A Green Learning Approach to Efficient Image Demosaicking

Mahtab Movahhedrad
University of Southern California
Los Angeles, California, USA
movahhed@usc.edu

Zijing Chen
University of Southern California
Los Angeles, California, USA
zijinch@usc.edu

C.-C. Jay Kuo
University of Southern California
Los Angeles, California, USA
jckuo@usc.edu

2024 IEEE International Conference on Big Data (BigData) | 979-8-3503-6248-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/BigData62323.2024.10825194

***Abstract*—Demosaicking is a critical process in the digital imaging pipeline, tasked with reconstructing full-color images from sampled data captured by R/G/B color sensors. The challenge arises from two-thirds of the pixel data being missing, which complicates the task of accurate reconstruction. Recent deep learning-based solutions have yielded considerable advancements in demosaicking performance. However, they are computationally intensive and rely on large model architectures, rendering them unsuitable for edge-devices deployment. This work introduces a novel demosaicking method to address these challenges based on green learning (GL), named green image demosaicking (GID). GID offers model transparency while significantly reducing the model size and computational complexity against deep learning methods. Notably, GID does not utilize neural networks. Instead, it is built upon unsupervised representation learning and supervised feature dimension reduction. GID effectively addresses the challenges of big data in vision applications and enhances predictive accuracy during decision-making. GID is engineered for rapid execution with parallel training, making it well-suited for real-time vision tasks on resource-constrained devices.**

***Index Terms*—demosaicking, green learning, dimension reduction, representation learning, edge devices.**

I. Introduction

An image sensor at each pixel location only detects a single color (typically red, green, or blue color channels). Digital cameras use the color filter array (CFA) to arrange R/G/B sensors for color image capturing. The Bayer array is the most popular CFA. It has two green sensors, one red sensor, and one blue sensor in a squared region of (2×2) pixels. Consequently, the raw sensor data only contains one-third of the color information. A process known as demosaicking is employed to reconstruct a complete color image from these partial data. The most straightforward solutions, such as bilinear and bicubic interpolation, for estimating missing color values often yield suboptimal results. As an alternative, heuristic methods that leverage the smoother nature of hue compared to individual color components are frequently employed in color image processing [1].

While these non-adaptive algorithms can yield acceptable results in smooth regions of an image, they tend to produce artifacts in more complex areas, particularly around textures and edges. Adaptive directional interpolation methods reconstruct images by aligning the interpolation process with the edges rather than crossing them to address these challenges. These techniques employ gradients or Laplacian operators to identify horizontal and vertical edges, ensuring that the reconstruction follows the natural contours of the image. [2]–[4]

A growing trend in image demosaicking is the application of deep learning techniques, which have now established new benchmarks in performance [5]–[7]. Despite their success, architectures such as convolutional neural networks and deep learning methods are inherently complex, comprising numerous layers and neurons that demand significant computational resources. This complexity challenges edge devices, which typically have limited processing capabilities compared to data centers or cloud-based GPUs/TPUs. Deploying these large and intricate models on edge devices can be computationally expensive and slow, often resulting in rapid battery depletion—a critical issue for mobile and portable devices.

Furthermore, deep learning models are frequently overparameterized to capture a wide array of patterns from training data, which, while enhancing performance, also leads to increased model sizes. The limited storage capacity of edge devices exacerbates the challenge of locally storing these large models. In cases where edge devices must offload processing tasks to a server or the cloud, the large model sizes contribute to significant data transfer costs and latency due to bandwidth constraints.

To overcome these limitations, we introduce a lightweight green learning (GL) [8] demosaicking approach tailored for edge devices. Unlike traditional deep learning methods, Green Learning operates independently of neural networks. Our proposed model is structured into three distinct stages:

Data Processing: In this initial stage, we utilize Menon et al. [9] interpolation to produce a preliminary estimation of the RGB pixel values. Additionally, We categorize each channel into three subchannels based on their respective positions within the CFA array, which significantly enhances the accuracy of predictions during the learning stage.

Feature Processing: This stage consists of three interconnected modules aligned with green learning pipelines:

1) **Unsupervised Representation Learning:** This typically involves subspace approximation techniques such as the Saab (Subspace Approximation with Adjusted Bias) [10] transform, or the Successive Subspace Learning (SSL) [11] method to extract representations from raw data efficiently.

2) **Supervised Feature Selection:** This module implements the Discriminant Feature Test (DFT) [12] and the Relevant Feature Test (RFT) [12] to identify and select pertinent features from the high-dimensional learned representations.
3) **Complementary Feature Generation:** In this module, features are generated based on the Least-Squares Normal Transform (LNT) [13], providing complementary features that further strengthen the feature space.

Supervised Decision Learning : This stage features a core predictor for residual learning, complemented by an auxiliary predictor that enhances the performance of the core model using three class-specific regressors. An illustration of these three stages is shown in Fig 1.

Our approach, characterized by its green and data-driven nature, is lightweight and transparent. Moreover, due to its minimal FLOPs, it enables rapid inference. Despite its simplicity, the method achieves results comparable to deep learning models.

