LEARNING AND EVALUATING IMAGE REPRESENTATIONS

BY

JASON ROCK

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Doctoral Committee:
Professor David Forsyth, Chair
Professor Svetlana Lazebnik
Professor Alexander G. Schwing
Dr. Jonathan Barron, Google
ABSTRACT

Prior to deep learning it was common to approach computer vision problems as describing a model that could be learned from a relatively small amount of data by incorporating domain knowledge. For example, image prediction tasks such as intrinsic image decomposition were approached by thinking about what reflectance and shading look like. In the case of reflectance, a Mondrian image; and in the case of shading, a smooth image. The difficult portion was how to formalize this prior domain knowledge into a model.

Deep learning has changed this paradigm. While deep learning hasn’t eliminated the value of domain knowledge, for many problems we now think in terms of model architectures and losses instead. While a choice of model architecture limits the types of results possible, neural networks tend to be less task dependent than domain specific methods. In fact, for almost any problem there is a fairly simple formula for using neural networks to get good results. 1. Collect labeled data, 2. Choose a network architecture, 3. Define a loss and train. However, there are still tasks where we might not be able to collect a lot of labeled data of a particular form (Grave OCR), or tasks where we can’t easily describe an unambiguous loss on easily collected data (Intrinsic Image Decomposition, Image correction including rain, cracks and glare), or a task where we want to do many similar tasks without having to train each one independently (face adjustment).

A unifying theme of my work is that generic representations can be learned from data and those learned representation can be used to make otherwise under-constrained problems tractable. Pre-deep learning this generic representation takes the form of a LEARCH-based model more recent work builds on auto-encoder representations. For authoring decompositions and removing rain, cracks, and glare, autoencoder models are learned from fake data and then shown to be applicable on real images. For learning to decompose rainy images cycle consistency losses are incorporated to learn without examples of de-rained images. In Face-to-Face transformation, an attribute sensitive image-to-image representation is pretrained and then a low dimensional representation for image attribute transformations is described. In Grave OCR we learn to generate data and learn the image decomposition model simultaneously, allowing us to learn how to predict image annotations without labeled data. Finally in evaluating intrinsic image decomposition, we explore evaluating intrinsic image models using human perception annotations. We show that human annotation evaluation has some issues and does not appear to differentiate between qualitatively different models. We propose a new task-specific procedure for evaluating intrinsic image decomposition using repainting and reshading and show that it can be used to identify differences between model that are currently unidentified.
I want to start by thanking my adviser Prof. David Forsyth. It was truly a privilege to work with him on such a broad and interesting range of projects. I will certainly miss your unfailing push to work on the cutting edge. I’d also like to thank Prof. Derek Hoiem for his advice and time, especially during the first few years of my Ph.D.

As with any graduate program, I have been honored to interact with so many smart and driven individuals. All the members of the Artificial Intelligence lab have provided me with a sounding board for ideas and friendships. In particular I’d like to thank the students that helped me get my footing including Daphne Tsatsoulis, Ian Endres, Saurabh Singh, Kevin Karsch, and Zicheng Liao. I’d like to thank Aditya Deshpande and Tanmay Gupta for the long conversations even after our projects diverged. I have to thank Kevin Shih, Aditya Deshpande, Zhizhong Li, and Daniel McKee for taking on the burden of keeping the lab servers running enabling my research. I also greatly appreciated Alice Lai and Yonatan Bisk’s friendship and the fairly constant reminders that there is more to artificial intelligence than images. Thank you to all of the faculty and staff at the University of Illinois at Urbana-Champaign, but thanks in particular to Chandra Chekuri and Matus Telgarsky for the always interesting conversations about life and thanks to Kathy Runk for her tireless reminders to register for classes. Among the other graduate students, thanks to Joe Degol, Rob Deloatch, Raj Kataria, and Ryan Musa for your friendship and camaraderie.

I would like to thank my family (David, Eileen, Linsey, and Travis) and the Unitarian Universalist Church of Urbana Champaign community for their unconditional encouragement and support. Thank you to my friends from Rensselaer Polytechnic Institute and the Illinois Mathematics and Science Academy, especially Ilya, Justin, Justin, Kristina, Michi, and Anita. I would also like to thank Brian Sea at IMSA and Prof. Charles Stewart at RPI for nurturing me and pushing me to achieve more.

I want to thank my committee. Prof. Svetlana Lazebnik, thank you for lending your focus on rigor. Prof. Alexander Schwing, thank you for lending your eye for detail. Dr. Jonathan Barron, than you for lending your deep insight. I greatly appreciate the time and energy that you put into helping me bring this document into existence.

Finally, thanks to my partner Katherine Wood who supported me in innumerable ways during my Ph.D.
# TABLE OF CONTENTS

**CHAPTER 1** INTRODUCTION .................................................. 1  
1.1 Thesis Statement ......................................................... 1  
1.2 Challenges and Contributions ........................................... 1  

**CHAPTER 2** REGRESSION AGAINST IMAGES .............................. 5  
2.1 Introduction .............................................................. 5  
2.2 Image Regression ....................................................... 7  
2.3 Implementation .......................................................... 9  
2.4 Applications ............................................................. 11  
2.5 Discussion ............................................................... 15  

**CHAPTER 3** AUTHORING IMAGE DECOMPOSITIONS WITH GENERATIVE MODELS .................................................. 18  
3.1 Introduction .............................................................. 18  
3.2 Background .............................................................. 19  
3.3 VAE Architecture ....................................................... 20  
3.4 Authored Training Data and Models ................................... 26  
3.5 Experimental Results .................................................... 28  
3.6 Comparison to Image-to-Image Models ............................... 33  
3.7 Discussion ............................................................... 33  

**CHAPTER 4** REMOVING RAIN, CRACKS, AND GLARE ................. 36  
4.1 Introduction .............................................................. 36  
4.2 Method Overview ....................................................... 37  
4.3 VAE Architecture ....................................................... 40  
4.4 Decomposition with VAEs ............................................. 42  
4.5 Image to Image Translation Baseline .................................. 45  
4.6 Experimental Results ................................................... 45  
4.7 Experiments on Real Images ............................................ 48  
4.8 Background .............................................................. 48  
4.9 Discussion ............................................................... 51  

**CHAPTER 5** LEARNING TO DECOMPOSE RAINY IMAGES ............. 52  
5.1 Introduction .............................................................. 52  
5.2 Rain Removal as Image to Image Transformation .................... 55  
5.3 Approach ............................................................... 56  
5.4 Data ................................................................. 59  
5.5 Experiments ............................................................. 60  
5.6 Discussion ............................................................... 62
CHAPTER 1: INTRODUCTION

1.1 THESIS STATEMENT

While neural networks have led to incredible performance gains in problems that require training a machine learning model and applying it on similar data, many problems are not obviously mapped into the paradigm. When a problem doesn’t map onto this framework in an obvious fashion, it can be difficult to know what to do or even how to evaluate the resulting performance. We present one work from before neural networks that used hand-defined prior models, and then present works that use neural network structures, pre-trained network modules, or prior losses to learn even when the target is ambiguous, the specific task is uncertain, or data is unavailable. We finally present an analysis of the current state of Intrinsic Image Decomposition and propose an evaluation for future Intrinsic Image work.

1.2 CHALLENGES AND CONTRIBUTIONS

1.2.1 Regression against Images

We show how a general image prior can be used to construct a learned loss function that is meaningful for image denoising, colorization, and intrinsic image decomposition. While this work was prior to neural networks, it’s illustrative to compare it to how neural-network based models learn vs how we constructed our image regression model. A neural network learns a function $f(I) = J$, where $I$ is some input image and $J$ is also an image. On the other hand, our method predicted functions of the image. The idea was that it would be difficult to predict a consistent color over any sort of distance unless we tied the pixels together in some way. As such, we allowed the method to predict various filtered versions of the image and aggregate them. The method described in this work was ultimately published as a colorization paper [1], because the denoising and intrinsic image applications did not outperform state of the art.

1.2.2 Authoring Image Decompositions with generative models

An natural extension of the regression against image work was to try to learn a neural network based image prior model. We introduce the conv-vaee and demonstrate how to use auto-encoders to learn a high quality generative image prior. We show that we can learn these priors from fake data and apply them on real data. This lets us introduce a decomposition layer (shading detail) that
would be impossible to learn to predict from data. We show that our channel priors can be used to decompose images from multiple datasets without re-training. While our image prior based method was flexible, it was unable to achieve performance better than state of the art methods on either Intrinsic Images in the Wild or the MIT dataset.

1.2.3 Removing Rain, Cracks, and Glare

We demonstrate the generic nature of the conv-vae decomposition model by apply it to a number of other tasks. In particular, we chose three image perturbation tasks where the perturbation has a strong spatial structure. In this chapter we also introduce the idea of combining optimization for estimation with direct prediction. In particular, we describe a representation that can be predicted directly from an image to produce a good seed, and then minimize starting from the seed to find a better solution.

1.2.4 Learning to Decompose Rainy Images

Focusing on rainy images, we identify an issue with applying cycle-consistency to some types of image-to-image problems. In particular for rain removal, if you fully remove the rain from an image, it’s impossible to recover the same rainy image, since rain effects have a random effect. This means that cycle-consistency must either hide information in the image or be reconstructed to allow for some latent variable representation. We propose using our conv-vae learned representations to decompose a rainy image into a “clean” image and a latent variable image, so that the exact input rainy image can be reconstructed. We show that the constraints afforded by the pre-trained representations allow for the loop closure required to train a cycle-consistent rain-removal network. However, the performance of the network on removing rain-streaks was less compelling that existing methods.

1.2.5 Face Editing

We show how to train an attribute aware auto-encoder that allows for a simple transformation construction for learning face-to-face edits. While the transformations could be learned with paired data, large scale pairs of image data with only single attribute changes don’t really exist. Instead, we propose learning the transformation by minimizing/maximizing attribute classifier responses regularized on the pretrained auto-encoder. While normally, directly minimizing a classifier would lead to either a bad image or imperceptible differences, our auto encoder regularizes the transformation making sure that the produced image looks like a real face. Unfortunately, while our
method is general, and can apply to a broad range of transformations, directly training for a single transformation is a more effective way to produce higher quality results.

1.2.6 Grave OCR

We present a method for improving a black-box optical character recognition (OCR) model for grave data. We motivate this problem by considering that black-box models are becoming more common and it’s unclear what, if anything, an end user could do to improve the results of those models. Black box models mean that you can’t simply collect additional labeled data or tack on a new layer and fine-tune the model. Instead, you have to approach the problem as finding a new image that you expect will work better than the original image. For OCR, we know what sort of image will work well. A clean black-and-white character mask should be easy for the OCR model to read. However, annotating data would be arduous. Instead, we generate plausible grave data, conditioned on a ground truth character mask. We then introduce the idea of max domain confusion to embed the generated images and real images in the same code space, and learn to predict the ground truth character masks. While this technique works well qualitatively, the predicted character masks only offer a very moderate improvement for the black-box OCR model.

