Searching for Fast Demosaicking Algorithms

Searching for Fast Demosaicking Algorithms

KARIMA MA, Massachusetts Institute of Technology
MICHAEL GHARBI, ANDREW ADAMS, and SHOAIB KAMIL, Adobe Systems Inc
TZU-MAO LI, University of California San Diego
CONNELLY BARNES, Adobe Systems Inc
JONATHAN RAGAN-KELLEY, Massachusetts Institute of Technology

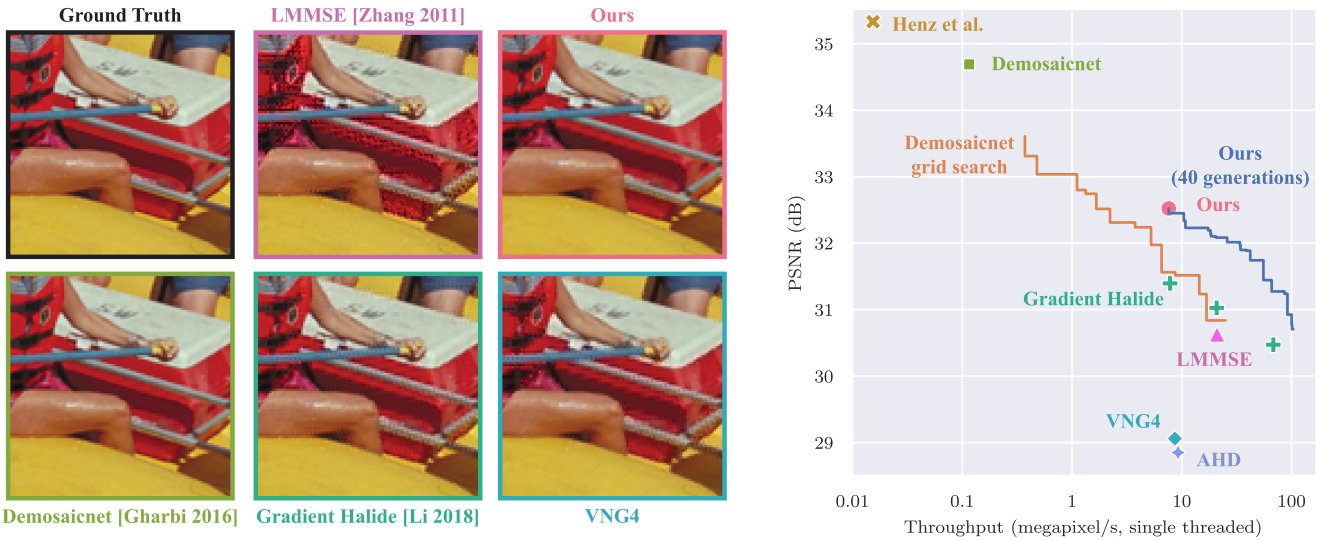


Fig. 1. We use a combination of learning and program search to automatically synthesize efficient, high-quality demosaicking algorithms. They significantly advance the Pareto frontier of cost vs. quality over prior state-of-the-art methods from 10 s to 1000 s of operations per pixel (plot, right). They are at least 1dB higher quality at the same cost, or 5–10× faster at the same quality, relative to prior published algorithms. Visual quality is noticeably improved on challenging image content (note the Bayer grid speckling and zippering artifacts in the LMMSE, GradientHalide, and VNG outputs). The only prior methods which offer higher quality than ours are large convolutional models 2–3 orders of magnitude more computationally expensive (Demosaicnet, Henz at al.). In addition to traditional Bayer demosaicking shown here, we present Pareto-dominant algorithms for demosaicking from X-Trans sensors, and for joint demosaicking superresolution and superresolution alone.

We present a method to automatically synthesize efficient, high-quality demosaicking algorithms, across a range of computational budgets, given a loss function and training data. It performs a multi-objective, discrete-continuous optimization which simultaneously solves for the program structure and parameters that best tradeoff computational cost and image quality. We design the method to exploit domain-specific structure for search efficiency. We apply it to several tasks, including demosaicking both Bayer and Fuji X-Trans color filter patterns, as well as joint demosaicking and super-resolution. In a few days on 8 GPUs, it produces a family of algorithms that significantly improves image quality relative to the prior state-of-the-art across a range of computational budgets from 10 s to 1000 s of operations per pixel (1 dB–3 dB higher quality at the same cost, or 8.5–200× higher throughput at same or better quality). The resulting programs combine features of both classical and deep learning-based demosaicking algorithms into more efficient hybrid combinations, which are bandwidth-efficient and vectorizable by construction. Finally, our method automatically schedules and compiles all generated programs into optimized SIMD code for modern processors.

CCS Concepts: • **Software and its engineering** → **Genetic programming**; • **Computing methodologies** → **Image processing**; **Machine learning**; **Artificial intelligence;**

Additional Key Words and Phrases: Demosaicking, super-resolution, domain specific programming, differentiable programming, neural architecture search, data driven methods

This work was partially supported by DARPA agreement HR00112090017 and NSF award CCF-1723445. GPU compute resources were provided by Crusoe Energy.
Authors' addresses: K. Ma and J. Ragan-Kelley, Massachusetts Institute of Technology; 32 Vassar St, Cambridge, MA 02139; emails: {karima, jrk}@mit.edu; M. Gharbi, A. Adams, S. Kamil, and C. Barnes, Adobe Systems Inc; 601 Townsend St, San Francisco, CA 94103, 345 Park Avenue San Jose, CA 95110, 104 Fifth Ave, 4th Fl, New York NY 10011, 801 N. 34th Street Seattle, WA 98103; emails: mgharbi@adobe.com, andrew.b.adams@gmail.com, kamil@adobe.com, connellybarnes@gmail.com; T.-M. Li, University of California San Diego, 9500 Gilman Drive, Mail Code 0404 La Jolla, CA 92093-0404; email: tzli@ucsd.edu.

0730-0301/2022/05-ART172
https://doi.org/10.1145/3508461

ACM Reference format:
Karima Ma, Michael Gharbi, Andrew Adams, Shoaib Kamil, Tzu-Mao Li, Connelly Barnes, and Jonathan Ragan-Kelley. 2022. Searching for Fast Demosaicking Algorithms. *ACM Trans. Graph.* 41, 5, Article 172 (May 2022), 18 pages.
https://doi.org/10.1145/3508461

1 INTRODUCTION

Demosaicking is among the most ubiquitous and performance-critical image processing tasks. As the critical first step, it can make or break the results of the entire camera imaging pipeline: any detail lost in demosaicking is gone forever, while any false detail introduced becomes a complex spatial structure nearly impossible to remove downstream. Balancing the two concerns is difficult, and the problem is ill-posed, so there is no correct answer. At the same time, demosaicking must often be performed under extreme computational budgets: a single stream of 4K 60 FPS video requires processing 0.5 gigapixels per second. Even if we dedicate one of the cores in a high-end mobile processor (CPU, GPU, or DSP) just to the task of demosaicking, with perfect SIMD utilization, this still leaves time for at most a few hundred operations per pixel.

Faced with this challenge, current demosaickers generally target one of two extremes (Figure 1). Most widely-deployed implementations, from cell phones to Adobe Camera Raw, are limited to at most 100s of operations per pixel of highly-optimized computation, hand-crafted to invert a single specific color filter array [Hirakawa and Parks 2006; Zhang and Wu 2005]. With this, they deliver reasonable image quality but struggle to avoid artifacts like Moiré and false detail in challenging situations. Meanwhile, deep learning and optimization-based methods have emerged which dramatically improve quality, and more easily generalize to different color filter arrays and other problem variants, but at the cost of 2–3 orders of magnitude more computation (hundreds of thousands to millions of operations per pixel), putting them out of reach of most practical use cases [Gharbi et al. 2016; Heide et al. 2014]. Depending on the chosen implementation, demosaicking can take anywhere from 25% to 85% of the Adobe Camera Raw ISP runtime.

We develop new families of efficient, learned demosaicking algorithms which significantly improve the state-of-the-art image quality achievable across the whole range from 10 s to 1000 s of operations per pixel. In addition to the common Bayer pattern, we also develop demosaicking algorithms for the Fuji X-Trans pattern, and that jointly solve demosaicking and super-resolution from a Bayer pattern.

Our programs are Pareto-dominant: they offer both significantly higher quality (1 dB–3 dB) at the same computational cost as any prior algorithm in the same range, and can deliver comparable or better image quality at dramatically lower computational cost (8.5–220× or more). They are designed for efficient streaming SIMD implementation, and automatically compile to highly-optimized kernels for modern processors.

We generate this family of new algorithms *automatically* by developing a multi-objective, discrete-continuous search which simultaneously solves for the program structure and parameters to find the best tradeoff between computational cost and image quality in a target range of computational budgets. The search is driven by the same loss function and training data as recent demosaicking and super-resolution neural networks [Anwar and Barnes 2020; Chu et al. 2021b; Dong et al. 2014; Gharbi et al. 2016; Henz et al. 2018; Shi et al. 2016; Wang et al. 2018]. However, we found standard **neural architecture search** (**NAS**) techniques to be insufficient for our task: these methods usually target highly regular and extremely over-parameterized models. We focus on low-cost models, which requires a careful design that exploits domain-specific structure. Our search produces state-of-the-art results in 4–5 days on 8 GeForce Titan Xp GPUs—on the same order as the cost of training a single neural network to convergence. The resulting programs combine features of both classical and deep learning-based demosaicking and super-resolution algorithms into more efficient hybrid combinations, composing building blocks into algorithms that are bandwidth-efficient and highly vectorizable by construction. Finally, our method automatically schedules and compiles any program produced by the search into highly-optimized SIMD code.

We believe our approach lays the foundation for automatically optimizing image processing pipelines for performance and quality, combining the advantages of both classical algorithms and deep learning to produce better, more efficient algorithms than currently exist. For example, in addition to three variants of the demosaicking problem, we show that our search method can also produce Pareto-dominant programs for the task of the high-performance super-resolution, alone.

In summary, we make the following contributions:

- New, state-of-the-art Bayer & X-Trans demosaicking, joint demosaicking with super-resolution, as well as standalone super-resolution algorithms that dramatically outperform prior work across the most commercially relevant range of computational budgets.
- A method for automatically generating such algorithms that span a wide range of compute budgets.
- We show that adding domain-specific primitives and search structures significantly improves the performance-quality tradeoffs achievable by differentiable program search on image processing tasks in the low-cost regime.
- We define a search space that generates SIMD and locality-friendly algorithms by construction, and a compiler that exploits this structure to automatically generate highly-optimized streaming implementations.

2 RELATED WORK

Our approach combines genetic program search with gradient-based optimization of differentiable programs, applying insights from machine learning and classical algorithms, to automatically search for efficient demosaicking programs that cover a large spectrum of the quality-performance tradeoff space.

2.1 Image Demosaicking

Reconstructing full-color images from color filter arrays is a well-researched, but inherently ill-posed, problem whose solutions must balance quality and efficiency [Li et al. 2008]. Demosaicking errors typically occur at edges, creating false "zipper" patterns or "maze" artifacts, but they can also affect large spatial regions, causing color fringing, false color Moiré patterns, or over-smoothing.

Classical algorithms share two key design elements: they use edge-adaptive directional filters to avoid smoothing over edges [Hamilton Jr and Adams Jr 1997; Hibbard 1995], and they exploit cross-channel correlations to guide the interpolation of the missing red and blue values, using an estimate of the more densely sampled green channel. For example, the smooth hue prior [Cok 1987] predicts smooth variations of differences or ratios between colors. Many proposed methods improve edge and color correlation detection, and sometimes jointly address denoising [Alleysson et al. 2005; Buades et al. 2009; Dubois 2005; Duran and Buades 2014; Hirakawa and Parks 2005, 2006; Kiku et al. 2013; Menon and Calvagno 2009; Niu et al. 2018; Zhang et al. 2009, 2011].

A different class of algorithms cast demosaicking as an inverse problem and solve for the full-color image using optimization [Chang et al. 2015; Condat and Mosaddegh 2012; Getreuer 2011; Heide et al. 2014; Kokkinos and Lefkimmiatis 2018; Tan et al. 2017a]. While these methods achieve high-quality demosaicking, the large computational cost of optimization limits their applicability.

