Efficient Neural Network Encoding for 3D Color Lookup Tables

Vahid Zehtab[*1, 2, 3], **David B. Lindell**[1, 2], **Marcus A. Brubaker**[1, 2, 3, 4], **Michael S. Brown**[3, 4]

[1]University of Toronto
[2]Vector Institute for Artificial Intelligence
[3]Samsung AI Center Toronto
[4]York University
{zehtab, lindell}@cs.toronto.edu, {mab, mbrown}@eecs.yorku.ca

Abstract

3D color lookup tables (LUTs) enable precise color manipulation by mapping input RGB values to specific output RGB values. 3D LUTs are instrumental in various applications, including video editing, in-camera processing, photographic filters, computer graphics, and color processing for displays. While an individual LUT does not incur a high memory overhead, software and devices may need to store dozens to hundreds of LUTs that can take over 100 MB. This work aims to develop a neural network architecture that can encode hundreds of LUTs in a single compact representation. To this end, we propose a model with a memory footprint of less than 0.25 MB that can reconstruct 512 LUTs with only minor color distortion ($\bar{\Delta}E_M \leq 2.0$) over the entire color gamut. We also show that our network can weight colors to provide further quality gains on natural image colors ($\bar{\Delta}E_M \leq 1.0$). Finally, we show that minor modifications to the network architecture enable a bijective encoding that produces LUTs that are invertible, allowing for reverse color processing.

Code — https://github.com/vahidzee/ennelut
Extended version — https://arxiv.org/abs/2412.15438

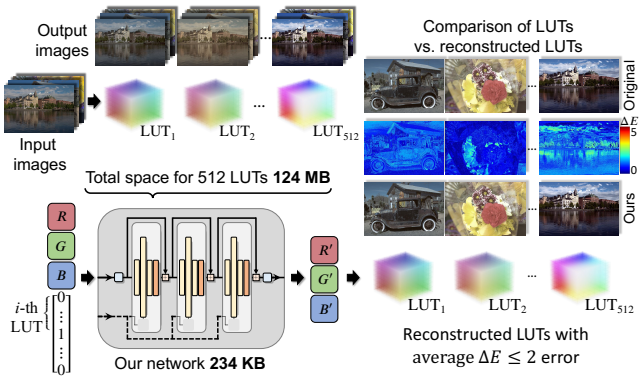

Figure 1: We propose a neural network architecture that encodes hundreds of LUTs into a single representation at a fraction of the memory requirements. Encoded LUTs can be reconstructed with minimal color distortion (i.e., $\Delta E \leq 2$).

Introduction

Color manipulation is a fundamental operation in computer vision and image processing, where input RGB values map to output RGB values. A widely used method for encoding such mappings is through a 3D color lookup table (LUT). LUTs are employed in a diverse range of applications, such as video editing, in-camera processing, photographic filters, computer graphics, and color processing for displays. Particularly, LUTs play a pivotal role in ensuring color accuracy and consistency across various display hardware.

An individual 3D LUT imposes a manageable memory overhead. For example, a standard 33×33×33 LUT at 16-bit precision requires approximately 70 KB. However, professional LUTs for color grading or color management often rely on a 65×65×65 resolution that requires approximately 0.5 MB at 16-bit precision. Storing a library of hundreds of such LUTs can quickly become a limitation, especially for applications running on resource-constrained devices, such as smartphones and camera hardware. Black-box compression (e.g., zip) provides limited reduction, for instance, a library of 512 zipped LUTs requires approximately 124 MB.

In this work, we propose a compact neural representation capable of reconstructing individual LUTs on the fly, reducing storage requirements and enhancing real-time color manipulation capabilities. We examine four network size variants for embedding different numbers of LUTs, with the largest requiring only 0.3 Mb storage to encode 512 LUTs. Our model recovers a full-size LUT in under 2 milliseconds, accommodating real-time applications. The recovered LUTs incur an average color difference (ΔE) of less than or equal to 2.0, an industry-standard definition for acceptable perceptual color distortion (Sharma and Bala 2017).

We demonstrate the versatility of our method with alternative loss functions and a straightforward weighting of input colors, enabling quality improvements for targeting natural image color gamuts. Our alternative weighted training achieves color differences (ΔE) of less than 1.0 for natural images. Lastly, we explore a modification to the network architecture, allowing for a bijective encoding of LUTs. This bijective encoding makes it possible to produce LUTs that are inherently invertible, opening up opportunities for inverse color processing and expanding the utility of LUTs.

[*]Work done while an intern at the Samsung AI Center Toronto.

Related Work

LUTs for Color Manipulation

As non-parametric functions, 3D Color LUTs (CLUTs or simply LUTs) are suitable for modeling complex transformations by sampling a target transformation on a 3D lattice. LUTs are instrumental for color grading in video editing (Postma and Chorley 2016) and used for correction in display colors (Shi and Luo 2021). LUTs are also essential tools in various stages of ISP pipelines (Kasson et al. 1995; Karaimer and Brown 2018), where they help ensure colorimetric accuracy or are used to render different picture styles (Karaimer and Brown 2016; Delbracio et al. 2021; Zhang et al. 2022).

LUTs have also been used as building blocks in frameworks that learn image enhancements (Zeng et al. 2020; Wang et al. 2021). For instance, work in (Yang et al. 2022a) learned LUTs using an adaptive lattice for color manipulation, while (Wang et al. 2021; Liu et al. 2023) learn contextualized LUTs for spatially varying and image-dependant color transformations. Although such works typically deal with less than a handful of LUTs, the memory footprint of LUTs has enticed attempts at less memory-intensive formulations such as through a combination of 1D and 3D LUTs (Yang et al. 2022b), or the decomposition of 3D LUTs into learned sub-tables and lower-rank matrices (Zhao, Abdelhamed, and Brown 2022). In this work, we assume the LUTs are provided and seek a neural architecture to encode them efficiently.