1.2.7 Evaluating Intrinsic Image Methods

While intrinsic image decomposition has been widely studied in computer vision, evaluation of intrinsic image methods remains a very difficult problem. Small-scale physical image data [2] is too small to train and too constrained to test methods learned on other data. Rendered data [3] can be collected at a large scale, making training possible, but modeling issues mean that performance might not transfer to real data. Human perception labels [4] have provided a way to train and evaluate on real images, however it’s unclear whether performance on these measures correlates with performance on downstream tasks. We propose focusing on evaluating reshading and repainting images since reshading and repainting are a task that Intrinsic Image decomposition enable. We show that reshading and repainting can be used for evaluation, in particular that datasets can be collected and an evaluation criteria can be defined. We show that performance on Intrinsic Images in the Wild (IIW) Weighted Human Disagreement Ratio (WHDR) [4] is not correlated with reshading and repainting evaluation, suggesting that 1. IIW WHDR should not be used as a standalone measure of performance and 2. models for image decomposition should take into account reshading and repainting during training. We propose two ways to use reshading data, the first which is better suited to evaluation and the second better suited to training. We do not
collect datasets large enough to train models on our reshading loss at this time, but hope that the identification and validation of reshading for evaluation will encourage the collection of the necessary data.
CHAPTER 2: REGRESSION AGAINST IMAGES

Figure 2.1: We unify a range of apparently distinct vision problems using image regression. Our framework produces one or more dependent image channels from one or more input images. Our method produces near state of the art results on three apparently different problems: denoising; intrinsic image estimation; and colorization. All figures are best viewed on a high resolution monitor.

2.1 INTRODUCTION

Many problems in computer vision can be formulated as regression. For example, denoising is regressing a clean image against a noisy version; computing intrinsic images is regressing albedo and shading against an image; colorization is regressing a color image against a monochrome image. These problems are not usually seen as regression problems, because the independent variables that are estimated (eg clean image; intrinsic images; color images) themselves have complex spatial structure that must be properly estimated. This paper describes methods to regress images — or image like objects, such as intrinsic image channels — against other images while respecting spatial structure.

Each of the example problems has attracted a variety of solutions, the vast majority of which are instances of this recipe: use image data and prior knowledge to set up an optimization problem, and solve to recover the desired representation. This is natural, because good image regression solutions should have three properties. First we wish to correctly predict individual pixels. Second we wish to avoid bad spatial patterns in the output, even over long scales. Third we should be capable of predicting multiple channels, even when those channels have complex interactions. For example, RGB channels in colorization are strongly correlated and reflectance and shading
in intrinsic image decomposition sum to the image. The properties are usually in tension. For example, the best independent prediction of pixel values generally contains bad patterns. Similarly, it is traditional to predict log shading by subtracting log reflectance from the log image because doing so exploits an important relationship between the channels. Traditionally, this tension is managed by an optimization problem. A learned data term attempts to predict each pixel correctly based on some local information while hand-chosen priors enforce spatial and channel coherence. While the data terms are often learned from data and therefore portable, priors are often specific to particular problems, and can be hard to identify.

**Contributions:** This paper describes a method to learn an optimization problem whose solution is the solution to an image regression problem. The problem is fully data dependent, tractable, produces solutions which predict pixels well, discourages bad patterns, and allows interactions between channels. Our method can be trained using supervised data specific to the regression problem (for denoising, we use pairs of noisy and clean images; for intrinsic images, we use tuples of image, reflectance and shading; for colorization, we use pairs of monochrome and color images). Because we believe our method is quite general, we apply it to three different standard problems, varying only in the features used and in details of spatial representations.

2.1.1 Related Work

**Denoising:** Image denoising is a standard problem. Dictionary based methods perform well [5]. Strong recent methods exploit deep network encoding of local image structure [6]. A recent review appears in [7].

**Intrinsic images:** Splitting an image into lightness and reflectance components is a classical computer vision problem [8]. There is a strong tradition of seeing the problem as inference on a generative physical model. Write $I$ for the log image, $A$ for the log albedo image, and $S$ for the log shading image. One then assumes that $I = A + S$, and seeks solutions for $A$ and $S$ that maximize priors. A good review, together with an extremely strong method, appears in [9]. This is tradition is odd for two reasons. It is easily verified with current datasets (eg. [2, 4]) that the constraint does not apply (as a result of glossy and subsurface scattering phenomena apparently). Furthermore, the generative model has a symmetry ($(A, S) \rightarrow (A + c, S - c)$) which complicates estimation. As a result, almost all methods report relative albedo and relative shading; in comparison, our method reports absolute albedo and absolute shading, because it is discriminative.

**Colorization:** Producing a color image from a monochrome image is again a standard problem. Most current solutions are intended to be part of an authoring pipeline, and have an interactive component. There is no standard quantitative measure of performance. A good review appears in [10].
2.2 IMAGE REGRESSION

2.2.1 Determining the Objective Function

We expect the dependent variables in image regression to be one or more images, and therefore to be large. For this reason our optimization must be no worse than quadratic in the dependent variables.

We write vectors in bold (eg \( \mathbf{b} \)) and matrices in script (eg. \( \mathcal{A} \)). Write \( \mathbf{b}(I) \) and \( \mathcal{A}(I) \) functions on the independent image \( I \). Write \( \mathbf{r} \) for the dependent variable, a vectorized image. The most generic quadratic optimization can be written

\[
\frac{1}{2} \| \mathbf{b}(I) - \mathcal{A}(I) \mathbf{r} \|^2. \tag{2.1}
\]

This clearly admits too many parameters to learn.

Write \( \Pi_u \) for the matrix which selects a patch about pixel \( u \). Then the form,

\[
\sum_{u \in \text{pixels}} \frac{1}{2} \| \mathbf{b}(I, u) - \mathcal{A}(I, u) \Pi_u \mathbf{r} \|^2, \tag{2.2}
\]

is a simplification of eq (2.1) which allows us to work with smaller \( \mathcal{A} \) and \( \mathbf{b} \). However this still involves a large \( \mathcal{A} \) if we wish to have large patches of \( \mathbf{r} \). Even with the simplification this is still a very general form of image regression. In fact if we set \( \mathcal{A}(I) \) to \( \text{Id} \) and \( \mathbf{b}(I) \) to a function which queries a database of patches for the most similar \( I \) we produce a patch matcher which averages overlapping patches at each pixel.

We present a different specialization of eq (2.2). Specifically, rather than limiting ourselves to small patches (necessary for direct solutions) or a simple construction of parameters (patch based reconstruction), we use a small number of filter responses for each patch. The filter construction depends on the application (See section 2.3.2). We write the linear operator the implements filters as \( \mathcal{F} \), giving us

\[
\sum_{u \in \text{pixels}} \frac{1}{2} \| \mathbf{b}(I, u) - \mathcal{A}(I, u) \mathcal{F} \Pi_u \mathbf{r} \|^2. \tag{2.3}
\]

In this form we expect \( \mathcal{A} \) to be \( n \times m \), and \( \mathcal{F} \) to be \( m \times r \). We set \( n < m \), possibly significantly less. Similarly, \( m < r \). Qualitatively \( \mathcal{F} \) looks for a range of significant patterns in \( \mathbf{r} \), \( \mathcal{A} \) identifies combinations of those filters to predict, and \( \mathbf{b} \) determines how the filters should be predicted based on independent image information. While \( \Pi_u \mathbf{r} \) is represented parsimoniously, we do not expect rank problems in the overall optimization problem because cost functions at each pixel interact with their neighborhood. We expect the diversity of pixels in a neighborhood to ensure the resulting
problem is well behaved.

2.2.2 Learning

LEARCH is a general strategy to search for continuous optimization solutions which produce the right answer. We show that image regression can be solved using the LEARCH strategy. \[11\]

Write $\Phi(\cdot; \theta)$ as our object function. As in LEARCH, we want $\Phi(r^*; \theta) \leq \Phi(r; \theta) \forall r$. We write a margin $H(\cdot, \cdot)$ which defines a distance between two samples

$$\sum_i \left[ \Phi(r^*_i; \theta) - \min_u \{ \Phi(u; \theta) - \lambda H(r^*_i, u) \} \right]. \quad (2.4)$$

Notice that in our case the parameters $\theta$ are functions of the image, $A(I), b(I)$. The standard strategy for learning under these conditions is functional gradient descent on the objective function.

Notice that our objective admits a trivial solution ($A = 0, b = 0$). We avoid this by requiring $AA^T = \text{id}$.

An important nuisance of solving LEARCH-style problems is that at every step for every example one may need to solve an inner optimization problem ($\min_u$ in eq (2.4)). For an appropriate choice of margin this can be avoided. In particular, we chose

$$H(r^*, u) = \|A(u - r^*)\|^2. \quad (2.5)$$

With this margin, we can complete the square to retrieve a closed form solution of eq (2.4),

$$\frac{\|b - Ar^*\|^2}{2} - \frac{b^Tb}{2} + \frac{\gamma^T \gamma}{2} + \lambda \frac{r^* A^T A r^*}{2}, \quad (2.6)$$

with $\gamma = \frac{b - \lambda A r^*}{\sqrt{1 - \lambda}}$.

Such a margin may not be appropriate for all learning problems because it enforces $\Phi(u; \theta)$ grows only in some (rather than all) dimensions. However, in our case 1) our patch filters form a sufficient (even if incomplete) representation of the diversity in real image patches and 2) $A(I)$ identifies the important combination of those filters for the specific image patches we are considering.

2.2.3 Function Specifics

In theory $A(I, u)$ and $b(I, u)$ represent arbitrary functions. In practice we chose to learn functions which are sums of regression trees. At a leaf $l$ we define a contribution to the function in
the following manner. Write $\Psi(I, u)$, an arbitrary feature construction for pixel $u$ in $I$. Then $\Delta A(I, u) = A_l$, a constant matrix. and $\Delta b(I, u) = \Psi(I, u)^T B_l + b_{cl}$. While this is linear in $\Psi(I, u)$, the features in $\Psi(I, u)$ allow us to use state of the art algorithms as features.

2.3 IMPLEMENTATION

2.3.1 Learning in practice

We learn a regression forest in a manner similar to [12]. At each leaf we will take a step which decrease our object value. We write the current estimate of the functions as $b^n(I, u)$ and $A^n(I, u)$. We can compute the gradient of the functions at a sample point. Computing the gradient with respect to $b(I, u)$ proceeds in closed form as

$$
\nabla_{b(I, u)} = \frac{b^n(I, u)}{1 - \lambda} - A^n(I, u)r_u^* - \lambda A^n(I, u)r_u^* \frac{1}{1 - \lambda}.
$$

(2.7)

We write $\nabla_{A(I, u)}$ as the gradient with respect to $A(I, u)$ but due to the orthonormalization, we compute it numerically.

