot be met due to mobile devices' limited computing power. Details about calculating such computing latency $T(\boldsymbol{\alpha})$ is in Section 5.1.

Second, the **Image quality loss** $\lambda_1 L_{struct} + \lambda_2 L_{percept}$ involves both perceptual metrics and structural metrics to maximize the perceptual quality of upscaled images. On one hand, since most perceptual image quality metrics measure the images' statistical deviation from human perception and are hence not differentiable, we instead train a differentiable NN-based perceptual quality estimator to mimic the behavior of these metrics in the learning process, and details of such training are in Section 5.2. On the other hand, we also involve structural metrics[4] to ensure that upscaled images approximate the ground truth. λ_1, λ_2 are hyper-parameters to balance each term's contribution to the learning process and will be further discussed in Section 5.3.

To apply gradient-based optimizers (e.g., SGD [2] and Adam [29]) in training, we adopt a continuous representation of $\boldsymbol{\alpha}$, by applying *sigmoid*$(\cdot)$ over a trainable weight w_i. As shown in Figure 9, there are two choices to ground the SR model split. Pre-selection places the weighting module before entering a NN layer. When $\alpha_i \to 1$, the input to the unquantized layer will be zeroed out, but this cannot guarantee that the layer's output will be zeroed due to internal bias parameters. Instead, we adopt post-selection to ensure that unselected layers do not affect the forward pass.

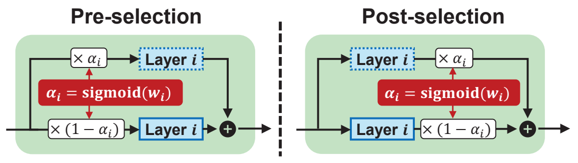


Figure 9: Design choices for *Model Split Learner*

Effectiveness of Model Split Learning. Model split can be decided in various ways. Compared to naive methods such as exhaustive enumeration or heuristic search [5], our proposed NN-based Model Split Learner can better depict the nonlinear correlation between different NN layers' computations on GPU/NPU and SR image quality, and also achieves higher compute efficiency with a smaller problem space.

To further verify such compute efficiency, we evaluated the amount of offline time consumed by deciding the optimal model split, using a Nvidia A5000 GPU with different NN models and datasets. Results in Table 4 show that our NN-based Model Split Learner spent at least 25% less compute time on small SR models such as EDSR. In particular, since such computing overhead of our Model Split Learner linearly increases with the number of layers in the SR model, on larger SR models such as ESRGAN, our compute efficiency is at least 2.2x higher than heurstic search methods.

Method	Dataset	NN Model	NN Layer count	Time (hour)
Exhaustive search	DIV2K	EDSR	37	3.5
Heuristic search	DIV2K	EDSR	37	1.7
Model Split Learner	DIV2K	EDSR	37	1.4
Exhaustive search	DIV2K	ESRGAN	171	78.6
Heuristics search	DIV2K	ESRGAN	171	14.4
Model Split Learner	DIV2K	ESRGAN	171	6.5

Table 4: Amount of offline computation time in finding the optimal SR model split

3.3 Data Format Converter

Since AI accelerators usually support only fixed-point arithmetic, data format conversion is needed whenever SR computations are switched between heterogeneous processors. Quantization and dequantization, as the most common form of such conversion, are computed as

$$\mathbf{q} = \mathbf{r}/s + z, \quad \mathbf{r} = (\mathbf{q} - z) \cdot s, \tag{3}$$

where $\mathbf{q}$ and $\mathbf{r}$ indicate quantized and dequantized data, respectively, and s and z are constants.

The major challenge of data format conversion in FYE-SR is that such conversion incurs non-negligible computing overhead. Table 5 shows that, when using smartphone CPU for such conversion on a single image, such computing latency could be up to 46ms. Even if we convert dequantization computations into table lookup, the computing latency is still about 27ms, which is even higher than the SR computing latency on NPU, as shown in Table 1 and Table 2.

Method	640×360×64 quant./dequant.	640×360×32 quant./dequant.
Arithmetic	46.4/46.1	27.7/28.9
Table lookup	-/20.0	-/10.5

Table 5: Computing latency (ms) of CPU-based data format conversion on a Google Pixel 6 smartphone

[4]Most structural metrics (e.g., SSIM and PSNR) are differentiable and can be directly used in the learning process.

To reduce the overhead, our approach is to conduct such conversion on mobile GPU with high parallelism. Unfortunately, such conversion is not supported by current deep learning frameworks on mobile devices (e.g., TFLite). Instead, we modify the commodity TFLite library that handles mobile NN execution, and implement our custom GPU-based parallel format conversion on top of the TFLite library. More details are in Section 6.

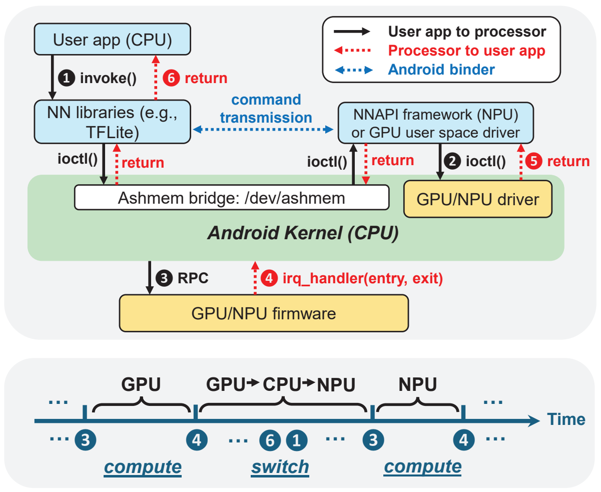


Figure 10: Hardware access events in Android OS when executing NNs on heterogeneous processors

4 RETRIEVING TIMESTAMPS FROM MOBILE OS

As shown in Figure 10, we measure the time needed to switch SR computations between different processors, by retrieving timestamps of the hardware access events in the Android OS kernel, including the Android kernel’s Remote Procedure Calls (RPCs) and hardware interrupt (`irq_handler`) sent by the processor (e.g., GPU or NPU). More specifically, the time switching between processors contains i) the overhead of function returns from NN computation on the current processor, corresponding to events from ❶ to ❸ ; and ii) the overhead of system calls to continue NN computation on the other processor, corresponding to events from ❹ to ❻ . The switching time between processors can then be calculated as the sum of elapsed time from ❹ to ❻ and the time from ❶ to ❸ .

To capture ❹ , we use the Linux tracing subsystem through the `/sys/kernel/tracing` pseudo-file interface. It can record various system events in the kernel, including interrupts (`irq`) and system calls such as `ioctl()`. The two system events, `irq_handler_entry` and `irq_handler_exit`, indicate start and end of IRQ interrupt handling, respectively, and we consider `irq_handler_entry` to be the exact timestamp ❹ , which indicates end of NN computation on the current processor and start of switching between processors.