LUT Compression

LUTs are stored as a 3D input-output lattice, where the input lattice is usually set to a uniform grid at fixed intervals along the R, G, and B color axes. As a result, it is necessary only to store the output colors. LUTs are commonly stored as standard ASCII or Unicode (i.e., `.cube` files) or as a binary array of floating-point values. Such encodings can be further compressed using off-the-shelf compression algorithms (e.g., zip). Alternatively, a LUT can be stored as an RGB image called a Hald image. The Hald image resolution can be adjusted to mimic different LUT resolutions. Hald images are stored in a lossless format such as `png`. Hald images also serve to visualize how an individual LUT manipulates colors over the entire color space.

Various lossless compression techniques have been proposed (Balaji et al. 2007, 2008; Shaw et al. 2012), achieving average compression rates of ≈30% (similar to the compression rates of `png`). In (Tang et al. 2016), lossy compressed LUTs were stored as ICC device-link profiles. In (Tschumperlé, Porquet, and Mahboubi 2019, 2020), sparse color key points were estimated that enabled the reconstruction of the original 3D LUT, providing an average compression rate of ≥95%. Existing LUT compression methods target individual LUTs. Our network implicitly compresses multiple LUTs, capitalizing on their inherent similarities, and achieves compression rates of ≥99%.

Neural LUTs

To our knowledge, (Conde et al. 2024) is the only work to target LUT embedding and reconstruction using a neural network. Specifically, (Conde et al. 2024) showed that 3 to 5 LUTs could be encoded using a model requiring approximately 750 KB storage. We show that our network architecture is better suited for LUT embedding. In addition, we describe how a straightforward modification to our network imposes bijectivity on the LUTs encoding. This allows an image processed with our LUT to be restored to its initial RGB values.

Proposed Approach

Formally, a 3D color LUT $F : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ is a function that maps input RGB colors to output RGB colors represented by a set of input-output pairs on a sparse lattice covering the input color space. F computes the output color of an arbitrary input using traditional interpolation techniques (Kasson et al. 1995).

Given a set of LUTs $\{F_1, F_2, \cdots, F_N\}$, we aim to find an implicit neural representation $f_\theta(\cdot, \mathbf{o}) : \mathbb{R}^3 \times \mathcal{L} \rightarrow \mathbb{R}^3$, where $\mathcal{L} \subset \mathbb{R}^N$ is the set of LUTs and f is a function modeled with a deep neural network parameterized by θ. The function f_θ takes on an `RGB` input in addition to a desired LUT F_i, represented with a one-hot encoded vector $\mathbf{o}_i \in \mathcal{L}$.

We learn the parameters θ such that $f_\theta(., \mathbf{o}_i) \approx F_i(.)$. We formalize this in terms of the optimization problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \underset{x \sim \mathcal{P}}{\mathbb{E}} \left[\sum_i^N \| f_\theta(x, \mathbf{o}_i) - F_i(x) \|_2^2 \right], \quad (1)$$

where $\mathcal{P}$ signifies a probability distribution over the input colors. Different choices for $\mathcal{P}$ result in different θ^*s trading off better reconstruction of certain colors over the others. The choice of $\mathcal{P}$ depends on the intended use case of f_{θ^*}. As we show in our experimental results, the closer $\mathcal{P}$ is to the evaluation distribution, the better f_{θ^*} trades off the quality of color reconstructions over out-of-distribution colors.

Network Architecture

To design an efficient architecture, we desire to capture the inherent characteristics of LUTs. We aim to devise a network structure that embodies the following traits:

1. LUTs frequently resemble an identity function in many regions of the input space,
2. the majority of LUTs exhibit local bijectivity, with exceptions being intentional manipulations that compress the color space into lower-dimensional manifolds (e.g., RGB to grayscale LUTs).

Residual networks (ResNet) (He et al. 2016) are a natural fit for the first property, given their inductive bias toward an identity function. To capture the second property, we propose to utilize architectures from the normalizing flows literature (Kobyzev, Prince, and Brubaker 2020).

Normalizing flows (Rezende and Mohamed 2015) are a class of bijective neural networks often used for generative modeling and density estimation. We take inspiration

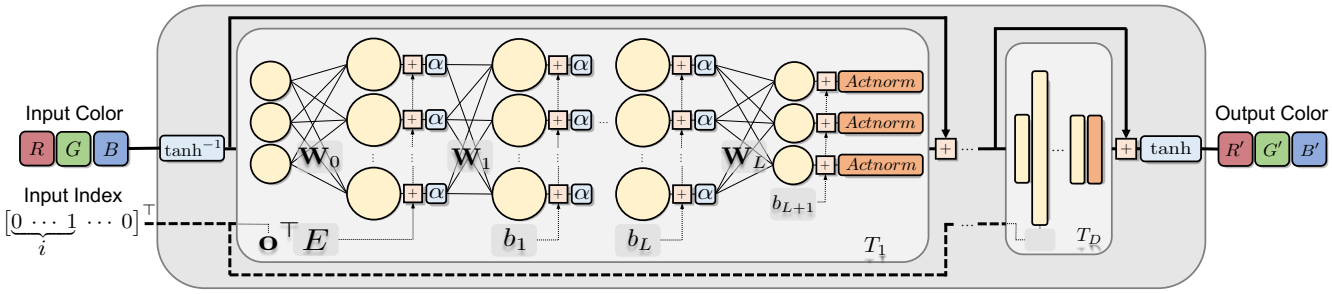

Figure 2: Our network consists of D transformations $T_i(\cdot, \mathbf{o})$, conditioned on a specific LUT by a one-hot encoded index $\mathbf{o}$ indicating the desired LUT to use. T_is contribute to the reconstructed output color through consecutive residual additions. T_is are modeled with multilayer perceptrons (MLP) with α non-linearities, where the biases of their first layer are selected based on $\mathbf{o}$. Activation normalization (Kingma and Dhariwal 2018) is used after each transformation to control the magnitude of the residual functions, ensuring stability in deeper architectures. $\tanh^{-1}$ and $\tanh$ respectively transform the inputs and outputs of the network, bringing it closer to the local identity.