Given gradients for the functionals at each pixel, we can then compute splits on the values of $\Psi(I, u)$ which maximize the mean gradient on either side of the split. We find splits by taking a random projection of $\Psi(I, u)$ and then search for the best splitting threshold. At a leaf, we perform a line search in the gradient direction to find an update which decreases the objective function $\Delta b_l$ and $\Delta A_l$. Write $\alpha_l(I, u)$, an indicator for leaf membership. We therefore write the update as

$$
b^{n+1}(I) = b^n(I) + \alpha_l(I, u)\Delta b_l
$$

(2.8)

$$
A^{n+1}(I) = \text{Orth}[A^n(I) + \alpha_l(I, u)\Delta A_l].
$$

(2.9)

We depart from tradition here in our computation of the step as we perform line search at each leaf independently. This allows us to make maximal progress on each leaf, regardless of the state of the tree. We believe this is an important feature for image regression as we expect the error to be dominated by a small number of difficult to predict patches. For example, shadowed locations make albedo and shading hard to predict in intrinsic image problems. We also differ from traditional regression trees due to the orthonormalization which means that at inference time we must traverse the trees and accumulate their effects in the same order they were learned.
2.3.2 Constructing Filters

Recall that we must define a set of filters $\mathcal{F}$ for our regression. In general we select filters for each channel independently, though it is straightforward to define a cross channel filter (i.e., averaging). When constructing our filters, we always include a center selection filter for each channel. This guarantees that our method can attempt to predict each pixel independently. We want the rest of our filters to provide descriptive power for image patches. An obvious choice is bars and spots at varying scales and orientations. We also learn PCA filters, filters created from the largest eigenvalue patches, to guarantee we can encode specific dataset peculiarities.

2.3.3 Hyperparameters

Our model has hyperparameters which determine tree structure, data handling, and objective function parameters. The number of trees ($t_n$) and the maximum depth ($t_d$) define the forest parameters. The number of samples per tree ($t_s$), the minimum number of samples per leaf ($l_s$), and the number of samples from each training image ($i_s$) determine how to handle the training data. The inner dimension of $A$, the margin $\lambda$, and the number of iterations per leaf ($t_{it}$) determine the function we will learn. We perform a search over the parameters for denoising and intrinsic image decomposition and find that similar parameters perform well for both tasks.

First we search values which affect the objective function first since improvements should be independent of the tree parameters. We try the inner dimension of $A$ at values 6, 9, 12, 15, and 19. Performance increases until 19 when we run out of memory. We try the margin, $\lambda$, for values of $10^{-5}$, .1, .25, .5, and .9. We find that setting it to .25 seems to give the best performance. Likely because smaller values do not encourage the objective to discriminate between nearby values enough and larger values cause the margin to dominate.

We then search over the tree parameters. Somewhat surprisingly increasing the number of trees ($t_n$) from 5 to 20 does not improve performance markedly. We suspect this is due to having quite deep and therefore discriminative individual trees. We try varying $l_s$ from 50 to 200 and also see minimal change. We search $t_s$ for values between 1000 and 7000 and find that increasing the value gives slight improvements at the cost of speed. We set $i_s$ according to memory constraints, though generally we set it to less than $t_s \times t_n$. 


2.3.4 Inference

Write $M_{iu} = A\mathcal{F}\Pi_{iu}$. Inference requires solving the linear system

$$[\sum_{u} M_{iu}^T M_{iu}]x = [\sum_{u} M_{iu}^T]b,$$

but we cannot form or store $\mathcal{W} = [\sum_{u} M_{iu}^T M_{iu}]$ because it is too large. We use preconditioned conjugate gradient to solve the linear system. We must be able to form products $\mathcal{W}x$ with arbitrary vectors $x$, but these products are easy to form. We form $x$ as an image, convolve the layers with the filters, multiply by a sparse matrix and then filter again.

2.4 APPLICATIONS

2.4.1 Denoising

<table>
<thead>
<tr>
<th>Method</th>
<th>Filter Forest[14]</th>
<th>Bilateral Filter</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSNR</td>
<td>29.65</td>
<td>25.71</td>
<td>22.62</td>
</tr>
</tbody>
</table>

Table 2.1: We expect our quantitative results on PSNR to decrease relative to the baseline of bilateral filters because our method minimizes an objective function over filters of the image rather than just the L2 error. Our method therefore accepts higher per pixel error to decrease per-filter and consequently per-patch error. See figure 2.2 for qualitative results that demonstrate this effect.

**Dataset:** We follow the normal protocol for generating noisy images described in [13]. Given a ground truth image $y$, we produce a noisy image $x = y + z$ for $z \sim \mathcal{N}(0, \sigma^2\text{id})$. As in [14] we use the BSDS500 benchmark [15]. We report results on $\sigma = 20$. We train using 50 images from the training set, and select 5000 patches per image. In computing PSNR we mask out 3 pixels around the edge of the image due to edge of image effects.

**Features:** Our feature representation of $I$ is an image pyramid of the noisy image, derivatives of the image and the gradient magnitude.

**Filters:** We also apply a bilateral filter, 18 PCA filters (learned from patches of size 5, 7, and 9), and bar and spot filters at 3 sizes and 4 orientations. We initialize our model to predict the bilateral filter of the noisy image. Our filters predicted for the regressed image are the center pixel, bars and spots at sizes 11, 15, and 21 with angles of 0, $\pi/3$, and $2\pi/3$ and 6 PCA filters of size 11.

**Results:** We report numeric results in Table 2.1 with a strong caveat. Our method minimizes an MSE which includes long scale filters of the denoised image. It is therefore willing to accept errors in per pixel error when doing so improves long scale filter error. We therefore expect a dip in the PSNR for denoising. However, our method makes sensible choices about the spatial properties of
error, and through choice of filters, it is possible to control where error will appear. In our specific case, because we provided bar filters, the error learns to hide in true image gradients. This makes our images appear to have significantly less error even though the numerical results are worse. See figure 2.2 to see the significance of structured versus unstructured error.

2.4.2 Colorization

**Dataset:** We perform our colorization experiments on the 3 largest indoor and outdoor scene categories of SUN dataset. For each scene category, we randomly select 40 color-grayscale image pairs as training data and hold out another 18 grayscale images for testing. In addition to this, colorization also requires some prior. Earlier work on grayscale colorization assumes a prior either in form of a stroked image [10] or a single reference image [16, 17]. Our method relaxes this constraint by requiring only a scene label as prior.
Table 2.2: Quantitative results for colorization, we report the RMS error between our reconstructed RGB image and ground truth color image for different categories of the SUN Dataset.

<table>
<thead>
<tr>
<th>Scene</th>
<th>Best Match</th>
<th>Average</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Living Room</td>
<td>0.305</td>
<td>0.236</td>
<td>0.074</td>
</tr>
<tr>
<td>Bedroom</td>
<td>0.344</td>
<td>0.256</td>
<td>0.070</td>
</tr>
<tr>
<td>Kitchen</td>
<td>0.335</td>
<td>0.259</td>
<td>0.071</td>
</tr>
<tr>
<td>Beach</td>
<td>0.308</td>
<td>0.236</td>
<td>0.095</td>
</tr>
<tr>
<td>Castle</td>
<td>0.343</td>
<td>0.254</td>
<td>0.068</td>
</tr>
<tr>
<td>Outdoor</td>
<td>0.355</td>
<td>0.256</td>
<td>0.073</td>
</tr>
</tbody>
</table>

Table 2.3: RMS error per pixel for colorization with and without scene label as prior. We use 5 test images per category. These results show that having a scene label as prior helps to achieve colorization that resembles ground truth colors.

<table>
<thead>
<tr>
<th></th>
<th>Living Room</th>
<th>Bedroom</th>
<th>Beach</th>
<th>Castle</th>
</tr>
</thead>
<tbody>
<tr>
<td>W/o Label</td>
<td>0.069</td>
<td>0.108</td>
<td>0.087</td>
<td>0.080</td>
</tr>
<tr>
<td>With Label</td>
<td>0.060</td>
<td>0.065</td>
<td>0.069</td>
<td>0.063</td>
</tr>
</tbody>
</table>

**Features:** For each image, we retrieve 10 most similar images from its scene category using bag-of-features retrieval [18]. These images are re-scaled to the size of query image. We compute the average and variance at each pixel using these 10 images. These average and variance images are used as features along with image pyramids, derivatives of grayscale image and its gradient magnitude. We also add the responses to filters in LM filter bank as features. These responses embody the nature of texture around a pixel and have strong co-relation to color. The average image (of top 10 matches), computed as a feature above, is also a good guess for the colorized output and we use it as an initialization.

**Filters:** Similar to denoising, we also apply bilateral filters, PCA filters and bar and spot filters.

**Results:** Since intensity information \( I = \frac{R+G+B}{3} \) is already present in the grayscale image, we only need to estimate 2 out of the 3 channels. We transform input images to BmY \( B - \frac{R+G}{I} \) and RmG \( \frac{R-G}{I} \) channels and estimate only these two channels in our output. RGB channels can be reconstructed by using the grayscale image and estimated BmY and RmG values. The grayscale, average, our reconstruction and ground truth color image are shown for a few test images in Figure 2.4. Starting from a not so good average image, our regressor is able to produce believable color images for all our test images. Note that, spatial coherence is built into the regressor and we do not perform any post processing specific to the application of colorization. This is unlike [17] which does a graph cut optimization post estimation and [16] which only produces micro-stroked (or scribbled) image and further interpolates based on luminance values for spatial consistency.

As far as the authors are aware, we present the first quantitative analysis of colorization in table 2.2. We report the root mean squared error (RMS) per pixel for RGB channels when: (i)
best matched image, (ii) average image and (iii) image regressed by our method are, respectively, reported as colorized output. We observe that our regressor significantly improves on the baseline color reported by average image and best matched image for all scene categories. Figure 2.3 shows that our reconstruction resembles the ground truth colors closely when scene label is used and table 2.3 shows the RMS error is larger when scene labels are ignored. Ignoring the scene label implies that the train images and matched images span across scene categories.

2.4.3 Reflectance and Shading

**Dataset:** We use the MIT intrinsic image dataset. We do not use the “natural” image in [19] because the images are produced using a simplisic spherical harmonic rendering technique which cannot reproduce phenomenon such as cast shadows and interreflections. We train using the train/test split from [19].

**Features:** We use the solution from SIRFS as a feature as well as three solutions from retinex at thresholds .05, .2 and .5. We create image pyramids, derivatives, and gradient magnitudes for the image, retinex, and SIRFS output. We initialize our model to produce SIRFS.

**Filters:** Our filters predicted for the regressed image are the center pixel, bars and spots at sizes 11, 15, and 21 with angles of 0, \( \pi/3 \), and \( 2\pi/3 \) and 6 PCA filters of size 11.

**Results:** We produce two independent reconstructions. The first predicts an RGB reflectance and a single grayscale shading. The second predicts an RGB reflectance but predicts a two channel laplacian pyramid for shading. Predicting multiple channels of shading has been done before in [20] where indirect and direct irradiance are predicted and [21] where shading and material signals are predicted using a dictionary method. However, both previous methods were tuned to predict a certain type of shading. Here we produce an arbitrary, yet easier to predict, shading decomposition. We present quantitative results in Table 2.4 using the same error metrics of [19]. RS-MSE is a patch based error computed on reflectance and shading. \( \alpha \)-R-MSE is an MSE with a scaling factor. We
introduce the “new” error metric of the unscaled MSE as our predictions, which do not explicitly incorporate a generative model, do not have an inherent scaling ambiguity. Figure 2.5 shows a sample of our reconstructions and compares it to that of SIRFS.

