SplitSR: An End-to-End Approach to Super-Resolution on Mobile Devices

XIN LIU, University of Washington, USA
YUANG LI, BUPT & University of Washington, USA
JOSH FROMM, University of Washington & OctoML, USA
YUNTAO WANG, Tsinghua Univiersity & University of Washington, USA
ZIHENG JIANG, University of Washington & OctoML, USA
ALEX MARIAKAKIS, University of Toronto, Canada
SHWETAK PATEL, University of Washington, USA

Super-resolution (SR) is a coveted image processing technique for mobile apps ranging from the basic camera apps to mobile health. Existing SR algorithms rely on deep learning models with significant memory requirements, so they have yet to be deployed on mobile devices and instead operate in the cloud to achieve feasible inference time. This shortcoming prevents existing SR methods from being used in applications that require near real-time latency. In this work, we demonstrate state-of-the-art latency and accuracy for on-device super-resolution using a novel hybrid architecture called SplitSR and a novel lightweight residual block called SplitSRBlock. The SplitSRBlock supports channel-splitting, allowing the residual blocks to retain spatial information while reducing the computation in the channel dimension. SplitSR has a hybrid design consisting of standard convolutional blocks and lightweight residual blocks, allowing people to tune SplitSR for their computational budget. We evaluate our system on a low-end ARM CPU, demonstrating both higher accuracy and up to 5× faster inference than previous approaches. We then deploy our model onto a smartphone in an app called ZoomSR to demonstrate the first-ever instance of on-device, deep learning-based SR. We conducted a user study with 15 participants to have them assess the perceived quality of images that were post-processed by SplitSR. Relative to bilinear interpolation — the existing standard for on-device SR — participants showed a statistically significant preference when looking at both images ($Z=-9.270$, $p<0.01$) and text ($Z=-6.486$, $p<0.01$).

CCS Concepts: • **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**; **Interactive systems and tools**; • **Computing methodologies** → **Computer vision tasks**;

Additional Key Words and Phrases: mobile computing, image super-resolution, edge computing, on-device machine learning

ACM Reference Format:
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. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 1, Article 25 (March 2021), 20 pages. https://doi.org/10.1145/3448104

Authors' addresses: Xin Liu, xliu0@cs.washington.edu, University of Washington, Seattle, WA, USA; Yuang Li, BUPT & University of Washington, Seattle, WA, USA; Josh Fromm, jwfromm@octoml.ai, University of Washington & OctoML, Seattle, WA, USA; Yuntao Wang, yuntaowang@tsinghua.edu.cn, Tsinghua Univiersity & University of Washington, Seattle, WA, USA; Ziheng Jiang, ziheng@cs.washington.edu, University of Washington & OctoML, Seattle, WA, USA; Alex Mariakakis, mariakakis@cs.toronto.edu, University of Toronto, Toronto, Canada; Shwetak Patel, shwetak@cs.washington.edu, University of Washington, Seattle, WA, USA.

2474-9567/2021/3-ART25 $15.00
https://doi.org/10.1145/3448104

1 INTRODUCTION

Image super-resolution (SR) is the task of reconstructing a high-resolution image from a low-resolution input. One domain where SR could have major impact is human-computer interaction. For example, most smartphones rely on digital zoom because they do not have an optical zoom lens, so on-device SR can provide high-resolution images when users examine static images [12]. Low-latency, on-device SR can also improve the quality of feedback during image capture as users adjust the camera's position and zoom factor. When users take photographs of documents, they can be confident that they will be able to read their documents later on since on-device SR can make individual characters more discernible [8, 32]. When users want to send photographs to others via computer-mediated communication, on-device SR can reduce data transmission requirements by allowing users to send low-resolution images that are later upsampled [7].

Another domain where on-device SR would have significant impact is healthcare [13, 36, 44], especially in the developing world where lower-end smartphones are more common. Plug-and-play smartphone accessories like portable ultrasound probes enable medical imaging, with the smartphone serving as a computation and communication hub [2]. Researchers have also developed smartphone accessories that augment the capabilities of the built-in camera for health screening [11, 40], including enclosed optical components for identifying refractive errors [30] and cataracts [31]. Furthermore, mobile health researchers have begun to utilize unmodified smartphone cameras for health screening. For example, Wadhawan et al. [45] created a smartphone app that analyzes images of skin lesions to classify them as benign or malignant, while Mariakakis et al. [28] proposed an app that analyzes short videos of pupil dilation for identifying traumatic brain injuries. Across all of these healthcare applications, low-latency on-device SR would ensure that small details are not overlooked during image analysis and not affect the user experience. By keeping computation on the smartphone itself and bypassing the need for data upload, connectivity in rural regions becomes less critical and sensitive patient information can be preserved.

Deep convolutional neural networks (CNNs) have yielded state-of-the-art accuracy for SR [21, 34, 49], but achieving this performance has required network architectures with many layers and a large spatial dimension across layers. As a result, these models usually have 1–5 million parameters, making them infeasible to directly deploy on resource-constrained platforms [17, 49, 51]. Therefore, such models are often deployed on cloud-based servers, which incurs latency penalties for data upload and result download. The inference time of the models themselves are also not trivial, falling short of the requirements for on-device scenarios [23]. Low-latency on-device SR would change this paradigm, eliminating unnecessary data transmission and power consumption during real-time applications.

To overcome these limitations, we propose a novel architecture for on-device SR called SplitSR. Our key insight is that a hybrid design consisting of standard convolutions and lightweight residual blocks can be used to maximize accuracy while satisfying a restricted computational budget. We apply this insight to extend RCAN [51], a state-of-the-art SR model, so that it can be run on a mobile device. To push the limits of SplitSR even further, we also introduce a novel lightweight residual block called the SplitSRBlock. This block leverages channel-splitting within standard 2D convolutions to retain spatial information and reduce computational cost. We study various configurations of our SplitSR architecture and SplitSRBlock to see how hyperparameters impact both inference latency and accuracy. We also leverage a modern deep learning compiler called TVM [4] to implement the proposed system and generate highly efficient machine code that accelerates inference speed on our target embedded device. In doing so, we demonstrate that SplitSR achieves approximately 5× speedup and superior accuracy compared to a state-of-the-art on-device SR model. We then deployed SplitSR in an Android app called ZoomSR to see if users could notice the difference in latency and image quality between SplitSR and bilinear interpolation, the latter being the existing standard for on-device SR. Through a 15-person user study, we find

that the participants preferred SplitSR over bilinear interpolation for both images of objects ($Z=-9.270$, $p<0.01$) and text ($Z=-6.486$, $p<0.01$). In summary, our contributions include:

- A hybrid architecture called SplitSR that combines standard convolutional blocks and lightweight residual blocks to enable on-device SR within a constrained computational budget,
- A lightweight residual block called a SplitSRBlock that applies channel-splitting to standard 2D convolutions in order to accelerate computation and retrain high-quality spatial transformation,
- A comprehensive analysis that evaluates the benefits of our SplitSRBlock and our SplitSR model against state-of-the-art alternatives,
- ZoomSR, the first-ever smartphone app to demonstrate on-device, deep learning-based SR, and
- A user study with ZoomSR demonstrating that people prefer the image quality engendered by SplitSR rather than bilinear interpolation.

2 RELATED WORK

In this section, we provide an overview of single-image SR solutions, including those designed for on-device systems. We then describe how SplitSR improves upon its predecessors.

2.1 Super Resolution

Historically, researchers have explored myriad methods for attacking the super-resolution task — ranging from example-based strategies to sparse-coding-based methods [42, 48–50]. However, convolutional neural networks (CNNs) have quickly overtaken these approaches given their ability to extract a high-level representation of image data. Recent studies have demonstrated the superiority of deep learning compared to alternative methods when it comes to state-of-the-art SR tasks [6, 51]. One of the earliest examples of deep learning for SR is by Dong et al. [9]. Kim et al. [18] later improved upon Dong et al.'s work with a much deeper convolutional neural network ($\geq$16 layers), using residual learning to overcome the challenges of so many layers. One of the major drawbacks of these works is that they require interpolating the low-resolution images to high-resolution inputs before they are fed into the network, which adds computational complexity and potentially loses significant details due to smoothing effects. By training a separate upsampler module at the tail of the network, Dong et al. [10] introduced a faster and more accurate network. The idea of using an independent upsampling block has become a common choice for many modern SR architectures.