from (Jacobsen, Smeulders, and Oyallon 2018; Chen et al. 2019; Behrmann et al. 2019) that advocate modifying a stock residual network with an inductive bias towards bijective maps. As demonstrated in (Jacobsen, Smeulders, and Oyallon 2018), if all the residual functions in a ResNet have a Lipschitz constant strictly smaller than 1 the entire network is invertible by the Banach fixed-point theorem. For such networks, termed residual flows, each residual function can be explicitly restricted through spectral normalization (Gouk et al. 2021; Miyato et al. 2018) and activation functions that have bounded derivatives. However, given that not all LUTs are bijective (e.g., RGB to greyscale), we do not want to rigidly restrict the network to bijectivity. Instead, we initialize it close to such a bijective transformation as a form of inductive bias to regularize learning.

To this end, we design our neural architecture with D residual components $T_1, T_2, \ldots, T_D$, as shown in Figure 2. Each residual component T_i, is an MLP with LipSwish non-linearities (Chen et al. 2019). T_is are conditioned on the desired LUT $\mathbf{o}$ by a learned matrix $E_i \in \mathbb{R}^{N \times h}$, where N is the number of embedded LUTs, and h is the width of the first hidden layer in T_i. The one-hot encoded vector $\mathbf{o}$, selects a row from E_i ($\mathbf{o}^\top E_i$), which is used as the biases for the first hidden layer in T_i.

We use activation normalization (Kingma and Dhariwal 2018) to mitigate numerical instabilities caused by stacking multiple residual functions together. To ensure that our model is as close to the space of identity and bijective functions, we initialize all the weights and biases of the network with small values and use LipSwish non-linearities with bounded derivatives.

Moreover, to deal with the bounded normalized input-output space, we convert the inputs using $\tanh^{-1}$ and then return to the bounded color space by applying $\tanh$ on the network outputs. This way, the learnable part of the model always operates in an unbounded space with similar input-output measures. This makes for a better computational model and helps our model deal with colors with high saturation, as the network weights no longer need to grow to produce saturated colors. In addition, compared to clipping the values (Conde et al. 2024), our method does not suffer from clipped gradient values, which can hinder training.

Training

Our training accommodates the simultaneous embedding of many LUTs on a custom color distribution $\mathcal{P}$. At each optimization iteration, we perform the following:

1. Randomly pick a batch of input colors from the 256^3 input color space based on $\mathcal{P}$ and normalize the source color values.
2. Compute the target colors by applying all (or optionally a random subset of LUTs) on the batch and normalize the target color values.
3. Compute the L_2 reconstruction error and update the weights. The network can also be trained using a ΔE loss which is described in our alternative training approaches experiments section.

Normalization. We normalize each color channel to fall within the range $[-0.83, 0.83]$ ensuring that the values passed through $\tanh$ and $\tanh^{-1}$ remain in a region with sufficiently large gradients, reducing the risk of vanishing gradients. Refer to supplemental materials for details.

Implementation. During training, input colors are processed with the LUT to produce target color values. We target all LUTs at each training step. We implemented a GPU-based trilinear interpolation using PyTorch (Paszke et al. 2019). An Nvidia RTX4090 GPU was used for training. With LUT interpolation implemented on GPU, we could process batch sizes of 2048 input colors, all transformed with up to 512 different LUTs.

We optimize our network using Adam (Kingma and Ba 2014) with default settings and use a stepped learning rate schedule, decreasing the learning rate at fixed intervals. See supplemental materials for more details.

Evaluation

To evaluate the quality of the LUT approximation, we use the $\mathtt{CIE}_{76}$ ΔE metric that has a direct interpretation in

Model (size)	Compression ratio (% ↑)	Evaluation on a 256^3-Hald image			Evaluation on natural images		
		$\bar{\Delta}E_M\downarrow$	$\bar{\Delta}E_{90\%}\downarrow$	PSNR (dB)↑	$\bar{\Delta}E_M\downarrow$	$\bar{\Delta}E_{90\%}\downarrow$	PSNR (dB)↑
Tiny (79 KB)	99.94	$4.47_{\pm 0.04}$	$8.61_{\pm 0.08}$	$32.37_{\pm 0.09}$	$4.76_{\pm 0.03}$	$8.23_{\pm 0.06}$	$33.77_{\pm 0.19}$
Small (157 KB)	99.87	$2.69_{\pm 0.03}$	$5.26_{\pm 0.07}$	$36.70_{\pm 0.10}$	$3.10_{\pm 0.04}$	$5.49_{\pm 0.07}$	$37.52_{\pm 0.10}$
Medium (235 KB)	99.81	$2.00_{\pm 0.02}$	$3.91_{\pm 0.04}$	$39.31_{\pm 0.08}$	$2.37_{\pm 0.02}$	$4.24_{\pm 0.04}$	$39.80_{\pm 0.07}$
Large (313 KB)	99.75	$1.64_{\pm 0.01}$	$3.23_{\pm 0.03}$	$41.03_{\pm 0.09}$	$1.98_{\pm 0.02}$	$3.60_{\pm 0.03}$	$41.37_{\pm 0.09}$

Table 1: This table reports the quality of our reconstructed LUTs for each model size when embedding 512 LUTs and trained using uniform sampling over the color space and an L_2 training objective in RGB. The results are computed for Hald images and natural images. Results are averaged over ten trial runs. We refer to the compressed file size of the model checkpoint as model size. Compression ratios compare the model size to the average compressed file size of 512 binary LUTs.