Data-driven techniques optimize the parameters of demosaicking algorithms using ground truth natural images [Go et al. 2000; Kapah and Hel-Or 2000; Khashabi et al. 2014; Kwan and Wu 2004; Li et al. 2018]. Recent approaches use convolutional neural networks [Gharbi et al. 2016; Henz et al. 2018; Klatzer et al. 2016; Kokkinos and Lefkimmiatis 2018, 2019; Liu et al. 2020; Ratnasingam 2019; Tan et al. 2018, 2017b]. Deep learning methods achieve state-of-the-art quality, but remain computationally expensive.

2.2 Super-resolution

Super-resolution recovers a high-resolution image from one or more low-resolution images. Classical iterative algorithms are often computationally expensive and rely on image degradation priors which can hinder their robustness [Yang and Huang 2017]. It is also difficult to avoid over blurring and introducing artifacts like false high frequencies and jagged edges. SRCNN introduced a modern convolutional neural network for super-resolution [Dong et al. 2014]. Subsequent work has introduced much larger models, some with over 1,000 convolutional layers [Zhang et al. 2018]. Despite producing state of the art image quality, these models are too expensive to run in most commercial applications. Unfortunately, existing fast super-resolution models like SRCNN and ESPCN [Shi et al. 2016] perform dramatically (2–3 dB) worse than large models (see Figure 8).

An even more challenging problem is super-resolving images directly from raw camera data. An overwhelming majority of photos are taken today by smartphone cameras. Their portability requires small sensors with limited resolution. Such cameras would benefit greatly from joint super-resolution and demosaicking programs. Traditional approaches to joint super-resolution and demosaicking require slow iterative optimization like coordinate descent [Farsiu et al. 2004] or clustering [Bennett et al. 2006]. Recent deep convolutional models [Qian et al. 2019; Xing and Egiazarian 2021] can take minutes to process a high-resolution image on a CPU, making them too slow to run in most commercial image processing pipelines.

2.3 Neural Architecture Search and Genetic Programming

NAS and genetic programming methods automatically generate programs that maximize some (often single) objectives, such as classification accuracy [Koza and Koza 1992; Zoph and Le 2016]. In graphics, genetic programming has been used for shader simplification [He et al. 2015; Sitthi-Amorn et al. 2011; Wang et al. 2014] and image pipeline optimization [Lou et al. 2016a].

The space of NAS and genetic programming approaches can be understood in terms of how they define their program search space, their search strategy, and their performance evaluation criteria [Elsken et al. 2019]. Our search algorithm can be viewed as a multi-objective NAS via genetic programming. Unlike most NAS methods, our search procedure focuses on low-cost algorithms and uses domain-specific program structures to design an efficient search space. It discovers fast and high-quality demosaicking programs that significantly outperform models produced by generic NAS baselines (Section 4.2) and super-resolution programs that achieve nearly comparable quality to models that are 84× more expensive produced by a prior multi-objective NAS technique [Chu et al. 2021b].

Search Space. To make the search tractable, NAS search spaces are often constrained to fixed-structure compositions (e.g., stacks) of repeated cells made of coarse-grained network building blocks (convolutions, skip connections, activations, etc.) [Zhong et al. 2018; Zoph et al. 2018]. This often leads to expensive models, with a limited structural variation. In contrast, because we are interested in efficient programs, we search over the complete program structure via local **directed acyclic graph** (**DAG**) mutations and include domain-specific operators beyond conventional NAS building blocks (Section 3.1). We factor our search space into semantically meaningful sub-tasks, which reduces the combinatorial complexity and improves quality (Section 3.2.3).

Search Strategy and Evaluation criteria. Although network pruning [Blalock et al. 2020] remains the most popular technique to speed up trained models, recent NAS work has explored multi-objective optimizations that account for model efficiency [Anderson et al. 2019; Chu et al. 2021b; Gong et al. 2019; Zhou and Diamos 2018]. However, few enable full exploration of the cost–quality tradeoff. [Tan et al. 2019; Zoph and Le 2016] collapse the two objectives into a single scalar reward, which prevents sampling along the Pareto curve during training, and limits user control. We use a variant of genetic search that allows us to sample the models we mutate and retain after each generation based on their dominance across the Pareto frontier. LEMONADE [Elsken et al. 2018] and FALSR [Chu et al. 2021a] also use multi-objective genetic search, but LEMONADE only samples models to mutate based on their costs.

Generated Model Efficiency. All prior NAS approaches we know search over cost regimes orders of magnitude more expensive than ours. MnasNet [Tan et al. 2019], which targets image classification on smartphones, produces models that are 10 to 100× more expensive than ours. FALSR [Chu et al. 2021a] produces super-resolution models with 326 k to 1021 k parameters, while our programs have under 10 k.

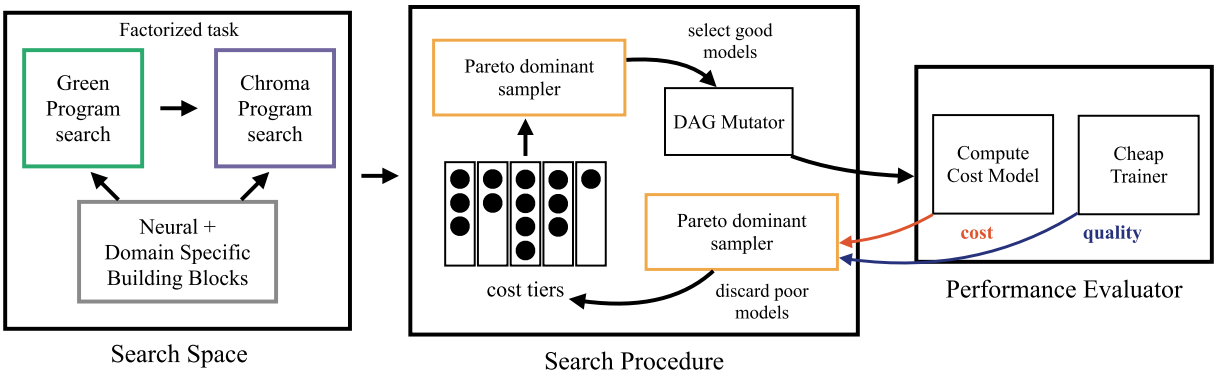


Fig. 2. System overview. We define a search space over demosaicking programs that consist of DAGs of both conventional neural network and domain-specific image processing building blocks. We further factor this space into a search over green prediction and chroma prediction sub-programs. We use genetic search combined with Pareto sampling to select which programs to mutate and maintain across generations. The Pareto dominance of our programs is measured by our cost model, which estimates the computational cost, and a cheap training process, which estimates the quality of a given program.

3 TECHNIQUE

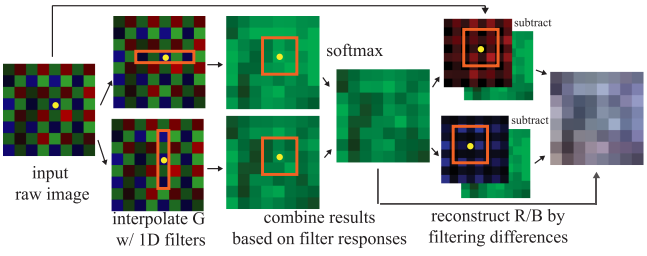
To generate demosaicking algorithms that balance cost with quality, we must efficiently search over a large number of candidate programs. Each program is differentiable and parameter-heavy, and needs to be trained on a large dataset. The key constraint driving our design decisions below is thus the time it takes to run the search. We take measures to decrease the time it takes to train each program, and to reduce the combinatorial complexity of our search space towards programs that are likely to be fast and high-quality.

We use a genetic search algorithm that populates each generation with mutations of the best-performing programs of the previous generation. Figure 2 illustrates the overall system. Because we are optimizing for multiple objectives, the notion of "best-performing" is not simple. We want high-quality programs that span a range of runtime budgets. We, therefore, divide the cost axis into *cost tiers*, and mutate and maintain programs from each tier based on their proximity to the Pareto frontier. We estimate program quality with a fast training procedure on a small dataset and we estimate computational cost with a simple model (a weighted sum of operations performed). From the final generation, we take the top 100 programs ranked by their Pareto-dominance, compile them to efficient Halide [Ragan-Kelley et al. 2012] implementations to measure true cost, and train them on the entire dataset to get true quality.

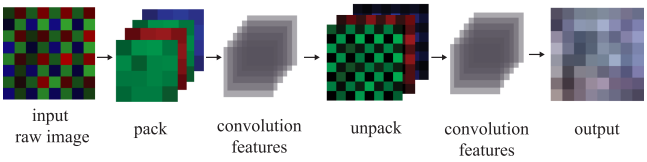
In the rest of this section, we first detail the domain-specific building blocks from which our search constructs programs (Section 3.1) then describe the genetic search process (Section 3.2), before finally describing our automatic compilation pipeline (Section 3.3).

3.1 Building Blocks

We design our search primitives based on four criteria. First, they need to be efficient to evaluate. Second, we need to be able to easily compose them into a meaningful pipeline. Third, they need to be differentiable to allow end-to-end training. Finally, they need to be sufficient to express existing demosaicking algorithms—both classical feed-forward demosaicking algorithms and deep-learning-based demosaicking and super-resolution algorithms (Figure 3).



(a) a classical edge-adaptive demosaicking algorithm



(b) a deep learning demosaicking algorithm

Fig. 3. We construct our program search space using building blocks inspired by both classical edge-adaptive (a) and deep-learning-based (b) demosaicking algorithms. We follow the classical regime by first reconstructing the green color (G), then reconstructing the red and blue (R/B) colors by predicting the difference between them and the reconstructed green color. Classical edge-adaptive demosaicking often selects between directional 1D filters for green colors to adapt to edges. Therefore, we include both 1D and 2D grouped convolutions, a softmax layer, element-wise multiplication, and sum reductions to reproduce this. For the deep-learning regime, we incorporate the commonly used *packing* and *unpacking* primitives (b) that pack a Bayer pattern into translation-invariant images and unpack them back to the input resolution. We also include standard deep learning primitives including convolutional and pooling layers, pixel-wise operations with potential inter-channel computation, including element-wise operations, a stack operator, and variants of 1×1 convolution layers.

Due to the higher sampling rate of the green channel in mosaic patterns, inspired by classical demosaicking algorithms [Cok 1987; Hibbard 1995], we factor our program search to first reconstruct the green image, and then use the green image to guide the reconstruction of the other two channels. Classical demosaicking algorithms often reconstruct the green channel using an

edge-adaptive selection from several local directional filters [Adams Jr 1995; Hirakawa and Parks 2005; Zhang et al. 2011]. Unlike conventional NAS, we include both 1D and 2D grouped convolution layers [Xie et al. 2017] to allow for directional filters and a softmax primitive, element-wise multiplication, and sum reductions to enable differentiable selection across filters. 1D convolutions also provide computational savings over linearly separable 2D convolutions. We also include variants of 1×1 convolution layers and channel-wise sum reductions to facilitate inter-channel communication.

Classical demosaicking algorithms often reconstruct red and blue channels by predicting their differences to the green channel. We include element-wise operations like addition and subtraction to model this technique. Our element-wise operators and stack operator also support residual connections [He et al. 2016] by combining intermediate outputs along the channel dimension.

Some deep-learning-based methods for demosaicking [Gharbi et al. 2016; Liu et al. 2020] use a packing layer that packs each repeating 2×2 grid in the Bayer pattern into four translation-invariant image channels (two green, one blue, and one red), and an unpacking layer that converts a 4 channel image back to the full resolution grid. But mosaic patterns may have different periodicities. For example, the Bayer pattern has a 2×2 period but the X-Trans pattern has a 6×6 period. We support the correct handling of different mosaic patterns with packing and unpacking operations parameterized by scale. Unpacking layers are also commonly used by deep super-resolution models [Anwar and Barnes 2020; Qian et al. 2019; Shi et al. 2016; Wang et al. 2018] in an upsampling layer to move information from the channel dimension into the spatial dimensions.

Upsampling and Downsampling are another set of important domain-specific building blocks for tasks like demosaicking and super-resolution. Downsampling is an efficient alternative for increasing a model's receptive field for recognizing high-frequency patterns and avoiding moiré artifacts, compared to the conventional approach of adding convolutional layers. We support multiple types of upsampling and allow our search to choose which ones to use. Our upsampling operators are: Unpack, Bicubic, and LearnedUpsample via transposed convolution. For Downsample we use a Pack or a LearnedDownsample which is a strided convolution with a filter width twice as large as the stride for proper anti-aliasing.