II. Related Work

Demosaicking

Demosaicking is a fundamental and extensively studied problem in image processing. The most traditional demosaicking techniques often apply a fixed filtering approach across all pixels, which usually performs adequately in flat regions of an image where interpolation is relatively straightforward. However, issues such as Moiré patterns and over-smoothing can still arise even in these flat regions. Gradient-based methods typically estimate edge directions and adjust the interpolation formulas so that filtering is performed preferentially along edge directions [2], [4]. In extend, variable number of gradients, as discussed in [14]. Combining interpolation results from horizontal and vertical directions is introduced in [15] color correlations, and edge-based methods are also addressed in [16]–[18]. Demosaicking can also be viewed as an optimization problem. The use of image priors is explored within optimization frameworks [19], [20]. Recent methods leverage convolutional neural networks [5]–[7], [21]–[23]. Demosaicking by Multiple Sub-Networks has also been studied in several papers. Using multiple sub-networks is studied in [20], [24].

While deep learning techniques deliver cutting-edge performance, they are still resource-intensive in terms of computation and model size. Furthermore, most of the works mentioned do not address demosaicking as an isolated problem; instead, they combine denoising with demosaicking in a joint approach.

Green Learning

Green Learning, introduced by Kuo and Madni in 2023 [8], represents a significant change from the deep learning methods commonly used in modern AI. This new approach focuses on more efficient machine learning techniques, aiming to address the growing computational demands of traditional AI systems.

A key feature of GL is moving away from backpropagation, which is essential in training neural networks. Instead, GL utilizes unsupervised feature extraction methods, such as the Saab Transform [10] or the channel-wise Saab Transform [11]. This change allows for more efficient data processing by extracting various features without the heavy computational load associated with backpropagation.

To further improve its effectiveness, GL incorporates advanced feature selection techniques, like the discriminant feature test (DFT) and the relevant feature test (RFT) [12]. These techniques are essential for identifying and using only the most pertinent features of model training, which helps optimize the training process and enhance the performance of the final models.

Green Learning uses several advanced algorithms to train these selected features, including XGBoost [25] and SLM [26]. Each algorithm brings unique strengths, enabling GL to adapt to different datasets and tasks, resulting in a flexible and robust decision-making process.

A major advantage of Green Learning is its operational efficiency, mainly because it avoids backpropagation and end-to-end training. It reduces the computational requirements and makes the framework more scalable and applicable to various fields.

III. Proposed method

A. Data Preprocessing

We start by processing the data derived from the color sensors using the interpolation algorithm proposed by Menon et al. [9]. This approach initially reconstructs the green channel at full resolution through edge-directed interpolation. Subsequently, the red and blue channels are estimated using the information from the reconstructed green channel. The interpolated RGB pixels are then classified into 6 groups based on their placement within the CFA array. Fig 2 illustrates the detail of this classification.

$$\text{R:}\begin{cases} R_0 : \text{R at the location of R sensor.} \\ R_1 : \text{R at the location of G sensor.} \\ R_2 : \text{R at the location of B sensor.} \end{cases}$$

$$\text{G:}\begin{cases} G_0 : \text{G at the location of G sensor.} \\ G_1 : \text{G at the location of R sensor.} \\ G_2 : \text{G at the location of B sensor.} \end{cases}$$

$$\text{B:}\begin{cases} B_0 : \text{B at the location of B sensor.} \\ B_1 : \text{B at the location of G sensor.} \\ B_2 : \text{B at the location of R sensor.} \end{cases}$$

The arrangement of R and B sensors at the locations of G sensors, R_1, B_1, is different based on whether they are placed in odd or even rows. However, this can be easily addressed by rotating the even rows by 90°. Since R_0, G_0, and B_0 are directly captured by the CFA sensors, they do not require any additional training. Consequently, each channel can be divided into two subchannels, R_1, R_2, G_1, G_2, B_1,B_2, which will be