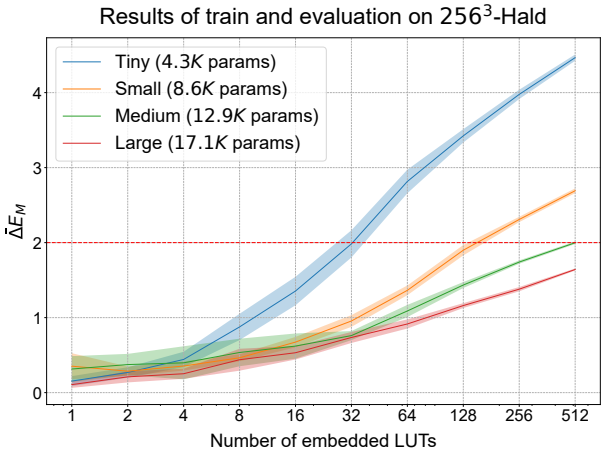

Figure 3: This plot shows how varying the number of embedded LUTs affects the performance of our models when training uniformly on the color-space using an L2 loss function in RGB and evaluating against 256^3 Hald images. The specified number of parameters represents the parameters in the T_i blocks without counting the E_is. Results are averaged over ten runs per #LUTs, each using a different set of LUTs.

terms of human perception. Specifically, two colors with a $\Delta E \leq 2$ are generally considered indistinguishable for an average observer (Sharma and Bala 2017). As ΔE is computed per-color pair, to get an estimate of the overall qualities of color reconstructions, we track the general statistics of ΔEs over the set of evaluation colors. We use $\Delta E_{q\%}$ and ΔE_M to denote the q-th quantile and the empirical mean of ΔEs over a particular evaluation set. Since our model embeds multiple LUTs at a time, we average such statistics over all the embedded LUTs in the model to get $\bar{\Delta}E$s. For instance, a model with $\bar{\Delta}E_{90\%} < 2$ can reconstruct 90% of the evaluated colors with a $\Delta E < 2$ on average over its embedded LUTs. Note that we also report PSNR values to be consistent with previous work.

Experiments

We begin by describing our training and testing data. We evaluate our network in two scenarios: (1) uniform sampling of the color space and (2) sampling based on natural images. Qualitative results are also shown on images processed by our reconstructed LUTs. Finally, we demonstrate the network's ability to invert a LUT.

Experimental Setup

LUTs. We obtained 543 open-source LUTs available under Creative Common licensing to train and test our method. The LUTs range in size between 16^3 to 35^3 and are encoded as `.cube` files. See supplemental material for details.

Evaluation. We report ΔE on 256^3 Hald images that capture all possible 8-bit color outputs produced by a given LUT. To estimate the reconstruction quality of colors in natural images, we report the metrics averaged over 100 randomly selected images from the Adobe-MIT 5K dataset (Bychkovsky et al. 2011). Our experiments showed minimal differences when using all of the images from the Adobe 5K dataset (Bychkovsky et al. 2011) for evaluation versous using a subset of 100 random selected images from the dataset. Therefore, for raster experimentation, we report the results of our evaluations on natural images using only 100 random images. See supplemental materials for details.

The selected images were converted to 8-bit sRGB format and rescaled such that their maximum dimension is 1024 pixels. We ensure the reproducibility of our experiments by retraining the same model with different sets of randomly selected LUTs to provide the average and 95% confidence interval for each metric. We provide the results for different model sizes and number of embedded LUTs.

Runtime. Our model reconstructs LUTs at any resolution. Execution time to recover LUTs with our medium-sized model at resolutions 65^3, 33^3, 11^3 and 7^3 are 4.11 ms, 1.64 ms, 1.27 ms, 1.21 ms respectively, measured on an Nvidia RTX4090 GPU. See supplemental materials for details and additional runtime results.

Quantitative Results

We start with training our models by sampling the LUTs over the entire color space. We refer to this as uniform sampling as every color in the color space is weighted equally. As previously mentioned, we use a GPU-based interpolation of the LUTs to generate these samples over the LUTs.

The number of trainable parameters for each model consists of: (1) the core parameters of each T_i, and (2) the embedding weights E_is, which grow linearly with the number of LUTs. With this in mind, we experimented with different

Training objective	Training distribution	Evaluation on a 256^3-Hald image			Evaluation on natural images		
		$\bar{\Delta}E_M\downarrow$	$\bar{\Delta}E_{90\%}\downarrow$	PSNR (dB)↑	$\bar{\Delta}E_M\downarrow$	$\bar{\Delta}E_{90\%}\downarrow$	PSNR (dB)↑
L_2	Uniform	$2.01_{\pm 0.02}$	$3.93_{\pm 0.05}$	$39.26_{\pm 0.10}$	$2.44_{\pm 0.03}$	$4.38_{\pm 0.06}$	$39.68_{\pm 0.08}$
	Natural images	$6.63_{\pm 0.12}$	$16.34_{\pm 0.37}$	$26.13_{\pm 0.18}$	$1.09_{\pm 0.01}$	$2.19_{\pm 0.02}$	$45.87_{\pm 0.21}$
ΔE	Uniform	$1.65_{\pm 0.02}$	$3.13_{\pm 0.05}$	$36.95_{\pm 0.12}$	$1.92_{\pm 0.01}$	$3.41_{\pm 0.02}$	$39.95_{\pm 0.15}$
	Natural images	$6.04_{\pm 0.21}$	$15.07_{\pm 0.53}$	$25.23_{\pm 0.28}$	$0.96_{\pm 0.02}$	$1.95_{\pm 0.06}$	$45.67_{\pm 0.19}$

Table 2: This table shows the results of using different color distributions $\mathcal{P}$ and training objectives for training and then evaluating the model (i.e., uniform versus natural images). Results are averaged over five trial runs, each fitting 512 different LUTs using our medium-sized architecture. Training on natural color distribution leads to better performance on natural images and reduced performance when evaluated over the entire color space (i.e., evaluated against Hald images). Using the ΔE loss function, as expected, reduces color reconstruction errors in terms of ΔE.