To capture ❸ , one intuitive approach is to modify the processor drivers so that they can send a message to the OS kernel’s ring buffer immediately before SR computation is launched on the processor. This message, as well as its timestamp (❸), could then be retrieved through `dmesg` program that reads the Linux kernel ring buffer. However, this approach cannot be applied to most current Android systems where GPU and NPU drivers are close-sourced. Instead, in FYE-SR we monitor the closest event `ACQUIRE_WAKE_LOCK` `ioctl()` (❷) as an approximation to ❸ .

5 CONSTRUCTING TRAINING LOSS FOR MODEL SPLIT LEARNER

In this section, we describe technical details of constructing the loss when training the *Model Split Learner.*

5.1 Calculating the Timing Loss

To properly calculate $T(\boldsymbol{\alpha})$ that indicates the end-to-end computing latency of SR model on heterogeneous processors, we separately calculate the computing time on processors and switching time between processors. First, letting t_i^G and t_i^N denote the computing time of SR model’s layer i on GPU and NPU, the total computing time on heterogeneous processors can be calculated as

$$T_{comp} = \sum\nolimits_{i=1}^{M} \left(\alpha_i t_i^N + (1-\alpha_i) t_i^G \right), \tag{4}$$

where M is the number of layers in SR model, and α_i is the binary variable indicating whether layer i is computed at NPU or GPU. α_i and $1-\alpha_i$ are hence contradictory to ensure each layer is only computed on one processor.

Switching between processors only happens on layer i when $\alpha_i \neq \alpha_{i+1}$ and layer $i+1$ is hence computed on a different processor. Letting $t_i^{G\to N}$ and $t_i^{N\to G}$ denote the time needed to switch computations from GPU to NPU and from NPU to GPU, respectively, the total switching time between processors during SR computations can be calculated as

$$T_{switch} = \sum\nolimits_{i=1}^{M-1} \left((1-\alpha_i)\alpha_{i+1} t_i^{G\to N} + \alpha_i (1-\alpha_{i+1}) t_i^{N\to G} \right), \tag{5}$$

where for any layer i, only one of the two additive terms will be non-zero.

Although T_{comp} and T_{switch} are both differentiable, it is inefficient to compute gradient-based feedback from their non-vectorized formulations in current NN libraries. Hence, we convert Eq. (4) and Eq. (5) to a vectorized formulation on a layer basis. For example, the vectorized $T(\boldsymbol{\alpha})$ with respect to layer 1 can be calculated as

$$\begin{aligned} T(\boldsymbol{\alpha}) = {} & \boldsymbol{\alpha} \odot \mathbf{t}^N + (\mathbf{1} - \boldsymbol{\alpha}) \odot \mathbf{t}^G \\ & + [(\mathbf{1} - \boldsymbol{\alpha}_{1:M-1})\boldsymbol{\alpha}_{2:M}] \odot \mathbf{t}^{G\to N} \\ & + [\boldsymbol{\alpha}_{1:M-1}(\mathbf{1} - \boldsymbol{\alpha}_{2:M})] \odot \mathbf{t}^{N\to G} \end{aligned} \tag{6}$$

where $\odot$ denotes the inner product and $\boldsymbol{\alpha}_{a:b} = [\alpha_a, \alpha_{a+1}, ..., \alpha_b]$. Since values in the continuous form of $\boldsymbol{\alpha}$ may not be close to 0 or 1 in training, we only use such continuous $\boldsymbol{\alpha}$ in the backward pass, but still use discrete $\boldsymbol{\alpha}$ in the forward pass based on the straight-through estimator method [4].

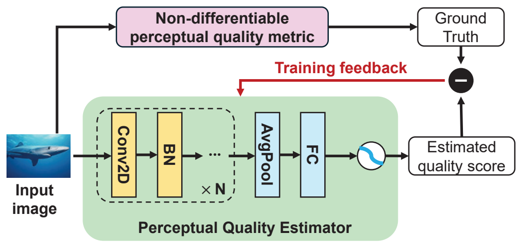


Figure 11: Training a differentiable PIQE estimator

5.2 Differentiable Perceptual Loss

As described in Section 3.2, the perceptual loss ($L_{percept}$) should also be differentiable to guide the training of *Model Split Learner*. Our approach to a differentiable perceptual loss is to construct a NN-based perceptual quality estimator that mimics the behavior of a perceptual image quality metric. As shown in Figure 11, the estimator consists of a few blocks of convolution and batch normalization (BN) layers to extract global and local features from the input image, and use average pooling (AvgPool) + fully connected (FC) layers to project the extracted features into a scalar. When training this estimator, we use the image quality score given by the original non-differentiable metric as the ground truth.

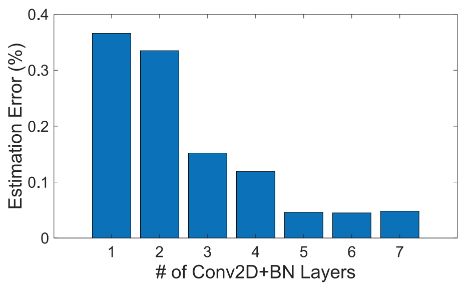


Figure 12: Error of estimating perceptual image quality with different numbers of Conv2D+BN blocks

To ensure that the knowledge that the NN-based estimator learned about the perceptual quality metric is complete, we use the LIVE image database [61] as the training dataset, where images are distorted with 5 distortion types (e.g., JPEG compression and Gaussian blur) and ensured to cover all the possible levels of perceptual quality. Our experiment results in Figure 12 show that, by using 5 Conv2D+BN blocks in the estimator, the estimator can restrain the quality estimation error within 4.7% with low computing cost, and we will use this design in the rest of the paper.

5.3 Combined Training Loss

As described in Section 3.2, the timing loss and image quality loss are combined in training by hyper-parameters λ_1 and λ_2, which affect the relative intensity of training feedback. When adopting SGD [2] as the optimizer, the update of $\boldsymbol{\alpha}$'s trainable weights $\boldsymbol{w}$ in each iteration is

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \cdot \left(\frac{\partial L_{time}}{\partial \boldsymbol{w}} + \lambda_1 \frac{\partial L_{percept}}{\partial \boldsymbol{w}} + \lambda_2 \frac{\partial L_{struct}}{\partial \boldsymbol{w}}\right), \quad (7)$$

where η is the learning rate.

In practice, we empirically determine the values of λ_1 and λ_2 based on the scales L_{time}, $L_{percept}$, and L_{struct}, so as to avoid training bias. For example, when PSNR and PIQE are used as the structural and perceptual image quality metrics, respectively, our experiment results show that the best values of λ_1 and λ_2 are 0.001 and 0.25 for the SRCNN model [11] running a Google Pixel 6 smartphone.