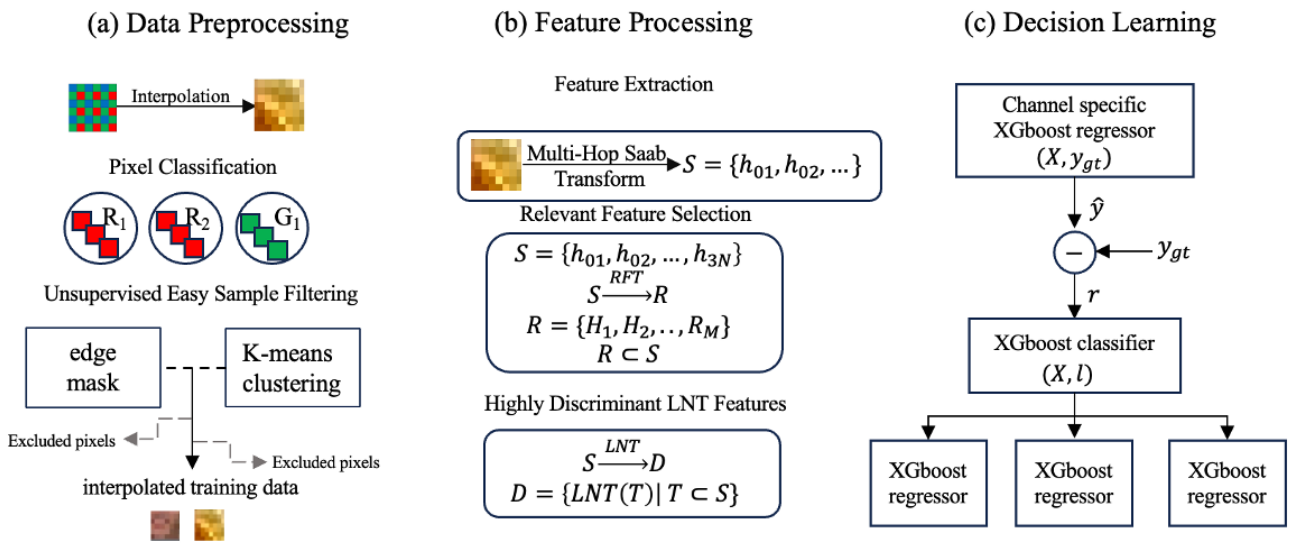

Fig. 1. An overview of the proposed pipeline in three stages: (a) Data Preprocessing, where initial interpolation is performed, pixels are divided into subchannels according to their positions within the CFA array, and easily predictable samples are excluded from subsequent training; (b) Feature Processing, which involves deriving feature representations, selecting relevant features, and augmenting with complementary discriminative features; and (c) Decision-Making, where two predictors—core and auxiliary—are employed to estimate the residuals of the interpolated values.

processed independently through the pipeline. This channel-wise, CFA-based approach ensures the consistency of representations derived from the directionally interpolated images. Another key consideration is that not all pixels require training.

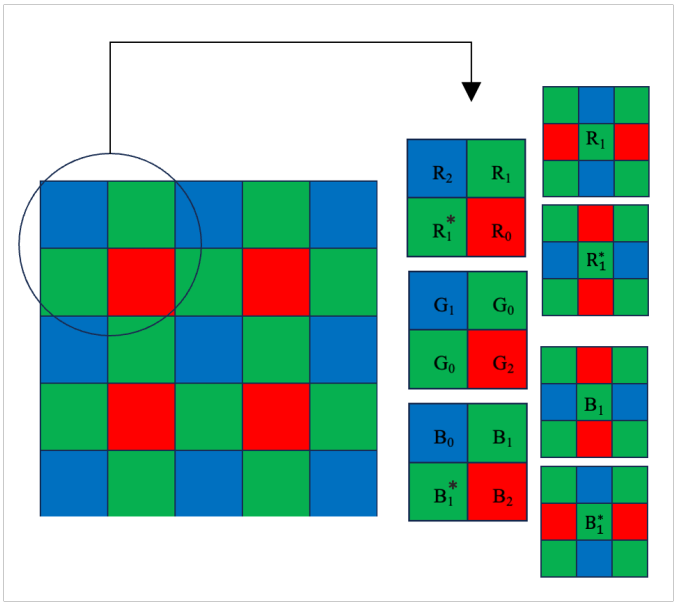

Fig. 2. Subchannels based on the CFA placement, $R_1{}^*$ and $B_1{}^*$, undergo a 90° rotation to align with the R_1 and B_1 arrangement.

Training can be bypassed for pixels in smooth regions, where interpolation is straightforward. To exclude these patches, we employ two cascaded modules. First, a patch consisting of its neighboring pixels is considered for each pixel. Patches with no edges, determined using Canny edge detection [27] within certain thresholds, are discarded. The remaining patches then undergo k-means clustering, where data with low MSE for interpolated values is classified as easy clusters. This process effectively filters out easy samples, enhancing the model's robustness.

B. Feature Processing

Representation Learning

We use a multi-hop Saab transform to obtain the latent representation space, capturing the spatial/spectral characteristics of the input patches. At the first hop for a target pixel, the local surrounding neighborhood forms an input tensor $\mathbf{v} \in \mathbb{R}^{n \times n}$, which we apply a Saab filter of size $m \times m$, $m < n$, to it. A Saab transform is a variation of the PCA transform that conducts eigenvalue analysis and projects the input tensor into an orthogonal subspace. Each hop can be expressed as follows:

$$z_k = \mathbf{u}_k^\top \mathbf{v} + c_k, \quad k = 0, 1, \ldots, K-1, \tag{1}$$

where $\mathbf{u}_k$ denotes the k-th principal component, and c_k represents a bias term associated with the k-th principal component score. Considering the set of principal components $\mathbf{U} = \{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{K-1}\}$, the projection of the input vector $\mathbf{v}$ onto $\mathbf{u}_0$ captures the overall magnitude across all dimensions.

The resulting subspaces can be decomposed into the direct sum of two orthogonal subspaces as follows:

$$\mathcal{Z} = \mathcal{Z}_{\text{DC}} \oplus \mathcal{Z}_{\text{AC}}, \tag{2}$$

where $\mathcal{Z}_{\text{DC}}$ corresponds to the subspace spanned by $\mathbf{u}_0 \in \mathbf{U}$, and $\mathcal{Z}_{\text{AC}}$ corresponds to the subspace spanned by the set $\{\mathbf{u} \in \mathbf{U} \mid \mathbf{u} \neq \mathbf{u}_0\}$. By subtracting the component $\mathbf{v}_{\text{DC}}$, which results from projecting the input $\mathbf{v}$ onto the subspace $\mathcal{Z}_{\text{DC}}$,

from the input tensor, we obtain the following expression for the $\mathbf{v}_{\text{AC}}$ component:

$$\mathbf{v}_{\text{AC}} = \sum_{k=1}^{k-1} \left(\mathbf{u}_k^{\top} \left(\mathbf{v} - \mathbf{v}_{\text{DC}} \right) + c_k \right). \tag{3}$$

Subsequently, the eigenvectors of $\mathbf{v}_{\text{AC}}$ can be determined by applying a Principal Component Analysis (PCA) transformation, after which the associated bias terms can be computed. The (i)-th hop, with a target neighborhood of size $n \times n$ and a Saab filter of size $m \times m$, will result in a tensor of size $I \in \mathbb{R}^{N \times N \times K^{i-1}}$, where $N = m - n + 1$ and K is the number of PCA components learned. Since the dimension of the output proliferates with each hop, max pooling is applied to the output before passing it to the next hop. In this work, we adopt a 3-hop Saab transform approach.