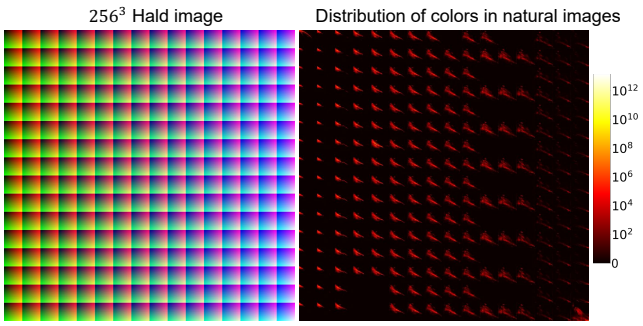

Figure 4: This figure shows a Hald image representing all input colors (left) and the distribution of these colors in the 100 Adobe-MIT5K images used for training (right).

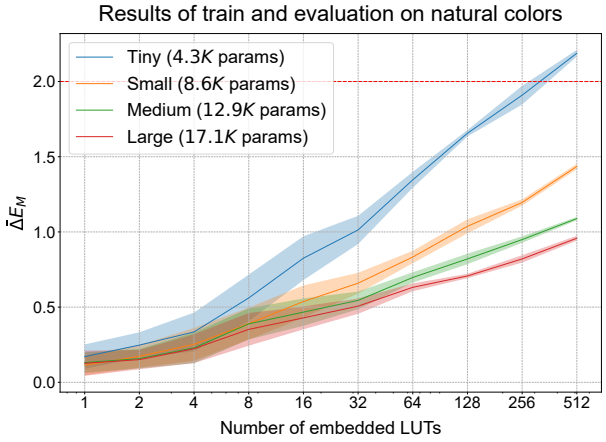

Figure 5: This figure shows the performance of different model sizes when embedding a varying number of LUTs trained and tested on natural images with an L_2 training objective. The number of parameters in the model size represents the parameters in the T_i blocks, not counting the E_is.

numbers of T_is (D) of varying depths and widths to pick the right set of hyper-parameters for our network. We found a hidden structure of $[32, 64, 32]$ neurons for T_is to work best with minimal memory requirements, as the core structure only has $4.3K$ trainable parameters, and embedding each LUT only requires learning an additional 32 parameters.

We consider four variants of our model (tiny, small, medium, large) with increasing modeling capacity by stacking 1 to 4 T_is together. Figure 3 shows the quality of the color reconstructions for each model variant when trained to embed a varying number LUTs. Smaller models can be used depending on the number of LUTs that need to be embedded. Figure 3 also shows that even the smallest model can embed up to 32 LUTs with a low enough ΔE.

Table 1 provides a more detailed look at the different models when embedding 512 LUTs. Here we report the compression ratios as the ratio of the compressed model checkpoints (including the weights of T_is and E_is) to the file size of the binary LUTs, compressed together in a single archive. See supplemental materials for details. As expected, the larger models achieve higher fidelity as the number of LUTs increases. Nonetheless, compared to the average compressed file size of 124.43 MBs for 512 LUTs, even our largest model has a small storage requirement of less than 350 KB. This represents a $\geq 99.7\%$ compression ratio while maintaining a minimal loss of perceivable quality of the reconstructed colors.

Alternative Training Approaches

Training using natural images. As previously mentioned, the choice of the training color distribution $\mathcal{P}$ plays a role in the overall quality of color reconstructions. To test this, we randomly select 200 images from the Adobe-MIT5K dataset (Bychkovsky et al. 2011), splitting them into a training and testing split of 100 images each. Figure 4 shows a visualization of the distribution of the colors in the natural training images as a heat map corresponding to a Hald image. The distribution heat map reveals that large regions of the color space have a low probability of occurring in the natural images, with approximately 85% of all colors rarely if ever, being observed.

This sparsity suggests that a model trained to focus on those regions of the color space which are more likely to occur in natural images should perform better when evaluated on images. We repeat our uniform distribution experiments but instead train by using the distribution of colors in the Adobe-MIT5K training split as our $\mathcal{P}$. We then evaluate the model on the images in the test split.