Numerous enhancements have been made to further improve SR accuracy. Zhang et al. [51] added an attention module to create a residual channel attention network (RCAN). To the best of our knowledge, RCAN's performance is currently rivaled by Dai et al. [6]'s second-order attention network (SAN) and Liu et al. [26]'s hierarchical back projection network (HBPN). However, these approaches incur significant computational costs, with SAN using self-ensembling across multiple SR models and HBPN requiring both RGB and YUV images as input. Sun and Chen [39] recently proposed a model called a content adaptive resampler (CAR), which improves the performance of a regular SR network by jointly training it with a model that downsamples high-resolution images. Like SAN, CAR incurs high computational costs since it requires a joint architecture, and CAR is also limited by the fact that it requires high-resolution images for inference. These models must be significantly downsized to accommodate the limited computational resources of mobile devices. However, reducing model complexity often leads to decreased accuracy, especially when the downsizing is done naïvely. In this work, we strive to create an accurate SR model that is efficient enough to run on a mobile device. SplitSR is a hybrid architecture that is intentionally designed to be tuned for a particular computational budget. For tighter budgets, SplitSR replaces standard convolutional blocks with lightweight residual blocks in a unique way to prioritize latency over accuracy. We apply this concept to the RCAN [51] architecture since it is more commonly used in the literature.

2.2 On-Device Super Resolution

Previous work has explored several methods for accelerating SR inference with custom hardware. For example, Kim et al. [19] demonstrated real-time SR on a highly optimized FPGA-based system. Although they attained a real-time performance from full HD (FHD) to 4K ultra HD (4K UHD), the algorithm was deployed on bulky and dedicated hardware that is not flexible to run other SR models. Dasari et al. [7] introduced a system for streaming 360° videos using SR. Although their work focused on network latency, they did not present results for how their approach would work on common 2D image datasets.

The most similar work to our own is MobiSR by Lee et al. [23]. MobiSR is an on-device SR system that runs on heterogeneous mobile processors, integrating the CPU, GPU, and digital signal processor (DSP). MobiSR achieves significant speedup compared to prior work, but the authors of that work were unable to implement their system on a mobile device because of the multi-processor load-balancing that was required. SplitSR achieves state-of-the-art mobile SR performance in terms of both speed and accuracy using a single neural network and a single CPU, making it accessible to many modern smartphones. In this work, we deploy SplitSR in an Android app called ZoomSR and evaluate whether users notice the difference between images processed by SplitSR versus bilinear interpolation.

2.3 Lightweight Residual Blocks

Standard convolutional layers are the fundamental building blocks for many computer vision applications. When many convolutional layers are stacked together to form very deep networks, residual connections can be used to facilitate gradient flow through the network during training. At a certain point, many networks become too computationally intensive for on-device systems, especially when the channel dimension of the feature maps is large. Assuming the input and output channels of a given convolutional layer are both N, the computational complexity of a standard convolutional layer is $O(N^2)$. In this section, we cover recent optimizations of convolutional layers that attempt to reduce convolutional computational complexity while maintaining high accuracy.

2.3.1 BottleneckBlock & ShuffleBlock. Howard et al. [15] proposed the use of depthwise separable convolutions to stack residual blocks and achieve a strong balance between accuracy and latency. These operations, also known as BottleneckBlocks, consist of two parts: a 3×3 depthwise convolution and by a 1×1 pointwise convolution. The depthwise convolution is used to expand features and makes use of single convolutional filters at every input channel to perform lightweight filtering, while the pointwise convolution constructs new features by calculating linear combinations of input channels. ShuffleBlock, introduced by Ma et al. [27], is an advanced version of the BottleneckBlock with several innovations (Figure 1A). First, ShuffleBlocks use channel-splitting to evenly split the input channels into branches. Only one branch is used for computation, where each channel is processed through two 1×1 pointwise convolutions and one 3×3 depthwise convolution. The branches are then concatenated, after which the channels are shuffled to allow information to be communicated between the two branches.

2.3.2 Inverted Residual Block & IdleBlock. Inverted residual blocks are designed to process significantly more spatial information during the depthwise convolution. To do this, these blocks add an additional pointwise layer at the beginning to expand low-dimensional information to a high-dimensional representation. After going through the depthwise convolution, these blocks use another pointwise layer to project the high-dimensional representation to the original low-dimensional space. Sandler et al. [38], who first proposed these blocks as part of their MobileNetV2 work, gained considerable performance improvements performance on the ImageNet dataset. Recently, Xu et al. [47] proposed the idea of combining an inverted residual block with a ShuffleBlock (without channel-shuffling) in order to prune superfluous connections. This construct is called an IdleBlock (Figure 1B. Xu et al. [47] also studied the use of IdleBlock in various hybrid configurations using two different

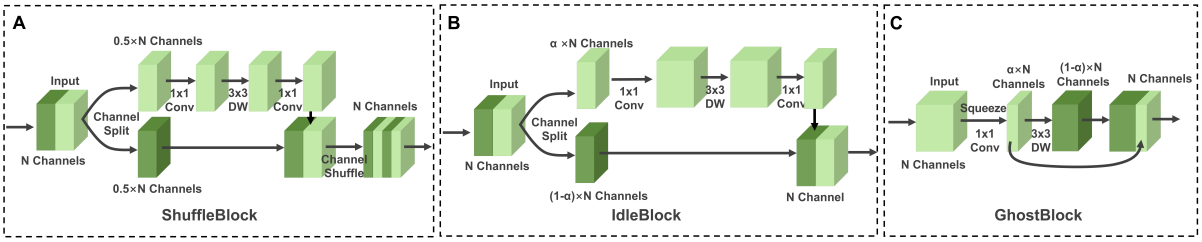

Fig. 1. Schematics showing several lightweight residual blocks from prior work: (A) ShuffleBlock [27], (B) IdleBlock [47], and (C) GhostBlock [14].

type of lightweight residual block for image classification and achieved the state-of-the-art accuracy against other mobile-specific neural networks.

2.3.3 GhostBlock. CNNs often contain redundancy in their feature maps. Calling these redundancies ghost features, Han et al. [14] recognized this inefficiency as an opportunity for optimization. They proposed the notion of a GhostBlock (Figure 1C) for generating redundant feature maps with cheap linear operations rather than ordinary convolutions. Within a GhostBlock, an ordinary convolution kernel is applied to generate a few intrinsic feature maps. Linear operations are used to generate redundant feature maps, which are then stacked with the intrinsic feature maps to produce the block's output. A pointwise convolution is used in the GhostBlock's first layer, depthwise convolutions are used for cheap linear operations.

We draw inspiration from this prior work to propose the SplitSRBlock, which improves performance by applying channel-splitting to standard 2D convolutions. We compare the SplitSRBlock against the aforementioned lightweight residual blocks in our SplitSR architecture, making us the first to evaluate these blocks in the context of image SR.

3 METHOD

In the following sections, we first describe the design of a novel lightweight residual block called the SplitSRBlock. We then describe how such blocks can be used in a novel hybrid architecture called SplitSR, which mixes standard convolutional blocks with lightweight residual blocks to enable on-device SR.

3.1 SplitSRBlock Design

Channel-wise attention has been used to help SR networks like RCAN [51] focus on informative features (i.e., channels) to boost accuracy. Because some channels are generally more important than others, the success of channel-wise attention has shown that not every channel is essential to generating the final high-resolution output. Moreover, channel-splitting can act like channel-attention to provide more complex weights on certain channels through back-propagation. Therefore, we propose a lightweight residual block design that uses channel-splitting to reduce computation, while maintaining high-quality spatial transformation from standard 2D convolutions to achieve high accuracy. The structure of SplitSRBlock is inspired by that of previously proposed lightweight residual blocks (Figure 1), which can be summarized in three stages. First, a tensor with N channels is split along the channel axis into two branches according to parameter α. Second, the branch with $\alpha \times N$ channels is used to perform either depthwise or pointwise convolutions while the other branch is left idle. When α=0.5, for instance, one branch uses half of the channels to conduct computation, and the other branch passes the remaining channels through the network. Finally, the two branches are concatenated to construct the block's output such that it has