Finally, we designed insertion functions that insert predicted color values into their proper color channels with the given color values from the mosaic to produce full red, green, and blue images. These specialized functions allow our programs to make location-aware color predictions and save computation by only predicting the missing values.

In all, we use the following building blocks:

- element-wise: `Add`, `Sub`, `Mul`, `Stack`, `ReLU`
- convolutional: `GroupedConv1D`, `GroupedConv2D`
- filter selection: `SoftMax`
- per-pixel inter-channel computation: `GroupedConv1x1`, `GroupedSum`, `InterleavedSum`
- upsample: `Unpack`, `Bicubic`, `LearnedUpsample`
- downsample: `Pack`, `LearnedDownsample`
- insertion: `GreenInsert`, `ChromaInsert`

The convolutional operators `GroupedConv1D`, `GroupedConv2D`, and `GroupedConv1x1` are parameterized by the number of groups and output channels. The filter width is 3 except in the 1×1 case. `SoftMax` is used without scaling parameters. `GroupedSum` and `InterleavedSum` both take an N channel image, and produce a N/k channel image by summing together values within the channel group. `GroupedSum` sums over consecutive channels while `InterleavedSum` outputs channel i by summing over channels $i + jN/k$ for $0 \leq j < k$. All operations are done in single precision floating point.

3.2 Search Algorithm

We combine multi-objective genetic search and gradient descent to find Pareto-dominant pipelines represented as DAGs of the building blocks described above. Our search space requires special mutation rules (Section 3.2.1) both to deal with complex DAG structures where nodes can have multiple downstream parents to share computation, and to respect the constraints of special operators (e.g., that all inputs to a stage be upsampled/downsampled to the same resolution, and that the output needs to be at the correct target resolution). Because our search space is very large and allows for arbitrary DAGs, we factor it using domain-specific knowledge to reduce its combinatorial complexity, leverage pruning rules to exclude obviously bad or inefficient programs, and use fast and robust training methods to quickly explore thousands of models (Sections 3.2.2 and 3.2.3).

The goal of our search is to explore the design space of fast demosaickers, so we initialize it with two efficient models, shown in Figure 4, that are based on prior state-of-the-art designs. One seed model is a small simplified version of Demosaicnet [Gharbi et al. 2016] with fewer layers and channel counts. We do not use the published full-size Demosaicnet as a seed model because its throughput is far below (almost two orders of magnitude lower) than the range of throughputs for efficient demosaickers. Our other seed model is inspired by the GradientHalide model [Li et al. 2018] but uses multiple resolutions via upsampling and downsampling operators.

To optimize program quality across a range of compute costs, we bin our programs into cost tiers. After each search generation, we keep the top 20 programs in each cost tier and sample 12 models from each tier to mutate for the next generation. The width of each bin doubles as the programs get larger. The cost tier bins start at 0–200 FLOPs for green interpolation programs (Section 3.2.3) and 0–300 FLOPs for blue/red interpolation programs. The blue/red interpolation programs are allowed to be more expensive because they use a green interpolation program to produce one of their inputs. During each generation, the newly mutated programs are trained on a small dataset for a few epochs. Programs are trained for 6 epochs on a 100 k image subset of the Demosaicnet dataset during the search. One could increase the number and width of the cost tiers to cover lower throughput programs. However, this would increase the search time by adding more expensive programs to train per generation.

Estimating Cost. We estimate the cost of each program by traversing the DAG and counting the number of floating point operations required by the computations performed at each node, treating operations that can be executed using **fused**

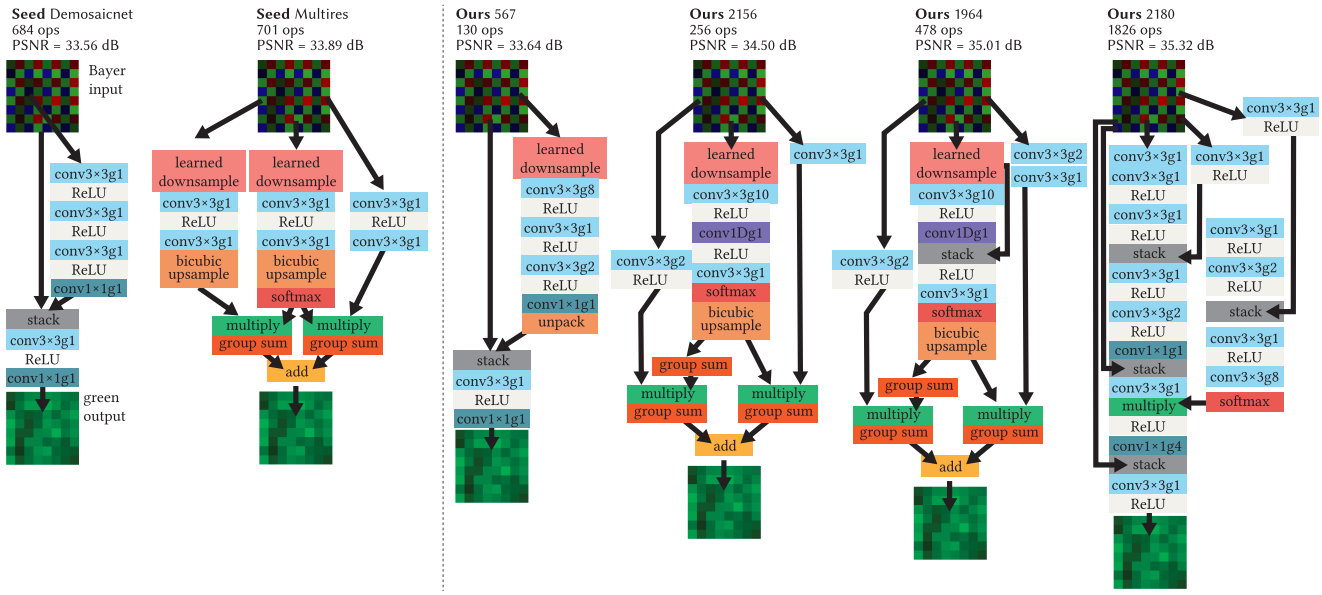


Fig. 4. Inputs and outputs of our search: seed programs are shown on the left, derived from prior work, and four green prediction models found by our search are shown on the right, ordered by increasing computational cost and quality. We use a Demosaicnet [Gharbi et al. 2016] variant and a multi-resolution demosaicking program inspired by AHD [Hirakawa and Parks 2005] as our seed models. Note how the diverse generated models differ significantly from the original seeds, how their DAG structures become increasingly complex as quality improves, and how they combine conventional neural building blocks with our domain-specific primitives. (Convolutional layers denote group count after their filter size).

multiply-add (**FMA**) as a single operation. The total cost of a program is the sum of the costs at each node. While this measure neglects data movement, it successfully estimates relative runtime, as demonstrated in Section 4, because our programs are constrained to operations that can be efficiently blocked or fused for locality.

Selecting Programs for Mutation. Within each cost tier, we select which programs to mutate and keep for the next generation using importance sampling. The likelihood of choosing a program is inversely proportional to its Pareto rank compared to other programs within its tier. Models on the Pareto frontier have rank 1. Models that would be on the frontier after removing all rank 1 models have rank 2, and so on. Thus, models are importance sampled based on their proximity to the current Pareto frontier.

3.2.1 Mutations. Each program selected for mutation is modified using one of the following operations:

- **Insertion** chooses, uniformly at random, a building block to add to the program at a random location in the DAG. If the chosen insertion location is invalid for the building block, this mutation rule continues randomly selecting locations until a valid location is found. If the operation to insert has two operands, then the second operand is chosen to be the output of some existing sub-DAG of the program. This induces recurrent connections and shared computations.
- **Deletion** chooses, uniformly at random, a DAG node to remove from the program.
- **Resolution Changes** Inserting, removing, and moving Upsamples and Downsamples is tricky. We must ensure that all resolution-changing mutations preserve the correct output resolution and do not allow intermediate outputs with different resolutions to be combined. To solve this problem, we use graph coloring and color each node by its output resolution. This representation allows us to perform resolution change mutations easily by manipulating color boundaries.
  - **Create Resolution Change**, randomly selects a subgraph from within a color boundary to compute at a lower resolution relative to its parent boundary. The resolution scale factor is randomly chosen as either 2 or 3.
  - **Remove Resolution Change**, randomly selects a color boundary and removes the resolution change it induces by deleting or inserting Upsamples and Downsamples.
  - **Shrink Resolution Subgraph**, randomly selects a node at a boundary and changes its resolution to that of the adjacent color across the boundary.
  - **Swap Resolution Operation**, randomly selects a resolution changing node and changes the type of Upsampling or Downsampling function used.
- **Decouple** randomly chooses a sub-DAG shared by more than one consumer node and duplicates it, allowing later mutations to modify each copy separately.
- **Channel Count Change** picks a random convolution operation and modifies its output channel count.[1]
- **Group Change** picks a random convolution operation and modifies its channel grouping [Xie et al. 2017], where the allowed groupings are any common factors of the input and output channel counts of the operation being mutated. Our

[1]The options for channel count are {8, 10, 12, 16, 20, 24, 28, 32}.

search finds that using grouped convolutions can provide useful computational savings (Figure 4).
- **Green Input Change**, used only during the search for red/blue interpolation programs (Section 3.2.3), modifies a red/blue program to use a different green subprogram as its input.

Search is biased towards insertion and deletion over other mutations, because we found this allows generations to increase quality and decrease cost more quickly.

3.2.2 Fast Training. Each mutated program is trained using gradient descent on a dataset of difficult mosaic patches [Gharbi et al. 2016]. In an ideal world, we would be able to train programs on the entire dataset to convergence; however, the training time (5–10 GPU-hours per program) would make search intractable. Instead, we employ a strategy to drastically cut down training time while still obtaining a good estimate of quality.

First, we constrain the training to use a fixed random subset of 100,000 (out of 2.6 million) images, and only train for 6 epochs. Both parameters were set based on the exploration of the training dynamics of our space of programs. We found this setting produces a good ranking of options that corresponds fairly closely to the fully-trained results, but in just 10–15 GPU-minutes per program.

Second, to avoid under-estimating quality due to catastrophic weight initialization [Frankle and Carbin 2019], we initialize three sets of weights. We simultaneously train all three sets for one epoch. After one epoch, we keep training the set of weights with the best validation PSNR. We observe that bad weight initializations can usually be detected within the first epoch by comparing performance across initializations. This strategy greatly improves the robustness of our quality measurements for a small increase in training time.

3.2.3 Efficiently Structured Search. We structure the search space in a number of ways to raise the likelihood of finding interesting points in the quality-efficiency tradeoff space within a fixed amount of time. Because of the large cost of training each program, we employ three key strategies, in addition to the structure of our building blocks, to steer the search towards useful programs:

— We bias the selection of mutations during search towards insertion and deletion (Section 3.2.1);
— We eagerly discard likely-poor programs before training them;
— We factor the search into the space of subprograms that estimate green values and the space of subprograms that use these green values to interpolate red and blue values.

Because our search is allowed to consider all possible DAGs consisting of our building blocks, mutations may result in programs that are unlikely to represent interesting points in the tradeoff between the amount of computation and the quality of the demosaicking result. For example, a mutation may insert an operation that is the inverse of the operation before it (such as an addition followed by subtraction, or a downsample followed by an upsample). Similarly, insertions may result in trivial operations (e.g., an `InterleavedSum` over a single channel). To avoid spending valuable training time on these programs, we eagerly discard them before the training phase.

We observe that some prior work on demosaicking algorithms uses different strategies to interpolate green values compared to blue and red values, due to the presence of twice as many known green values versus the other colors. Green values are also predicted first because they serve as a useful guide for predicting red and blue. We, therefore, reduce the combinatorial complexity of the search by factoring it into two separate search spaces: one for green prediction programs and another for red and blue prediction programs. We first perform the search over green-interpolation programs and select a set of generated programs that are Pareto-dominant over a range of computing costs. Then the red-blue search can select any of these Pareto-dominant green-interpolation programs to use as an additional input to the mosaic to help guide the red and blue interpolation. We show that this strategy is more efficient than the alternative of searching for a single program that interpolates all values jointly in Section 4.2.

3.3 Compiling Programs to Optimized Implementations

Compiling our programs to efficient SIMD code is straightforward and fully automatic, as our building blocks were intentionally selected with this in mind. We compile our programs by lowering them to Halide [Ragan-Kelley et al. 2012]. We traverse the model graph, mapping each node to its corresponding Halide implementation. These are typically 5–10 lines of code each.