Table 2 shows the performance of our medium-sized

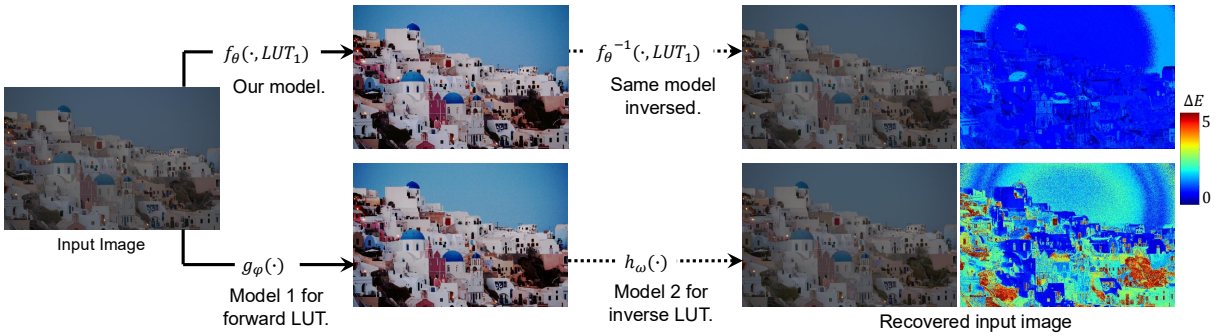

Figure 6: This figure compares the inversion accuracy of our modified invertible model versus a non-invertible architecture for estimating the forward and inverse pass through a single LUT. We use a deeper architecture ($D = 32$, hidden structure $[16, 32, 16]$) to embed 32 LUTs, ensuring it remains bijective during training. We train a *small* architecture to estimate the LUT (g_ψ) and its inverse (h_ω). Results are shown on an Adobe 5K dataset image processed by reconstructed LUTs, with ΔE error maps computed per pixel against the ground truth image.

model trained and tested on different $\mathcal{P}$s to embed 512 LUTs. Training uniformly across the entire color space improves generalization across all possible colors. However, as seen in Figure 5 and Table 2, it is evident that knowing the target evaluation color distribution (e.g., when LUTs are consistently applied to natural images) allows for the utilization of specific training distributions $\mathcal{P}$ to enhance the quality of the embedded LUTs.

Training with ΔE. We can also train our network using ΔE directly as the loss function. Table 2 shows that this results in better ΔE values, but at the cost of lower performance in terms of PSNR.

Qualitative Results

Figure 7 provides qualitative results on images processed by reconstructed LUTs. Our results are computed using our medium-sized model with 32 LUT embedding and trained with uniform sampling. We compared this with the best-performing architecture reported by Conde et al. (Conde et al. 2024). Our model can achieve higher fidelity color reconstructions with a significantly smaller model size. See supplemental material for further details on these experiments and architectural ablation studies.

Figure 8 shows qualitative results on our model trained using uniform sampling and natural image sampling. As the quantitative experiments also indicate, training our models on natural images improves performance when the reconstructed LUTs are applied to natural images. Nevertheless, uniform training might be preferred when consistency of predictions is required, as the model trained on a sparse distribution might fail to faithfully reconstruct colors that appear infrequently in the training distribution. For example, in Figure 8 (last row), the purple-colored flowers have higher ΔE than those processed by the uniformly trained model.

LUT Inversion

We designed our model to be initialized near the space of bijective transformations. Here we examine the effect of keeping the network strictly bijective. We use spectral normalization (Miyato et al. 2018; Gouk et al. 2021) to normalize the weights of each residual transformation T_i by its largest singular value, enforcing a Lipschitz constant of 0.97. This restriction, along with our choice of activation function, forces the model to remain bijective during training, which allows the network to be inverted (Behrmann et al. 2019) with a fixed-point iteration algorithm. As a result, we can compute an inverse color LUT by reversing the order of computations in the network and inverting each transformation. See supplemental materials for details.

Restricting the architecture to be bijective may slightly decrease its modeling capacity, and we find more depth is needed to approximate the LUTs comparably. For more details, see the supplemental material.

We visualize the results of our LUT inversion mechanism on a single image and LUT in Figure 6. For a point of comparison, we fit our small-sized architecture on the LUT applied (referred to as LUT_1) as follows. First, we fit the LUT as normal (g_ψ). Next, we also fit the LUT but swap inputs and outputs to estimate its inverse (h_ω). As seen in Figure 6, the invertible architecture works well at inverting the LUT, effectively up to the numerical precision of the fixed-point iteration. In contrast, attempting to directly estimate the LUT and its inverse separately results in significant errors.

Concluding Remarks

This paper has introduced a network architecture designed to efficiently encode 3D color lookup tables (LUTs) into a single compact representation. Our proposed model achieves this with a minimal storage footprint, consuming less than 0.25 MB. The reconstruction capability of the model extends to 512 LUTs, introducing only minor color distortion ($\bar{\Delta}E_M \leq 2.0$) across the entire color space. Furthermore, we demonstrate that the network has the ability to weight LUT colors, yielding additional quality improvements, particularly evident in natural image colors with $\bar{\Delta}E_M \leq 1.5$. Our network architecture is also able to accommodate bijective encoding, enabling the production of invertible LUTs and facilitating reverse color processing.