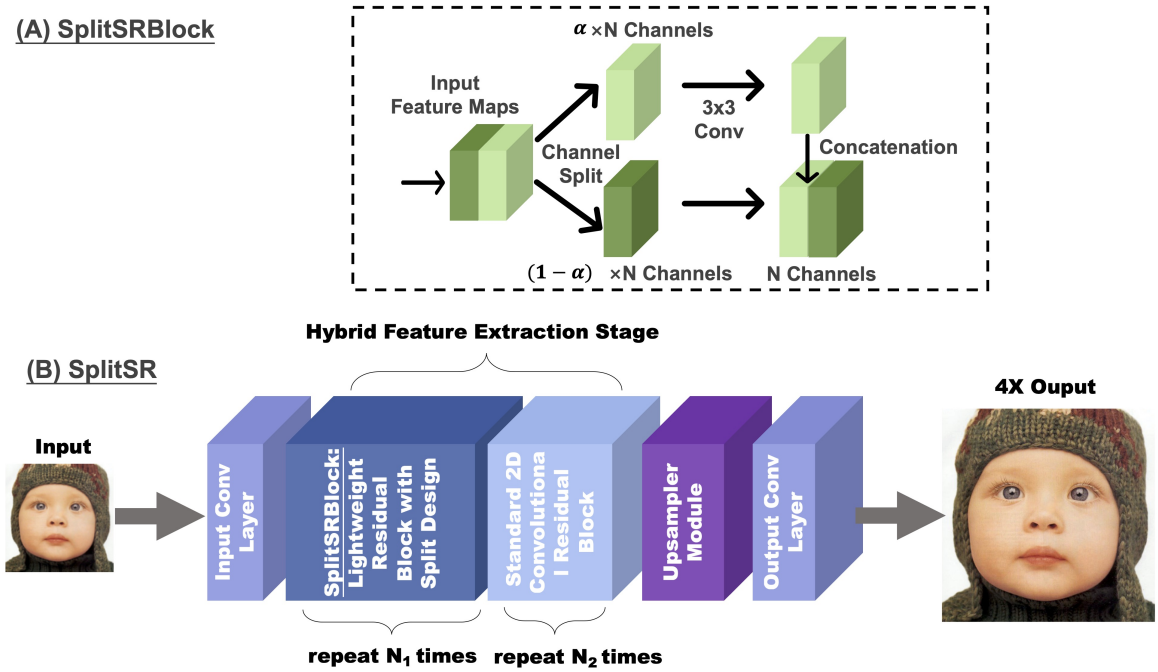

Fig. 2. (A) SplitSRBlock is a lightweight residual block based on standard 2D convolution to retain the quality of spatial transformation and reduces the computational cost by channel-split. (B) SplitSR is a hybrid architecture that leverages a combination of lightweight residual blocks (e.g., SplitSRBlock) and standard 2D convolutions so that people can optimize the tradeoff between latency and performance to enable SR on a constrained computational budget.

the same number of channels as the input. As α decreases, more layers can be added while keeping the same number of total network parameters. This structure is both computationally efficient and able to achieve high accuracy in SR because it is able to learn enough information about the image using only a subset of the input channels and backpropagation.

Existing lightweight residual blocks that leverage channel-splitting are currently designed for discriminative computer vision tasks like image classification. SR, however, is a generative task that usually requires significant fine-grained spatial information. When a small number of channels are involved in the computation after the channel-splitting stage, depthwise and pointwise convolutions sacrifice spatial information with only a small benefit in computational efficiency. Therefore, we propose the SplitSRBlock (Figure 2-A) specifically for efficient SR. Instead of depthwise and pointwise convolutions, the SplitSRBlock uses standard 2D convolution after channel-splitting to retain spatial transformation. Standard convolutions preserve spatial information by having receptive fields, and the computation requirements for such operations are significantly diminished after channel-splitting. As with the other lightweight residual blocks, the results of the two branches (processed and unprocessed) are concatenated at the end to retain the same number of output channels. However, the SplitSRBlock reverses the direction of concatenation at the output so that the first $\alpha \times N$ channels (used for convolution) become the last $\alpha \times N$ channels of the output. By doing this, every single channel will be involved in a computation after $\frac{1}{\alpha}$ blocks. The theoretical computation reduction that can be obtained by using SplitSR is α^2, where $\alpha \in (0, 1]$.

3.2 SplitSR Architecture Design

Most modern SR networks share a similar four-stage structure [6, 24, 51]: (1) an input layer like a single 2D convolutional layer, (2) a feature extraction stage with a large number of 2D convolutional layers to extract a high-dimensional representation of the image, (3) an upsampler stage made up of convolutional and pixel-shuffle layers to rearrange information along the depth dimension into blocks of spatial information, and (4) an output layer that generates the final upscaled image. Past attempts to make SR networks efficient enough to be run on mobile devices have involved replacing all of the standard convolution blocks with lightweight residual blocks [23]. Modern SR architectures usually have a large number of standard convolutional blocks, but the majority of them are clustered at the feature extraction stage. Each stage in a standard SR architecture serves a different purpose, so although naïvely replacing standard convolution blocks at all locations increases efficiency, it often negatively impacts accuracy.

We introduce a hybrid architecture called SplitSR to help balance the tradeoff between accuracy and latency for on-device SR (Figure 2-B). SplitSR is based on the RCAN model [51], although our hybrid approach could be applied to any other state-of-the-art SR model. RCAN uses 64 feature maps (output channels) in its convolutional layers and over 400 layers to achieve high accuracy. SplitSR modifies RCAN by leveraging a combination of standard convolutional layers and lightweight residual blocks. Lightweight residual blocks with channel-splitting (e.g., SplitSRBlock) can only be used at the feature extraction and upsampler stages since the input and output stages change the number of channels. As more lightweight residual blocks are introduced, SplitSR becomes more efficient at the cost of sacrificing model complexity and degrees of freedom. With this framing in mind, we introduce four parameters as potential design decisions for our proposed SR architecture:

(1) **Channel-Split Ratio:** the fraction of channels that are used for computation within each block
(2) **Hybrid Index:** the number of standard convolutional blocks that are replaced by lightweight residual blocks
(3) **Hybrid Mode:** the ordering of standard convolutional blocks and lightweight residual blocks (e.g., standard-lightweight, lightweight-standard)
(4) **Replacement Location:** the stages where standard convolutional blocks are replaced (e.g., feature extraction, upsampler)

4 PERFORMANCE EVALUATION & RESULTS

In this section, we describe the experiments we conducted to evaluate the effects of the various hyperparameters in our SplitSR architecture. We also measure the performance benefits provided by our SplitSRBlock for SR tasks compared to other block designs. To conclude this section, we compare SplitSR against a state-of-the-art on-device SR model, MobiSR [23].

4.1 Implementation

One of the challenges in deploying on-device deep learning models is the gap between high-level graph optimization and low-level operator optimization on the target platform. Unfortunately, very few hand optimization tools are designed to handle highly customized neural network architectures. Even after an architecture has been hand-optimized for a target device, moving that architecture to a new platform, or even changing input shapes, can require a complete rewrite of the optimized kernels. However, recent deep learning compilers like TVM [4] and Halide [33] have emerged to simplify the optimization process. These compilers separate functional definitions from scheduling (i.e., how the code should be executed). This separation allows developers to quickly explore a schedule space without rewriting the kernel at each step.

In this work, we extended TVM to support the operations required for the SplitSRBlock and the other three residual blocks described in the Section 2.3. Our TVM-based on-device SR system directly converts a TensorFlow

graph to a Relay representation [37] and compiles that code to multiple target devices using LLVM. We utilize TVM's scheduling primitives for core operations in our proposed SR architecture (e.g., pixel shuffle). The scheduling primitives we apply are summarized as follow:

- **Tiling:** Splits the entire computation process into small blocks to better reuse data and utilize the cache. As a result, tiling improves computational efficiency and reduces memory traffic.
- **Packing:** Reformats the input tensor based on the tiled configuration to utilize memory more efficiently and reduce the cache misses.
- **Vectorization:** Takes advantage of single instruction, multiple data (SIMD) instructions in the target hardware to enable simultaneous multi-operator computation.
- **Unrolling:** Diminishes branch penalties and reduces latency while accessing memory and removing control flow operations.

Since we have implemented the operators following TVM's convention, we can utilize TVM's built-in graph optimization processes. These include dead code elimination, common sub-expression elimination, constant folding, and operator fusion.

4.2 Experimental Setup

We deploy SplitSR on an open-source embedded system called the Firefly-RK3399[1]. The board is powered by dual 1.8 GHz clusters: one with two large Cortex-A72 cores and another with four small Cortex-A53 cores. The RK3399 also has a Mali-T860 MAP4 mobile GPU, but our work focus on evaluating the performance on CPU. By focusing on CPUs, the SplitSR system can generalize to any ARM-based CPU mobile engine, which are common across contemporary mobile devices. Furthermore, GPUs can have unique architectures that are not available on other computing platforms (e.g., Raspberry Pi, low-end smartphones). We use single-precision floating-point arithmetic (FP32) as the precision of our models during evaluation.

We implemented our system in TensorFlow [1] and TVM [4]. During training, 96×96 cropped high-resolution images and their low-resolution counterparts were pulled from the DIV2K dataset [41] in batches of 16. Random horizontal flipping and random rotation at 90°-intervals were used for data augmentation. The Adam optimizer [20] was used to minimize L1 loss with the following parameters: initial learning rate = 1e−4, β_1 = 0.9, β_2 = 0.999 and ϵ = 1e−7. Networks were trained for 6×10^5 steps with learning decay by a factor of 2 every 2×10^5 iterations.

In this work, we focus on models that increase resolution by ×4. To expedite model convergence, we first pre-train the model by training it to increase resolution by ×2 for 30×10^5 iterations. This training regiment has been widely used in past SR work [23, 24, 51].