Halide code also requires a "schedule" that specifies how the algorithm runs on the hardware. Generating this is also straightforward and fully automatic, again thanks to our choice of search space. All of our nodes have small spatial support, so we fuse the entire pipeline in tiles. We compute stages as needed per tile of output.[2] Nodes that only have one consumer are inlined into that consumer whenever this would not incur recompute (i.e., when the consumer is not a stencil or reduction over channels). Nodes that perform a computation that varies across the output channels (e.g., Stack) are unrolled across output channels. All nodes that reduce across input channels (e.g., conv layers) are fully unrolled across input channels. Upsamples and downsamples are never inlined, to avoid complicating the addressing patterns of the consuming stage.

This simple heuristic scheduler gives a performance that correlates extremely well with our cost model, as shown in Figure 10. Halide includes more complex automatic schedulers (e.g., [Adams et al. 2019]), but they proved unnecessary in this work.

4 EVALUATION

We evaluate our search technique and the programs it produces according to cost (program throughput) and quality (PSNR relative to ground truth) on three demosaicking applications: Bayer demosaicking, X-Trans demosaicking, and joint Bayer demosaicking with super-resolution. In addition, we evaluate our technique's

[2] We used a tile size of 768×240, which was empirically determined on our target x86 processor. Tile size would ideally be re-tuned if compiling for a different target.

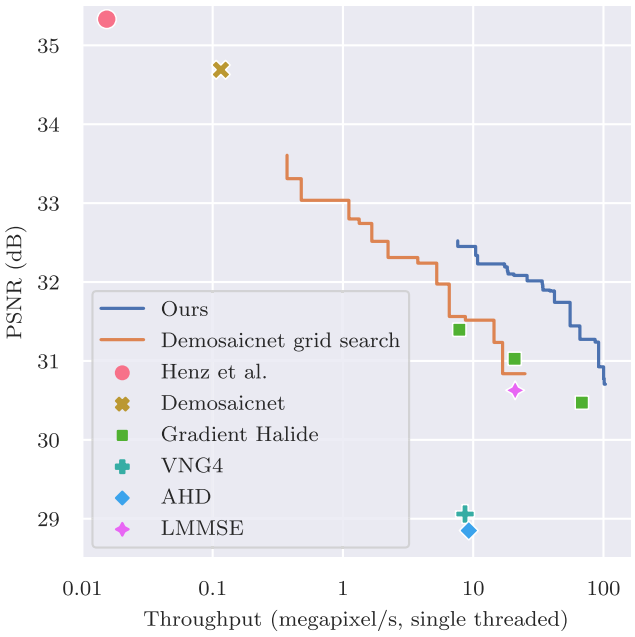


Fig. 5. Bayer demosaicking quality vs. throughput on the HDR-VDP and Moiré datasets for our automatically generated Pareto frontier of programs, prior state-of-the-art algorithms, and a grid search over Demosaicnet variants representative of structured pruning. Our search process significantly improves the quality vs. performance tradeoff of existing programs in the real-time performance regime, and spans a frontier of state-of-the-art algorithms covering a throughput range of 10—100 Megapixels per second on a single CPU core. Our search found a program that is > 8× faster than the best baseline model, Gradient Halide with 15 7×7 filters, at the same quality and another program that is >1 dB better at the same throughput.

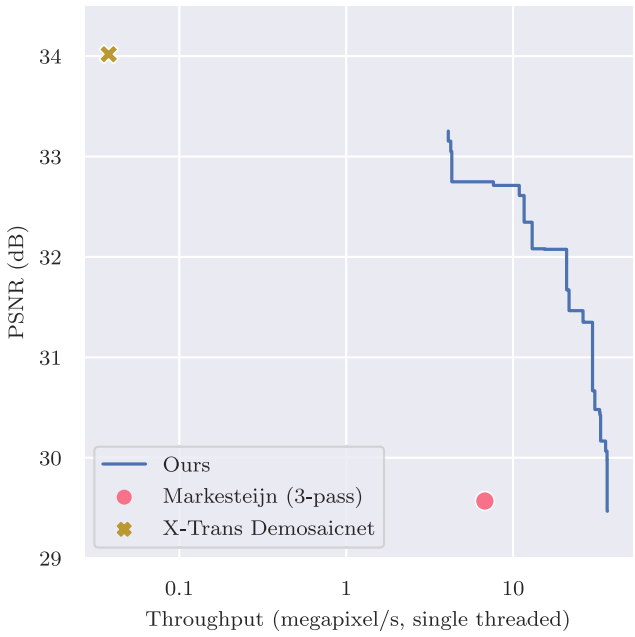


Fig. 6. X-Trans demosaicking quality vs. throughput for our automatically generated Pareto frontier of programs and state-of-the-art algorithms on the HDR-VDP and Moiré datasets. Very few demosaicking algorithms exist that support X-Trans, making our technique vital for finding algorithms at different points in the quality-performance tradeoff space. Our search produces a program that is 5.4× faster than Markesteijn with 0.4 dB higher quality. The highest quality program found by our search is 3.7 dB better than Markesteijn and more than 100× faster than Demosaicnet, with very high quality reconstruction at 33.25 dB test set performance.

generalization to the standalone task of super-resolution. We compare our programs against the cost and quality of existing demosaicking, super-resolution, and joint demosaicking super-resolution methods. We also visualize the outputs of programs produced by our method that span a wide range of throughputs and compare them qualitatively to the outputs of our baselines. We further evaluate our three key design choices:

(1) We study the usefulness of our domain-specific search primitives by comparing to our search algorithm restricted to traditional convolutional network primitives.
(2) We evaluate the importance of decomposing the search into green-prediction followed by red/blue prediction, by searching for programs that directly predict all three color channels.
(3) We study the accuracy of our cost model by comparing the estimated cost with the actual runtimes of generated programs.

All results are evaluated on held-out test datasets, which we did not touch during the search process. These are the Set5, Set14, Urban100, BSD100, Kodak, and McMaster datasets along with 1,000 randomly-selected 128×128 patches from each of the challenging HDR-VDP and Moiré datasets used by Gharbi et al. [2016]. All performance measurements are for a single core on an Intel i9-9960X CPU @ 3.10 GHz with HyperThreading disabled.[3]

4.1 Pareto-Dominant Programs

Figures 5, 6, 7, 8 show the Pareto frontier of programs found by our search along with several existing demosaicking, joint demosaicking with super-resolution, and super-resolution algorithms as baselines. Table 1 lists for each application and test dataset, the PSNRs and throughputs of all our baseline methods, and a selection of models found by our search. For demosaicking and joint demosaicking with super-resolution tasks, our programs are the result of 40 generations of search over green reconstruction programs, and 40 generations on red and blue reconstruction programs. For the super-resolution task, our programs are the result of 40 search generations. Our search process significantly improves the quality vs. performance tradeoff of existing programs in the real-time performance regime, and spans a frontier of state-of-the-art algorithms covering a throughput range of one to two orders of magnitude in megapixels per second depending on the application.

[3]We use single-core performance because all algorithms trivially parallelize in tiles, and parallelism increases bench-marking noise while not impacting relative performance. For our neural network baselines, since there were no fast implementations available at the time of writing, we used the correlation equation for our programs between modeled cost and measured runtime to estimate throughput.

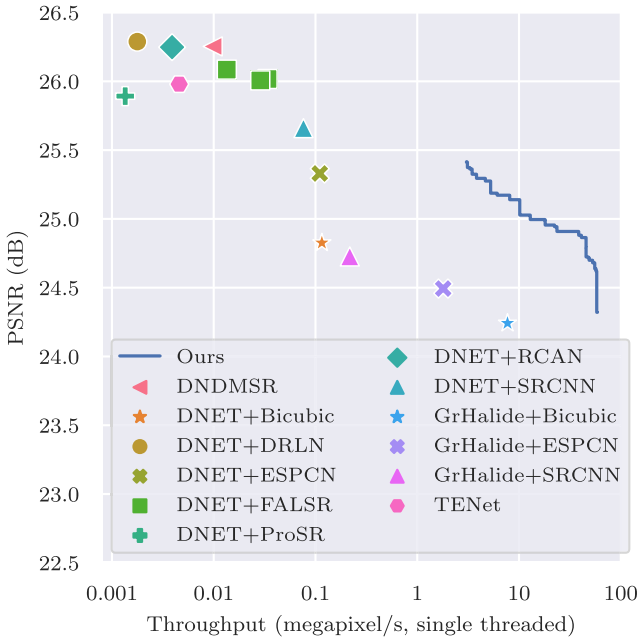


Fig. 7. Joint demosaicking and super-resolution quality vs. throughput for our automatically generated Pareto frontier of programs and state-of-the-art algorithms on the HDR-VDP and Moiré datasets. Most work on joint demosaicking and super-resolution (DNDMSR & TENet) has been focused on designing deep neural networks that are very computationally expensive. ESPCN+GradientHalide is the best performing baseline out of those that operate near the regime of throughputs we consider in our search. Our search produces a program that dominates ESPCN+GradientHalide by > 30× in throughput and 0.13 dB in quality and another program that is nearly 1dB better with 1.7× faster throughput. The best joint demosaicking and super-resolution deep model is DNDMSR, which has 0.84 dB higher quality than our best program but is more than 300× slower. Our search produces a dense range of Pareto-dominant joint demosaicking and super-resolution algorithms across a throughput regime where there were previously very few options.

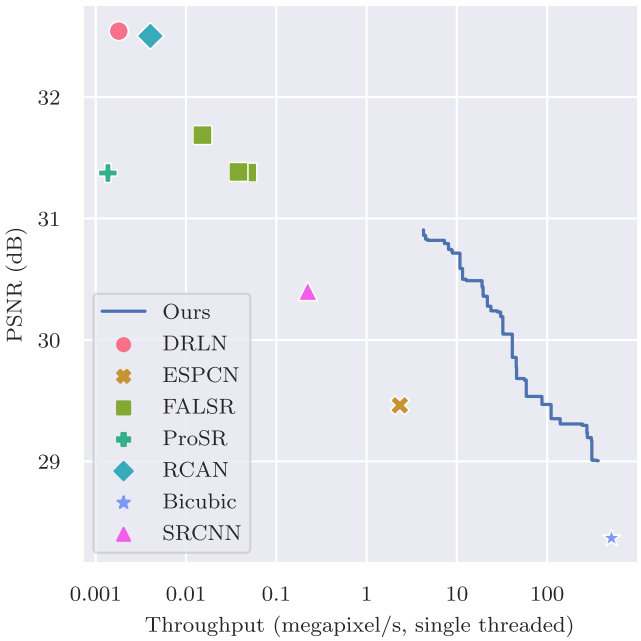


Fig. 8. Super-resolution quality vs. throughput for our automatically generated Pareto frontier of programs and state-of-the-art algorithms, evaluated on standard super-resolution datasets: BSD100, Urban100, Set5, and Set14. ESPCN is the best performing baseline out of those that operate near the regime of throughputs we consider in our search. Our method found a program that achieves the same PSNR as ESPCN while being nearly 50× faster and another program that achieves 1.45 dB better quality while being 2× faster. SRCNN and ESPCN are significantly Pareto-dominated by our models which shows that finding efficient high-quality super-resolution models requires more than just scaling down a deep convolutional neural network. The fastest large neural network, FALSR-B, is 0.47 dB better than our best model but 90× slower.

4.1.1 Bayer Demosaicking. For the Bayer demosaicking task, we compare quality and throughput against implementations of AHD, LMMSE, and VNG4 from `librtprocess` [CarVac 2021], a library extracted from RawTherapee [The RawTherapee Team 2021] (Figure 5). Parameters are set to place these algorithms as close to the Pareto frontier as possible, typically by tuning for maximum performance. For AHD, we add a final chroma median filtering step to replicate what is done in RawTherapee. We also compare against variants of two published demosaicking algorithms, Gradient Halide [Li et al. 2018] and Demosaicnet [Gharbi et al. 2016]. We scale the Gradient Halide model by increasing the size and number of filters and we structurally prune Demosaicnet via grid search over layer and channel counts to span a range of compute budgets. For reference we also show where the published version of Demosaicnet and another even larger neural network demosaicker [Henz et al. 2018] lie on the tradeoff space. [Henz et al. 2018] has higher quality than Demosaicnet but is 7.5× slower; both are orders of magnitude too expensive for most production use cases. The programs produced by our method cover a wide range of compute budgets and significantly outperform all of our baselines. Our method produces programs that achieve significantly better image quality (> 1 dB) at the same compute budget of existing baselines, or comparable image quality with nearly 10× higher throughput. Figure 11 visualizes the demosaicked outputs of 3 programs produced by our method compared to the outputs of our baseline methods and Demosaicnet on images from our test sets. Our programs avoid producing the artifacts seen in the baselines, such as zippering in the first column of LMMSE, AHD, and Gradient Halide, color fringing and Bayer grid artifacts in the second column of AHD and GradientHalide, and the Bayer grid and zippering artifacts in the fifth and sixth columns of LMMSE. As expected, the output quality of our programs improves as their cost increases.