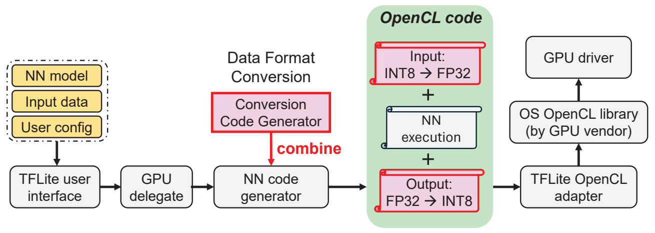


Figure 13: Generating OpenCL codes for data format conversion, by modifying the existing OpenCL code generator in TFLite backend library

6 DATA FORMAT CONVERSION

To minimize the runtime computing cost of data format conversion between fixed-point (e.g., INT8) and floating-point (e.g., FP32) precisions, we perform such conversion on mobile GPU with high parallelism. To do this in Android, an intuitive solution is to add additional NN layers in the SR model when defining the model structure in user space. However, current deep learning libraries on Android, such as TFLite, do not support efficient operators for such conversion like `tf.cast()` provided in TensorFlow desktop APIs.

Instead, we directly append and prepend conversion codes to the OpenCL program that is generated by TFLite library for GPU execution. As shown in Figure 13, TFLite converts the computing graph of the NN model to OpenCL GPU programs for parallel GPU execution. The original NN code generator works by stitching OpenCL code strings of each NN operator, and we combine it with our custom Conversion Code Generator by modifying the original code to handle INT8 data input and output. Figure 14 shows a code piece of the implemented Conversion Code Generator. It generates code strings that call `convert_char()` to cast input data from FP32 to INT8 for quantization. Dequantization from INT8 to FP32 can be supported in the similar way.

```
std::string c;
c += "__kernel void vector_f32toint8_gpu( __global const float* src_f,";
c += "        __global char* res, const int num ){";
c += "    const int idx = get_global_id(0);";
c += "    if (idx < num) {";
c += "        res[idx] = convert_char(src_f[idx]);";
c += "    }";
c += "}";
```

Figure 14: Code piece of generating data format conversion code

We conducted preliminary experiments to explore the computing efficiency of GPU-based data format conversion, by comparing it with multithreading CPU execution. As shown in Table 6, GPU-based conversion is up to 5× faster than CPU conversion, 14× faster than naive Java implementation, and its overhead if 25% of data transmission overhead between GPU and NPU. Note that, such data conversion latency is also incorporated when measuring the switching times between processors in SR computations.

Time (ms)	layer 1 to 2 GPU→NPU	layer 1 to 2 NPU→GPU	layer 2 to 3 GPU→NPU	layer 2 to 3 NPU→GPU
Data Tx	21.32	19.83	11.22	10.91
Naive Java	46.41	46.14	27.73	28.88
CPU conversion	-	20.01	-	10.50
GPU conversion	**4.31**	**5.28**	**1.64**	**1.98**

Table 6: Latency (ms) of data format conversion on a Google Pixel 6 smartphone with SRCNN [11] model

7 IMPLEMENTATION

Offline Phase: We implement our SR Timing Profiler using a rooted smartphone with Android OS and a workstation. A bot program written in Python runs on the workstation to control the profiling on smartphone. The bot program first generates SR model files containing the target NN layer and dummy input/output layers with TensorFlow. It then sends the generated model files in `.tflite` format to smartphone, and controls the test program on smartphone to perform profiling via ADB debugging connection. The test program written in C++ runs in the Termux emulator on rooted Android OS. It executes the `.tflite` models using TFLite C++ library, measures execution time using timestamps from Linux kernel ring buffer and Linux tracking subsystem, and sends the profiling results back to the bot program on workstation.

With the time profiling results, we implement the learning program for Model Split Learner on the workstation. The learning program utilizes TensorFlow's Model Optimization Toolkit (`tfmot`) to handle training with quantized model components. Split model files with quantized and unquantized components, together with a metadata file, are generated for the online phase based on the training results of model split.

Online Phase: SR computation is implemented as an Android App, which incorporates our modified TFLite library to execute pre-defined split SR model files. The Data Format Converter written in C++ and OpenCL is integrated into modified TFLite library.

8 PERFORMANCE EVALUATION

Our evaluations use three NN-based SR models: EDSR [40], ESRGAN [73], and ENet-PAT [59] with different designs and model complexities. EDSR and ESRGAN models are re-implemented with the latest TensorFlow 2 API [22, 30] and pre-trained on the DIV2K dataset [1]. ENet-PAT's weights are loaded from the originally released model trained on MSCOCO dataset [41].

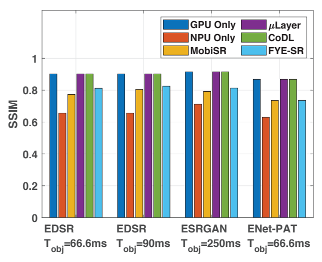
To evaluate SR performance, the following datasets are used in training the Model Split Learner:

- **DIV2K [1]:** It contains 800 training and 100 validation images at 2K resolution, and covers a wide range of outdoor and indoor photos. Since the test data in DIV2K is not publicly available, we randomly take 100 images from the original 800 training images for testing, and use the rest 700 images for training.
- **OST300 [72]:** It contains 10,324 training images and 300 testing images with various resolutions of outdoor scenes in 7 categories (e.g., buildings, sky, and mountains). We randomly select 1,000 images from testing images for validation, and use the original testing dataset for evaluation.
- **Flickr2K [40]:** It contains 2,650 images of 2K resolution in urban and natural landscapes. We randomly select 2,000 images for training, 300 images for validation and the rest 350 images for testing.

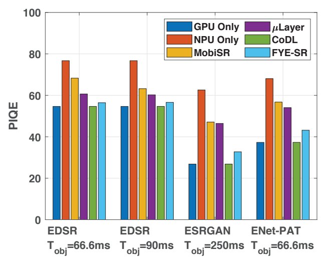
We first apply common pre-processing methods to obtain augmented HR images from the original datasets, and then use bicubic down-sampling over HR images to obtain paired LR images. In evaluations, these LR images are used as input to the SR model, whose outputs are compared with the original HR images. The output image quality is measured by both structural metrics (PSNR and SSIM) and perceptual metrics (PIQE [70] and LPIPS [81]). We measure the end-to-end latency of SR computations, including the computing times on all processors and the switching times between processors including both OS operations (as shown in Figure 10) and data format conversion (as described in Section 6).

We compare FYE-SR with the following baselines that use heterogeneous mobile processors for SR:

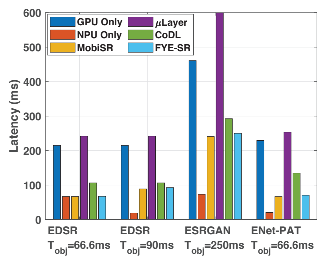
- **MobiSR [37]** splits input images into patches and dispatches them to CPU, GPU and AI accelerator based on SR difficulty.
- ***μ*Layer [28]** speeds up per-layer execution by splitting each NN layer by channels onto different processors. Each layer's execution requires synchronization and communication for data consistency.
- **CoDL [26]** performs image SR in full precision between GPU and CPU by splitting image data. It reduces communication overhead by performing less GPU-CPU communication, at the cost of computing extra image features on both processors.