4.3 Evaluation Procedure and Metrics

To assess the performance of the models we deployed, we report both speed and accuracy as evaluation metrics. For speed, we ran each model 10 times and report averaged inference latency in milliseconds. For accuracy, we fixed random seed in training and report the standard metrics for SR image quality: peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) [46]. PSNR is computed on the Y channel of the YCbCr color space, which corresponds to image luminance. **PSNR is a logarithmic metric; hence, seemingly incremental differences are deceptively significant**. SSIM is a normalized metric (range: 0–1) for assessing the perceptual similarity between two images. We conduct our experiments on four datasets that are commonly used for SR model evaluations: Set5 [3], Set14 [50], B100 [43], and Urban100 [16]. These datasets contain 5, 14, 100, and 100 images, respectively.

[1] http://en.t-firefly.com/product/rk3399.html

4.4 Reference Model Configurations

We applied our hybrid design and SplitSRBlock to the RCAN [51] architecture, a state-of-the-art SR model. RCAN includes blocks of convolutional operations, which are clustered together to form residual groups with a short residual connection. By introducing channel-splitting, our approach enabled us to significantly extend depth of a SR network while maintaining the same inference latency. Because of RCAN's structure, which uses clusters of blocks as groups, we adapt our definition of hybrid index (HI) to refer to groups rather than blocks. We removed RCAN's channel attention layer because it is only beneficial when the number of channels in each convolution is high. Using our split design reduces the number of channels that are processed within each block to $\alpha \times N$, making the channel attention layer less necessary. Moreover, channel-splitting in the SplitSRBlock provides some degree of attention to certain channels (a subset of feature maps that go through convolution).

Since SplitSR is an architecture that can be tuned for computational budget, we created two versions of our model:

- **Accuracy-focused:** 7 residual groups each with 7 residual blocks
- **Latency-focused:** 5 residual groups each with 6 residual blocks

The accuracy-focused network configuration was meant to be directly comparable to MobiSR's reference model. The latency-focused model was more suitable for mobile applications given its shallowness and lower operation count. We specifically designed the latency-focused configuration to have an average inference time around 500 ms, which we thought would demonstrate a better user experience and would be more accessible for low-end smartphones and more applicable to more real-world application. It is also worth noting that these model configurations represent two instantiations of the accuracy-latency tradeoff. The number of residual groups and residual blocks per group can be tuned depending on the computational budget and how much accuracy is valued over latency for a target application. The same can be applied for the number of feature maps (i.e., convolution channels) in the models.

We used MobiSR [23] as a competitive benchmark since it also uses an RCAN-based design. SplitSR is able to hold more residual blocks than MobiSR for the same computational budget because of the channel-splitting provided by the SplitSRBlock. All of the models in our experiments had 16 feature maps to remove that hyperparameter as a possible confound.

4.5 Hyperparameter Selection for SplitSR

SplitSR has many hyperparameters that can be adjusted to balance accuracy and performance. Below, we describe the effects of some of these hyperparameters on our latency-focused model.

4.5.1 Channel-Split Ratio. Table 1 shows the performance implications of different values for the channel-split ratio α. As α increases, more channels are used for computation. This creates a more complex model that is able to generate a higher-order representation of the data, leading to higher accuracy. However, increasing α also leads to higher latency. The latency when $\alpha = 0.25$ is very close to the latency when $\alpha = 0.125$ while yielding better PSNR for all datasets. Although $\alpha = 0.5$ and $\alpha = 1$ provide better accuracy, they take much longer for inference. To establish a good trade-off between latency and accuracy, we use α=0.25 for remainder of our evaluation.

4.5.2 Hybrid Index. Table 2 shows the performance of different values for the hybrid index (HI). When HI increases, more standard 2D convolution groups are replaced with lightweight residual groups, resulting in both lower latency and accuracy; nevertheless, mixing standard and lightweight convolutions can lead to a sweet spot for accuracy and latency. The accuracy gap between HI=3 and HI=4 is substantially larger than that between HI=2 and HI=3. Meanwhile, the latency difference between those pairs is practically equivalent. For these reasons, we use HI=3 for future evaluation.

Table 1. The performance comparison for different settings for the channel-split ratio α on our latency-focused SplitSR model (HI = 3, hybrid mode = front, replacement location = feature extraction).

			Average PSNR (dB) / SSIM			
Alpha	**Params**	**Latency (ms)**	**Set5**	**Set14**	**B100**	**Urban100**
α = 0.125	90k	603	31.46 / 0.8942	28.12 / 0.7876	27.25 / 0.7451	25.14 / 0.7686
α = 0.250	94k	629	31.53 / 0.8950	28.18 / 0.7887	27.28 / 0.7458	25.20 / 0.7704
α = 0.500	110k	725	31.57 / 0.8958	28.20 / 0.7897	27.29 / 0.7467	25.24 / 0.7724
α = 1.000	172k	856	31.75 / 0.8980	28.30 / 0.7918	27.36 / 0.7486	25.41 / 0.7787

Table 2. The performance comparison for different settings for the hybrid index HI on our latency-focused SplitSR model (α = 0.25, hybrid mode = front, replacement location = feature extraction).

			Average PSNR (dB) / SSIM			
Hybrid Index (HI)	**Params**	**Latency (ms)**	**Set5**	**Set14**	**B100**	**Urban100**
HI = 2	120k	706	31.61 / 0.8960	28.22 / 0.7900	27.31 / 0.7472	25.28 / 0.7739
HI = 3	94k	629	31.53 / 0.8950	28.18 / 0.7887	27.28 / 0.7458	25.20 / 0.7704
HI = 4	68k	555	31.33 / 0.8920	28.05 / 0.7856	27.19 / 0.7432	25.02 / 0.7633

Table 3. The performance comparison for three different hybrid modes — front, end, and mixed — on our latency-focused SplitSR model (α = 0.25, HI = 3, replacement location = feature extraction).

			Average PSNR (dB) / SSIM			
Hybrid Mode	**Params**	**Latency (ms)**	**Set5**	**Set14**	**B100**	**Urban100**
Front	94k	629	31.53 / 0.8950	28.18 / 0.7887	27.28 / 0.7458	25.20 / 0.7704
End	94k	629	31.47 / 0.8943	28.12 / 0.7876	27.25 / 0.7451	25.15 / 0.7688
Mixed	94k	629	31.46 / 0.8945	28.14 / 0.7880	27.25 / 0.7450	25.15 / 0.7688

Table 4. The performance comparison for different lightweight block placements on our latency-focused SplitSR model (α = 0.25, HI = 3, hybrid mode = front).

			Average PSNR (dB) / SSIM			
Replacement Location	**Params**	**Latency (ms)**	**Set5**	**Set14**	**B100**	**Urban100**
Feature Extraction (FE)	94k	629	31.53 / 0.8950	28.18 / 0.7887	27.28 / 0.7458	25.20 / 0.7704
FE + Upsampling	80k	504	31.45 / 0.8937	28.11 / 0.7874	27.24 / 0.7447	25.11 / 0.7675
Throughout	67k	468	31.10 /0.8877	27.91 / 0.7827	27.10 / 0.7403	24.84 / 0.7563

4.5.3 Hybrid Mode. Table 3 shows the performance implications of different hybrid modes. We examined three different hybrid modes:

(1) **Front:** The first HI residual groups are replaced
(2) **End:** The last HI residual groups are replaced
(3) **Mixed:** HI residual groups are replaced with a regular spacing

Because HI is fixed in this experiment (HI=3), the model complexity remains the same; only the order of operations changes across the different hybrid modes. Therefore, the inference latency is identical across all three hybrid modes. The "front" configuration outperforms the other two modes in regards to accuracy, so we use that setting for future evaluation.

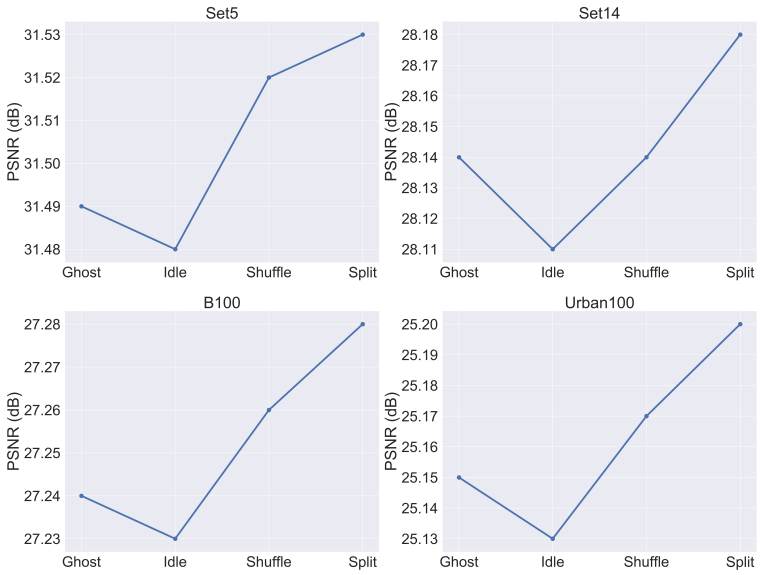

Fig. 3. The effect that residual block selection has on PSNR for (top-left) Set5 [3], (top-right) Set14 [50], (bottom-left) B100 [43], (bottom-right) and Urban100 [16].