2.5 DISCUSSION

This work was done prior to the wide adoption of deep learning. The approach was ultimately published as a colorization paper in [1], with an extension to allow for adjustment of colorization results for color-histograms, techniques which did not apply to the other problems explored in this paper. This form of learning a prior model and then using optimization techniques during inference
Figure 2.5: Here we present results for our two channel laplacian shading estimation and compare to SIRFS. Reflectance images are shown on a white background in the first row, and shading are shown below on a black background. Notice that our shading better captures the high frequency texture on the turtles shell and fawns back. Also, while the SIRFS shading smoothness term dominates the prediction of the turtle and frog’s head our method reconstructs spatial structure.
<table>
<thead>
<tr>
<th>Method</th>
<th>SIRFS [19]</th>
<th>Retinex</th>
<th>Bayesian [22]</th>
<th>Ours (1)</th>
<th>Ours (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS-MSE</td>
<td>.0170</td>
<td>.0298</td>
<td>.0136</td>
<td>.0354</td>
<td>.0411</td>
</tr>
<tr>
<td>α-R-MSE</td>
<td>.0137</td>
<td>.0285</td>
<td>.0074</td>
<td>.0238</td>
<td>.0263</td>
</tr>
<tr>
<td>α-S-MSE</td>
<td>.0079</td>
<td>.0335</td>
<td>.0092</td>
<td>.0211</td>
<td>.0271</td>
</tr>
<tr>
<td>R-MSE</td>
<td>.0304</td>
<td>.217</td>
<td>92.21</td>
<td>.0383</td>
<td>.0435</td>
</tr>
<tr>
<td>S-MSE</td>
<td>.0287</td>
<td>92.21</td>
<td>-</td>
<td>.0364</td>
<td>.0384</td>
</tr>
</tbody>
</table>

Table 2.4: Quantitative results for reflectance and shading inference. Our first method (1) predicts the shading as a single channel. The second method (2) predict shading as two channels, a laplacian pyramid.

appears in multiple places in this document and those approaches are informed by this work. While it is now standard to think about these tasks as related by learned image-to-image regression tasks [23], at the time of this work, the idea of combining these three problems, describing a generic architecture, and learning the representation was a novel departure from the literature.
CHAPTER 3: AUTHORING IMAGE DECOMPOSITIONS WITH GENERATIVE MODELS

Figure 3.1: We learn models for platonic albedo, shading, and detail independently from platonic images. We can then combine them as shown to perform decompositions tasks. For example, we can decompose the shading and detail from the vase, or the albedo and shading of the room using the same shading model trained on platonic shading. Note that the decompositions of shading for the vase captures the generalized cylindrical shape, and the shading of the corners of the room are noticeably darker. Figures best viewed in high resolution in color.

3.1 INTRODUCTION

We wish to decompose images into their component parts. Traditionally we think of objects as smooth surfaces with albedo maps on them. This leads to the plausible assumption that shading effects are smooth, and locally a function of surface normals and lighting [24]. However, as discussed in [21], real objects are not like this. Objects have shading detail; small bumps, pits, grooves, scratches etc. on the surface. These mesoscopic effects are formally due to shape, but are not captured by current intrinsic image methods because they create effects which are qualitatively different from smooth shading. It is compelling to consider methods that are capable of decomposing objects into smooth shading, shading detail, and albedo.
Unfortunately, collecting images and decomposing them into ground truth albedo, shading, and shading detail appears to be impossible. It is comparably easier to collect datasets which represent the Platonic ideals for each layer independently. For example, albedo images are piecewise constant Mondrians, shading images are realistically rendered 3D primitives, and shading detail (roughly the contribution of surface bumps) is swatches of reasonable materials (e.g., stucco walls, sand, crumpled paper) that have minimal long scale shading effects. These independently trained generative models form a generative basis for images that can be combined to produce layers that explain a full image.

Using generative models for the decomposition task won’t work unless the models are capable of producing something that looks like images. This requires significant architectural innovation, since current generative models create rather small and blurry images. Our innovation is the convolutional variant of the Variational Auto Encoder (VAE) that is capable of producing high quality images. We call this variant, a conv-VAE and evaluate its representational power on images.

**Contributions:** 1) We describe a Convolutional Variational Auto Encoder that is capable of representing high frequency image information. 2) We author models using the conv-VAE for specific platonic phenomena which generalize to the real phenomena. 3) We decompose images into intrinsic layers even when ground truth decomposition of images into the layers cannot be constructed.

### 3.2 BACKGROUND

**Image Prediction using Neural Networks:** Neural networks have been applied in relatively direct ways to predict various per pixel measures including colorization [25, 26], superresolution [27], intrinsic image decomposition [28], depth [29], surface normals, semantic labels [30], pixel values [31], various combinations [32, 33], and [23] introduce a generally applicable image-to-image translation tool.

Minimizing perceptual losses allows for the production of stylized images [34, 35] and textures [36, 37]. These perceptual losses can be used to train feedforward networks as in [38] and [39]. Perceptual losses can also be learned with GANs as in [40].

**Generative Models for images** Other recent work builds on generative models like encode-decoders, VAEs [41], or GANs [42]. VAEs have been used on images [43], faces [44, 45], inpainting [46], prediction of motion [47, 48], and room surface normals and textures [49]. GANs have been used to generate images [50] and 3D shapes [51]. When combined with VAEs, GANs can be used to learn losses [52]. The most similar work in this space to ours is [53] who use VAEs to represent an image manifold. However, our conv-VAE framework allows us to generate high
resolution images directly from a VAE.

**Intrinsic Image Decomposition:** Splitting an image into shading and albedo components is a classical computer vision problem [8], as is explaining shading by reconstructing surface normals [54]. There is a strong tradition of obtaining intrinsic images as inference on a generative physical model. Write $I$ for the log image, $A$ for the log albedo image, and $S$ for the log shading image. One then assumes that $I = A + S$, and seeks solutions for $A$ and $S$ that maximize priors. Similarly, reconstructing surface normals from shading is seen as inference on a generative physical model – one seeks a normal field $n(x)$ that explains the shading $S$ under some rendering model [54]. Barron and Malik show that attacking these problems together yields an extremely strong method for recovering albedo and shading, as well as the best current shape reconstructions [9].

These traditions are odd. It is easily verified with current datasets (eg. [2, 4]) that $I \neq A + S$ (as a result of mesostructure, glossy, and subsurface scattering phenomena). Even in constrained lab environments built to determine ground truth, it is difficult to fully separate the phenomena into the correct layers, in ground truth images from [2] the mesostructure shading bleeds into the albedo layer. Similarly, all shading models that yield tractable reconstruction methods are physically incorrect [55].

Prior to neural networks, methods for intrinsic images [56, 9, 57] used hand defined priors for the albedo and shading channels in [2]. Recent works, which use neural networks to predict intrinsic image decompositions [28, 58] directly use SINTEL [3] or IIW [4] to augment the MIT dataset because they otherwise will not have enough data. Our method, does not need ground truth decompositions because we do not train the albedo, shading, or shading detail models jointly. Instead, we train independent representations on Platonic ideals, and then combine them to form a decomposition model.

### 3.3 VAE Architecture

We will briefly describe VAEs in practice to motivate our conv-VAE. In depth theoretical discussion can be found in [41] and a nice tutorial is [59].

In essence, one can think about training a VAE as training two networks, an encoder $E(I)$ which is trained to map images $I$ to latent variables, usually called codes $z$, and a decoder $D(z)$ that is trained to map these codes to images. A variational criterion is used to ensure that (a) codes are distributed as $z \sim \mathcal{N}(0, 1)$ (b) decoding a code $D(z)$, with $z = E(I)$ yields the image $I$ and (c) decoding a code near some $z = E(I)$ yields an image close to $I$. 
3.3.1 Convolutional VAE

The basic VAE has some problems as a model for images, especially high resolution images. First, the VAE model has difficulties producing high spatial frequencies. Second, the global codes make it difficult to learn that images are shift and rotationally invariant. Third, there is no way to apply a VAE to images of varying sizes. Previous works have tried to solve these issues by generating images pixel-by-pixel, conditioning on previously seen pixels.[31]

We propose the conv-VAE for achieving these goals. Rather than creating a single global code for an image, we create a field of codes that describe local regions. This means that our latent space is a code “image” rather than a code vector as shown in figure 4.3. For example, on a 64x64 image, we would write a 128 bit code as a 4x4x8 code, where each “pixel” location impacts about one quarter of the output image. Since we replaced a fully connected layer that produced a 128 bit code with a convolution that produces 8 bit codes, it reduces the number of parameters in the network. However, the conv-VAE is still capable of reproducing images better because it strictly enforces locality in the latent space. This comes at the cost of independence between the dimensions of the code, which makes drawing valid codes from the latent space more difficult. However, for decomposition this is not a problem because we have other constraints.
3.3.2 Laplacian Conv-VAE

Training conv-VAEs to produce sharp images is still difficult because the typical reconstruction loss for $D(z)$ is $\|D(E(I)) - I\|^2$ which does not capture the importance of high frequency edges. As such, generated images are often blurry. Techniques for improving encoder-decoder results include filtering the image with a Laplacian filter to emphasize edges [45] and discretizing the continuous color space and using a cross entropy loss [31].

We extend the Laplacian filter idea by learning a VAE for each layer of a Laplacian Pyramid. This is good for three reasons. First, predicting the high frequency layers of a Laplacian Pyramid is easy because they are often 0. Second, an L2 Loss can be applied at each layer of the Laplacian Pyramid, requiring that all frequencies of the input image are correctly captured. Third, predicting each layer with a conv-VAE explicitly places additional code capacity on the higher frequencies since they are encoded as larger images. We are not the first to apply Laplacian Pyramids in generative networks. [60] train a GAN to predict a Laplacian Pyramid. Our approach differs as we do not condition across scales and instead treat our conv-VAEs at each level as independent predictors.

3.3.3 Modeling Examples with VAEs

<table>
<thead>
<tr>
<th>Filter</th>
<th>Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>128x128</td>
<td>64x64</td>
</tr>
<tr>
<td>5x5</td>
<td>64x64</td>
</tr>
<tr>
<td>5x5</td>
<td>32x32</td>
</tr>
<tr>
<td>3x3</td>
<td>32x32</td>
</tr>
<tr>
<td>4x4</td>
<td>29x29</td>
</tr>
<tr>
<td>1x1</td>
<td>29x29</td>
</tr>
</tbody>
</table>

Table 3.1: The VAE architecture we use for all of our authored models is a Laplacian conv-VAE with 4 Laplacian layers, and a code size of 4 per code “pixel”. We only describe the encoder portion, as the decoder is the reverse. For a 128x128 image, this compresses to about 8% of the original image size. In the limit as image size increases, the compression rate is about 11%.