Relevant Feature Selection

After completing the 3-hop representation learning module, we generate a distinct set of representations for each hop. Given the high dimensionality of these features, it is essential to identify the most relevant ones from each set. We apply a supervised feature selection technique as described in [12] to accomplish this. The Relevant Feature Test (RFT) method, tailored for regression tasks, is employed to pinpoint the most significant features by partitioning the feature dimensions and assessing their impact on prediction error. The method involves several crucial steps, as follows.

First, for each feature dimension j, the training data is partitioned into two subsets, $S_{L,t}^j$ and $S_{R,t}^j$, using a threshold f_t^j, where $S_{L,t}^j$ and $S_{R,t}^j$ represent the subsets of data on the left and right of the threshold f_t^j, respectively. The RFT loss $\mathcal{L}_t^j$ at this threshold is defined as the weighted sum of the mean squared errors (MSEs) of the regression models fitted to the two subsets. Mathematically, the RFT loss is given by:

$$\mathcal{L}_t^j = \frac{N_{L,t}^j \mathcal{L}_{L,t}^j + N_{R,t}^j \mathcal{L}_{R,t}^j}{N}$$

where N is the total number of samples, $N_{L,t}^j$ and $N_{R,t}^j$ are the sizes of the left and right subsets, respectively, and $\mathcal{L}_{L,t}^j$ and $\mathcal{L}_{R,t}^j$ are the corresponding regression mean squared errors (MSEs) for the left and right subsets.

Next, the method searches for the optimal threshold f_{op}^j that minimizes the RFT loss for each feature dimension. The optimized RFT loss for feature j is defined as

$$\mathcal{L}_{\text{min}}^j = \min_{t \in T} \mathcal{L}_t^j,$$

where T is the set of candidate thresholds. This process identifies the threshold that results in the smallest possible regression MSE for each feature, where $\mathcal{L}_{\text{min}}^j$ represents this minimized loss.

Finally, to select the most relevant features, the optimized RFT losses $\mathcal{L}_{\text{min}}^j$ are ordered across all feature dimensions j. The smaller the value of $\mathcal{L}_{\text{min}}^j$, the more relevant the corresponding feature dimension f^j. The top K features with the lowest $\mathcal{L}_{\text{min}}^j$ values are selected as the most relevant features for the regression task. The RFT loss curves select the relevant features at each hop. Fig. 3illustrates the RFT loss of sorted feature indices for each hop, with lower RFT loss values indicating higher feature importance. As expected, the first hop representations exhibit less loss than those from subsequent hops. It is worth noting that this supervised method is not computationally expensive and requires only a small number of training samples. A validation set is utilized to avoid overfitting and ensure accurate predictions for unseen data. Fig4 shows the rank of each feature dimension, where a lower rank indicates higher energy for both the training and validation sets. The thresholds determined by the RFT loss curves for each hop define a searching window of strong feature dimensions for the test and validation sets. The overlapping dimensions between the training and validation sets are identified at each hop. Finally, the selected features from each hop are combined to form the most informative feature space.

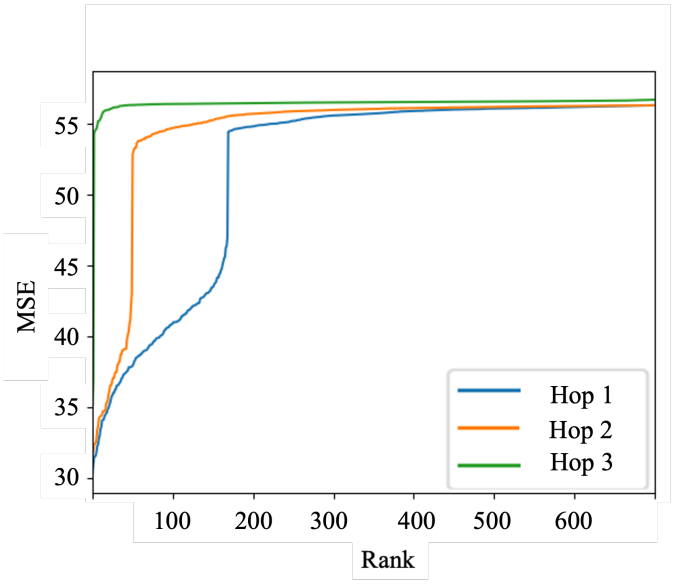

Fig. 3. Feature relevance and loss comparison for three Hops

Complementary Feature Generation

To generate Least-Squares Normal Transform (LNT) features, we begin by recasting the multi-class classification problem as a regression task. Given a dataset comprising ℓ training samples and κ classes, we define an indicator matrix $\mathcal{T} \in \mathbb{R}^{m \times \ell}$, where each element $\tau_{m,\ell}$ corresponds to the superclass label for a specific split configuration m, with $\tau_{m,\ell} \in \{0, 1\}$.