4.5.4 Replacement Location. Table 4 shows the performance implications of where the lightweight residual blocks are used to replace standard convolutional blocks within the RCAN architecture. As discussed in the Section 3.2, the two major stages where it makes the most sense to utilize lightweight residual blocks are the feature extraction and upsampling stages. However, standard convolutional blocks can also be replaced at the output of each residual group and at the output of the feature extraction stage. We investigate the following replacement strategies:

(1) **Feature extraction only:** Convolutions are replaced only in the feature extraction stage
(2) **Feature extraction + upsampling:** Convolutions are replaced in both the feature extraction and upsampling stages
(3) **Throughout:** Convolutions are replaced in the feature extraction stage, the upsampling stage, at the end of each residual group, and the output of the feature extraction stage

Additional strategies could have included replacing just the convolutions in the upsampling stage or the other non-major stages, but those stages are much smaller than the main feature extraction stage and would therefore not significantly impact the network's performance. We find that that replacing convolutions outside of the feature extraction stage deteriorates accuracy without commensurate latency benefits, so we only replace convolutions in the feature extraction stage for future investigation.

4.5.5 Lightweight Residual Block Type. We compare the performance engendered by four lightweight residual block designs: he ShuffleBlock [27], the IdleBlock [47], the GhostBlock [14], and our proposed SplitSRBlock. Figure 4 illustrates the performance RCAN achieves with the various lightweight residual blocks for the four validation datasets. SplitSRBlock outperforms the other blocks in terms of both latency and accuracy. The ShuffleBlock is the clear runner-up in regards to both metrics. The IdleBlock results in the worst accuracy, whereas the GhostBlock produces the worst latency. Both the IdleBlock and the GhostBlock expand feature maps

Table 5. The comparison between the results reported in the MobiSR paper [23] and SplitSR deployed on TVM [4].

			Average PSNR(dB) / SSIM			
Model	**Params**	**Latency (ms)**	**Set5**	**Set14**	**B100**	**Urban100**
MobiSR	152k	4570	31.73 / 0.8873	28.24 / 0.7729	27.33 / 0.7283	25.34 / 0.7610
MobiSR (w/ shuffle)	17k	1023	N/A	N/A	N/A	24.57 / N/A
MobiSR (dual-model)	182k	1426*	31.40 / N/A	28.10 / N/A	27.30 / N/A	25.30 / N/A
Bilinear	N/A	N/A	27.56 / 0.80	25.51 / 0.69	25.54 / 0.66	22.69 / 0.65
SplitSR (accuracy)	174k	947	31.76 / 0.8982	28.29 / 0.7916	27.39 / 0.7491	25.46 / 0.7795
SplitSR (latency)	94k	629	31.53 / 0.8950	28.18 / 0.7887	27.28 / 0.7458	25.20 / 0.7704

*This experiment required a DSP and a CPU.

close to their input, whereas the ShuffleBlcok and SplitSRBlock do not. Our empirical results show that expansion does not lend to improved SR performance.

Figure 3 expands on the accuracy of the various models across the validation datasets. Our proposed SplitSRBlock surpasses the other three residual blocks across all the datasets, and each dataset exhibits the same ranking for the different blocks. These findings validate our hypothesis that utilizing the split design with conventional 2D convolution achieve a superior balance in accuracy and speed.

4.6 Overall Performance of SplitSR

We also compare SplitSR against the current state-of-the-art mobile SR system, MobiSR [23]. In that work, Lee et al. evaluate three different configurations:

(1) **MobiSR:** This is the most standard network with normal convolutions, but fewer layers so that their model can operate on a mobile device
(2) **MobiSR w/ Shuffle:** In this model, standard convolutional blocks are replaced with ShuffleBlocks since those were determined to be the best performing lightweight residual block in Lee et al.'s experiments.
(3) **MobiSR Dual-Model:** In this model, MobiSR is split into two models that are run on heterogeneous mobile processors, integrating the CPU, GPU, and digital signal processors (DSP). A heavy model is run on the DSP while a lightweight model is run on the CPU/GPU. To utilize these models, MobiSR splits a raw image into small patches (90×160) and feeds them into an load-balancing unit that assesses their difficulty; 'hard' patches are sent to the DSP and 'easy' patches are sent to the CPU/GPU for SR inference.

To test the performance of MobiSR, Lee et al. used a DSP and an Octa-core CPU with four large customized ARM A75 cores and four A55 cores with a clock speed up to 2.8GHz (Qualcomm Snapdragon 845 SoC[2]). In contrast, we evaluate SplitSR on a CPU with four A72 cores and two A53 cores; the clock speed of the cores can go only go up to 1.8 GHz, comparable to the much older Qualcomm Snapdragon 650[3].

Table 5 shows the performance of two SplitSR configurations (latency- and accuracy-focused) against the aforementioned MobiSR configurations. We also include the performance of bilinear interpolation, which is the existing standard for on-device SR. Note that Lee et al. did not evaluate all three of their configurations on all possible datasets; thus leaving gaps in our comparison. We found that SplitSR is 4.78× faster and achieves higher accuracy on every validation dataset compared to the standard MobiSR model using single processor. When we compare our latency-focused model against MobiSR with shuffle, we demonstrate superior accuracy in the Urban100 dataset as well as 33% acceleration in CPU latency. When we compare SplitSR against MobiSR's dual-model configuration, our accuracy-based model not only achieves higher accuracy but also a ~34% speedup.

[2]https://www.qualcomm.com/products/snapdragon-845-mobile-platform
[3]https://www.qualcomm.com/products/snapdragon-650-mobile-platform

Fig. 4. The latency and accuracy of our latency-focused SplitSR model across four datasets: Set5 [3], Set14 [50], B100 [43], and Urban100 [16].

In summary, SplitSR achieves better accuracy and an inference speedup of up to 4.8× despite the fact that MobiSR uses higher performance hardware and an intricate system to maximize performance. By leveraging a compiler dedicated to deep learning for generative vision tasks like SR, our results suggest that people should integrate lightweight residual blocks along with a deep learning compiler into their heavy mobile computing system to gain substantial speedup. Moreover, our proposed SplitSRBlock exhibits superior performance against alternatives proposed by prior work, showing that depthwise separable convolution is not always necessary for new lightweight residual blocks.

5 USER STUDY

To contextualize the results of our offline evaluation, we took a step further to deploy SplitSR on a smartphone and investigate whether individuals can notice the difference in image quality provided by SplitSR over the existing standard of bilinear interpolation. We first describe the way we implemented SplitSR as part of a standard image gallery app called ZoomSR to provide as seamless of an experience as possible. We then describe the two tasks that participants were asked to perform and the associated findings.

5.1 ZoomSR Design

For our study, we developed ZoomSR as an image gallery app for viewing images saved on the smartphone. We deployed this app on a Google Pixel 4, which has a 5.7-inch OLED touch screen with a resolution of 2280×1080 and a Qualcomm Snapdragon 855 processor. Images that are loaded in ZoomSR are displayed in a 1742×1080-pixel area, with the rest of the screen being reserved for instructions and interface controls. Whenever the user loads an image onto the screen, ZoomSR randomly selects either SplitSR or bilinear interpolation for upsampling during zoom. Using a button at the bottom of the screen, the user can see the same image post-processed by the alternate SR method.

Most images that people view on their smartphones nowadays have a larger resolution than the images in the datasets that we used to evaluate our technique, and processing larger images incurs longer delays. Bilinear interpolation is able to process such images with relative ease, but we had to intelligently utilize SplitSR to achieve tolerable processing times. Within ZoomSR, we use the latency-focused SplitSR model to upsample

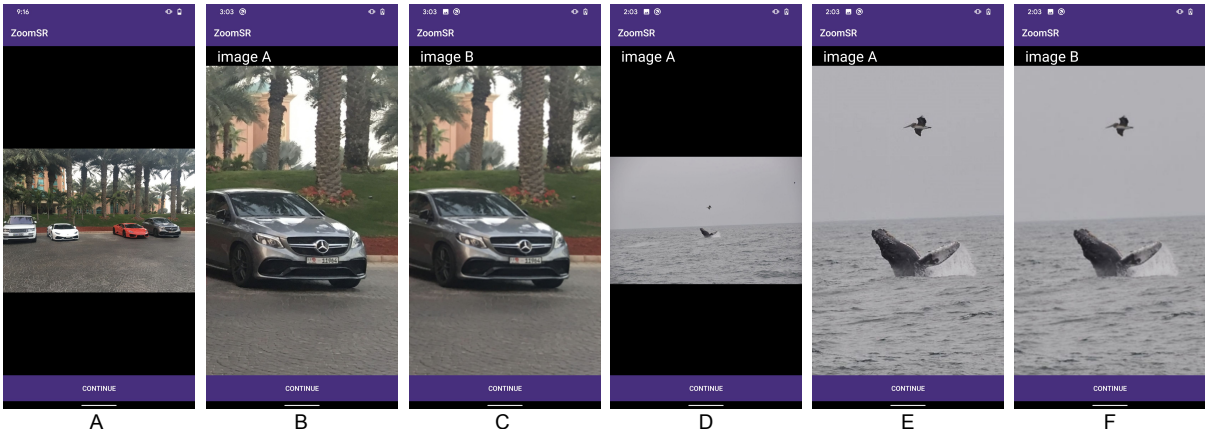

Fig. 5. Screenshots taken from ZoomSR for the image task: (A–C) an image of cars and (D–F) an image of animals. The screenshots show the (A, D) original images, (B, E) images after upsampling with SplitSR, and (C, F) images after upsampling with bilinear interpolation. Note that the presentation order of SplitSR and bilinear interpolation was randomized to mitigate ordering effects.