(a) Structural image quality (SSIM↑)

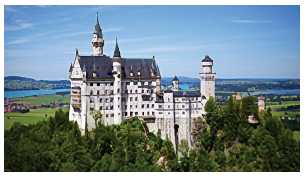


(b) Perceptual image quality (PIQE↓)

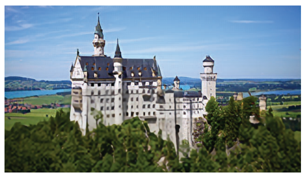


(c) SR computing latency

Figure 15: SR image quality and computing latency using EDSR, ESRGAN, and ENet-PAT models on DIV2K dataset with 4× 720×480 output on a Pixel 6 smartphone


(a) GPU Only (PIQE=26.84)

(b) NPU Only (PIQE=62.58)


(c) MobiSR (PIQE=47.10)

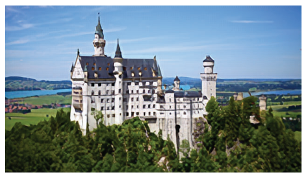

(d) FYE-SR (PIQE=32.75)

Figure 16: Samples of upscaled images using ESRGAN model with 4× 720×480 output. FYE-SR can effectively suppress the distortions and visual inconsistency at detailed objects (windows on buildings).

In all experiments, we use the Adam optimizer [29] with a learning rate of 1×10^{-4} and batch size of 8 to train the Model Split Learner. With the SR model split being learned, we adopt our implementation in Section 7 and conduct experiments on Google Pixel smartphones, including Pixel 6, 7 and 8 models, using their on-board GPU and Tensor NPU.

8.1 Image Quality and Computing Latency

We first compare FYE-SR with other baseline schemes on DIV2K dataset for 4× SR with 720×480 image output. We use EDSR and ESRGAN models, and set the timing constraint (T_{obj}) to range between 66.6ms (15 FPS) and 250ms (4 FPS). To better demonstrate the advantage of FYE-SR, we also compared it with GPU-only computation and NPU-only computation. Note that GPU-only computation is always at full precision and hence achieves the best SR image quality, but also incurs the highest computing latency. NPU-only computation, on the other hand, minimizes the SR computing latency but also results in the lowest SR image quality.

As shown in Figure 15, despite slight drop in structural image quality (measured by SSIM) compared to GPU Only, FYE-SR always approximates the perceptual quality of upscaled images to that of GPU Only, and outperforms other baselines by up to 50%. In particular, FYE-SR performs best on larger SR models such as ESRGAN that consists of >100 convolutional layers. The samples of upscaled images in Figure 16 further show that, compared to MobiSR, FYE-SR can effectively suppress the distortions and visual inconsistency at detailed objects. This result verifies the effectiveness of our Model Split Learner in exploring the best split of SR models.

Besides, FYE-SR can always meet the given time constraint and reduce the SR computing latency by 1.8×-5.6×. Especially when compared with μLayer and CoDL, FYE-SR can improve the SR performance by avoiding the large overhead of synchronization and communication between processors. Note that the performance difference between FYE-SR and baselines varies by T_{obj}. Such difference is small when T_{obj} is relaxed but will be significant when T_{obj} is tight. In Figure 15, when T_{obj}=66.6ms, FYE-SR reduces the latency by 47.7% compared to CoDL, but its image quality drop is 3.2%.

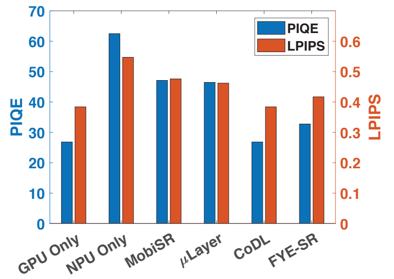
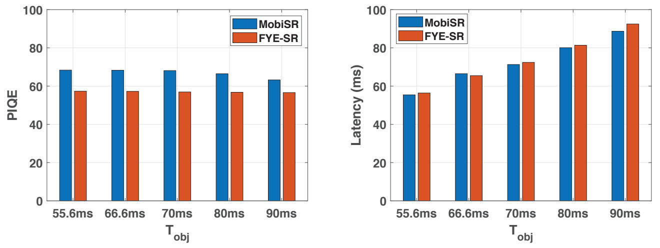


Figure 17: SR image quality measured by different perceptual metrics, including PIQE↓ and LPIPS↓

Due to the difficulty of aligning with human perception, a single perceptual quality metric may not capture all aspects of the images' perceptual quality. As shown in Figure 17, when using different perceptual image quality metrics, FYE-SR can always achieve similarly higher perceptual quality of upscaled images and hence demonstrates high generality over different application scenarios.

8.2 Performance with Different T_{obj}

We vary the timing constraint (T_{obj}) of SR computations and then compare FYE-SR with baselines. Since the latency of CoDL and μLayer are fixed and relatively longer as shown in Figure 15(c), we only compare with the most competitive baseline, i.e., MobiSR. As shown in Figure 18, when the timing constraint varies, FYE-SR constantly outperforms MobiSR on image quality while achieving on-par SR computing latency. This is because MobiSR splits the input image across processors based on pixel variations, which is less accurate and underestimates inter-pixel consistency.



(a) Perceptual image quality (PIQE↓) (b) SR computing latency

Figure 18: SR image quality and computing latency w.r.t different T_{obj} using EDSR

In particular, as shown in Figure 19, when T_{obj} increases, FYE-SR is able to adaptively adjust the SR model split and allows more SR computations to be executed on GPU, hence resulting in extra improvement of upscaled image quality.

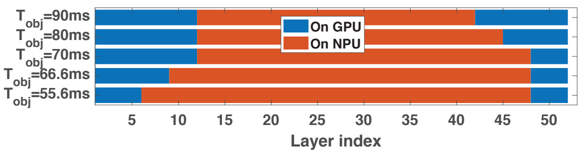


Figure 19: Model split on EDSR with different T_{obj}

8.3 Ablation Study

Figure 20 shows the breakdown of SR computing latency, with ESRGAN and EDSR models, when T_{obj} is 250ms and 66.6ms, respectively. In general, more complicated SR model results in more switches between processors in SR computations, but even for the complicated ESRGAN model, FYE-SR can still constrain such switching overhead within 40%. Similarly, with the more complicated ESRGAN model, more GPU computations are needed to maximize the image quality.

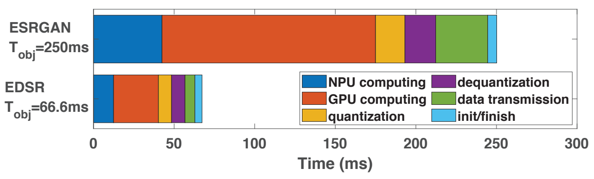


Figure 20: Breakdown of SR computing latency

8.4 Using Different Smartphone Models

To demonstrate FYE-SR's generality, we compare its performance on Google Pixel 6, 7 and 8 smartphones, which have highly different computing power. For example, e.g., Pixel 7's Mali-G78 MP20 GPU can do 1.94 TFLOPS but Pixel 6's Mali-G710 MP7 can only do 0.7 TFLOPS in FP32. As shown in Table 7, FYE-SR can satisfy timing constraints and maintain high image quality on all device models. With stronger devices, FYE-SR achieves better quality-latency tradeoff due to better hardware efficiency, e.g., 2% better PIQE and 6% lower latency with EDSR on Pixel 8 compared to Pixel 6.