We validate conv-VAEs by comparing to conventional VAEs reproduction on images from ImageNet. Unlike previous works, which resize images to a small size, we take 64x64 pixel crops during training. This makes the assumption that images are translation invariant explicit. We compare a conv-VAE with and without the Laplacian Pyramid to two conventional VAEs. VAE-1 has roughly the same number of network parameters, and VAE-2 has roughly the same number of latent values as our conv-VAEs. Due to the fully connected layers in the conventional VAE, it is impossible to provide both at once. Details of the full parameterization is provided in table 4.1. We
Table 3.2: Architecture for the encoding portion of the VAE, conv-VAE, and Laplacian conv-VAE for comparison. The decoding portion is the same in reverse. For a fair comparison, we compare two VAEs. VAE-1 has roughly the same number of parameters as the conv-VAE, while VAE-2 has the same code dimensionality, but drastically more parameters. When converting from Laplacian conv-VAE to conv-VAE we matched the intermediate sizes when possible, but always rounded up when necessary.

3.3.4 Decomposition with conv-VAE Models

We have described a method that can learn high quality representations for images. Assume that we have a generative model for log Platonic albedo and a generative model for log Platonic
shading. We want to use these models to decompose an image into its component parts. We note that some albedo (resp. shading) codes are more common than others. Furthermore, when there are phenomena that can be explained by either layer (eg. cast shadows) we want to force our decomposition to choose – the layers should not have strong correlation. We would like to obtain a decomposition that (a) uses common codes for each layer (b) explains the image and (c) produces decorrelated layers.

The conv-VAE models are composed of $E_a(\cdot)$ and $D_a(\cdot)$ for the albedo encoder and decoder. Similarly, $E_s(\cdot)$ and $D_s(\cdot)$ describe the VAE for shading. For the Laplacian conv-VAE, the encoder and decoder produce a set of code images and a Laplacian decomposition respectively. We build a probabilistic model of common codes per code “pixel” for albedo and shading. First, we encode
Figure 3.4: Laplacian conv-VAE trained with 64x64 patches on full images. On the left is the input image. On the right is an image created by encoding and then decoding the input image. Our model typically does a good job of handling image phenomena, though there are some mild checkerboard patterns. Best viewed in color at high resolution.
ground truth patches to recover code images. For the Laplacian conv-VAE, we recover the set of code “images”, and concatenate the codes, resizing the smaller “images” to the size of the largest “image”. We then treat each pixel as an iid draw from a distribution and fit a probability density model to the set. In practice we use a Gaussian model with a spherical covariance. Let the probability models for albedo and shading be $P_a(z)$ and $P_s(z)$. Let $z_i^a$ be the $i$th pixel of the albedo code, (resp $z_i^s$). We write the full code loss as a negative log likelihood $P(z^a, z^s) = -1/N \sum \log(P_a(z_i^a)) + \log(P_s(z_i^s))$.

As is traditional in intrinsic images, we enforce the image be explained by the codes by minimizing the residual $\mathcal{R}(z^s, z^a, I) = \| \log(I) - D_s(z^s) - D_a(z^a) \|^2$.

To decorrelate the layers, we introduce a term which is meant to force the decomposition to “make up its mind” about where a signal should live. We define an upper bound on the correlation with the Frobenius norm of the 3x3 covariance matrix of the spatially corresponding pixel values in the albedo and shading. This correlation measure is attractive because it is simple and can be applied patch-wise, at varying scales, and across Laplacian layers. Let $\text{cov}(\cdot, \cdot)$ be the operation that computes the covariance matrix by centering two signals, and then taking the mean of the outer products. Let $P$ be the number of patches and $L$ the number of Laplacian layers. Let $A_p^{(l)}$ be the $p$th patch in the $l$th Laplacian layer of the albedo prediction. The correlation is $\text{corr}(A, S) = 1/L \cdot P \sum_l \sum_p \| \text{cov}(A_p^{(l)}, S_p^{(l)}) \|_F$

Our full decomposition equation is

$$\arg\min_{z^a, z^s} \mathcal{R}(z^s, z^a, I) - \lambda_p P(z^a, z^s) + \lambda_c \text{corr}(D_s(z^s), D_a(z^a))$$ (3.1)

### 3.4 AUTHORED TRAINING DATA AND MODELS

#### 3.4.1 Albedo

Platonic albedo, figure 3.5a, is piecewise constant. We create a set of 2-color Mondrian images, where a polygon at the center has a different color than the rest of the image. The colors for these are drawn from the palette of colors from the 10-train set of MIT images. We generate 500 Mondrian images of size 150x150 to allow for 128x128 crops that are shifted or rotated.

#### 3.4.2 Shading

Platonic shading, figure 3.5b, is the effect of light on a simple smooth surface. We generate platonic shadings by rendering 3D primitives with no surface color using LuxRender. We use a
directional light source which rotates about the the object with a fixed camera view. We either provide a fill light where dark pixels get about 10% of pure white, or a weak fill light where dark pixels get about 1% of pure white. The weak fill light matches the lighting conditions found in MIT, while the stronger fill light is a good proxy for more typical lighting, where diffuse components are quite large. There are 70 images, cropped to the bounding box of the rendered object so that the images are about 500x500 pixels. During training, we take 128x128 crops from these, such that the crops lie mostly inside of the masked object.
3.4.3 Shading Detail

Platonic shading detail, figure 3.5c is the effect from small bumps on a flat smooth surface under uniform lighting. We collect swatches of materials from the Internet including sand which has wavy texture, stucco which has repetitive bumps, and creased paper which varies between the two. These images are similar to those used in video games for texturing. These swatches are mostly planar and have minimal long-scale shading effects and almost no albedo effects.\footnote{While sand technically has albedo effects caused by the varying colors of sand grains, this is different from what we think of as platonic albedo in a similar manner to how shading detail is different from platonic shading.} We remove contributions of shading and center the images at .5 by fitting a linear shading gradient to the image. We also crop to \([0, 1]\), to guarantee existence of the log image. Our dataset consists of 45 images of about 300x300 pixels.

3.4.4 Laplacian conv-VAE Training Details

Our VAE model’s architecture is described in table 3.1. We use a 4-layer Laplacian decomposition, but it is otherwise the same as the Laplacian conv-VAE used on ImageNet. We train the model to encode and decode log versions of the images so that they can be used directly for intrinsic image decomposition, where it is common to write \(\log(I) = \log(A) + \log(S)\). This also means that we do not use a sigmoid activation layer for the final output layer. We use the Adam optimizer with an exponentially decaying learning rate starting at .001 decaying every 500 iterations by .9. For \(t\), the iteration and \(\sigma(\cdot)\) the sigmoid function, our KL divergence terms weight is \(20\sigma(.02 \cdot t − .5)\). The image residual loss is weighted by 1000. An image prior, the L1 of image gradients, is weighted by .1.

3.5 EXPERIMENTAL RESULTS

We emphasize that all results are obtained with the same models of Platonic albedo (A), shading (S), and shading detail (D). Different decompositions are obtained by using different combinations of models (i.e. A,S; S,D; A,S,D; etc).

3.5.1 Albedo and Shading Decomposition

We decompose images into Albedo and shading using our authored models. We use the 10-image MIT train set to perform a cursory search of parameters for weighting the probability models, correlation, and image reconstruction term. We present quantitative results in table 3.3 on the...
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>WHDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bell et al. [4]</td>
<td>21.1</td>
</tr>
<tr>
<td>Zhao et al. [61]</td>
<td>23.7</td>
</tr>
<tr>
<td>Garces et al. [62]</td>
<td>25.9</td>
</tr>
<tr>
<td>Retinex (gray) [2]</td>
<td>27.3</td>
</tr>
<tr>
<td>Retinex (color)</td>
<td>27.4</td>
</tr>
<tr>
<td>Ours (Linear .25)</td>
<td>27.9</td>
</tr>
<tr>
<td>Shen et al. [63]</td>
<td>32.4</td>
</tr>
<tr>
<td>Ours (Linear .1)</td>
<td>34.9</td>
</tr>
<tr>
<td>Baseline (const R)</td>
<td>36.6</td>
</tr>
<tr>
<td>Ours (sRGB .1)</td>
<td>47.3</td>
</tr>
<tr>
<td>Baseline (const S)</td>
<td>51.6</td>
</tr>
</tbody>
</table>

Table 3.4: Results on IIW. All other algorithms were tuned for WHDR with a threshold of .1, ours was not. As such, we search for an optimal value for the threshold (.25) on the first 100 images from IIW, and report results. Other numbers from [4].

10-MIT test dataset for real color images. We use the typical scaled MSE measures from [9]. Our qualitative results are shown in figure 3.6. We also present quantitative results on WHDR on IIW from [4] using the same model in table 3.4. Since our model is not trained to perform decompositions which maximize the WHDR measure, we search for an optimal threshold using the first 100 images from IIW. We compare to decompositions from [4] in figure 3.7.

We set parameters as follows: .0001 for the probability models, 10k for the correlation, and 100 for the image reconstruction term. While the loss we are optimizing is drastically nonconvex, a trick that we found helpful for determining weights was to initialize the model near the ground truth decomposition, and look for weights that were stable at that point. Another important trick was finding a good initialization. For albedo and shading, we initialize shading at a smoothed version of the image, and albedo to the residual.
3.5.2 Shading and Shading Detail

Another cool application is decomposing shading into shading and shading detail. It is especially interesting because, unlike albedo and shading there is no easy way to capture images of an object with and without shading detail. As such, there are no ground truth decompositions of images into shading and shading detail. However, the effect of the decomposition is obvious. The
Figure 3.7: We compare the best results (top 5 rows, WHDR = 0.0%) from Bell et al. to our results on the same images. We also compare to good results (bottom 3 rows, WHDR = 6.5%). We did not pick any results based on our WHDR scores. Notice that our results are occasionally locally better. For example, in the third row, look at the shadow from the bowl, in the last row look at the bright light on the center of the floor.
shading image captures the global shape of an object, while the shading detail captures the texture or “feel” of the object. It is also worth noting that it works well on MIT even when the detail is in deep shadow. It is also general, and can decompose images of white generalized cylinders (vases). We show both in figure 3.8.

The details of our model are similar to the albedo and shading decomposition. We use .0001 for the probability weights, 10k for the correlation, and 100 for the image reconstruction term. We initialize the shading image to be a smoothed version of the input image on MIT (the image for vases), and the material to be a constant image.

3.5.3 Albedo, Shading, and Shading Detail

Finally, we can generalize our model to decompose images into three phenomena at once. We evaluate how well this works on MIT by composing shading and shading detail to form a single shading image. As we can see in table 3.3, incorporating these three channels improves quantitative performance, since we accurately determine that shading detail should be attributed to shading. We show output images in figure 3.9.

We use parameters .0001 for the probability models, 10k for the correlation, and we have to increase the image reconstruction term to 10000 because the correlation model contains the contribution from all pairs.
Figure 3.9: Our model can extend to decompose three layers: shading, detail, and albedo. The outputs are qualitatively very good, especially the albedos which are generally constant and the shading details which take small bumpy details. Best viewed in high resolution in color. All images are produced using the same platonic models, but in different combinations (here A,S,D).

3.6 COMPARISON TO IMAGE-TO-IMAGE MODELS

Image-to-Image translation [23], a concurrent paper with ours, demonstrates how to predict images from other images. We evaluate image-to-image translation on MIT and IIW qualitatively in figure 3.11. We find that the model is capable of learning MIT albedo prediction when trained on MIT, but the model does not generalize to IIW.