4.1.2 X-Trans Demosaicking. It is even more difficult to write good demosaicking algorithms for the X-Trans mosaic due to its large periodicity and irregular pattern. This is evidenced by the fact that very few X-Trans demosaicking algorithms have been created compared to those for Bayer. In such cases, it is especially beneficial to have a system that can automatically generate good demosaickers. For the X-Trans demosaicking task we

compare against the Markesteijn algorithm, also from RawTherapee, and the X-Trans variant of Demosaicnet (Figure 6). Our search produced a program that is 3.2 dB better in quality than Markesteijn at a similar throughput, and another program that is 5.4× faster than Markesteijn and 0.4 dB better in quality. The highest quality program found by our search is a staggering 3.7 dB better than Markesteijn and more than 100× faster than Demosaicnet while producing high quality outputs with a test set performance of 33.25 dB on the difficult HDR-VDP and Moiré datasets. Figure 12 visualizes the outputs of 3 programs produced by our method that cover a range of throughputs as well as the outputs of the Markesteijn algorithm and the X-Trans Demosaicnet variant. Our programs avoid the artifacts produced by the Markesteijn algorithm such as the moire artifacts in the first and third column, the color fringing in columns 2 through 6, and the maze pattern in the third column.

4.1.3 Joint Bayer Demosaicking and Super-resolution. Most recent work on super-resolution has been focused on designing high quality models in a very low throughput regime of less than 0.002 Megapixels per second (e.g., DRLN [Anwar and Barnes 2020], RCAN [Zhang et al. 2018], PROSR [Wang et al. 2018]). FALSR [Chu et al. 2021b] uses genetic search to find cheaper super-resolution models but the fastest model produced by their approach, which operates at less than 0.05 Megapixels per second, is still 50× slower than the slowest model produced by our method with less than 0.5 dB higher quality. ESPCN [Shi et al. 2016] and SRCNN [Dong et al. 2014] are two smaller super-resolution models that operate closer to the throughput regime we consider for real-time performance programs and we compare to them as baselines for our super-resolution applications.

For the joint Bayer demosaicking and super-resolution task we compared against the best variant of the GradientHalide model (15 filters, each 7 × 7) for demosaicking chained with either SRCNN, ESPCN, or bicubic upsampling. We also show performance of Demosaicnet chained with SRCNN, ESPCN, and bicubic upsampling for completeness. For context, we report the performance of several large super-resolution deep neural networks: DRLN, ProSR, RCAN, and FALSR, each chained with Demosaicnet, and two joint demosaicking super-resolution deep neural networks: TENet [Qian et al. 2019] and DNDMSR [Xing and Egiazarian 2021], which all lie outside the typical commercial application throughput requirements.

ESPCN+GradientHalide is the best performing baseline out of those that operate near the regime of throughputs we consider in our search. Our search found a program that dominates ESPCN+GradientHalide by >30× in throughput at similar quality and another program that is nearly 1 dB better and 1.7× faster. The best joint demosaicking and super-resolution deep model is DNDMSR, which has 0.84 dB higher quality than our best program but is more than 300× slower. Note that chaining an expensive model like Demosaicnet with SRCNN produces a model with 40× lower throughput than our best program with only 0.25 dB higher quality, but for only a 3× decrease in throughput our search can improve program quality by more than 0.25 dB, indicating that it is more efficient to search over a joint model. Our search produces a dense range of Pareto dominant joint demosaicking and super-resolution algorithms across a throughput regime where there were previously very few options. Figure 13 shows the outputs of one of our Pareto dominant programs along with several of other methods. Our program avoids the high frequency speckling seen in the first and fifth columns, the color fringing in the third column, and the zippering artifacts in the last column of Bicbuic+GradientHalide, ESPCN+GradientHalide and SRCNN+GradientHalide.

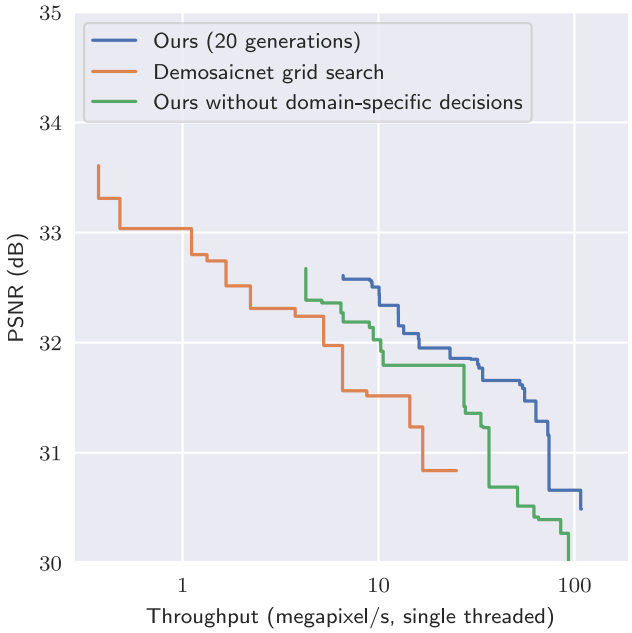
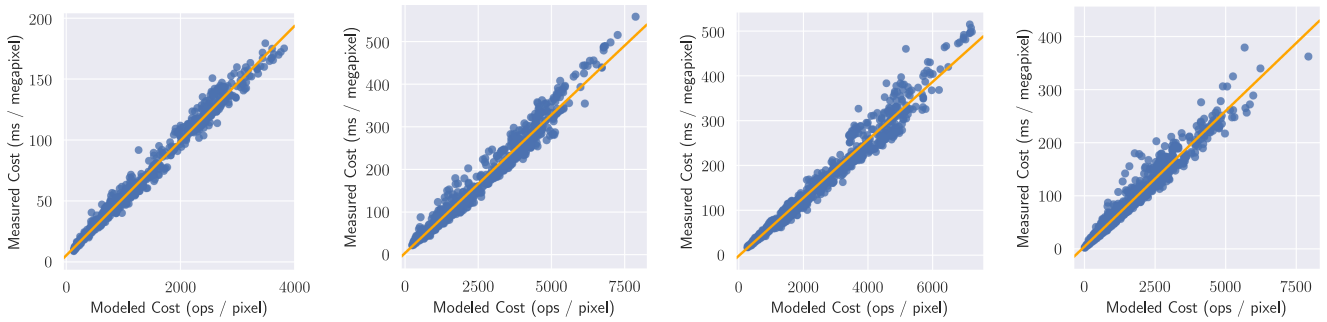


Fig. 9. Search space design ablation. Our domain-specific building blocks and search space factorization yield significant improvements to the achievable Pareto frontier. The gap in program quality between our search method with and without these domain-specific choices is significant, with a difference of up to 1 dB between programs with the same throughput, or a gap of up to 3× in throughput between programs of the same quality. The further gap in quality between our method without domain specific choices and a grid search over structurally pruned versions of Demosaicnet shows that our method of using genetic search over a space of programs represented as mutable DAGs is crucial as well.

4.1.4 Super-resolution. We also applied our method to the super-resolution task alone and compared against the same super-resolution models from the joint task excluding TENet and DNDMSR which are joint models. ESPCN is the best performing baseline out of those that operate near the regime of throughputs we consider in our search. Our method found a program that achieves the same PSNR as ESPCN while being nearly 50× faster and another program that is 1.45 dB better and almost 2× faster. SRCNN and ESPCN are significantly Pareto dominated by our models which shows that finding Pareto-dominant super-resolution models require more than just scaling down a deep convolutional neural network. The fastest deep neural network, FALSR-B, is 0.47 dB better than our best model but 90× slower. In Figure 14, we show the outputs of several other methods along with one of our Pareto dominant models. SRCNN tends to produce over-blurred outputs and ESPCN introduces false high frequencies as seen in



(a) Demosaicking from Bayer ($R^2 = 0.99$)
(b) Demosaicking from X-Trans ($R^2 = 0.98$)
(c) Super-resolution from Bayer ($R^2 = 0.97$)
(d) Super-resolution from RGB ($R^2 = 0.96$)

Fig. 10. Modeled vs. measured runtimes for the four tasks. Our search procedure is guided by estimated program costs from our cost model. It is crucial that these costs correlate well with the actual runtimes of our programs (or at least monotonically increase with respect to actual runtime) in order for our search to effectively explore the Pareto frontier of programs. Actual benchmarked runtimes of our programs show that there is a strong correlation between our estimated costs and true runtimes with R^2 ranging from 0.96 to 0.99 across all four applications. Our cost model is accurate in absolute terms, as well: one unit of cost translates to 1.6 CPU cycles per SIMD vector of output when compiled to AVX2 code on a single core of our Skylake X test machine.

the first, third, and sixth columns. SRCNN also produces zippering artifacts as shown in the third column.

4.2 Ablation: Search Space Design

We investigate the benefits of our two main design choices: (1) using domain specific building blocks and (2) modularizing the demosaicking task into chroma reconstruction guided by a green reconstruction sub-task. In Figure 9, we compare our search space to an alternative design, "Ours without domain-specific decisions", which is restricted to using only conventional NAS building blocks: convolution, ReLU, element-wise addition, and stacking. It is still allowed to mutate channel and group counts for convolutional operations. This alternative search space construction also does not factor the green reconstruction task from the red and blue reconstruction task. Note that this search space is larger than or equivalent to that considered by other NAS approaches like FALSR and LEMONADE, with the exception that we do not use BatchNorm (used by LEMONADE which searches for image classification models) because is not used for regression tasks like demosaicking and super-resolution. For fair comparison, this alternative design was run for 40 generations and we compare it to our method run for 20 generations for green reconstruction and another 20 for chroma reconstruction.

We also compare our search method to a grid search over variants of Demosaicnet with different layer and filter counts to see how our method compares to structured pruning. We do not compare to unstructured pruning because in unstructured pruning, removing parameters does not guarantee or directly translate to a faster model.

Figure 9 shows that our two search space design choices yield significant improvements to the Pareto frontier. The gap in program quality between our method and "Ours without domain-specific decisions" is significant, with a difference in favor or our method of up to 1 dB between programs with the same throughput, or up to 3× difference in throughput between programs of the same quality, indicating that a modularized search with domain specific building blocks is crucial for producing Pareto dominant programs.

The further gap in quality between "Ours without domain-specific decisions" and the grid search shows that even without domain specific intelligence, our method of using genetic search over a space of programs represented as mutable DAGs yields notable improvements to the frontier on its own. We limited the range of program throughputs that our search considers to stay within the real-time performance regime, and the grid search method explored a few models that were orders of magnitude more expensive than those considered by our search. Thus, the quality range of the grid search programs extends above those produced by our search. However, since the grid search's search space is a subset of our method's search space, our system can be easily configured to cover lower throughput ranges if desired, to expand the range of program quality. For the range of throughputs considered by our search, both our method and "Ours without domain-specific decisions" produce frontiers that significantly Pareto dominate structured pruning via grid search. Our domain specific search method produces programs that are up to 6.4× faster at the same quality or up to 1 dB better at the same throughput compared to the grid search models.

4.3 Correlation Between Program Cost and Runtime

Given that our search procedure is guided by program costs based on estimated floating point operations, it is important to know that these costs correlate well with the actual runtimes of our programs. Figure 10 shows a strong correlation between our estimated costs and actual benchmarked runtimes of our programs ($R^2 > 0.96$ for all tasks). Performance is also good in absolute terms: one unit of cost in our cost model translates to around 1.6 CPU cycles per SIMD vector of output when compiled to AVX2 code on a single core of our Skylake X test machine.