Device	SR Model	T_{obj}(ms)	PSNR/SSIM/PIQE/time(ms)
Pixel 6	EDSR	66.6	26.43/0.812/56.45/67.3
Pixel 6	ESRGAN	250	27.32/0.813/32.75/250.4
Pixel 7	EDSR	66.6	26.51/0.817/55.34/64.25
Pixel 7	ESRGAN	250	27.32/0.813/32.03/241.7
Pixel 8	EDSR	66.6	26.51/0.817/55.12/63.72
Pixel 8	ESRGAN	250	27.38/0.814/32.24/240.85

Table 7: SR image quality and computing latency w.r.t different smartphone models

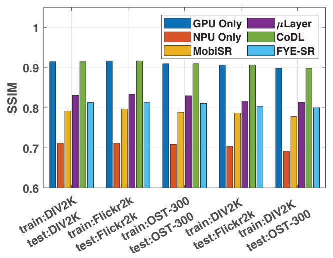
8.5 Performance on Different Datasets

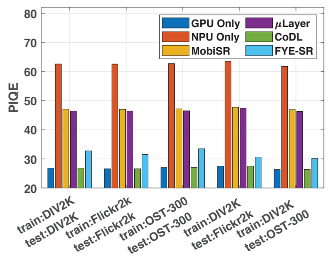
Datasets for SR tasks are typically small (<10,000) compared to other tasks (e.g., image classification) and contain partial world knowledge about colors, texture, and object relationships. Such domain difference challenges the FYE-SR's optimality. Nevertheless, Figure 21 shows that FYE-SR constantly outperforms other baselines on all three datasets. Compared to MobiSR, FYE-SR improves SSIM and PIQE by up to 3% and 55% respectively. Besides, like most existing learning-based methods, the training of Model Split Learner may overfit the training set. However, we verified that FYE-SR maintains high performance when trained on DIV2K and tested on Flickr2K, and shows minor performance loss between DIV2K and OST-300.

8.6 Performance with Different SR Configurations

We vary the SR configurations with different SR ratios and output image resolutions, and train the SR model split accordingly. As shown in Figure 22, FYE-SR speeds up SR computations by 3× compared to GPU Only, while significantly improving the image quality in all configurations. In particular, FYE-SR can speed up more when SR ratio=2×, because a lower SR ratio needs less precision and more layers can hence be computed at NPU.

In these cases, as shown in Figure 23, instead of reducing the number of layers computed on GPU, FYE-SR demonstrates adaptability and can retain the number of layers on GPU but instead reduce the number of switches between processors. In this way, it saves the overall computing latency while minimizing the loss of image quality.


(a) Structural image quality (SSIM↑)


(b) Perceptual image quality (PIQE↓)

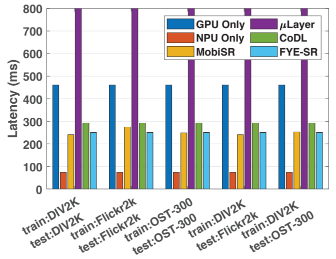
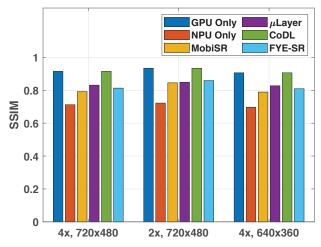
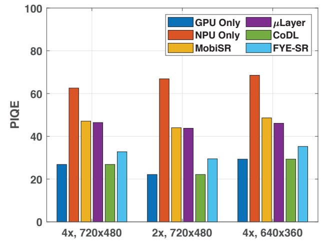

(c) SR computing latency

Figure 21: SR image quality and computing latency using ESRGAN model on different datasets with T_{obj}=250ms and 4× 720×480 output


(a) Structural image quality (SSIM↑)


(b) Perceptual image quality (PIQE↓)

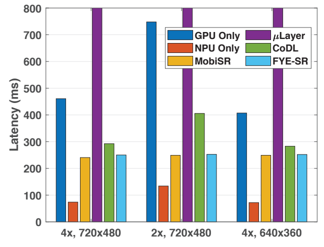

(c) SR computing latency

Figure 22: SR image quality and computing latency using ESRGAN model on DIV2K dataset and different SR configurations on a Pixel 6 smartphone. T_{obj}=250ms

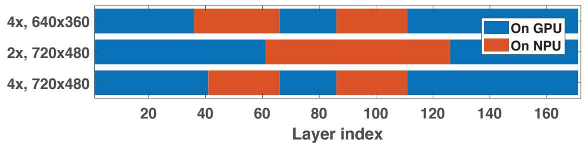


Figure 23: Model split on ESRGAN with different SR configurations

8.7 Power Consumption

FYE-SR is power-efficient because it dispatches the majority of SR computations for NPU execution, which by design consumes less power. To test FYE-SR's power consumption, we keep the SR program running in the foreground on smartphones and set the screen brightness to 50%. The device has WiFi connection but no cellular connection. As shown in Figure 24, the battery drainage after 1-hr use is <20%, enabling multiple hours of continuous SR computations.

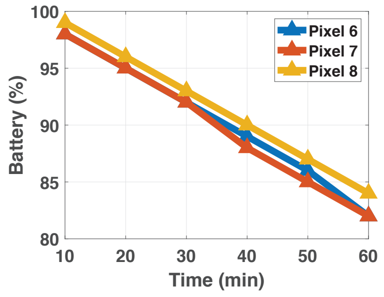


Figure 24: Power consumption on different devices

In addition, we also measured the per-frame energy usage on different smartphone SoCs, including Snapdragon M865, Snapdragon M888 and Google Pixel 6. Such per-frame energy usage is measured on an average basis from the battery usage of continus SR inference over 53,484 frames. As shown in Table 8, FYE-SR consumes less energy per frame compared to MobiSR. Its energy usage is slightly higher than NAWQ-SR [69] which only uses energy-efficient NPU for SR computations, but note that NAWQ-SR requires NPUs to support mixed-precision computations that are not available on most smartphone models.

Device	Method	Per-frame energy usage (J)
SDM888	MobiSR [37]	0.910
SDM865	MobiSR [37]	0.386
SDM888	MobiSR [37]	0.292
SDM865	NAWQ-SR [69]	0.114
SDM888	NAWQ-SR [69]	0.073
Pixel 6	FYE-SR	0.216

Table 8: Per-frame energy usage comparison across different SR methods

9 RELATED WORK

Computationally efficient image SR. FYE-SR is related to the existing work on computationally efficient image SR. Most existing AI work focuses on developing more lightweight SR models with methods such as compressed convolutional filter [12, 62] and skip connections [77]. However, on mobile devices with limited computing power, even such lightweight models cannot achieve high FPS, due to large feature maps in NN inference. Aggressively compressing the SR model (e.g., binarized model [44]) can improve speed but significantly impair the image quality.