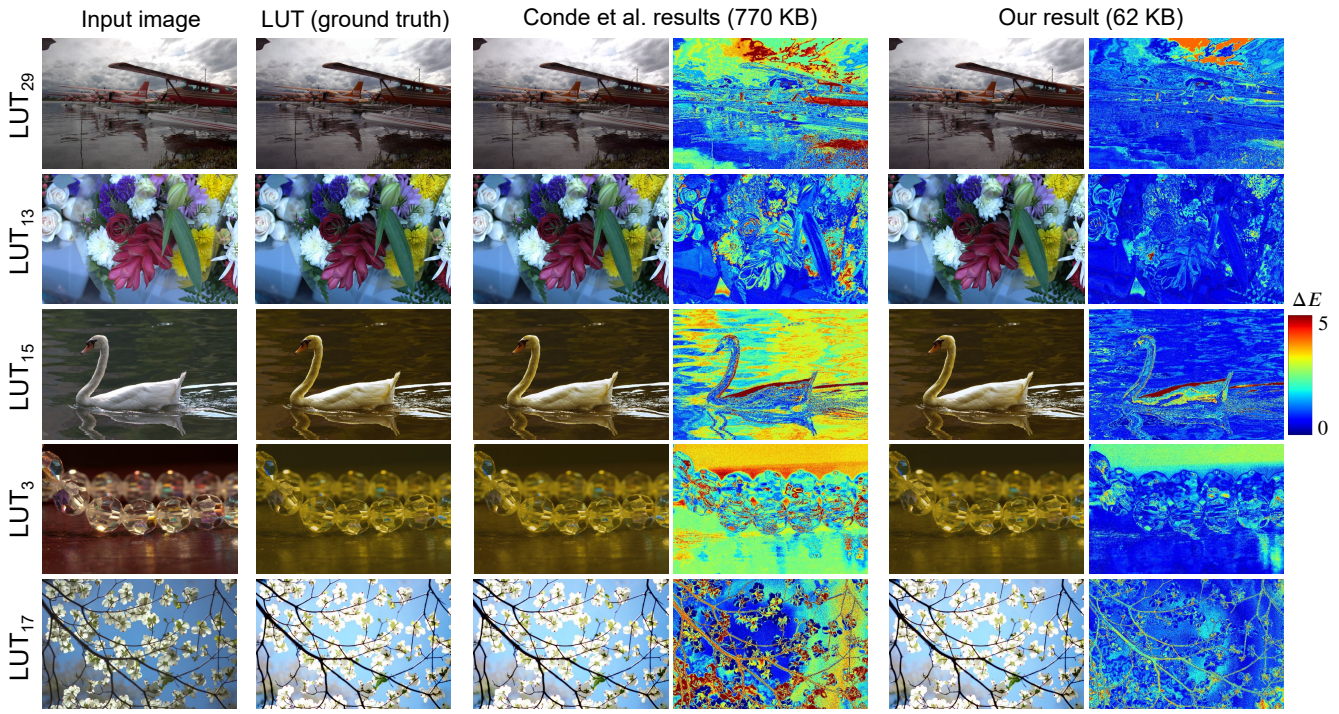

Figure 7: Comparison of our results with the best-performing model from (Conde et al. 2024). We use our medium-sized variant that has embedded 32 LUTs trained on 256^3 Hald images. Results are shown on images selected from the Adobe 5K dataset (Bychkovsky et al. 2011) processed by reconstructed LUTs. ΔE error maps are computed per pixel against the ground truth images that have been processed directly by the corresponding LUT.

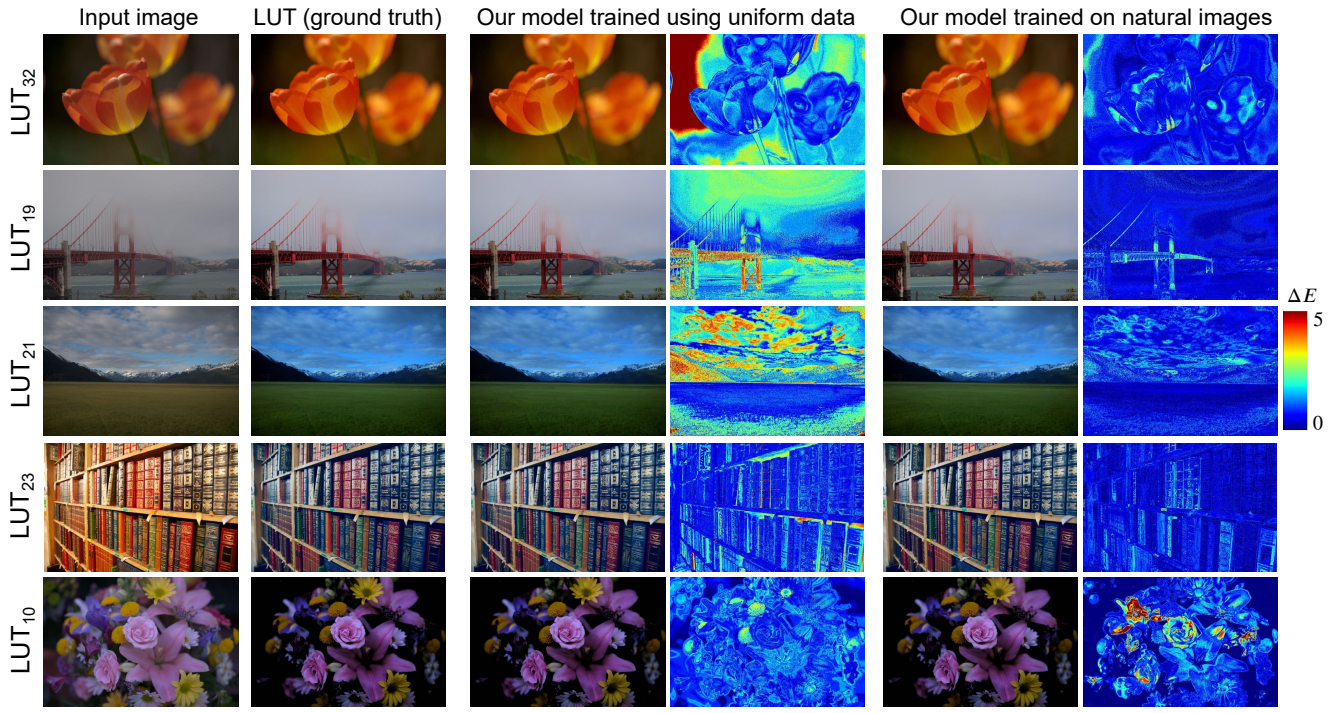

Figure 8: Comparison of the result of training directly on 100 natural color images versus uniform training. The medium-sized variant of our network is trained to embed 32 LUTs. The images shown are selected from the testing image split. Training on natural images provides a notable improvement when applied to natural images.