The objective is to determine the weight matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ that projects the input feature space $\mathbf{X} \in \mathbb{R}^{n \times \ell}$ into the target space $\mathcal{T}$. This is achieved by solving the linear least-squares regression problem:

$$\mathbf{AX} + \mathbf{B} = \mathcal{T},$$

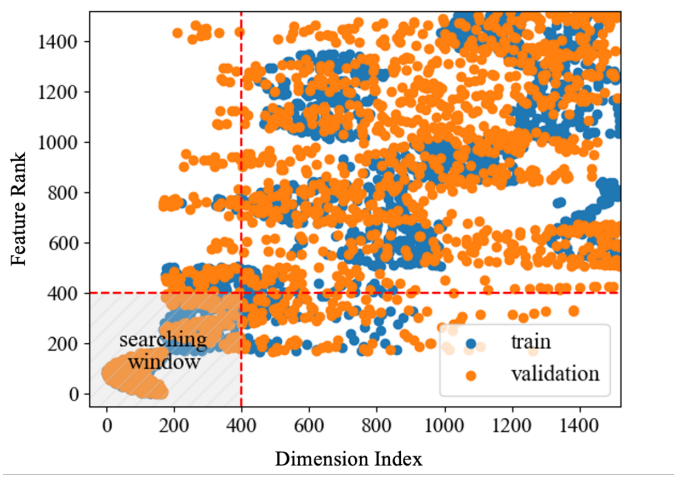

Fig. 4. Ranked features based on RFT loss for train and validation sets.

where $\mathbf{B}$ is a rank-one bias matrix. By taking the expectation of the equation, we isolate the term involving $\mathbf{A}$, as the bias matrix $\mathbf{B}$ only shifts the features without impacting their discriminative ability:

$$\mathbf{B} = \mathbb{E}[\mathcal{T}] - \mathbf{A}\mathbb{E}[\mathbf{X}].$$

The weight matrix $\mathbf{A}$ is then derived from the least-squares normal equation:

$$\mathbf{A} = \mathcal{T}\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}.$$

With the matrix $\mathbf{A}$ computed, the m complementary LNT features for any given input feature vector $\mathbf{x} \in \mathbb{R}^n$ are generated through

$$\mathbf{d} = \mathbf{A}\mathbf{x},$$

where $\mathbf{d} = (d_1, \ldots, d_m)^\top$ represents the set of LNT-generated features. To effectively generate the LNT features, it is crucial to identify feature subsets that yield high discriminative power.

In this study, we partitioned the features into n subsets and traced the XGBoost tree paths within each subset. These paths serve as indicators of discriminative feature sets within the trees. Fig. 5 visually represents the feature subsets corresponding to each path. The enhancements achieved through this methodology will be thoroughly examined in Section C. In conjunction with the selected features derived from the RFT, the features generated through LNT will be employed for decision learning in the subsequent stage.

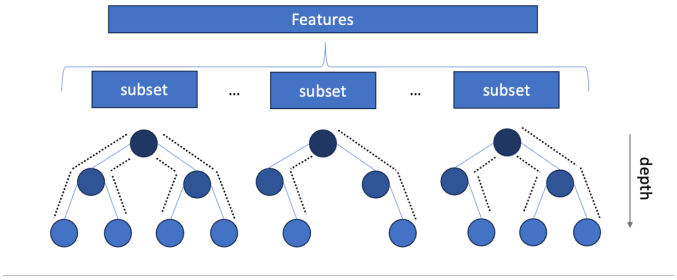

Fig. 5. Discriminative feature subsets for LNT feature generation

C. Supervised Decision Learning

Our decision-learning stage consists of two modules: core and auxiliary predictors.

Core predictor:
The core predictor is an XGBoost regressor, which is utilized to predict the residuals of the interpolated target pixels. The model was trained independently on each of the six subchannels R_1, R_2, G_1, G_2, B_1, and B_2.

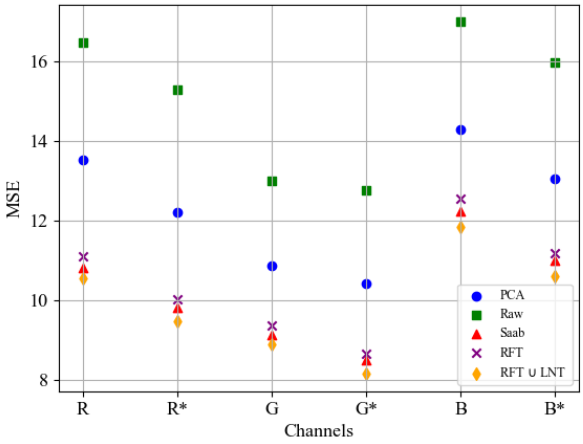

Fig. 6. Mean square error of the core predictor trained on different feature spaces

We demonstrate the effectiveness of the data processing and feature extraction stages by training the core regressor with different strategies. Fig6 compares the mean squared error (MSE) of predicted residuals across different feature sets: raw features, Saab features, RFT features, and a combination of RFT and LNT features. The comparison is made for channel- and CFA-based subchannel-wise (denoted with a star) training. The results reveal a significantly lower prediction error when using subchannel-wise training. Additionally, the method achieves substantial feature reduction compared to Principal Component Analysis (PCA). Including complementary LNT features further enhances prediction accuracy, leading to a reduced MSE.