To further speed up SR, system efforts were made to distribute SR computations to heterogeneous processors on mobile devices [36, 37, 69], but suffer from visual inconsistency in upscaled images. Later work [42] mitigates such visual inconsistency via fine-grained partitioning of NN features, but incurs large communication overhead between processors. In contrast, our design of FYE-SR aims to fully utilize the computing power of heterogeneous processors but minimize the loss of perceptual quality of upscaled images.

Efficient NN computations with heterogeneous processors. Our design of FYE-SR is inspired by the existing work that utilizes heterogeneous on-device processors for efficient NN computations. Early work focuses on reducing the overall energy consumption but cannot speed up [32]. Later work speeds up NN inference [25, 26] and training [76] via concurrent execution of different NN substructures on different processors, but ignores the possible accuracy loss caused by AI accelerators with low arithmetic precision.

Recent work also utilized the capabilities of mixed-precision computation on mobile processors for SR computations [69], such that SR computations are split for different arithmetic precisions based on the corresponding difficulty of computation and impact on image quality. These methods, nevertheless, only utilize one physical processor and are orthogonal to FYE-SR, which involves more system factors for switching between different processors, including the switching delay, data format conversion and data transfer delay.

Neural architecture search. Our design of Model Split Learner in FYE-SR is related to the existing work in neural architecture search (NAS) [57]. Existing NAS methods search for the optimal model structures that satisfy performance objectives (e.g., accuracy, latency, and memory cost), but the searched models are limited to single-processor execution with fixed precision. In contrast, our design in FYE-SR seeks for the optimal split of SR model to best balance between SR computations with different arithmetic precisions.

10 DISCUSSIONS

Online adaptation of SR model split. In our current design of FYE-SR, we train the Model Split Learner offline and use the learned model split online, and our experiment results in Section 8.5 show that such learned model split performs well over different testing datasets. It may be necessary to adjust the model split if the SR timing constraint changes online, and we can adopt the existing incremental learning [7] and approximate learning [63] for such online adaptation.

Generality to other NN structures. Our evaluations in this paper mainly involve CNN-based SR models, but our design of FYE-SR can be generally applied to other NN model structures. For example, recent diffusion [54] and diffusion transformer [53] models can also be used to upscale and enhance the image quality. To apply FYE-SR, we can convert the diffusion model to a sequential model by unrolling through time [75]. The involvement of α in model split learning will also adapt to the parameter sharing in the unrolled model.

Integrating advanced quantization methods. The performance of FYE-SR can be further improved if the quantized portion of SR model running on AI accelerators can be more accurate. To improve such accuracy, advanced post-quantization techniques [68] and quantization-aware training [48] can be integrated into Model Split Learner, to jointly optimize the model split and model parameters. Automatic hyper-parameter tuning methods [20] can also assist in deciding the best weights to balance different loss terms.

Adapting to mixed-precision processors. FYE-SR focuses on fully utilizing the computing power of heterogeneous mobile processors (GPU and NPU) that operate different arithmetic precisions. Some recent methods, such as NAWQ-SR [69], instead exploit the capability of mixed-precision computation at the single NPU. Hence, although both FYE-SR and NAWQ-SR split the SR model to fit different computation requirements, they adopt different splitting methods due to the difference in targeted computing hardware. On devices with mixed-precision NPUs available, FYE-SR and NAWQ-SR can complement each other, such that FYE-SR splits computations between GPU and NPU, and NAWQ-SR further splits computations on NPU at a finer granularity.

Extending to video SR. Our design of FYE-SR can be generally applied to speed up video SR on mobile devices, by focusing on temporal fluency across consecutive frames and incorporating video quality metrics [16] into the loss. In addition, the NNs for video SR usually contain additional branches to process temporal information [60] and hence complicate the modeling of end-to-end SR computing latency. We will further study such modeling in our future work.

11 CONCLUSION

In this paper, we present FYE-SR, a new image SR technique that maximizes the perceptual quality of upscaled images on mobile devices. The basic approach of FYE-SR is to split the SR model and dispatch different NN layers to heterogeneous processors, and decide the optimal model split with an end-to-end learning approach. Experiment results show that FYE-SR can significantly enhance the perceptual image quality while satisfying the timing constraint of SR computations.

ACKNOWLEDGMENTS

We thank the anonymous shepherd and reviewers for their comments and feedback. This work was supported in part by National Science Foundation (NSF) under grant number IIS-2205360, CCF-2217003, CCF-2215042, and National Institutes of Health (NIH) under grant number R01HL170368.

REFERENCES

[1] Eirikur Agustsson and Radu Timofte. 2017. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 126–135.

[2] Shun-ichi Amari. 1993. Backpropagation and stochastic gradient descent method. *Neurocomputing* 5, 4-5 (1993), 185–196.

[3] Saeed Anwar, Salman Khan, and Nick Barnes. 2020. A Deep Journey into Super-Resolution: A Survey. *ACM Comput. Surv.* 53, 3, Article 60 (may 2020), 34 pages. https://doi.org/10.1145/3390462

[4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[5] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.

[6] Yochai Blau and Tomer Michaeli. 2018. The perception-distortion tradeoff. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 6228–6237. https://doi.org/10.1109/cvpr.2018.00652

[7] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. 2018. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*. 233–248.

[8] Xiangyu Chen, Xintao Wang, Jiantao Zhou, Yu Qiao, and Chao Dong. 2023. Activating more pixels in image super-resolution transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 22367–22377.

[9] Gunhee Cho, Yusuke Shinyama, Jin Nakazato, Kazuki Maruta, and Kei Sakaguchi. 2022. Object recognition network using continuous roadside cameras. In *2022 IEEE 95th Vehicular Technology Conference:(VTC2022-Spring)*. IEEE, 1–5. https://doi.org/10.1109/VTC2022-Spring54318.2022.9860677

[10] Saptarsi Das, Arnab Roy, Kiran Kolar Chandrasekharan, Ankur Deshwal, and Sehwan Lee. 2020. A systolic dataflow based accelerator for CNNs. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5. https://doi.org/10.1109/ISCAS45731.2020.9180403

[11] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2015. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence* 38, 2 (2015), 295–307.

[12] Chao Dong, Chen Change Loy, and Xiaoou Tang. 2016. Accelerating the super-resolution convolutional neural network. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*. Springer, Springer International Publishing, 391–407.

[13] Tingxing Tim Dong, Hao Yan, Mayank Parasar, and Raun Krisch. 2022. Rendersr: A lightweight super-resolution model for mobile gaming upscaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3087–3095. https://doi.org/cvprw56347.2022.00348

[14] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems* 33 (2020), 18518–18529.

[15] Claude E Duchon. 1979. Lanczos filtering in one and two dimensions. *Journal of Applied Meteorology and Climatology* 18, 8 (1979), 1016–1022.