256×256 px patches to 4×. If the user's desired zoom level is below 2×, ZoomSR simply uses bilinear upsampling to process the patches. If the desired zoom level is between 2–4×, ZoomSR pushes patches through the model to upsampling them by a factor of 4× and then downsamples the results to the desired resolution. If the desired zoom level is greater than 4×, ZoomSR still pushes the patches through the SplitSR model, but then applies bilinear interpolation to the results to upsample further. We limit the maximum zoom level to 5× for our study. We prioritize processing the patches closest to where the user initiates their pinch-to-zoom gesture to ensure that the most important parts of the image are done first. Patches outside of the user's final view of the image are still processed since users should still be allowed to explore the rest of the image during or after a zoom gesture.

5.2 Protocol

We conducted a user study with 15 participants (8 male, 7 female) to assess the users' perception of the image quality engendered by SplitSR. The study was split into two parts — an image task and a reading task — since people can fixate on different characteristics of objects and text, respectively.

For the image task, participants were asked to sequentially examine 10 randomly ordered images of subjects like animals, people, and nature (see Figure 5). Participants were free to pan and zoom the image as they pleased. When they were done examining the same image using the two different SR methods — SplitSR and bilinear interpolation — participants were asked to rate the quality of each version of the image. Participants were asked to rate the images according to how they were rendered (e.g., resolution, focus, blurriness) rather than their content. The ratings were made along a 7-point Likert scale (1: lowest quality, 7: highest quality).

For the reading task, participants were asked to sequentially examine 10 images of text that was photographed on a distant blackboard (see Figure 6). The blackboard was 2 meters away and the text had roughly 60–70 characters, so participants had to zoom-in to resolve the characters. The text was also placed at a different location on the blackboard each time to prevent participants from learning the same zooming pattern over time. Participants were asked to read text and to rate the ease with which they were able to do it in along a 7-point Likert scale (1: very easy, 7: very hard). Unlike with the previous task, participants only saw each image with

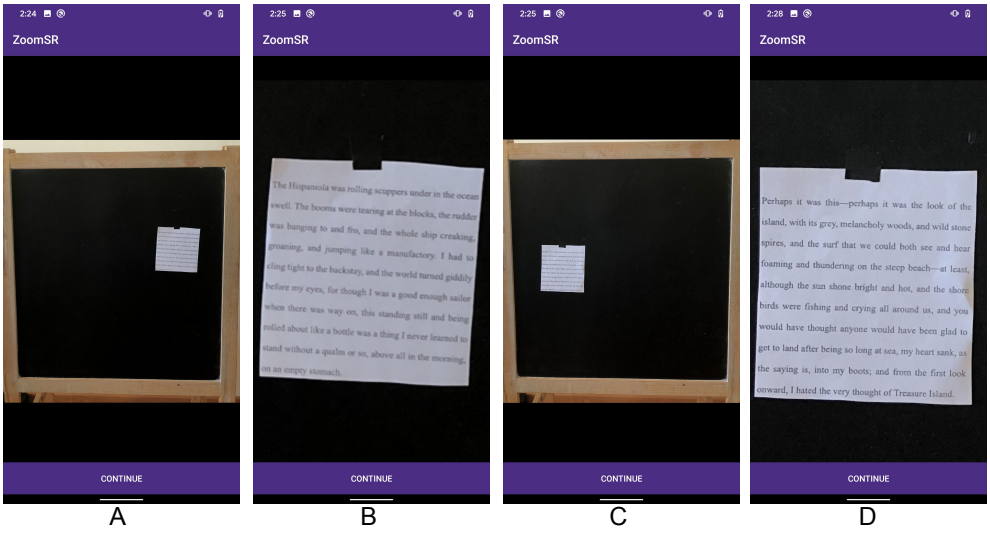

Fig. 6. Screenshots taken from ZoomSR for the reading task. The screenshots show the (A, C) original images, (B) an image after upsampling with bilinear interpolation, and (D) an image after upsampling with SplitSR. Note that the presentation order of SplitSR and bilinear interpolation was randomized to mitigate ordering effects.

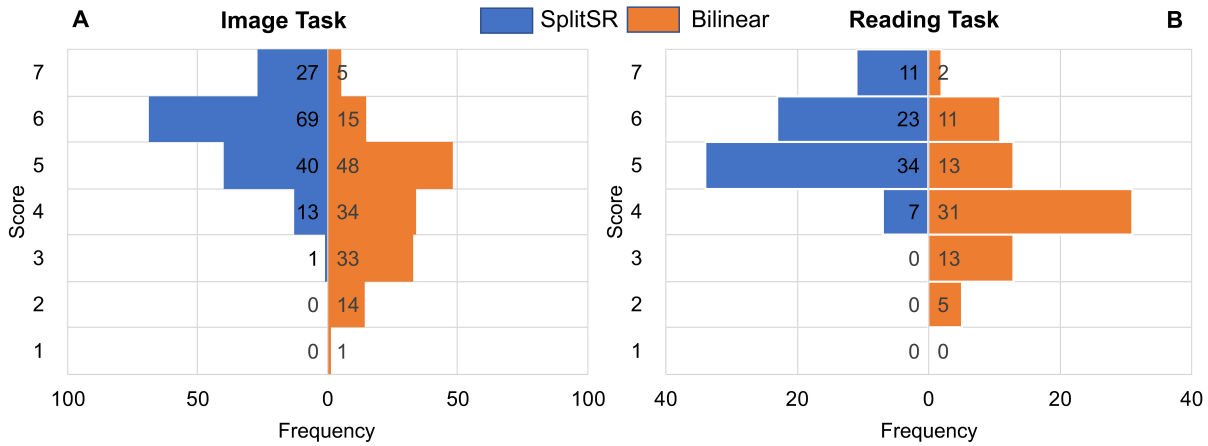

Fig. 7. The distributions of Likert scores in the (left) image and (right) reading tasks.

either SplitSR or bilinear interpolation (evenly distributed) since seeing the same image twice would have made the latter far easier to read.

5.3 Results

Participants in both studies did not report noticing the additional latency incurred by SplitSR versus bilinear interpolation. Pinch-to-zoom gestures take roughly 1000 ms on average, and the processing time of the latency-focused SplitSR configuration is takes 629 ms on average. ZoomSR starts upsampling images as soon as a gesture

Table 6. The performance comparison between MobiSR and SplitSR when both are implemented with TVM [4].

			Average PSNR(dB) / SSIM			
Model	**Params**	**Latency (ms)**	**Set5**	**Set14**	**B100**	**Urban100**
MobiSR (w/ TVM)	172K	941	31.73/0.8976	28.25/0.7908	27.33/0.7481	25.39/0.7780
SplitSR (accuracy)	174K	947	**31.76/0.8982**	**28.29/0.7916**	**27.39/0.7491**	**25.46/0.7795**
Light MobiSR (w/ TVM)	101K	626	31.38/0.8923	28.07/0.7860	27.21/0.7439	25.07/0.7655
SplitSR (latency)	94K	629	**31.53/0.8950**	**28.18/0.7887**	**27.28/0.7458**	**25.20/0.7704**

is initiated, which means that SplitSR typically finished processing images before people completed their gestures and returned to examining the images' details.

5.3.1 Part 1: Image Task. Figure 7A summarizes the distribution of Likert scores for the image task. A statistically significant difference was found between the Likert scores for bilinear interpolation and SplitSR ($Z = -9.270, p < 0.01$) according to the Wilcoxon signed-rank test. The median score earned by SplitSR was 2 points higher than that of bilinear interpolation (6 vs. 4). This result was upheld in a random effects model that accounted for the image ID, participant ID, and presentation order of the different conditions ($F = 285, p < 0.01$). Across the 150 pairwise comparisons (15 participants × 10 images), participants perceived that SplitSR improved image quality over bilinear interpolation for 85.3% of the images. Participants only preferred the images from bilinear interpolation 4.7% of the time, and the remaining cases has no perceptible difference according to participants.