5 FUTURE WORK, LIMITATIONS, AND CONCLUSION

Our work opens exciting avenues for future work in synthesizing high-performance hybrid algorithms that combine aspects of deep learning and task-specific classical operations. For instance, for image processing, our method could potentially be extended to

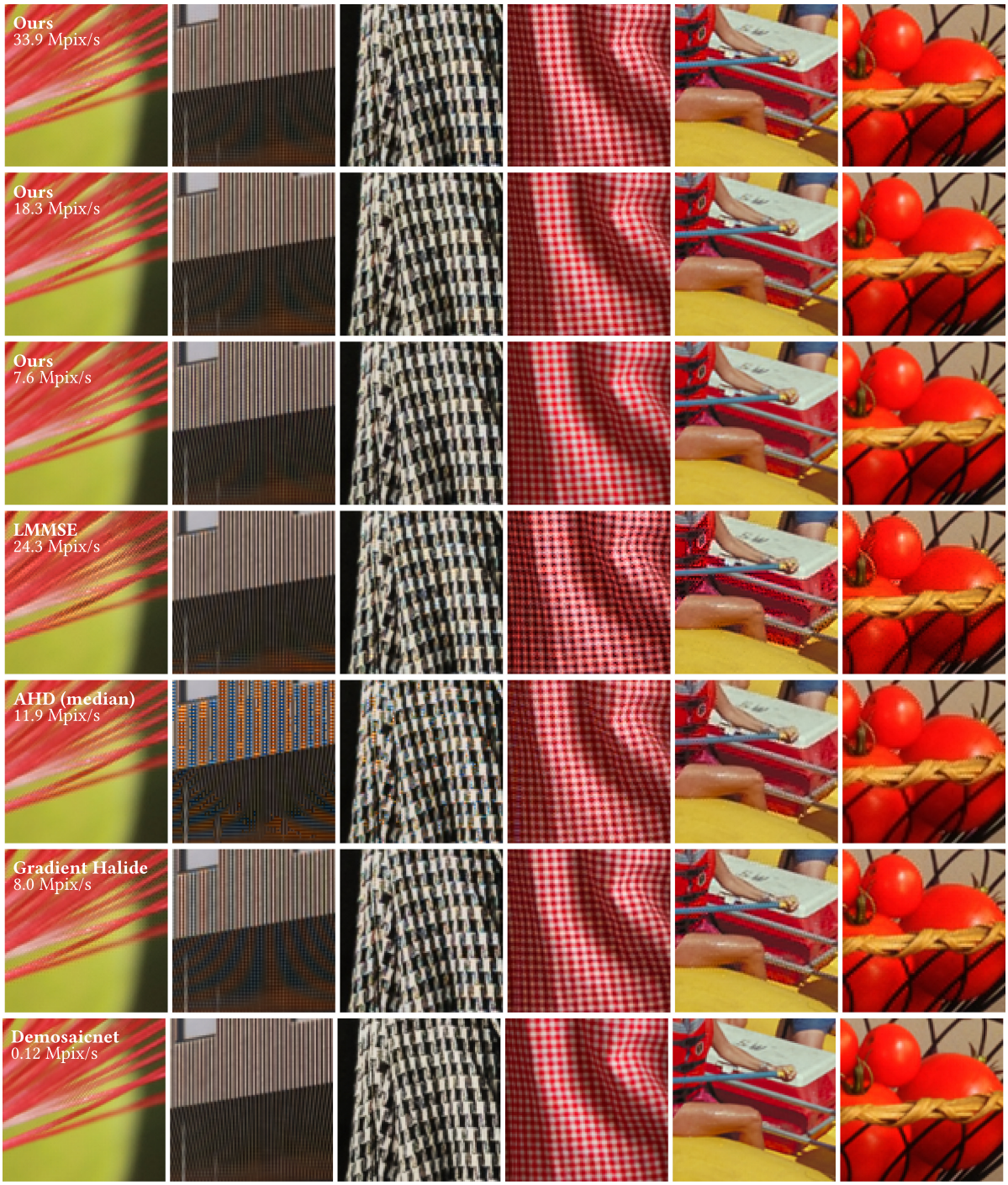


Fig. 11. Bayer demosaicking qualitative comparison. We show a selection of results from the Bayer demosaicking task for 3 algorithms we discovered, covering a range of computation budgets, and a selection of baselines. The Gradient Halide implementation uses 15 7×7 filters. We report the speed of each algorithm in megapixels per second (Mpix/s). Our programs avoid producing the artifacts seen in the baseline methods, such as zippering in the first column of LMMSE, AHD, and GradientHalide, color fringing and Bayer grid artifacts in the second column of AHD and GradientHalide, and the Bayer grid and zippering artifacts in the fifth and sixth columns of LMMSE. The output quality of our programs improves as their cost increases.

Table 1. Quantitative Comparison for the Four Tasks

	PSNR (dB)				Throughput (Mpixels/s)
Bayer Demosaicking	hdrvdp	moire	kodak	mcm	
Demosaicnet	33.73	35.65	40.75	39.05	0.116
Gradient Halide	30.53	32.26	37.47	35.72	7.82
VNG4	28.19	29.93	35.58	34.28	8.65
AHD	27.69	30.01	35.41	33.81	9.25
LMMSE	29.47	31.79	35.61	33.71	21.0
ours	31.05	32.98	38.05	36.28	33.9
ours	31.27	33.12	38.60	36.91	18.3
ours	31.74	33.31	39.25	37.50	7.6
X-Trans Demosaicking	hdrvdp	moire	kodak	mcm	
Demosaicnet	32.52	35.51	39.58	37.80	0.038
Markesteijn	28.35	30.79	36.38	34.58	6.78
ours	30.67	33.48	37.44	35.23	20.9
ours	31.27	34.15	38.13	36.14	10.89
ours	31.74	34.76	38.48	36.48	4.09

	PSNR (dB)				Throughput (Mpixels/s)
Super-resolution from Bayer	hdrvdp	moire	kodak	mcm	
Dnet+DRLN	24.83	27.75	32.56	32.02	0.002
Dnet+ESPCN	24.00	26.66	31.18	30.44	0.110
Dnet+FALSR-A	24.71	27.46	32.39	31.73	0.013
Dnet+FALSR-B	24.64	27.39	32.31	31.68	0.034
Dnet+FALSR-C	24.64	27.37	32.30	31.68	0.029
Dnet+ProSR	24.53	27.25	32.19	31.46	0.001
Dnet+RCAN	24.81	27.68	32.60	32.02	0.0039
Dnet+SRCNN	24.32	27.00	31.89	31.37	0.076
Dnet+Bicubic	23.49	26.16	30.84	30.12	0.116
GrHalide+Bicubic	22.95	25.53	30.10	28.99	7.70
GrHalide+ESPCN	23.25	25.74	30.03	28.85	1.80
GrHalide+SRCNN	23.45	26.00	30.56	29.36	0.217
TENet	24.73	27.23	32.59	32.12	0.005
DNDMSR	25.01	27.50	32.77	32.37	0.010
ours	23.69	26.30	31.05	30.05	18.0
ours	23.83	26.45	31.27	30.35	10.2
ours	24.01	26.54	31.48	30.60	5.26
ours	24.16	26.67	31.75	31.00	3.04
Super-resolution	bsd100	set14	set5	urban100	
DRLN	30.88	32.11	36.20	31.00	0.002
ESPCN	28.93	29.51	33.16	26.24	2.34
FALSR-A	30.57	31.27	35.50	29.40	0.015
FALSR-B	30.42	31.05	35.28	28.77	0.048
FALSR-C	30.41	31.05	35.32	28.76	0.038
ProSR	30.23	31.09	34.51	29.68	0.001
RCAN	30.86	32.04	36.20	30.91	0.0040
SRCNN	29.66	30.30	34.40	27.25	0.223
Bicubic	28.03	28.46	31.87	25.10	517
ours	29.72	30.43	34.55	27.25	19.0
ours	29.88	30.59	34.79	27.57	10.8
ours	29.93	30.64	34.83	27.70	7.90
ours	29.99	30.76	34.90	27.91	3.20

For all demosaicking tasks, we report PSNR on test images from the Gharbi et al. [2016], Kodak, and McMaster datasets for our baselines and several algorithms we discovered that span a range of runtime costs. For the superresolution task, we report PSNR on the test images from the standard BSD100, Set14, Set5, and Urban100 superresolution test datasets. Figures 11–14 show qualitative results for programs produced by our search compared to baseline algorithms.

cover other operations that have classical solutions such as multi-scale tone and detail transfer, photographic style transfer, dehazing [Chen et al. 2017], blurring and sharpening [Lou et al. 2016b], or matting [Li et al. 2019]. Our method could also be extended to explore whether the task-specific set of building blocks or primitives could be automatically extracted from example classical programs, unlike in this article's case where the building blocks are pre-specified and fixed.

One limitation and area for future work is that we have not yet explored fixed-point quantization of the network weights [Lin et al. 2016], which would further improve throughput, and would be necessary for implementation on a DSP or imaging ASIC.

In conclusion, we have presented a discrete and continuous search for demosaicking and super-resolution algorithms that are able to synthesize pipelines that Pareto-dominate state-of-the-art algorithms when both quality *and* performance matter. These algorithms lower to highly-efficient SIMD code and combine the benefits of classical and deep methods. We believe our approach opens up an important new direction for task-specific learning via program search.

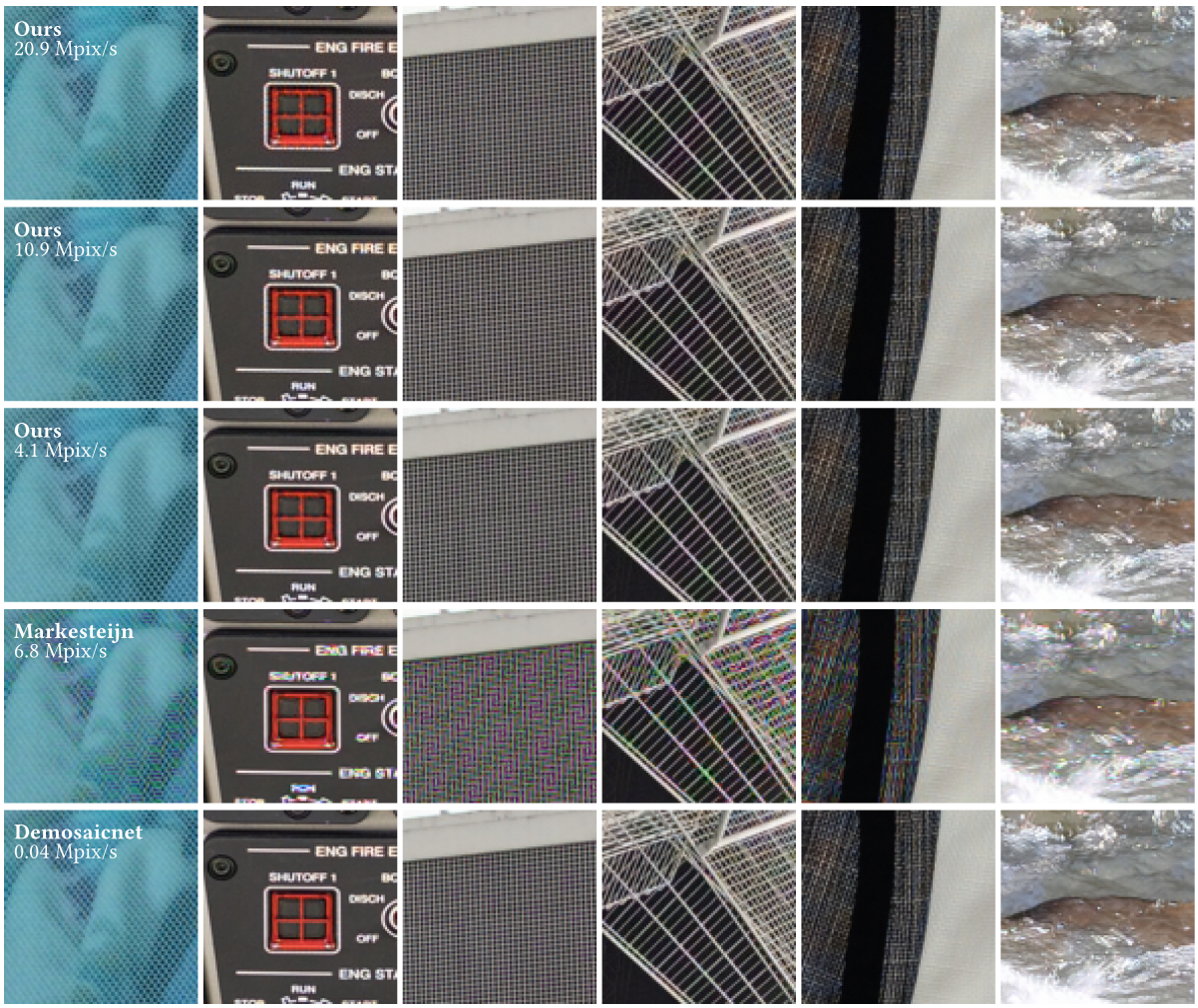


Fig. 12. X-Trans demosaicking qualitative comparison. We show a selection of outputs from the X-Trans demosaicking task for 3 algorithms we discovered, covering a range of computation budgets, as well as the outputs of the Markesteijn algorithm and the X-Trans Demosaicnet. We report the speed of each algorithm in megapixels per second (Mpix/s) Our programs avoid the artifacts produced by the Markesteijn algorithm such as the moire artifacts in the first and third column, the color fringing in columns 2 through 6, and the maze pattern in the third column while being up to 3× faster. Our programs obtain visual quality close to that of Demosaicnet, which is 100× more expensive than our most expensive program.