Auxiliary Predictor:
This module encompasses both XGBoost multi-class classification and XGBoost regression models. Our primary objective for the classification task is categorizing the pixel data into three classes: negatively large, small, and positively large

target values. The supervised classification leverages the DFT features extracted from the training data.

The motivation for this classified approach arises from the significant imbalance in our target residue distribution, where small residue values are disproportionately larger. If this imbalance is not addressed, it can severely hinder the performance of the auxiliary regressor, resulting in suboptimal predictions. To mitigate this issue, each cluster is assigned its own XGBoost regressor, tailored to the specific characteristics of that cluster. Table I compares the auxiliary predictor's prediction error across different sampling strategies: a combined approach (unclassified), a sampled approach where the regressor samples more from the less dominant data, and our classified method.

TABLE I
COMPARISON OF THE AUXILIARY TRAINING STRATEGIES

Decision Method	MSE		
	R	***G***	***B***
Core	9.21	7.87	10.25
Auxiliary: combined	9.11	7.53	10.18
Auxiliary: sampling	8.84	7.31	9.96
Auxiliary: classified	8.12	6.75	9.41

IV. EXPERIMENTS

We employed the classic Kodak and McMaster (MCM) datasets to evaluate our algorithm's performance on demosaicking and denoising tasks. The Kodak dataset comprises 24 images, each with a resolution of 768 × 512 pixels. In comparison, the McMaster dataset includes 18 images, cropped to 500 × 500 pixels from the original high-resolution images of 2310 × 1814 pixels for the noise-free CFA image. Unlike many recent models that perform demosaicking and denoising jointly, our approach focuses solely on demosaicking, making direct comparisons based on noise-influenced results less relevant. Our evaluation criteria include performance, model complexity, and computational cost.

A. Performance Analysis

We compare our results with several state-of-the-art methods, including residual interpolation (RI) [28], minimized-Laplacian residual interpolation (MLRI) [29], adaptive residual interpolation (ARI) [30], directional difference regression and efficient regression priors (DDR-ERP) [31], and CNN based models: DJDD [6], NTSDCN [32], Inc [33]. Despite not employing deep learning techniques, our model achieves comparable performance. Notably, our model was trained on only approximately 60% of the MCM dataset and 55% of the Kodak dataset while utilizing a significantly reduced feature dimension set for training. TableII presents our comparable performance with deep demosaicking models.

B. Model Complexity Analysis

Many contemporary demosaicking neural models utilize a single unified network to reconstruct the red, green, and blue channels. While this approach allows the RGB channels to share common representations, it can result in overfitting for some channels and underfitting for others. Although we trained separate models for each channel, each model is relatively small. For data processing, our first hop uses 729 parameters for an input size of $15 \times 15 \times 3$ and a filter size of 3×3. The second hop, with an input size of $7 \times 7 \times 27$, involves 59,049 parameters. Hops with $i > 2$ were not trained because their representations were deemed uninformative based on the RFT analysis.

The selection of relevant features involves storing the partition loss for each feature with an array that holds the indices of features sorted by their respective losses. This approach is computationally lightweight, with the model containing 18,252 parameters for hop1 and 12,150 for hop2 across all six subchannels. The model employs an XGBoost model with depth= [2,3,4] to generate complementary features, resulting in 93,600 parameters. Our core predictor utilizes an XGBoost regressor with a depth of 4, comprising 193,200 parameters, while the auxiliary predictor, also with the same depth, has a total size of 386,400 parameters. Our model has a total of 700K parameters. We compare the size of our model with several neural models, as shown in the table. Despite the small model size, it's important to note that we trained our models on 6 subsets for three channels. Using independent models may enhance robustness to noise and outliers in the data. Additionally, training models on these subsets of data enables parallel processing, which can significantly decrease the time needed to train large and complex models.
TableIII compares our modelsize to convolutional models, DJDD [6], PCNN [24], NTSDCN [32] and InC [17].

TABLE II
AVERAGE CPSNR OF DIFFERENT MODELS

Method	Dataset	
	Kodak (24 images)	**McMaster (18 images)**
RI [28]	39.05	36.79
MLRI [29]	39.52	36.51
ARI [30]	39.63	37.42
DDR-ERP [31]	40.46	37.50
DJDD [6]	41.20	39.13
NTSDCN [32]	42.79	39.48
InC [33]	42.81	39.62
GID (ours)	**41.25**	**39.21**

C. Computational Cost Analysis

As previously discussed, our model is specifically designed for deployment on edge devices with limited computational resources. Since demosaicking is a real-time process for mobile cameras, it requires low-latency responses. The FLOP count, which quantifies the number of floating-point operations (additions, multiplications, etc.) necessary to run a model, is a critical indicator of the computational demands. Each operation consumes resources, making the total FLOP count a helpful estimate of the required computational power. We use FLOPs as a metric instead of runtime because algorithm complexity, parallel computing capabilities, and memory access

times often influence runtime. These variables can obscure the accurate computational load of an algorithm, making FLOPs a more consistent measure of performance across different environments.