[16] Qiang Fan, Wang Luo, Yuan Xia, Guozhi Li, and Daojing He. 2019. Metrics and methods of video quality assessment: a brief review. *Multimedia Tools and Applications* 78 (2019), 31019–31033.

[17] Sina Farsiu, Dirk Robinson, Michael Elad, and Peyman Milanfar. 2004. Advances and challenges in super-resolution. *International Journal of Imaging Systems and Technology* 14, 2 (2004), 47–57. https://doi.org/10.1002/ima.20007

[18] AMD GPUOpen. 2021. AMD FidelityFX Super Resolution 1 (FSR 1). https://gpuopen.com/fidelityfx-superresolution/

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

[20] A Ali Heydari, Craig A Thompson, and Asif Mehmood. 2019. Softadapt: Techniques for adaptive loss weighting of neural networks with multi-part loss functions. *arXiv preprint arXiv:1912.12355* (2019).

[21] Alain Hore and Djemel Ziou. 2010. Image quality metrics: PSNR vs. SSIM. In *2010 20th international conference on pattern recognition*. IEEE, 2366–2369.

[22] Kuan-Yu Huang. 2020. esrgan-tf2. https://github.com/peteryuX/esrgan-tf2.

[23] Andrey Ignatov, Andres Romero, Heewon Kim, and Radu Timofte. 2021. Real-Time Video Super-Resolution on Smartphones With Deep Learning, Mobile AI 2021 Challenge: Report. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2535–2544.

[24] Andrey Ignatov, Radu Timofte, Maurizio Denna, and Abdel Younes. 2021. Real-time quantized image super-resolution on mobile npus, mobile ai 2021 challenge: Report. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2525–2534.

[25] Joo Seong Jeong, Jingyu Lee, Donghyun Kim, Changmin Jeon, Changjin Jeong, Youngki Lee, and Byung-Gon Chun. 2022. Band: coordinated multi-dnn inference on heterogeneous mobile processors. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. 235–247.

[26] Fucheng Jia, Deyu Zhang, Ting Cao, Shiqi Jiang, Yunxin Liu, Ju Ren, and Yaoxue Zhang. 2022. CoDL: Efficient CPU-GPU Co-Execution for Deep Learning Inference on Mobile Devices. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '22)*. Association for Computing Machinery, New York, NY, USA, 209–221. https://doi.org/10.1145/3498361.3538932

[27] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.

[28] Youngsok Kim, Joonsung Kim, Dongju Chae, Daehyun Kim, and Jangwoo Kim. 2019. μLayer: Low Latency On-Device Inference Using Cooperative Single-Layer Acceleration and Processor-Friendly Quantization. In *Proceedings of the Fourteenth EuroSys Conference 2019 (EuroSys '19)*. Association for Computing Machinery, New York, NY, USA, Article 45, 15 pages. https://doi.org/10.1145/3302424.3303950

[29] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[30] Martin Krasser. 2022. Single Image Super-Resolution with EDSR, WDSR and SRGAN. https://github.com/krasserm/super-resolution.

[31] Koushik Sivarama Krishnan and Karthik Sivarama Krishnan. 2021. SwiftSRGAN-Rethinking Super-Resolution for Efficient and Real-time Inference. In *2021 International Conference on Intelligent Cybernetics Technology & Applications (ICICyTA)*. IEEE, 46–51. https://doi.org/10.1109/ICICyTA53712.2021.9689188

[32] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim 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)*. IEEE, 1–12.

[33] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. 2017. Photo-realistic single image

super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4681–4690.

[34] Chulhee Lee, Seongyoun Woo, and Sangwook Baek. 2019. Bitrate and transmission resolution determination based on perceptual video quality. In *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*. IEEE, 1–6. https://doi.org/10.1109/IISA.2019.8900685

[35] Jounghoo Lee, Jinwoo Choi, Jaeyeon Kim, Jinho Lee, and Youngsok Kim. 2021. Dataflow mirroring: Architectural support for highly efficient fine-grained spatial multitasking on systolic-array npus. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 247–252. https://doi.org/10.1109/DAC18074.2021.9586312

[36] Juhyoung Lee, Jinsu Lee, and Hoi-Jun Yoo. 2020. SRNPU: An energy-efficient CNN-based super-resolution processor with tile-based selective super-resolution in mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 10, 3 (2020), 320–334.

[37] Royson Lee, Stylianos I. Venieris, Lukasz Dudziak, Sourav Bhattacharya, and 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)*. Association for Computing Machinery, New York, NY, USA, Article 54, 16 pages. https://doi.org/10.1145/3300061.3345455

[38] Huixia Li, Chenqian Yan, Shaohui Lin, Xiawu Zheng, Baochang Zhang, Fan Yang, and Rongrong Ji. 2020. Pams: Quantized super-resolution via parameterized max scale. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXV 16*. Springer, 564–580.

[39] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. 2021. Swinir: Image restoration using swin transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*. 1833–1844.

[40] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. 2017. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 136–144. https://doi.org/10.1109/cvprw.2017.151

[41] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 740–755.

[42] Xin Liu, Yuang Li, Josh Fromm, Yuntao Wang, Ziheng Jiang, Alex Mariakakis, and Shwetak Patel. 2021. Splitsr: An end-to-end approach to super-resolution on mobile devices. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 1 (2021), 1–20.

[43] Zhisheng Lu, Juncheng Li, Hong Liu, Chaoyan Huang, Linlin Zhang, and Tieyong Zeng. 2022. Transformer for single image super-resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 457–466.

[44] Yinglan Ma, Hongyu Xiong, Zhe Hu, and Lizhuang Ma. 2019. Efficient super resolution using binarized neural network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 0–0.

[45] Evan M Masutani, Naeim Bahrami, and Albert Hsiao. 2020. Deep learning single-frame and multiframe super-resolution for cardiac MRI. *Radiology* 295, 3 (2020), 552–561.

[46] Kaisa Miettinen. 1999. *Nonlinear multiobjective optimization*. Vol. 12. Springer Science & Business Media.

[47] Anish Mittal, Rajiv Soundararajan, and Alan C Bovik. 2012. Making a "completely blind" image quality analyzer. *IEEE Signal processing letters* 20, 3 (2012), 209–212. https://doi.org/10.1109/LSP.2012.2227726

[48] Markus Nagel, Marios Fournarakis, Yelysei Bondarenko, and Tijmen Blankevoort. 2022. Overcoming oscillations in quantization-aware training. In *International Conference on Machine Learning*. PMLR, 16318–16330.

[49] Young H Oh, Seonghak Kim, Yunho Jin, Sam Son, Jonghyun Bae, Jongsung Lee, Yeonhong Park, Dong Uk Kim, Tae Jun Ham, and Jae W Lee. 2021. Layerweaver: Maximizing resource utilization of neural processing units via layer-wise scheduling. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 584–597. https://doi.org/10.1109/HPCA51647.2021.00056

[50] Jean-François Pambrun and Rita Noumeir. 2015. Limitations of the SSIM quality metric in the context of diagnostic imaging. In *2015 IEEE international conference on image processing (ICIP)*. IEEE, 2960–2963.