5.3.2 Part 2: Reading Task. For this analysis, we use the Mann-Whitney U test to compare Likert scores since the measurements were unpaired. Figure 7B demostrates the distribution of Likert score of the reading task. As before, a statistically significant difference was found in favor of SplitSR ($Z = -6.486, p < 0.01$). The median Likert score for SplitSR was 1 points higher than that of bilinear interpolation (5 vs. 4). This results was upheld in a random effects model that accounted for the image ID, participant ID, and presentation order of the different conditions ($F = 155, p < 0.01$).

6 DISCUSSION & LIMITATIONS

In this work, we demonstrate the first instance of deep learning-based SR on a mobile device using our SplitSR architecture. The channel-split design of our SplitSRBlock made this goal tenable by yielding superior accuracy over prior approaches while achieving faster inference speed. These innovations not only made it possible to perform on-device SR, but also improved image quality by a noticable amount for end-users. In this section, we describe potential avenues of future work and the limitations of our findings.

6.1 Importance of TVM

As noted in Section 4.1, one of the challenges in deploying deep learning models onto platforms like mobile devices is the gap between high-level graph optimization and low-level operator optimization on the target platform. TVM addresses this issue by automatically generating efficient machine-readable code for deep learning models, which is why we use it to develop SplitSR and ZoomSR. In order to quantify the performance improvement that can be attributed to our integration with TVM, we also deployed SplitSR using the desktop-based Tensorflow without TVM. Note that the desktop-based TensorFlow is not designed for ARM-based platforms, and we were unable to use TensorFlow Lite since it does not support some of the operations we needed to implement SplitSR. The accuracy-focused configuration of SplitSR takes an average of 2560 ms for inference across all the benchmark datasets and the latency-focused configuration takes 1802 ms, highlighting that optimization was critical for on-device deployment. These results do not invalidate the results presented earlier since prior literature has also leveraged deep learning optimization to achieve their results. Our performance baseline, MobiSR [23], used

the Snapdragon Neural Processing Engine to optimize operations on their deep learning architecture. MobiSR also uses the Snapdragon 845, which has superior hardware specifications to the Firefly-RK3399 we used in our evaluations.

We felt that comparing SplitSR with TVM against the results presented in the MobiSR paper (with its own optimization) was the most honest comparison that could be made since MobiSR's source code was not available and we were unable to reproduce their results. Nevertheless, we conducted an additional evaluation comparing SplitSR and MobiSR with standardized hardware and TVM as the compiler (Table 6). To produce comparable results to those shown in the MobiSR paper, we deployed their single-model architecture with a non-reduced upsampler module, adding an additional 20k parameters out of 170k. Comparing this model to the accuracy-focused SplitSR model, we find that SplitSR outperforms MobiSR by 0.05 dB on average across all four validation datasets while achieving similar inference latency. We also compare the latency-focused SplitSR model against a lightweight version of MobiSR with 6 residual blocks in each group instead of 10 residual blocks. We find that SplitSR outperforms MobiSR in this evaluation as well, providing a significant accuracy boost of 0.13 dB on the Urban100 dataset.

6.2 Future Applications of SplitSR

In this paper, we based our model architecture on RCAN [51], but our hybrid architecture and use of channel-splitting can be extended to other modern SR architectures. We also hypothesize there may be an opportunity to apply SplitSRBlock to the comparable task of on-device audio SR [22], which increases the resolution of an audio spectrogram to achieve higher quality audio. In general, we encourage researchers to explore new ways of integrating our SplitSRBlock in places where they would normally use standard convolution to support on-device computation on different hardware platforms.

We foresee many potential applications that could benefit from on-device SR beyond our ZoomSR app. Studies have shown that SR can play a major role in medical imaging [13, 35]. As medical devices become more ubiquitous, particularly in low- and middle-income regions, the ability to run clinical assessments without requiring a data connection will become increasingly important. Mobile health would also benefit from SR. For example, Liu et al. [25] recently demonstrated that smartphones can be used for remote physiological testing (i.e., photoplethysmography), and McDuff [29] showed that SR improves the accuracy of such techniques. Another application that would benefit from SR is PupilScreen [28], a screening tool for traumatic brain injuries that utilizes a smartphone's camera measure the pupillary light reflex. Because the camera is being used to measure the size of a person's pupils, the video resolution has a direct impact on PupilScreen's precision and diagnostic accuracy. Although computations for both of these applications could be done on the cloud, bypassing the need for data upload would help maintain the privacy of protected health information.

6.3 Pinch-to-Zoom Gesture Prediction for Lower Latency

Regardless of a model's performance offline, the way the model is integrated into the user experience has a significant impact on latency (and therefore, perceived performance). For our study app, we used a single SplitSR model to upsample an image by 4× and then adjusted the image based on the final zoom level. This decision was a compromise that had consequences for images that needed to be upsampled at both lower and higher zoom levels. For images that had to be upsampled past 4×, we used a smaller model with quicker latency to ensure that ZoomSR was responsive, but that led to sacrifices in image quality. For images that did not have to be upsampled that high, we achieve the maximum image quality possible, but the model was larger than necessary and thus ZoomSR spent more time processing the image than it would have required had it used a smaller model. These drawbacks can be mitigated by simultaneously running a small set of models that upsample images to varying scales (i.e., 2×, 3×, and 4×). However, this is problematic to do even with our state-of-the-art latency, and the

memory footprint of these models also imposes its own challenges. We believe that the final zoom level can be anticipated in some cases by observing the way a person performs a pinch-to-zoom gesture on the touchscreen, with larger gestures indicating higher upsampling. Knowing this information, a scheduler can prioritize the model with the closest scaling factor to minimize the amount of post-processing required. Investigating such a prediction mechanism and determining the ideal number of models to optimize latency remain as topics for future investigation.

6.4 Limitations

We investigated five hyperparameters in the SplitSR system: channel split ratio (α), hybrid index, hybrid mode, replacement location, and block type. In our experiments, we varied a single hyperparameter while keeping the others constant, thus allowing us to report the effect of each parameter on their own. We hypothesize that a complete grid search across all possible hyperparameters may lead to a more optimal configuration and improved performance, but we suspect that the configuration we presented for our SplitSR system is close to that target. AutoTVM [5], which is made by the same researchers who created TVM, provides the additional functionality of supporting automatic hyperparameter search, which we anticipate would be particularly useful for identifying the optimal patch scheduling primitives (e.g., patch size) to attain even lower latency.

Other limitations of our evaluation lie in the fact that we only tested one deployment configuration for SplitSR: an image gallery app on a Google Pixel 4 with a single model for upsampling images by 4×. We chose to deploy ZoomSR on Android over iOS since TVM is more compatible with Android, and we chose the zoom factor of 4× to have the fairest comparison possible with Lee et al.'s MobiSR work [23]. Nevertheless, SplitSR can be used on a variety of platforms with different configurations, and we leave this exploration to future work.

7 CONCLUSION

In this paper, we have proposed a novel and end-to-end mobile super-resolution system called SplitSR. By introducing a split design into modern SR architectures and leveraging a modern deep learning compiler to perform low-level operator optimization, we have achieved up to 5× inference speedup while outperforming accuracy compare against the previous state-of-the-art systems. We also developed and tested ZoomSR, the first-ever smartphone app for on-device SR using deep learning. We believe our proposed approach will enable many future on-device SR applications, particularly healthcare applications in low-resource settings where cloud connectivity and on-device compute capacity is limited. It is our hope that other researchers apply our findings to this domain and others in the future.

ACKNOWLEDGMENTS

This research is supported by University of Washington Endowment and Gift Funds. We would also like to thank Xinbei Gong for helping edit some of the figures.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283.

[2] Dawn M Becker, Chelsea A Tafoya, Sören L Becker, Grant H Kruger, Matthew J Tafoya, and Torben K Becker. 2016. The use of portable ultrasound devices in low-and middle-income countries: a systematic review of the literature. *Tropical Medicine & International Health* 21, 3 (2016), 294–311. https://doi.org/10.1111/tmi.12657

[3] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie Line Alberi-Morel. 2012. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. (2012). https://doi.org/10.5244/c.26.135

[4] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 578–594.

[5] Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. Learning to optimize tensor programs. In *Advances in Neural Information Processing Systems*. 3389–3400.

[6] Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Lei Zhang. 2019. Second-order attention network for single image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 11065–11074. https://doi.org/10.1109/cvpr.2019.01132

[7] Mallesham Dasari, Arani Bhattacharya, Santiago Vargas, Pranjal Sahu, Aruna Balasubramanian, and Samir R. Das. 2020. Streaming 360-Degree Videos Using Super-Resolution. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. IEEE. https://doi.org/10.1109/infocom41043.2020.9155477

[8] Dmitry Datsenko and Michael Elad. 2007. Example-based single document image super-resolution: a global MAP approach with outlier rejection. *Multidimensional Systems and Signal Processing* 18, 2-3 (2007), 103–121.