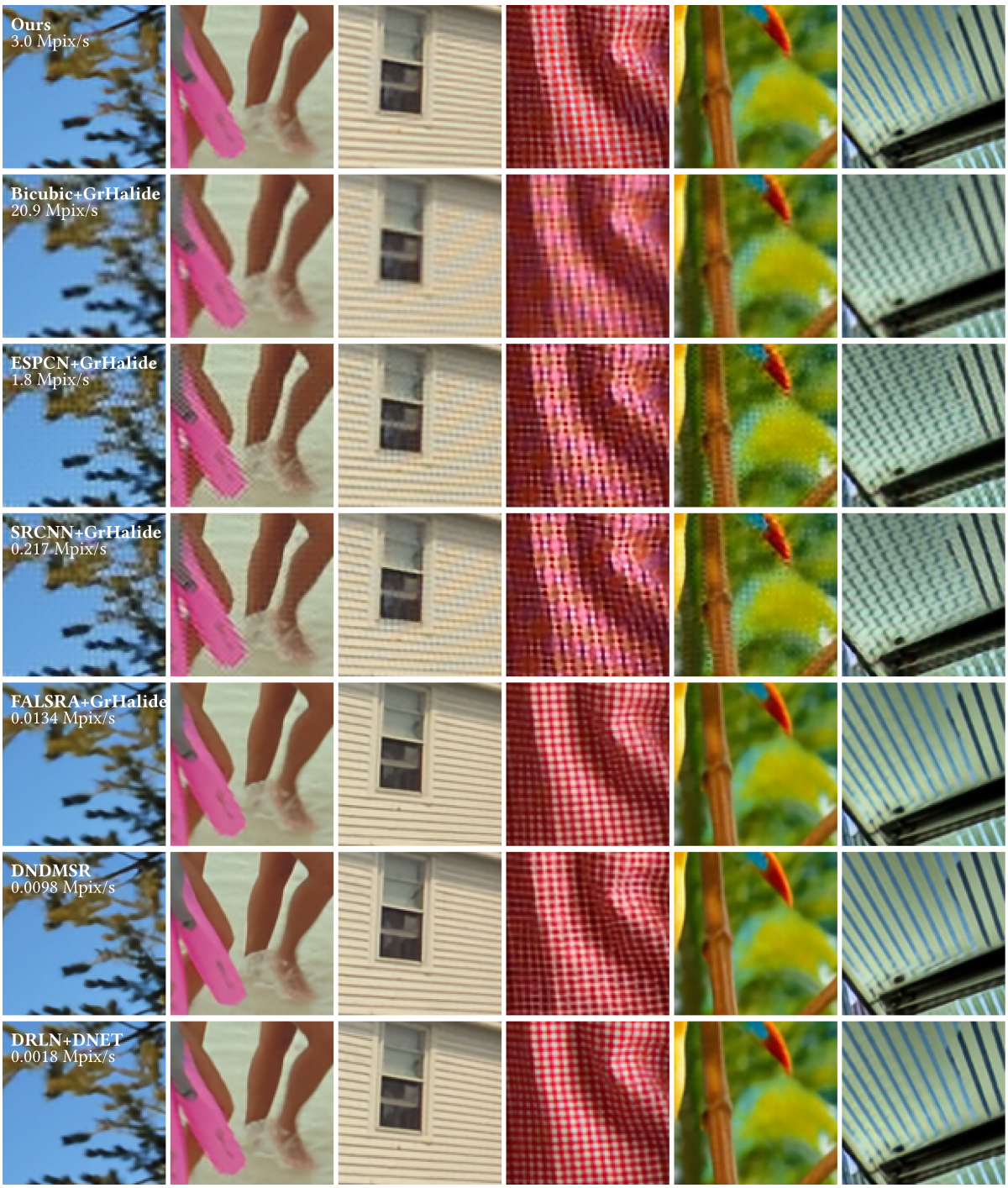


Fig. 13. Joint demosaicking with super-resolution qualitative comparison. We show a selection of results on the joint demosaicking and super-resolution task from a program we discovered and a selection of baselines. The Gradient Halide implementation uses 15 7×7 filters. We report the speed of each algorithm in megapixels per second (Mpix/s). Our program is 1.5× faster than ESPCN+GradientHalide and 15× faster than SRCNN+GradientHalide while avoiding the high-frequency speckling seen in the first and fifth columns, the color fringing in the third column, and the zippering artifacts in the last column of Bicbuic+GradientHalide, ESPCN+GradientHalide, and SRCNN+GradientHalide.

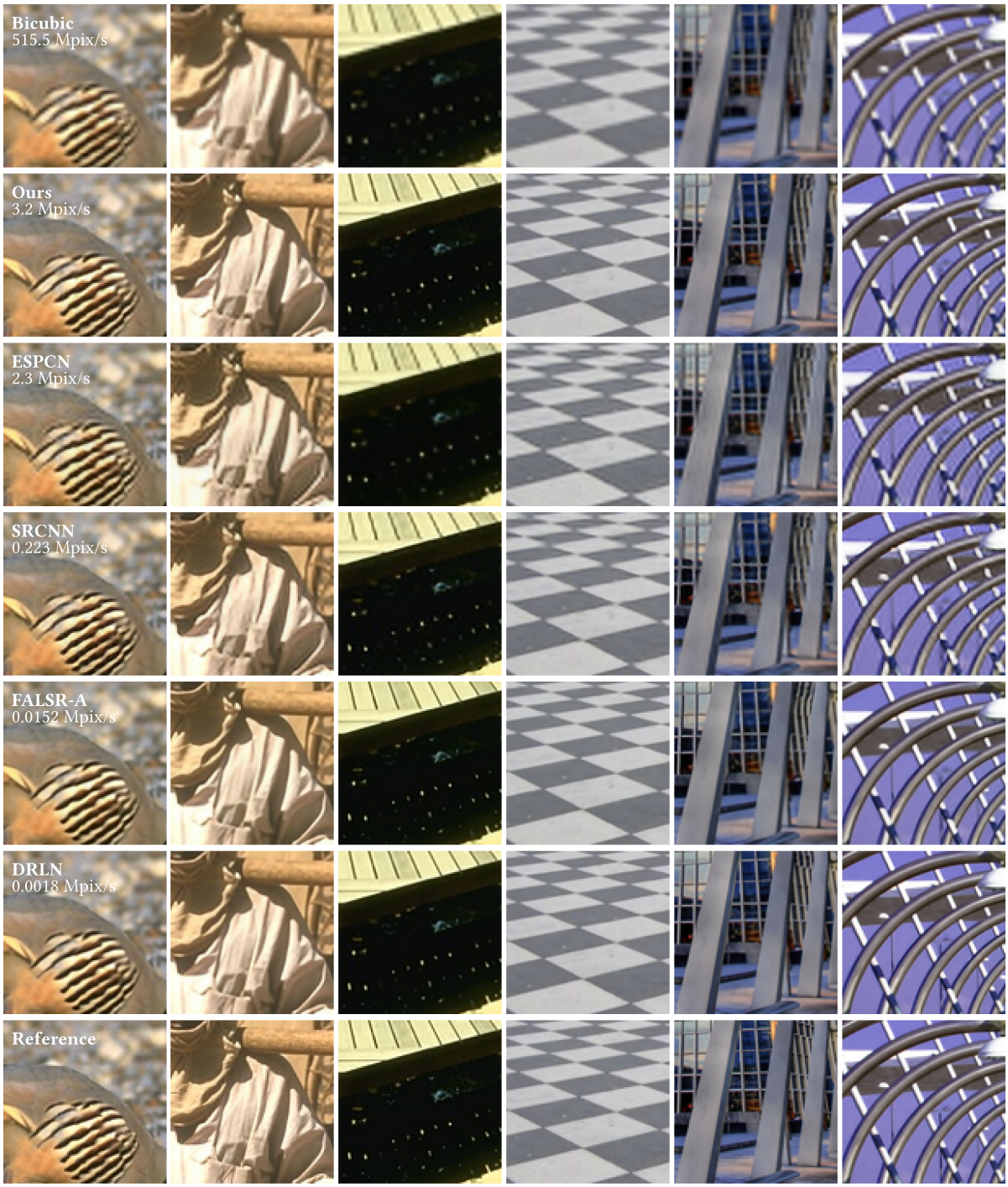


Fig. 14. Super-resolution qualitative comparison. We show a selection of results on the super-resolution task from a program we discovered and a selection of baseline algorithms. We report the speed of each algorithm in megapixels per sec (Mpix/s). SRCNN tends to produce over-blurred outputs and ESPCN introduces false high frequencies as seen in the first, third, and sixth columns. SRCNN also produces zippering artifacts as shown in the third column. For nearly 300× faster throughput our program's output quality approaches that of FALSR-A, a large expensive deep neural network.

ACKNOWLEDGMENTS

We thank Dillon Sharlet for providing valuable inspiration for the design of building blocks for demosaicking, and the design considerations of production demosaickers.

REFERENCES

Andrew Adams, Karima Ma, Luke Anderson, Riyadh Baghdadi, Tzu-Mao Li, Michaël Gharbi, Benoit Steiner, Steven Johnson, Kayvon Fatahalian, Frédo Durand, and Jonathan Ragan-Kelley. 2019. Learning to optimize halide with tree search and random programs. *ACM Transactions on Graphics* 38, 4 (2019), 12 pages. DOI: https://doi.org/10.1145/3306346.3322967

James E. Adams Jr. 1995. Interactions between color plane interpolation and other image processing functions in electronic photography. In *Proceedings of the Cameras and Systems for Electronic Photography and Scientific Imaging*. 144–151.

David Alleysson, Sabine Susstrunk, and Jeanny Hérault. 2005. Linear demosaicing inspired by the human visual system. *IEEE Transactions on Image Processing* 14, 4 (2005), 439–449.

A. Anderson, J. Su, Rozenn Dahyot, and D. Gregg. 2019. Performance-oriented neural architecture search. In *Proceedings of the International Conference on High Performance Computing & Simulation*. 177–184.

Saeed Anwar and Nick Barnes. 2020. Densely residual laplacian super-resolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 3 (2022), 1192–1204. https://doi.org/10.1109/TPAMI.2020.3021088

Eric P. Bennett, Matthew Uyttendaele, C. Lawrence Zitnick, Richard Szeliski, and Sing Bing Kang. 2006. Video and image Bayesian demosaicing with a two color image prior. In *Proceedings of the Computer Vision – ECCV 2006*. Aleš Leonardis, Horst Bischof, and Axel Pinz (Eds.), Springer Berlin, 508–521.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. 2020. What is the state of neural network pruning? In *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*, Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze (Eds.). mlsys.org. https://proceedings.mlsys.org/book/296.pdf.

Antoni Buades, Bartomeu Coll, Jean-Michel Morel, and Catalina Sbert. 2009. Self-similarity driven color demosaicking. *IEEE Transactions on Image Processing* 18, 6 (2009), 1192–1202.

CarVac. 2021. librtprocess Github Page. Retrieved 31 March 2022 from https://github.com/CarVac/librtprocess.