[51] Eunhyeok Park, Junwhan Ahn, and Sungjoo Yoo. 2017. Weighted-entropy-based quantization for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5456–5464.

[52] Sung Cheol Park, Min Kyu Park, and Moon Gi Kang. 2003. Super-resolution image reconstruction: a technical overview. *IEEE Signal Processing Magazine* 20, 3 (2003), 21–36. https://doi.org/10.1109/MSP.2003.1203207

[53] William Peebles and Saining Xie. 2023. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4195–4205.

[54] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. 2023. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952* (2023).

[55] Haotong Qin, Yulun Zhang, Yifu Ding, Xianglong Liu, Martin Danelljan, Fisher Yu, et al. 2024. QuantSR: Accurate Low-bit Quantization for Efficient Image Super-Resolution. *Advances in Neural Information Processing Systems* 36 (2024).

[56] Qualcomm. 2021. Qualcomm Neural Processing SDK. https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk

[57] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2021. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)* 54, 4 (2021), 1–34.

[58] Google Research. 2021. Improved On-Device ML on Pixel 6, with Neural Architecture Search. https://blog.research.google/2021/11/improved-on-device-ml-on-pixel-6-with.html

[59] Mehdi SM Sajjadi, Bernhard Scholkopf, and Michael Hirsch. 2017. Enhancenet: Single image super-resolution through automated texture synthesis. In *Proceedings of the IEEE international conference on computer vision*. 4491–4500. https://doi.org/10.1109/iccv.2017.481

[60] Mehdi SM Sajjadi, Raviteja Vemulapalli, and Matthew Brown. 2018. Frame-recurrent video super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 6626–6634.

[61] H Sheikh. 2005. LIVE image quality assessment database release 2. *http://live.ece.utexas.edu/research/quality* (2005).

[62] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. 2016. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1874–1883.

[63] Xu Sun, Xuancheng Ren, Shuming Ma, and Houfeng Wang. 2017. meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting. In *International Conference on Machine Learning*. PMLR, 3299–3308.

[64] Tianxiang Tan and Guohong Cao. 2021. Efficient execution of deep neural networks on mobile devices with npu. In *Proceedings of the 20th International Conference on Information Processing in Sensor Networks*

(Co-Located with CPS-IoT Week 2021). 283–298.

[65] Tianxiang Tan and Guohong Cao. 2022. Deep learning on mobile devices through neural processing units and edge computing. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 1209–1218. https://doi.org/10.1109/INFOCOM48880.2022.9796929

[66] Chengzhou Tang, Yuqiang Yang, Bing Zeng, Ping Tan, and Shuaicheng Liu. 2022. Learning to Zoom Inside Camera Imaging Pipeline. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 17552–17561. https://doi.org/10.1109/CVPR52688.2022.01703

[67] Radu Timofte, Shuhang Gu, Jiqing Wu, and Luc Van Gool. 2018. Ntire 2018 challenge on single image super-resolution: Methods and results. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 852–863.

[68] Zhijun Tu, Jie Hu, Hanting Chen, and Yunhe Wang. 2023. Toward Accurate Post-Training Quantization for Image Super Resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5856–5865.

[69] Stylianos I Venieris, Mario Almeida, Royson Lee, and Nicholas D Lane. 2023. NAWQ-SR: A Hybrid-Precision NPU Engine for Efficient On-Device Super-Resolution. *IEEE Transactions on Mobile Computing* (2023). https://doi.org/10.1109/TMC.2023.3255822

[70] N Venkatanath, D Praneeth, Maruthi Chandrasekhar Bh, Sumohana S Channappayya, and Swarup S Medasani. 2015. Blind image quality evaluation using perception based features. In *2015 twenty first national conference on communications (NCC)*. IEEE, 1–6. https://doi.org/10.1109/NCC.2015.7084843

[71] Xuezhi Wang, Guanyu Gao, Xiaohu Wu, Yan Lyu, and Weiwei Wu. 2022. Dynamic DNN model selection and inference off loading for video analytics with edge-cloud collaboration. In *Proceedings of the 32nd Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '22)*. Association for Computing Machinery, New York, NY, USA, 64–70. https://doi.org/10.1145/3534088.3534352

[72] Xintao Wang, Ke Yu, Chao Dong, and Chen Change Loy. 2018. Recovering realistic texture in image super-resolution by deep spatial feature transform. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 606–615.

[73] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. 2018. Esrgan: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European conference on computer vision (ECCV) workshops*. 0–0.

[74] Zhihao Wang, Jian Chen, and Steven CH Hoi. 2020. Deep learning for image super-resolution: A survey. *IEEE transactions on pattern analysis and machine intelligence* 43, 10 (2020), 3365–3387. https://doi.org/10.1109/TPAMI.2020.2982166

[75] Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 10 (1990), 1550–1560.

[76] Daliang Xu, Mengwei Xu, Qipeng Wang, Shangguang Wang, Yun Ma, Kang Huang, Gang Huang, Xin Jin, and Xuanzhe Liu. 2022. Mandheling: Mixed-precision on-device dnn training with dsp offloading. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 214–227.

[77] Jin Yamanaka, Shigesumi Kuwashima, and Takio Kurita. 2017. Fast and accurate image super resolution by deep CNN with skip connection and network in network. In *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14-18, 2017, Proceedings, Part II 24*. Springer, 217–225.

[78] Hyunho Yeo, Chan Ju Chong, Youngmok Jung, Juncheol Ye, and Dongsu Han. 2020. NEMO: Enabling Neural-Enhanced Video Streaming on Commodity Mobile Devices. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom '20)*. Association for Computing Machinery, New York, NY, USA, Article 28, 14 pages. https://doi.org/10.1145/3372224.3419185

[79] Juheon Yi, Seongwon Kim, Joongheon Kim, and Sunghyun Choi. 2020. Supremo: Cloud-assisted low-latency super-resolution in mobile devices. *IEEE Transactions on Mobile Computing* 21, 5 (2020), 1847–1860. https://doi.org/10.1109/TMC.2020.3025300

[80] Kaihao Zhang, Dongxu Li, Wenhan Luo, Wenqi Ren, Bjorn Stenger, Wei Liu, Hongdong Li, and Yang Ming-Hsuan. 2021. Benchmarking Ultra-High-Definition Image Super-resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

[81] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 586–595. https://doi.org/10.1109/cvpr.2018.00068

[82] Wenlong Zhang, Yihao Liu, Chao Dong, and Yu Qiao. 2019. Ranksrgan: Generative adversarial networks with ranker for image super-resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 3096–3105. https://doi.org/10.1109/iccv.2019.00319

[83] Yinjie Zhang, Yuanxing Zhang, Yi Wu, Yu Tao, Kaigui Bian, Pan Zhou, Lingyang Song, and Hu Tuo. 2020. Improving quality of experience by adaptive video streaming with super-resolution. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 1957–1966. https://doi.org/10.1109/INFOCOM41043.2020.9155384