[9] 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. https://doi.org/10.1109/tpami.2015.2439281

[10] Chao Dong, Chen Change Loy, and Xiaoou Tang. 2016. Accelerating the super-resolution convolutional neural network. In *European conference on computer vision*. Springer, 391–407. https://doi.org/10.1007/978-3-319-46475-6_25

[11] Zhaoxin Geng, Xiong Zhang, Zhiyuan Fan, Xiaoqing Lv, Yue Su, and Hongda Chen. 2017. Recent progress in optical biosensors based on smartphone platforms. *Sensors* 17, 11 (2017), 2449. https://doi.org/10.3390/s17112449

[12] Nikolaos Georgis, Fredrik Carpio, and Paul Jin Hwang. 2013. Super-resolution digital zoom. US Patent 8,587,696.

[13] Hayit Greenspan. 2009. Super-resolution in medical imaging. *The computer journal* 52, 1 (2009), 43–63.

[14] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. 2020. GhostNet: More Features From Cheap Operations. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. https://doi.org/10.1109/cvpr42600.2020.00165

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

[16] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. 2015. Single image super-resolution from transformed self-exemplars. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5197–5206. https://doi.org/10.1109/cvpr.2015.7299156

[17] Zheng Hui, Xiumei Wang, and Xinbo Gao. 2018. Fast and accurate single image super-resolution via information distillation network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 723–731. https://doi.org/10.1109/cvpr.2018.00082

[18] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. 2016. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1646–1654. https://doi.org/10.1109/cvpr.2016.182

[19] Yongwoo Kim, Jae-Seok Choi, and Munchurl 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* 29, 8 (2018), 2521–2534. https://doi.org/10.1109/tcsvt.2018.2864321

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

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105. https://doi.org/10.1145/3065386

[22] Volodymyr Kuleshov, S Zayd Enam, and Stefano Ermon. 2017. Audio super resolution using neural networks. *arXiv preprint arXiv:1708.00853* (2017).

[23] 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* (Los Cabos, Mexico) *(MobiCom '19)*. Association for Computing Machinery, New York, NY, USA, Article 54, 16 pages. https://doi.org/10.1145/3300061.3345455

[24] 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

[25] Xin Liu, Josh Fromm, Shwetak Patel, and Daniel McDuff. 2020. Multi-Task Temporal Shift Attention Networks for On-Device Contactless Vitals Measurement. *arXiv preprint arXiv:2006.03790* (2020). https://arxiv.org/pdf/2006.03790.pdf

[26] Zhi-Song Liu, Li-Wen Wang, Chu-Tak Li, and Wan-Chi Siu. 2019. Hierarchical back projection network for image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 0–0.

[27] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 116–131. https://doi.org/10.1007/978-3-030-01264-9_8

[28] Alex Mariakakis, Jacob Baudin, Eric Whitmire, Vardhman Mehta, Megan A Banks, Anthony Law, Lynn McGrath, and Shwetak N Patel. 2017. PupilScreen: Using Smartphones to Assess Traumatic Brain Injury. In *Proc. IMWUT '17*, Vol. 1. 81:1—-81:27. https://doi.org/10.1145/3131896

[29] Daniel McDuff. 2018. Deep super resolution for recovering physiological information from videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 1367–1374.
[30] Vitor F Pamplona, Ankit Mohan, Manuel M Oliveira, and Ramesh Raskar. 2010. NETRA: interactive display for estimating refractive errors and focal range. *ACM transactions on graphics (TOG)* 29, 4 (2010), 77. https://doi.org/10.1145/1833349.1778814
[31] Vitor F Pamplona, Erick B Passos, Jan Zizka, Manuel M Oliveira, Everett Lawson, Esteban Clua, and Ramesh Raskar. 2011. Catra: cataract probe with a lightfield display and a snap-on eyepiece for mobile phones. In *Proc. SIGGRAPH '11*. 7–11. http://doi.acm.org/10.1145/1964921.1964942
[32] Ram Krishna Pandey, K Vignesh, AG Ramakrishnan, et al. 2018. Binary document image super resolution for improved readability and OCR performance. *arXiv preprint arXiv:1812.02475* (2018).
[33] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *Acm Sigplan Notices* 48, 6 (2013), 519–530. https://doi.org/10.1145/2499370.2462176
[34] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788. https://doi.org/10.1109/cvpr.2016.91
[35] M Dirk Robinson, Stephanie J Chiu, J Lo, C Toth, J Izatt, and Sina Farsiu. 2010. New applications of super-resolution in medical imaging. *Super-Resolution Imaging* 2010 (2010), 384–412.
[36] M Dirk Robinson, Stephanie J Chiu, Cynthia A Toth, Joseph A Izatt, Joseph Y Lo, and Sina Farsiu. 2017. New applications of super-resolution in medical imaging. In *Super-Resolution Imaging*. CRC Press, 401–430. https://doi.org/10.1201/9781439819319-13
[37] Jared Roesch, Steven Lyubomirsky, Logan Weber, Josh Pollock, Marisa Kirisame, Tianqi Chen, and Zachary Tatlock. 2018. Relay: A new ir for machine learning frameworks. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. 58–68. https://doi.org/10.1145/3211346.3211348
[38] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520. https://doi.org/10.1109/cvpr.2018.00474
[39] Wanjie Sun and Zhenzhong Chen. 2020. Learned image downscaling for upscaling using content adaptive resampler. *IEEE Transactions on Image Processing* 29 (2020), 4027–4040.
[40] Yulung Sung, Fernando Campa, and Wei-Chuan Shih. 2017. Open-source do-it-yourself multi-color fluorescence smartphone microscopy. *Biomedical optics express* 8, 11 (2017), 5075–5086. https://doi.org/10.1364/boe.8.005075
[41] Radu Timofte, Eirikur Agustsson, Luc Van Gool, Ming-Hsuan Yang, and Lei Zhang. 2017. Ntire 2017 challenge on single image super-resolution: Methods and results. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 114–125. https://doi.org/10.1109/cvprw.2017.150
[42] Radu Timofte, Vincent De Smet, and Luc Van Gool. 2013. Anchored neighborhood regression for fast example-based super-resolution. In *Proceedings of the IEEE international conference on computer vision*. 1920–1927. https://doi.org/10.1109/iccv.2013.241
[43] Radu Timofte, Vincent De Smet, and Luc Van Gool. 2014. A+: Adjusted anchored neighborhood regression for fast super-resolution. In *Asian conference on computer vision*. Springer, 111–126. https://doi.org/10.1007/978-3-319-16817-3_8
[44] Dinh-Hoan Trinh, Marie Luong, Francoise Dibos, Jean-Marie Rocchisani, Canh-Duong Pham, and Truong Q Nguyen. 2014. Novel example-based method for super-resolution and denoising of medical images. *IEEE Transactions on Image processing* 23, 4 (2014), 1882–1895. https://doi.org/10.1109/tip.2014.2308422
[45] Tarun Wadhawan, Ning Situ, Hu Rui, Keith Lancaster, Xiaojing Yuan, and George Zouridakis. 2011. Implementation of the 7-point checklist for melanoma detection on smart handheld devices. In *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*. IEEE, 3180–3183. https://doi.org/10.1109/IEMBS.2011.6090866
[46] 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, 4 (2004), 600–612. https://doi.org/10.1109/tip.2003.819861
[47] Bing Xu, Andrew Tulloch, Yunpeng Chen, Xiaomeng Yang, and Lin Qiao. 2019. Hybrid Composition with IdleBlock: More Efficient Networks for Image Recognition. *arXiv preprint arXiv:1911.08609* (2019).
[48] Jianchao Yang, John Wright, Thomas S Huang, and Yi Ma. 2010. Image super-resolution via sparse representation. *IEEE transactions on image processing* 19, 11 (2010), 2861–2873. https://doi.org/10.1109/tip.2010.2050625
[49] Wenming Yang, Xuechen Zhang, Yapeng Tian, Wei Wang, Jing-Hao Xue, and Qingmin Liao. 2019. Deep learning for single image super-resolution: A brief review. *IEEE Transactions on Multimedia* 21, 12 (2019), 3106–3121. https://doi.org/10.1109/tmm.2019.2919431
[50] Roman Zeyde, Michael Elad, and Matan Protter. 2010. On single image scale-up using sparse-representations. In *International conference on curves and surfaces*. Springer, 711–730. https://doi.org/10.1007/978-3-642-27413-8_47
[51] 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 *Proceedings of the European Conference on Computer Vision (ECCV)*. 286–301. https://doi.org/10.1007/978-3-030-01234-2_18