Kan Chang, Pak Lun Kevin Ding, and Baoxin Li. 2015. Color image demosaicking using inter-channel correlation and nonlocal self-similarity. *Signal Processing: Image Communication* 39, PA (2015), 264–279.

Qifeng Chen, Jia Xu, and Vladlen Koltun. 2017. Fast image processing with fully-convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*.

Xiangxiang Chu, Bo Zhang, Hailong Ma, R. Xu, Jixiang Li, and Qingyuan Li. 2021b. Fast, accurate and lightweight super-resolution with neural architecture search. In *Proceedings of the 2020 25th International Conference on Pattern Recognition*. 59–64.

Xiangxiang Chu, Bo Zhang, Hailong Ma, Ruijun Xu, and Qingyuan Li. 2021a. Fast, accurate and lightweight super-resolution with neural architecture search. In *Proceedings of the 2020 25th International Conference on Pattern Recognition*. IEEE, 59–64.

David R. Cok. 1987. Signal processing method and apparatus for producing interpolated chrominance values in a sampled color image signal. US Patent 4,642,678. Accessed 31 March 2022.

Laurent Condat and Saleh Mosaddegh. 2012. Joint demosaicking and denoising by total variation minimization. In *Proceedings of the International Conference on Image Processing*. IEEE, 2781–2784.

Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2014. Learning a deep convolutional network for image super-resolution. In *Proceedings of the Computer Vision – ECCV 2014*. Springer International Publishing, Cham, 184–199.

Eric Dubois. 2005. Frequency-domain methods for demosaicking of Bayer-sampled color images. *Signal Processing Letters* 12, 12 (2005), 847–850.

Joan Duran and Antoni Buades. 2014. Self-similarity and spectral correlation adaptive algorithm for color demosaicking. *IEEE Transactions on Image Processing* 23, 9 (2014), 4031–4040.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018. Efficient multi-objective neural architecture search via lamarckian evolution. In *Proceedings of the 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=ByME42AqK7.

Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. 2019. Neural architecture search: A survey. *The Journal of Machine Learning Research* 20, 55 (2019), 1–21. http://jmlr.org/papers/v20/18-598.html.

Sina Farsiu, Michael Elad, and P. Milanfar. 2004. Multiframe demosaicing and super-resolution from undersampled color images. In *Proceedings of the IS&T/SPIE Electronic Imaging*.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=rJl-b3RcF7.

Pascal Getreuer. 2011. Color demosaicing with contour stencils. In *Proceedings of the International Conference on Digital Signal Processing*. IEEE, 1–6.

Michaël Gharbi, Gaurav Chaurasia, Sylvain Paris, and Frédo Durand. 2016. Deep joint demosaicking and denoising. *ACM Transactions on Graphics* 35, 6 (2016), 191:1–191:12.

Jinwook Go, Kwanghoon Sohn, and Chulhee Lee. 2000. Interpolation using neural networks for digital still cameras. *Transactions on Consumer Electronics* 46, 3 (2000), 610–616.

Chengyue Gong, Zixuan Jiang, Dilin Wang, Yibo Lin, Qiang Liu, and David Z. Pan. 2019. Mixed precision neural architecture search for energy efficient deep learning. In *Proceedings of the International Conference on Computer-Aided Design*. IEEE, 1–7.

John F. Hamilton Jr and James E. Adams Jr. 1997. Adaptive color plan interpolation in single sensor color electronic camera. US Patent 5,629,734. Accessed 31 March 2022.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*. 770–778.

Yong He, Tim Foley, Natalya Tatarchuk, and Kayvon Fatahalian. 2015. A system for rapid, automatic shader level-of-detail. *ACM Transactions on Graphics* 34, 6 (2015), 1–12.

Felix Heide, Markus Steinberger, Yun-Ta Tsai, Mushfiqur Rouf, Dawid Pająk, Dikpal Reddy, Orazio Gallo, Jing Liu, Wolfgang Heidrich, Karen Egiazarian, Jan Kautz, and Kari Pulli. 2014. FlexISP: A flexible camera image processing framework. *ACM Transactions on Graphics* 33, 6 (2014), 231:1–231:13.

Bernardo Henz, Eduardo S. L. Gastal, and Manuel M. Oliveira. 2018. Deep joint design of color filter arrays and demosaicing. *Computer Graphics Forum* 37, 7 (2018), 389–399.

Robert H. Hibbard. 1995. Apparatus and method for adaptively interpolating a full color image utilizing luminance gradients. US Patent 5,382,976. Accessed 31 March 2022.

Keigo Hirakawa and Thomas W. Parks. 2005. Adaptive homogeneity-directed demosaicing algorithm. *IEEE Transactions on Image Processing* 14, 3 (2005), 360–369.

Keigo Hirakawa and Thomas W. Parks. 2006. Joint demosaicing and denoising. *IEEE Transactions on Image Processing* 15, 8 (2006), 2146–2157.

Oren Kapah and Hagit Zabrodsky Hel-Or. 2000. Demosaicking using artificial neural networks. In *Proceedings of the Applications of Artificial Neural Networks in Image Processing*. International Society for Optics and Photonics, 112–120.

Daniel Khashabi, Sebastian Nowozin, Jeremy Jancsary, and Andrew W. Fitzgibbon. 2014. Joint demosaicing and denoising via learned nonparametric random fields. *IEEE Transactions on Image Processing* 23, 12 (2014), 4968–4981.

Daisuke Kiku, Yusuke Monno, Masayuki Tanaka, and Masatoshi Okutomi. 2013. Residual interpolation for color image demosaicking. In *Proceedings of the International Conference on Image Processing*. IEEE, 2304–2308.

Teresa Klatzer, Kerstin Hammernik, Patrick Knobelreiter, and Thomas Pock. 2016. Learning joint demosaicing and denoising based on sequential energy minimization. In *Proceedings of the International Conference on Computational Photography*. IEEE, 1–11.

Filippos Kokkinos and Stamatios Lefkimmiatis. 2018. Deep image demosaicking using a cascade of convolutional residual denoising networks. In *Proceedings of the European Conference on Computer Vision*. 303–319.

Filippos Kokkinos and Stamatios Lefkimmiatis. 2019. Iterative joint image demosaicking and denoising using a residual denoising network. *IEEE Transactions on Image Processing* 28, 8 (2019), 4177–4188.

John R. Koza and John R. Koza. 1992. *Genetic Programming: On the Programming of Computers By Means of Natural Selection*. MIT Press.

Cindy Kwan and Xiaolin Wu. 2004. A classification approach to color demosaicking. In *Proceedings of the International Conference on Image Processing*. IEEE, 2415–2418.

Tzu-Mao Li, Michaël Gharbi, Andrew Adams, Frédo Durand, and Jonathan Ragan-Kelley. 2018. Differentiable programming for image processing and deep learning in Halide. *ACM Transactions on Graphics* 37, 4 (2018), 139:1–139:13.

Xin Li, Bahadir Gunturk, and Lei Zhang. 2008. Image demosaicing: A systematic survey. In *Proceedings of the Visual Communications and Image Processing*. International Society for Optics and Photonics, 68221J.

Xiaoqiang Li, Jide Li, and Hong Lu. 2019. A survey on natural image matting with closed-form solutions. *IEEE Access* 7 (2019), 136658–136675.

Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *Proceedings of the International Conference on Machine Learning*. PMLR, 2849–2858.

Lin Liu, Xu Jia, Jianzhuang Liu, and Qi Tian. 2020. Joint demosaicing and denoising with self guidance. In *Proceedings of the Computer Vision and Pattern Recognition*.

Liming Lou, Paul Nguyen, Jason Lawrence, and Connelly Barnes. 2016a. Image perforation: Automatically accelerating image pipelines by intelligently skipping samples. *ACM Transactions on Graphics* 35, 5 (2016), 1–14.

Liming Lou, Paul Nguyen, Jason Lawrence, and Connelly Barnes. 2016b. Image perforation: Automatically accelerating image pipelines by intelligently skipping samples. *ACM Transactions on Graphics* 35, 5 (2016), 1–14.

Daniele Menon and Giancarlo Calvagno. 2009. Joint demosaicking and denoising with space-varying filters. In *Proceedings of the International Conference on Image Processing*. IEEE, 477–480.

Yan Niu, Jihong Ouyang, Wanli Zuo, and Fuxin Wang. 2018. Low cost edge sensing for high quality demosaicking. *IEEE Transactions on Image Processing* 28, 5 (2018), 2415–2427.

Guocheng Qian, Jinjin Gu, Jimmy S. J. Ren, Chao Dong, Furong Zhao, and Juan-Ting Lin. 2019. Trinity of pixel enhancement: A joint solution for demosaicking, denoising and super-resolution. arXiv:1905.02538. Retrieved from https://arxiv.org/abs/1905.02538.

Jonathan Ragan-Kelley, Andrew Adams, Sylvain Paris, Marc Levoy, Saman Amarasinghe, and Frédo Durand. 2012. Decoupling algorithms from schedules for easy optimization of image processing pipelines. *ACM Transactions on Graphics* 31, 4 (2012), 32:1–32:12.

Sivalogeswaran Ratnasingam. 2019. Deep camera: A fully convolutional neural network for image signal processing. In *Proceedings of the International Conference on Computer Vision Workshops*.

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 2016 IEEE Conference on Computer Vision and Pattern Recognition*. 1874–1883. DOI: https://doi.org/10.1109/CVPR.2016.207

Pitchaya Sitthi-Amorn, Nicholas Modly, Westley Weimer, and Jason Lawrence. 2011. Genetic programming for shader simplification. *ACM Transactions on Graphics* 30, 6 (2011), 1–12.

Daniel Stanley Tan, Wei-Yang Chen, and Kai-Lung Hua. 2018. DeepDemosaicking: Adaptive image demosaicking via multiple deep fully convolutional networks. *IEEE Transactions on Image Processing* 27, 5 (2018), 2408–2419.

Hanlin Tan, Xiangrong Zeng, Shiming Lai, Yu Liu, and Maojun Zhang. 2017a. Joint demosaicing and denoising of noisy Bayer images with ADMM. In *Proceedings of the International Conference on Image Processing*. IEEE, 2951–2955.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2820–2828.

Runjie Tan, Kai Zhang, Wangmeng Zuo, and Lei Zhang. 2017b. Color image demosaicking via deep residual learning. In *Proc. IEEE I International Conference on Multimedia & Expo*. 793–798.

The RawTherapee Team. 2021. RawTherapee Home Page. Retrieved 31 March 2022 from https://rawtherapee.com.

Rui Wang, Xianjin Yang, Yazhen Yuan, Wei Chen, Kavita Bala, and Hujun Bao. 2014. Automatic shader simplification using surface signal approximation. *ACM Transactions on Graphics* 33, 6 (2014), 1–11.

Yifan Wang, Federico Perazzi, Brian McWilliams, Alexander Sorkine-Hornung, Olga Sorkine-Hornung, and Christopher Schroers. 2018. A fully progressive approach to single-image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1492–1500.

Wenzhu Xing and Karen Egiazarian. 2021. End-to-end learning for joint image demosaicing, denoising and super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3507–3516.

Jianchao Yang and Thomas Huang. 2017. Image super-resolution: Historical overview and future challenges. *Super-Resolution Imaging*. CRC Press. DOI: https://doi.org/10.1201/9781439819319

Lei Zhang, Rastislav Lukac, Xiaolin Wu, and David Zhang. 2009. PCA-based spatially adaptive denoising of CFA images for single-sensor digital cameras. *IEEE Transactions on Image Processing* 18, 4 (2009), 797–812.

Lei Zhang and Xiaolin Wu. 2005. Color demosaicking via directional linear minimum mean square-error estimation. *IEEE Transactions on Image Processing* 14, 12 (2005), 2167–2178.

Lei Zhang, Xiaolin Wu, Antoni Buades, and Xin Li. 2011. Color demosaicking by local directional interpolation and nonlocal adaptive thresholding. *Journal of Electronic Imaging* 20, 2 (2011), 023016.

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*.

Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. 2018. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2423–2432.

Yanqi Zhou and Gregory Diamos. 2018. Neural architect: A multi-objective neural architecture search with performance prediction. In *Proceedings of the Conference on SysML*. 1–3.

Barret Zoph and Quoc V. Le. 2016. Neural architecture search with reinforcement learning. In *Proceedings of the 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017*. OpenReview.net. https://openreview.net/forum?id=r1Ue8Hcxg.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8697–8710.

Received August 2021; revised December 2021; accepted December 2021