We also train an image-to-image prediction model on our authored data by creating shaded Mondrian images. These are formed by compositing Mondrians and shading images. As shown in figure 3.10, the model can predict the Mondrian albedos, but the model does not generalize to predicting albedos on MIT or IIW (figure 3.11).

3.7 DISCUSSION

This work was contemporary with Image-to-Image [23]. The general premise behind this work was an attempt to combine the value of the optimization portion of Regression Against Images (chapter 2) with the generic framework of deep learning, specifically the generative representational power of VAEs. Development of the conv-VAE architecture was an attempt to create a
higher quality generative model. While our conv-VAE model can reproduce fine image detail, it is hard to sample a conv-VAE, because codes at each pixel are not independent. While we believed that sampling codes from the conv-VAE code space would be possible, work in that direction in particular using Pixel-CNNs [64] to generate image codes, was unsuccessful. Another work that is somewhat related to our conception of using a network as a prior is [65], which estimates the parameters of a deconvolution network during inference. Read against this work, the deep image prior describes the latent code space as the deconvolution network parameters. One of the positives of this work, that still hasn’t been widely explored in the literature was the value of being able to learn from platonic ideals of the decomposition layers in the absence of real image decompositions while still offering generalization to the real data.
Figure 3.11: While Isola et. al. [23] performs well on MIT when trained with MIT (column C, rows 3 and 4), the model does not generalize to IIW as small details are eliminated (rows 1 and 2). When trained on our Mondrian and shading image patches (column D) the results are poor on both IIW and MIT as colors bleed across boundaries. Results from our model is shown in column E.
CHAPTER 4: REMOVING RAIN, CRACKS, AND GLARE

Figure 4.1: An overview of our architecture. We learn encoders, decoders, and codes \((z_A, z_I)\) for clean images and perturbations (cracks shown here). We define a code likelihood as a learned likelihood on data plus a similarity to the initial codes. We learn how to predict these codes directly from observed images and define a correlation term, which we do not visualize in the figure. We then search for modified codes that minimize \(\min_{z_A, z_I} L(z_A, \hat{z}_A) + L(z_I, \hat{z}_I) + \lambda \| R \|^2 + \text{corr}(D_A(z_A), D_I(z_I))\). We start the search at the direct predictions, and anneal \(\lambda\).

4.1 INTRODUCTION

Many computer vision problems follow this recipe: observed images are a sum of an original image and a perturbation, and one must recover the original image. Examples include: reflection removal, rain removal, glare removal and haze removal. The perturbation typically has quite strong spatial structure. However, it is hard to build learned methods that exploit this spatial structure, because there is little or no training data.

In this paper, we show that it is possible to learn methods to recover the original image and the perturbation using synthetic data, then apply these methods successfully to real data. We use novel and general learned generative models, combined with an annealed search, to obtain estimates of both image and perturbation. We show performance improvements available by using models of
both image and perturbation.

Our approach works as follows. First, we build learned generative models for both image $I$ and perturbation $P$ using synthetic data. Our models use a novel auto-encoder architecture to map a relatively small code to image (resp. perturbation). We use a direct prediction strategy to recover estimates of image and perturbation codes from an observed image. We then start a search for image code $z_I$ and perturbation code $z_P$ that are (a) close to the direct prediction codes; (b) have high likelihood under a GMM model of code probability; and (c) result in image and perturbation that sum to the observed image. A system overview is given in figure 4.1.

Using generative models for the decomposition task won’t work unless the models are capable of producing something that looks like images. This requires significant architectural innovation, since current generative models create rather small and blurry images. Our innovation is the convolutional variant of the Variational Auto Encoder (VAE) that is capable of producing high quality images. We call this variant, a conv-VAE and evaluate its representational power on images.

**Contributions:** (1) We show that our model produces strong results on four problems hitherto attacked with quite different, unlearned, methods. (2) We describe a Convolutional Variational Auto Encoder that is capable of representing image information better than current VAEs, and so makes it possible to use generative methods to solve real and established computer vision problems. (3) We show that our model beats a strong baseline (Im2im), because it is helpful to have a spatial model of the perturbation.

4.2 METHOD OVERVIEW

4.2.1 Task

We explore problems of the form $I_{obs} = I + A$. Where $I$ is an image and $A$ is a perturbation which have different structure. This is different from the problems in [23], where the task is to predict $I$ from $I_{obs}$ without assuming that anything is known about the transformation function. Problems of this form have the nice property that it is reasonable to expect that there is a single correct $I$ which is recoverable from the information contained in $I_{obs}$.

We select a diverse set of perturbations, which are feasible to simulate, difficult to find ground truth for, and are of general interest to the image editing community. For all tasks, $I$ is clean images, and $A$ is a perturbation that should be removed in an image correction pipeline. We can easily collect clean images. We chose about 3 images per class from the SUN 397 image dataset, eliminating small and grayscale images. We resize the images so that they are all roughly the same size. This gives us just over 1076 images for training and 1050 images for testing. Examples of
true phenomena and our generated phenomena are shown in figure 4.2.

- **Rain removal**: [66] provides rain streak images which can be added back to images. We use 6 with rotations for training, and 6 with rotations for testing.

- **Reflection removal**: [67] adds blurred and original images to construct artificial data. We blur the images from our clean image set, and then construct observed images by combining two random images.

- **Crack/Scratch removal**: a common problem in older photographs is scratch degradation. We use versions of cracked surfaces to construct crack alpha mats which we can add to the images to construct cracked images. We assume that all cracks are white, though this is not necessarily the case.

- **Glare removal**: This is an extreme version of reflection removal, when the reflected source is a bright light that creates a patch of saturated pixels. This effect is seen most commonly when a flash is used behind a pane of glass, it is also frequently visible in photographs that include reflections of the sun. It is also a nuisance problem in art photography. We simulate the effect by shining a flashlight at a pane of glass on a black background.

4.2.2 Approach

Our approach is similar to the GMM patch-prior of [68]. Rather than learning GMMs on patches, we learn GMMs on the code layer of a Variational Auto Encoder (VAE). This makes the prior more informative, but comes at the cost of representation simplicity. Most obviously, the initialization of the code matters because the generator network causes any loss to be strongly non-convex at the code layer. We therefore learn a translation network, similar to [23] which allows us to constrain our search to likely areas of the code space.

While the initialization is a feedforward pass through a translation encoder, the inference procedure most resembles image stylization [34] or texture generation [36]. Though our method optimizes VAE codes, rather than pixels. If our VAEs are properly trained, most codes should produce things that look locally like images.

1. **Train conv-VAEs**: For both images and perturbations, we train VAEs independently. The standard VAE loss is used, such that the resulting encoders and decoders reproduce the images and generate codes which are meaningful and compact.
Figure 4.2: Our generated images have similar structure to real photographs. From top to bottom, cracks are a serious problem in archival photographs, rain streaks are a nuisance problem, reflection can ruin photographs of animals at the zoo, and glare is a common problem in outdoor imagery. Best viewed in color at high resolution.
2. **Train translation encoders**: With the decoders from the independently trained VAEs fixed, a translation encoder is trained to predict codes for clean images and perturbations from observed images. The loss is a combination of the clean image prediction, perturbation prediction, the residual with the reconstructed observed image, and a correlation measure between image and perturbation.

3. **Inference with optimization**: During inference, we initialize at the translation encoders prediction. An optimization problem is solved which takes into account prior beliefs about code likelihood, belief in the translation prediction, observed image reproduction, and image and perturbation decorrelation. This produces a code which is better than the direct translation prediction.

4.3 **VAE ARCHITECTURE**

We need a representation for images that has a compact probabilistic representation. VAEs clearly fit these requirements. In practice, training a VAE can be thought of as training two networks, an encoder \( E(I) \) which is trained to map images \( I \) to latent variables, usually called codes \( z \), and a decoder \( D(z) \) that is trained to map these codes to images. A variational criterion is used to ensure that (a) codes are distributed as \( z \sim \mathcal{N}(0, 1) \) (b) decoding a code \( D(z) \), with \( z = E(I) \) yields the image \( I \) and (c) decoding a code near some \( z = E(I) \) yields an image close to \( I \). In depth theoretical discussion can be found in [41] and a nice tutorial is [59].

A model that has codes that are representative of images and a Gaussian interpretation is the Variational Auto Encoder (VAE). We briefly describe VAEs in practice to motivate our conv-VAE. In depth theoretical discussion can be found in [41] and a nice tutorial is [59].

In essence, one can think about training a VAE as training two networks, an encoder \( E(I) \) which is trained to map images \( I \) to latent variables, usually called codes \( z \), and a decoder \( D(z) \) that is trained to map these codes to images. A variational criterion is used to ensure that (a) codes are distributed as \( z \sim \mathcal{N}(0, 1) \) (b) decoding a code \( D(z) \), with \( z = E(I) \) yields the image \( I \) and (c) decoding a code near some \( z = E(I) \) yields an image close to \( I \).

4.3.1 **Convolutional VAE**

While VAE codes have good properties, the results from VAEs to this point are not suitable for our problem. This is because the VAE has some problems as a model for images, especially high resolution images. First, it is difficult to make a VAE model produce high spatial frequencies. Second, the global codes make it difficult to learn that images are shift invariant. Third, there is
no way to naturally apply a VAE to images of varying sizes. Previous works have tried to solve these issues by generating images pixel-by-pixel, conditioning on previously seen pixels [31], but this technique is not appropriate for optimization since the pixel by pixel prediction is explicitly correlated and would be very difficult to minimize into.

We introduce the conv-VAE, which are a fully convolutional variant of the VAE. The transformation is similar to fully convolutional networks [30] for pixel classification. That is, rather than creating a single global code for an image, the conv-VAE creates a field of codes that describe local patches. This means that the latent space is a code “image” rather than a code vector. The conv-VAE is better at reproducing images because it strictly enforces locality in the latent space. This comes at the cost of independence between dimensions of the code, since neighboring codes have overlapping impact on the image pixels. This makes drawing codes from the latent space impossible. However, for our tasks this is not a problem since there are other natural constraints. A description of the general architecture is shown in figure 4.3.

4.3.2 Modeling Examples with VAEs

We validate conv-VAEs by comparing to conventional VAEs for image reproduction for a known code. This is a good proxy for our ultimate task, but is not similar to how VAEs are typically evaluated. We train on 48900 images from the ImageNet validation set and test on 200 held out
images. Unlike previous works, which resize whole images to a small size, we take 64x64 pixel crops during training. This make the assumption that images are translation invariant explicit. We compare the conv-VAE to two conventional VAEs. VAE-1 has the same code dimensionality as conv-VAE, but drastically more parameters. VAE-2 has a small code dimensionality, but only one order of magnitude more parameters than the conv-VAE. We use the same convolutional portions of the encoder and decoder, and adjust only the code layers.

The full architecture definition are shown in table 4.1. We use strided convolutions and deconvolutions, and leaky relus at all except the code and output image layer where we do not use a nonlinearity. Each model is trained for 5 epochs using Adam with an exponentially decaying step rate. We follow VAE best practice and have an annealed KL-divergence weight that starts small and increases as a sigmoid.