Due to their intricate architectures and the numerous operations required to process inputs through multiple layers of interconnected nodes, Neural networks typically exhibit high FLOP counts. These layers involve extensive matrix multiplications, additions, and non-linear activation functions, all contributing to the overall computational burden. Our model requires 31948 FLOPs per sample, demonstrating its efficiency and speed during inference, as it effectively manages the computational demands despite the complexity of its tasks. Table III demonstrates the FLOP count of our model compared to deep learning methodologies.

TABLE III
COMPARISON OF MODEL PARAMETERS AND FLOPS

Method	No. of Parameters (M)	FLOPs per Sample (K)
PCNN [24]	20.6 (29.42×)	-
DJDD [6]	36.2 (51.71×)	249.12 (7.79×)
NTSDCN [32]	1.2 (1.71×)	230.08 (7.20×)
InC [33]	1.3 (1.85×)	249.12 (7.79×)
GID (ours)	**0.7** (1×)	**31.94** (1×)

V. CONCLUSION

We propose a subchannel-wise green model for demosaicking that delivers performance on par with state-of-the-art deep learning methods. Our approach emphasizes representation learning and dimensionality reduction to effectively capture the underlying data patterns instead of relying on additional layers and parameters characteristic of black-box models. This design choice enhances the interpretability of our model while mitigating the risk of overfitting. Notably, our model is approximately half the size of other lightweight deep learning alternatives and achieves a computational efficiency that is one-seventh that of deep learning models. Our model is particularly well-suited for deployment on edge devices, offering minimal latency, low battery consumption, parallel training, and reduced RAM usage.

VI. APPENDICES

A. FLOPs calculation for i-hop Saab transform

Let $\mathbf{X} \in \mathbb{R}^{c\times p\times p}$ be the input tensor, where c denotes the number of channels, and $p \times p$ is the spatial resolution of each channel. We apply a transformation $Saab$ to $\mathbf{X}$, defined by a kernel of size $m \times m$ with a stride of 1, resulting in an output tensor $\mathbf{X}' \in \mathbb{R}^{c\times n\times n\times m\times m}$, where n denotes the spatial resolution of the output channels.

Mathematically, this transformation can be expressed as:

$$\mathbf{X}' = Saab(\mathbf{X})$$

Given that the stride is 1, the spatial dimensions of $\mathbf{X}$ reduce from $p\times p$ to $n\times n$ after the transformation, where n depends on the input size p, the kernel size m, and any padding applied.

The computational complexity in terms of FLOPs for this transformation is given by:

$$\text{FLOPs} = 2 \times n^2 \times m^{2i} \times c$$

where i represents the iteration index (i-th hop) corresponding to how many times the transformation T has been applied successively.

B. FLOPs calculation for XGboost models

Consider an ensemble of T decision trees, each of depth D, where the classification task involves C classes. For each decision tree, the computational cost primarily arises from the comparison operations at each decision node and the subsequent addition operation required to determine the branch to follow.

Comparisons: Each tree, having a depth of D, requires D comparison operations as each decision node involves a comparison to decide the path between the left and right subtrees.

Additions: At each decision node, a single addition operation is required to determine which child node to traverse next.

Given these factors, the total FLOPs required for processing a single sample through the ensemble can be expressed as:

$$F = T \times C \times (D + 1)$$

C. Model size calculation for XGboost models

The model size in terms of the number of parameters can be computed as follows. Consider an ensemble of T decision trees, each with a depth D, applied to a classification task with C classes. The total number of parameters in the model is determined by the number of decision nodes and leaf nodes in each tree, multiplied by the number of trees and classes.

Decision Nodes: Each decision tree of depth D contains $2 \times (2^D - 1)$ decision nodes. The factor of 2 accounts for the decision at each node to traverse the left or right subtree.

Leaf Nodes: Each tree also contains 2^D leaf nodes, representing the final outcomes after all decisions have been made.

Given these factors, the total number of parameters P in the ensemble can be expressed as:

$$P = T \times C \times \left[2 \times (2^D - 1) + 2^D\right]$$

This formula provides a comprehensive measure of the model size, accounting for all parameters involved in the decision-making process across the entire ensemble.

ACKNOWLEDGMENT

Computation for the work was supported by the University of Southern California's Center for Advanced Research Computing (carc.usc.edu).

REFERENCES

[1] J. E. Adams Jr, "Interactions between color plane interpolation and other image processing functions in electronic photography," in *Cameras and Systems for Electronic Photography and Scientific Imaging*, vol. 2416. SPIE, 1995, pp. 144–151.

[2] R. H. Hibbard, "Apparatus and method for adaptively interpolating a full color image utilizing luminance gradients," Jan. 17 1995, uS Patent 5,382,976.

[3] C. Laroche, "Apparatus and method for adaptively interpolating a full color image utilizing chrominance gradients," *United States Patent, no. 5373322*, 1994.

[4] J. F. Hamilton Jr, "Adaptive color plan interpolation in signal sensor color electronic camera," *United State Patent, 5,629,734*, 1997.

[5] D. S. Tan, W.-Y. Chen, and K.-L. Hua, "Deepdemosaicking: Adaptive image demosaicking via multiple deep fully convolutional networks," *IEEE Transactions on Image Processing*, vol. 27, no. 5, pp. 2408–2419, 2018.

[6] M. Gharbi, G. Chaurasia, S. Paris, and F. Durand, "Deep joint demosaicking and denoising," *ACM Transactions on Graphics (ToG)*, vol. 35, no. 6, pp. 1–12, 2016.