References

Balaji, A.; Sharma, G.; Shaw, M.; and Guay, R. 2008. Preprocessing methods for improved lossless compression of color lookup tables. *J. Imaging Sci. Technol.*, 52(4).

Balaji, A.; Sharma, G.; Shaw, M. Q.; and Guay, R. 2007. Hierarchical compression of color lookup tables. In *Proc. CIC*.

Behrmann, J.; Grathwohl, W.; Chen, R. T.; Duvenaud, D.; and Jacobsen, J.-H. 2019. Invertible residual networks. In *Proc. ICML*.

Bychkovsky, V.; Paris, S.; Chan, E.; and Durand, F. 2011. Learning photographic global tonal adjustment with a database of input/output image pairs. In *Proc. CVPR*.

Chen, R. T.; Behrmann, J.; Duvenaud, D. K.; and Jacobsen, J.-H. 2019. Residual flows for invertible generative modeling. *Proc. NeurIPs*.

Conde, M. V.; Vazquez-Corral, J.; Brown, M. S.; and Timofte, R. 2024. NILUT: Conditional Neural Implicit 3D Lookup Tables for Image Enhancement. In *Proc. AAAI*.

Delbracio, M.; Kelly, D.; Brown, M. S.; and Milanfar, P. 2021. Mobile computational photography: A tour. *Annu. Rev. Vis. Sci.*, 7(1): 571–604.

Gouk, H.; Frank, E.; Pfahringer, B.; and Cree, M. J. 2021. Regularisation of neural networks by enforcing Lipschitz continuity. *Mach. Learn.*, 110: 393–416.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proc. CVPR*.

Jacobsen, J.-H.; Smeulders, A.; and Oyallon, E. 2018. i-RevNet: Deep Invertible Networks. In *Proc. ICLR*.

Karaimer, H. C.; and Brown, M. S. 2016. A software platform for manipulating the camera imaging pipeline. In *Proc. ECCV*.

Karaimer, H. C.; and Brown, M. S. 2018. Improving color reproduction accuracy on cameras. In *Proc. CVPR*.

Kasson, J. M.; Nin, S. I.; Plouffe, W.; and Hafner, J. L. 1995. Performing color space conversions with three-dimensional linear interpolation. *J. Electron. Imaging*, 4(3): 226–250.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv*.

Kingma, D. P.; and Dhariwal, P. 2018. Glow: Generative flow with invertible 1x1 convolutions. *Proc. NeurIPS*.

Kobyzev, I.; Prince, S. J.; and Brubaker, M. A. 2020. Normalizing flows: An introduction and review of current methods. *IEEE Trans. Pattern Anal. Mach. Intell.*, 43(11): 3964–3979.

Liu, C.; Yang, H.; Fu, J.; and Qian, X. 2023. 4D LUT: learnable context-aware 4d lookup table for image enhancement. *IEEE Trans. Image Process.*, 32: 4742–4756.

Miyato, T.; Kataoka, T.; Koyama, M.; and Yoshida, Y. 2018. Spectral normalization for generative adversarial networks. *arXiv*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Proc. NeurIPs*.

Postma, P.; and Chorley, B. 2016. Color Grading with Color Management. *SMPTE Motion Imaging J.*, 125(8): 69–74.

Rezende, D.; and Mohamed, S. 2015. Variational inference with normalizing flows. In *Proc. ICML*.

Sharma, G.; and Bala, R. 2017. *Digital color imaging handbook*. CRC press.

Shaw, M.; Guay, R. G.; Sharma, G.; and Rajagopalan, A. B. 2012. Lossless compression of color look-up table via hierarchical differential encoding or cellular interpolative prediction. US Patent 8,294,953.

Shi, K.; and Luo, M. R. 2021. Methods to improve colour mismatch between displays. In *Proc. CIC*.

Tang, C.; Wang, W.; Collison, S.; Shaw, M.; Gondek, J.; Reibman, A.; and Allebach, J. 2016. ICC profile color table compression. In *Proc. CIC*.

Tschumperlé, D.; Porquet, C.; and Mahboubi, A. 2019. 3D Color CLUT compression by multi-scale anisotropic diffusion. In *Proc. CAIP*.

Tschumperlé, D.; Porquet, C.; and Mahboubi, A. 2020. Reconstruction of Smooth 3D Color Functions from Keypoints: Application to Lossy Compression and Exemplar-Based Generation of Color LUTs. *SIAM J. Imaging Sci.*, 13(3): 1511–1535.

Wang, T.; Li, Y.; Peng, J.; Ma, Y.; Wang, X.; Song, F.; and Yan, Y. 2021. Real-time image enhancer via learnable spatial-aware 3d lookup tables. In *Proc. ICCV*.

Yang, C.; Jin, M.; Jia, X.; Xu, Y.; and Chen, Y. 2022a. AdaInt: Learning adaptive intervals for 3D lookup tables on real-time image enhancement. In *Proc. CVPR*.

Yang, C.; Jin, M.; Xu, Y.; Zhang, R.; Chen, Y.; and Liu, H. 2022b. SepLUT: Separable image-adaptive lookup tables for real-time image enhancement. In *Proc. ECCV*.

Zeng, H.; Cai, J.; Li, L.; Cao, Z.; and Zhang, L. 2020. Learning image-adaptive 3d lookup tables for high performance photo enhancement in real-time. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(4): 2058–2073.

Zhang, F.; Zeng, H.; Zhang, T.; and Zhang, L. 2022. CLUT-Net: Learning Adaptively Compressed Representations of 3DLUTs for Lightweight Image Enhancement. In *Proc. ACM-MM*.

Zhao, L.; Abdelhamed, A.; and Brown, M. S. 2022. Learning Tone Curves for Local Image Enhancement. *IEEE Access*, 10: 60099–60113.