We evaluate the VAEs by measuring the reconstruction error on an image. While our conv-VAE is size invariant, the traditional VAEs are only capable of handling images of the same size as their training data. We therefore, choose 200 crops one from each held out image to evaluate. The conv-VAE significantly outperforms the traditional VAEs. PSNR for the conv-vae is 23.36 vs 16.30 and 19.97 for vae-1 and 2 respectively. Reconstructed images are shown in figure 4.4.

### Table 4.1: Architecture for the encoding portion of the VAE and conv-VAE. The decoding portion is the same in reverse. For a fair comparison, we compare two VAEs. VAE-1 has the same code dimensionality as a conv-VAE, but has $2^{25}$ parameters compared to $2^{17}$ for the conv-VAE. VAE-2 has a far smaller code dimensionaly than a conv-VAE, yet still has more parameters ($2^{18}$). Layer sizes for the conv-VAE are illustrative, the image (resp. code) can be scaled together to any size.

<table>
<thead>
<tr>
<th>Filter Layer</th>
<th>Filter Layer</th>
<th>Filter Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>64x64x3</td>
<td>64x64x3</td>
</tr>
<tr>
<td>5x5 32x32x64</td>
<td>5x5 32x32x64</td>
<td>5x5 32x32x64</td>
</tr>
<tr>
<td>5x5 16x16x64</td>
<td>5x5 16x16x64</td>
<td>5x5 16x16x64</td>
</tr>
<tr>
<td>3x3 16x16x64</td>
<td>3x3 16x16x64</td>
<td>3x3 16x16x64</td>
</tr>
<tr>
<td>fc 2048</td>
<td>fc 8</td>
<td>1x1 16x16x8</td>
</tr>
<tr>
<td>Code</td>
<td>fc 2048</td>
<td>fc 8</td>
</tr>
</tbody>
</table>
Figure 4.4: The conv-VAE (d) outperforms traditional VAEs (b) and (c). VAE (b) has the same number of parameters as the conv-VAE, and VAE (c) has the same number of latent parameters as the conv-VAE. It is not possible to achieve both at the same time due to the difference between fully-connected layers and convolution layers. Best viewed in color at high resolution.

rain, cracks, and glare, we change the first layer to have stride one, and decreased the code size to 2 from 8.

VAEs are trained with two losses. One is the L2 reconstruction error $\| I - D(E(I)) \|^2$, which we compute in the YCgCo color space. We weight this with 1800. The second loss is the KL-divergence between the codes from the encoder and $\mathcal{N}(0, 1)$, which is computed in closed form with the parameterization trick. We use a sigmoid annealing rate for the KL loss. For $t$, the iteration and $\sigma(\cdot)$ the sigmoid function, the KL divergence term’s weight is $8.0\sigma(0.02 \cdot t - 0.5)$. We use the Adam optimizer with a initial learning rate of .001, with a decay of .8, after 1000 iterations.
4.4.2 Translation Encoder Details

The translation encoders $T_I(I_{obs}) = \hat{z}_I$ and $T_A(I_{obs}) = \hat{z}_A$ for image and aberation have the same architecture as their respective VAE encoders. They are trained together with a four term loss. The first term is the l2 reconstruction errors $\|I - D_I(T_I(I_{obs}))\|$ similarly for $A$. The second term is the KL-Divergence between the codes produced by the translation encoders and $\mathcal{N}(0, 1)$. The third term is the observation reproduction, $\|I_{obs} - D_A(T_A(I_{obs})) - D_I(T_I(I_{obs}))\|^2$. The fourth term is a correlation to encourage the decomposition to “make up its mind” about where a signal should live.

The correlation term is an upper bound on the spatial correlation, defined as the sum of Frobenius norms of local covariance matrices. Let $A_p$ be a patch from the perturbation image and $I_p$ be a patch from the clean image, and $I_p(i)$ be the $i$th pixel then $\text{cov}(A_p, I_p) = \sum_i A_p(i)I_p(i)^T$. The upper bound on the correlation is $\text{corr}(A, I) = \frac{1}{P} \sum_{p=1}^{P} \|\text{cov}(A_p, I_p)\|_F$.

We weight the terms as follows. The l2 reconstruction errors get a weight of 1800, the KL-divergence is a sigmoid annealing rate $8.0 \sigma(0.02 \cdot t - 0.5)$, the observation reproduction is only given a small weight of 1.0, and the correlation is given a weight of 1000000 (it takes values orders of magnitude smaller than the other terms). We use the Adam optimizer with an initial learning rate of .001, with a decay of .8, after 1000 iterations.

4.4.3 Inference

The inference loss is defined in four terms. The first term penalizes extreme departures from the translation encoder, enforced as $\|\hat{z}_A - z_A\|$ (resp. $z_I$). The second term encourages the usage of high likelihood codes. This term is learned by encoding a large set of patches from the training data, and fitting a GMM to the codes. These first two terms are code likelihoods, which we collapse into a single term for compactness here $\mathcal{L}(z_A, \hat{z}_A) = \|\hat{z}_A - z_A\| - \lambda_{gmm}\log p_{GMM}(z_A)$. The third term is the observation residual $\|R\|^2 = \|I_{obs} - D_A(T_A(I_{obs})) - D_I(T_I(I_{obs}))\|^2$. The fourth term is the correlation term $\text{corr}(D_A(z_A), D_I(z_I))$ described above. This gives us an inference model

$$\min_{z_A, z_I} \lambda_L(\mathcal{L}(z_A, \hat{z}_A) + \mathcal{L}(z_I, \hat{z}_I)) + \lambda_r\|R\|^2 + \lambda_c\text{corr}(D_A(z_A), D_I(z_I)).$$

Adam is used to optimize the loss initialized at the translation encoders predictions of the codes $\hat{z}_A$ and $\hat{z}_I$. The weights of the terms are set to, $\lambda_{gmm} = .1$, $\lambda_L = 1.0$, $\lambda_c = 10000$, and $\lambda_r$ is annealed on an exponential schedule $1.0 \cdot 10^t/300$. We show the effect of optimizing this objective with an annealed residual rate in figure 4.5. Notice that while the residual decreases it still contains high frequency image information. An inability to represent high frequencies is a limitation of
any encoder-decoder architecture that does not include skip layers, and while the conv-VAE is better, it is not immune. However, since the minimization explicitly maintain a residual, the high frequencies can be recovered by adding the residual back to the predicted image. We find that this improves the results.

4.5 IMAGE TO IMAGE TRANSLATION BASELINE

An alternative approach to the image correction problem is to treat it as Image to Image Translation using [23]. Image to Image Translation is trained by learning a translation “generator” which takes in an input image of one class and produces an output image of another. It uses an adversarial loss to learn what sorts of errors are more egregious than others as well as an L1 loss. We would expect that the method should perform quite well on these tasks. After all, a naive baseline of simply returning the input image performs quite well. In theory, this means that the generator should quickly learn the identity and then the adversary should learn the statistics of the additive error, and encourage the generator to remove it.

4.6 EXPERIMENTAL RESULTS

We present quantitative results for our four tasks in table 4.2. A naive baseline of treating the uncorrected image as correct performs well for the tasks. This demonstrates how difficult it is to achieve high performance for these tasks. We outperform image to image translation on rain removal and crack removal. We perform similarly at reflection removal, and perform worse at glare. However, looking at qualitative results in figure 4.6, we notice that neither method performs particularly well on glare removal. Our method replaces saturated pixels with gray, and image to image replaces them with blocky textures that are roughly the correct color. This is especially apparent in the windmill example.

For rain removal we notice that the results are very similar, however, occasionally, image to image translation introduces significant artifacts like those seen in the wheat picture. Similarly, for cracks, the results are often similar, but for larger cracks, the image to image technique fills them with noticeable artifacts, while our method favors an inoffensive smooth patch. These errors are indicative of a failure in the adversarial training, either failing to converge, or finding a “blind spot” of the discriminator.
Figure 4.5: Annealing the observed image reconstruction term improves results because early iterations optimize the likelihood of the codes and later iterations enforce the constraint. In the first example, the initial prediction is the wrong color, and annealing is necessary to make it predict correctly. In the second example, the direct prediction has wrongly placed some cracks in the image (see below the giraffe’s leg). These produce unlikely codes which are removed in the early iterations, and do not reappear in the later iterations. The input image is shown on the far left, just to its right is the direct prediction, from left to right, we show increasing iterations. We show the image, perturbation, and residual in the three rows.

<table>
<thead>
<tr>
<th></th>
<th>Crack</th>
<th>Rain</th>
<th>Glare</th>
<th>Refl</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ssim</td>
<td>psnr</td>
<td>ssim</td>
<td>psnr</td>
</tr>
<tr>
<td>Naive Baseline</td>
<td>0.9167</td>
<td>23.4106</td>
<td>0.9201</td>
<td>29.0304</td>
</tr>
<tr>
<td>Im-to-Im[23]</td>
<td>0.9241</td>
<td>26.5216</td>
<td>0.9299</td>
<td>29.3065</td>
</tr>
<tr>
<td>Ours (Direct)</td>
<td>0.9308</td>
<td>26.7175</td>
<td>0.9471</td>
<td>31.4956</td>
</tr>
<tr>
<td>Our (Optimize)</td>
<td><strong>0.9374</strong></td>
<td><strong>27.3175</strong></td>
<td><strong>0.9483</strong></td>
<td><strong>31.9525</strong></td>
</tr>
</tbody>
</table>

Table 4.2: Quantitative results on tasks.
Figure 4.6: Qualitative results. We show outputs for 4 images for all perturbations. Notice that failure cases of our method are oversmooth texture, while im-2-im [23] creates blocky texture. Figure best viewed at high resolution and in color.
4.7 EXPERIMENTS ON REAL IMAGES

We validate the method by applying our learned models for crack removal and rain removal on real images. For cracked photographs, we find archival portraits. While these photographs are sepia toned, and unlike anything in our training dataset, our results, shown in figure 4.7 are good. They wouldn’t be mistaken for a finished professional restoration, however they clearly improve the images. One obvious shortcoming of the results is that our method does not remove the black cracks. However, this is not a failure of our method. The method is observing the structure of the training data, and does not try to generalize outside the phenomena it was trained to correct. The correct way to handle these perturbations would be to create a new black crack dataset. This is in a way, a bias-variance tradeoff. While it might be nice to have a method that would try to generalize to fix the black cracks, it is likely that such a method would make mistakes more often and try to remove small features of the image.

For rain removal, we use photographs from [66] and compare qualitatively. Our result removes the sharpest effect of the rain, but leaves behind more of a longscale nuisance structure. Results are shown in figure 4.8.

4.8 BACKGROUND

Image Prediction using Neural Networks: Neural networks have been applied in relatively direct ways to predict various per pixel measures including colorization [25, 26], superresolution [27], intrinsic image decomposition [28], depth [29], surface normals, semantic labels [30], pixel values [31, 23] and various combinations [32, 33]. Minimizing perceptual losses allows for the production of stylized images [34, 35] and textures [36, 37]. These perceptual losses can be used to train feedforward networks as in [38] and [39]. Perceptual losses can also be learned with GANs as in [40].