[7] L. Liu, X. Jia, J. Liu, and Q. Tian, "Joint demosaicing and denoising with self guidance," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2240–2249.

[8] C.-C. J. Kuo and A. M. Madni, "Green learning: Introduction, examples and outlook," *Journal of Visual Communication and Image Representation*, vol. 90, p. 103685, 2023.

[9] D. Menon, S. Andriani, and G. Calvagno, "Demosaicing with directional filtering and a posteriori decision," *IEEE Transactions on Image Processing*, vol. 16, no. 1, pp. 132–141, 2006.

[10] N. Li, Y. Zhang, and C.-C. J. Kuo, "Explainable machine learning based transform coding for high efficiency intra prediction," 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:229340651

[11] Y. Chen and C.-C. J. Kuo, "Pixelhop: A successive subspace learning (ssl) method for object classification," *arXiv preprint arXiv:1909.08190*, 2019.

[12] Y. Yang, W. Wang, H. Fu, C.-C. J. Kuo *et al.*, "On supervised feature selection from high dimensional feature spaces," *APSIPA Transactions on Signal and Information Processing*, vol. 11, no. 1, 2022.

[13] X. Wang, V. K. Mishra, and C.-C. J. Kuo, "Enhancing edge intelligence with highly discriminant lnt features," in *2023 IEEE International Conference on Big Data (BigData)*. IEEE, 2023, pp. 3880–3887.

[14] E. Chang, S. Cheung, and D. Y. Pan, "Color filter array recovery using a threshold-based variable number of gradients," in *Sensors, Cameras, and Applications for Digital Photography*, vol. 3650. SPIE, 1999, pp. 36–43.

[15] X. Wu and N. Zhang, "Primary-consistent soft-decision color demosaic for digital cameras," in *Proceedings 2003 International Conference on Image Processing (Cat. No. 03CH37429)*, vol. 1. IEEE, 2003, pp. I–477.

[16] D. Menon and G. Calvagno, "Joint demosaicking and denoisingwith space-varying filters," in *2009 16th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2009, pp. 477–480.

[17] Y. Niu, J. Ouyang, W. Zuo, and F. Wang, "Low cost edge sensing for high quality demosaicking," *IEEE transactions on image processing*, vol. 28, no. 5, pp. 2415–2427, 2018.

[18] J. Duran and A. Buades, "Self-similarity and spectral correlation adaptive algorithm for color demosaicking," *IEEE transactions on image processing*, vol. 23, no. 9, pp. 4031–4040, 2014.

[19] F. Heide, M. Steinberger, Y.-T. Tsai, M. Rouf, D. Pajak, D. Reddy, O. Gallo, J. Liu, W. Heidrich, K. Egiazarian *et al.*, "Flexisp: A flexible camera image processing framework," *ACM Transactions on Graphics (ToG)*, vol. 33, no. 6, pp. 1–13, 2014.

[20] H. Tan, X. Zeng, S. Lai, Y. Liu, and M. Zhang, "Joint demosaicing and denoising of noisy bayer images with admm," in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 2951–2955.

[21] B. Henz, E. S. Gastal, and M. M. Oliveira, "Deep joint design of color filter arrays and demosaicing," in *Computer Graphics Forum*, vol. 37, no. 2. Wiley Online Library, 2018, pp. 389–399.

[22] F. Kokkinos and S. Lefkimmiatis, "Iterative joint image demosaicking and denoising using a residual denoising network," *IEEE Transactions on Image Processing*, vol. 28, no. 8, pp. 4177–4188, 2019.

[23] S. Ratnasingam, "Deep camera: A fully convolutional neural network for image signal processing," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, pp. 0–0.

[24] T. Yamaguchi and M. Ikehara, "Image demosaicking via chrominance images with parallel convolutional neural networks," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 1702–1706.

[25] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[26] H. Fu, Y. Yang, V. K. Mishra, and C.-C. J. Kuo, "Subspace learning machine (slm): Methodology and performance evaluation," *Journal of Visual Communication and Image Representation*, vol. 98, p. 104058, 2024.

[27] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[28] D. Kiku, Y. Monno, M. Tanaka, and M. Okutomi, "Beyond color difference: Residual interpolation for color image demosaicking," *IEEE Transactions on Image Processing*, vol. 25, no. 3, pp. 1288–1300, 2016.

[29] ——, "Minimized-laplacian residual interpolation for color image demosaicking," in *Digital Photography X*, vol. 9023. SPIE, 2014, pp. 197–204.

[30] Y. Monno, D. Kiku, M. Tanaka, and M. Okutomi, "Adaptive residual interpolation for color image demosaicking," in *2015 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2015, pp. 3861–3865.

[31] J. Wu, R. Timofte, and L. Van Gool, "Demosaicing based on directional difference regression and efficient regression priors," *IEEE transactions on image processing*, vol. 25, no. 8, pp. 3862–3874, 2016.

[32] Y. Wang, S. Yin, S. Zhu, Z. Ma, R. Xiong, and B. Zeng, "Ntsdcn: New three-stage deep convolutional image demosaicking network," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 9, pp. 3725–3729, 2020.

[33] Y. Niu, L. Zhang, and C. Li, "Independent and collaborative demosaicking neural networks," in *Proceedings of the 5th ACM International Conference on Multimedia in Asia*, 2023, pp. 1–7.