Generative Models for images Other recent work builds on generative models like encode-decoders, VAEs [41], or GANs [42]. VAEs have been used on images [43], faces [44, 45], inpainting [46], prediction of motion [47, 48], and room surface normals and textures [49]. GANs have been used to generate images [50] and 3D shapes [51]. When combined with VAEs, GANs can be used to learn losses [52]. Patch priors are an important comparison point; they are not usually used to generate images, except in restoration. Zoran and Weiss demonstrate very strong performance by a GMM patch prior [68]. The most similar work in this space to ours is [53] who use VAEs to represent an image manifold. However, our conv-VAE framework allows us to generate high resolution images directly from a VAE.

Rain removal from a single image is an established computer vision problem (video involves
Figure 4.7: Automatic crack removal from archival photographs using our method trained on synthetic data. The improvement is subtle, but apparent, while the major crack is not removed because it is not white, many of the minor scratches have been removed. On the bottom, the improvement is apparent, especially on her clothing and hair. Black cracks are not removed because the training data does not contain black cracks. This is a limitation of the training data, not the method.
Figure 4.8: Our results are comparable on rain removal to the previous method of [66] which was specifically created to remove rain from images. Our images have a bit more longscale nuisance structure, but also seems to have kept more of the high frequency information which is visible in the umbrella and railroad crossing sign.

distinct strategies, review in [69]). Generally, approaches build patch models of image and rain streak, then infer by optimization, but differ by model. Kang et al. decompose images into spatial frequency components, then suppress rain signal components in the high frequency layer using a dictionary ([70]; variants in [71, 72]). Luo et al. recover image and rain fields encoded using a dictionary, close to data, and summing to the observed image [73]. These strategies tend to create difficulties in the high spatial frequency components. Li et al. use GMM priors for image and rain [66]. Our approach is a distinct, but natural, extension, in using a global autoencoded model of image and of rain.

**Crack removal** with manual input is quite successful (inpainting review in [74]; note that manual identification of cracks with inpainting gets quite high PSNR’s, in the 30’s [75]). Cornelis et al. detect cracks automatically in a panel painting with edge detection methods, then use inpainting [76]. Giakoumis et al. detect cracks with morphological methods, then inpaint [77]. No PSNR is available in either case, but qualitatively the method is successful. Our approach is distinct, in modelling the overall (rather than local) spatial structure of large cracks.

**Reflection removal** algorithms differ by reflection model. Duplicate reflections from thick
sheets of glass produce ghosting, a cue exploited by Shih et al. [78]. Their method is regularized with an image model. Springer and Weiss use a GMM mixture model, but select components by hand [79]. Levin and Weiss reconstruct layers from gradients assigned to each layer exploiting manual input [80]. Our approach is distinct in modelling the overall (rather than local) spatial structures of both image and reflection.

Glare removal, especially as it relates to taking photographs of glossy artwork or artwork behind glass is a known photography problem, however, it does not appear to be studied directly in computer vision. Gu et al. study removal of artifacts from dirty lenses and other thin occluders [81]. Koreban and Schechner remove lense flare using a physical model [82]. Li et al. study nighttime haze, where light sources induce a “glow” which bears some resemblance to our glare [83].

4.9 DISCUSSION

In an attempt to extend and prove the usefulness of the conv-VAE architecture, as well as to present a testing framework for image-to-image prediction, we proposed approaching three tasks that loosely fit the image formation model that an output image is the combination of a clean image and some perturbation. We show, similarly to chapter 2 that the perturbations can be learned from platonic collected data and then a decomposition can be applied. One significant difference with chapter 2 is that the initialization of the conv-VAE code mattered a lot more. Rather than simply initializing it at some random location, we incorporated ideas from Image-to-Image translation [23], to predict an initialization point, and then used our optimization based inference procedure to create a better “finalized” output.
CHAPTER 5: LEARNING TO DECOMPOSE RAINY IMAGES

5.1 INTRODUCTION

Many computer vision problems fit the following framework: decompose an image into components, each representing some aspect of the original image (examples include: intrinsic image decomposition, image deraining, crack removal, building image material decompositions, and so on). However, there are two very challenging features in this framework. First, models of how the components combine to produce the original image are, at best, approximate. Second, it is hard to get examples of images and components to train learned regression methods.

If real rainy and rain-free aligned images were available, there would be nothing to do: existing image to image prediction techniques (e.g., [23]) would be the best option. However, such a dataset is impossible to collect. There are two consequences. First, direct learning and evaluation of a decomposer is not available. More important, quantitative evaluation is untrustworthy, because models are evaluated on synthetic data from the same source used to train the model. This leads to a confirmation bias in favor of simple models of rain, typically models that simply insert and remove streaks.

In this paper, we use the established problem of removing rain effects from an image as a model. We show that cycleGAN methods perform poorly, because they cannot destroy information. We describe a novel method that exploits a rain effects map (REM) to encode the effects of rain. In contrast to existing models, our REM is not just added to an image to produce a rainy image. Instead, it operates as a set of latent variables representing all the complex spatial and chromatic effects produced by rain. We use a learned composer to model how image and REM combine to produce all the effects of rain. This is learned jointly with a decomposer that models how to separate the two components using spatial models of rain and of images. Our method is trained using both synthetic data (as is usual for this problem) and real data (which is very unusual).

A latent variable map is required Recent methods for mapping images learn maps from populations to populations. CycleGAN is a recent procedure for learning to map images of class A (say, rainy images) to class B (say, rain-free images) that ensures that an image passed from class A (resp. B) to B (resp. A) and then back arrives where it started ([84]). However, cycleGAN is not adapted to deraining images (figure 5.1). First, the rainy images and rain-free images do not have the same content. Rainy images tend to show outdoor scenes, and lots of greenery, making it hard to select a good class B. Second, the cycle consistency principle strongly resists destroying information. For example, start from a clean image, and generate some valid rain effects for that image. Cycle consistency requires that the rain effects are such that, when removed, we get the
Figure 5.1: Deraining results for [84], on images not seen during training (top row) and images seen during training (bottom row); these results use the original code published by the authors. Notice that the cycles are consistent and the predicted images in the rain/derained domain make sense. The de-rained rainy images are image-like and have somewhat vibrant colors. The rainy images are greener and grayer. The issues are obvious, the method has found an unsatisfactory solution to the problem. Rather than learning how to remove rain from images, it has learned something strange.

original image back. It is clear that we can generate any valid rain “onto” the image. However, start from a rainy image, and remove the rain effects. Cycle consistency requires that when rain effects are re-applied, the result is the original rainy image. This means that the de-raining step cannot remove the rain fully, because if it did it couldn’t close the loop. This phenomenon can be seen in the original paper, too ([84], Fig. 1). For example, the horse predicted from a zebra shows a faint pattern of zebra stripes in its hide. In contrast, our method can derain successfully because it creates a separate rain effect map component into which rain information can be put.

**The latent variable map must encode complex effects** Current models are weak because they assume that an image is obtained by applying a known generation procedure involving a single (or sometimes two) effect maps (detailed review below; the generation procedure typically involves adding a map to the image). These maps are defined by priors. With priors of the effects, inference is used to recover the rain-free images (eg. [66]). An alternative is to generate samples of effect maps and create a synthetic dataset to train a direct recovery as in [85]. There is a serious problem with this approach. The true generation procedure isn’t really known and so the models do not represent the effects of real rain. Real rain causes a wide range of complex effects in images (figure 5.2). Some of these, like rain streaks, puddles, and splashes, involve adding texture patterns to the image. Others involve smoothing processes such as spatial smoothing due to atmospheric haze. Finally, per pixel color shifts caused by atmospheric scattering or airlight are visible at unusually short geometric scales (because the air is wet) and so change visibly with depth even in shallow scenes. This means the effects of rain are complex and scene dependent. Questions about how heavy the rain should be, where puddles should appear, how much should colors shift,
and others complicate the generation procedure. Being able to learn a composer would alleviate many of these concerns. A learned composer can compensate for problems in the underlying prior models of the effect maps, and apply adjustments for the content of a specific image. In contrast to current models, our approach learns to compose a perturbation map with an image in a way that produces a rain-like result.

![Figure 5.2: Rain is a combination of a number of phenomena. From left to right: 1. Bluring effects cause objects in the distance to appear out of focus. 2. Colors shift to be less vibrant. 3. Puddles and splashes on surfaces can obscure textures. 4,5,6. Rain streaks can be present. Typically they appear as a oriented high frequency bright splotches though distance to the camera plays a role in how prominent they are. Most recent models handle only streaks and remove the rain effects by smoothing.](image)

**Contributions:** (1): We show that current image-to-image models are unable to handle the established problem of rain removal. (2): We show that our augmentation of cycleGan handles rain removal. (3): We show that our model performs as well or better than previous rain-removal models on real images.

5.1.1 Related work

The literature on **rain removal** in a single image typically focuses on removal of rain streaks. [70] decompose images into spatial frequency components, then suppress rain in the high frequency layers using a dictionary; variants appear in [71] and [72]. [73] recover an image and effect image from a dictionary using the constraint they sum to the image. [66] uses GMM layer priors to perform a decomposition into rain streaks and image. [85] use a neural network architecture trained to perform decomposition into effect image (rain streaks or haze) and image. Rain removal from videos is a distinct problem (there are more images of the background), and is reviewed in [69]; however, as with static images, methods focus on streaks, rather than haze removal or color correction. In contrast, our method is the first to our knowledge to attack all the effects of rain simultaneously with a learned model.

Scattering effects in wet air make parts of rain images hazy. There is a literature on **dehazing** procedures typically exploit an explicit physical model to estimate the amount of airlight and then correct each pixel for per-pixel airlight contributions which will depend on depth. [86], [87] and [88] explicitly estimate depth. Various forms of constraints on haze and image have been explored including smoothness (of various forms) [89], [90], [91], and [92]. Color-based constraints were
used in [93] and [94]. [95] explore constraints on learned features. In somewhat tangential work, [96], [97], and [83] consider nighttime dehazing which requires a different model since the light sources can be visible in the image. In contrast, our method is required to learn to dehaze with no explicit physical model or explicit depth reconstruction.

Scattering effects in wet air cause color shifts in rain images. **Color constancy** methods estimate corrections to account for illuminant color effects. Early algorithms ([8]) are strong, but it is usual to assume that illuminant color varies slowly over space (eg see [98], [99], [100]). The most relevant comparison is to [101], which uses a sliding window classification framework to estimate a spatially varying illuminant color. In contrast, our method is required to learn to correct image colors for a narrower range of effects which are mostly depth dependent, and so can vary quite quickly over space.

5.2 RAIN REMOVAL AS IMAGE TO IMAGE TRANSFORMATION

We see rain removal as a variant of the image to image transformation problem. There are three important constraints that inform our architecture, training scheme, and losses. The method must be willing to destroy and create information; it must be possible to train the method without having true rain/rain-free pairs; and we must learn models of rain effects while avoiding nuisance solutions.

Our method can appear to **destroy and create information** because we use a **decomposer** to separate the image and rain effects, and a **composer** to apply rain effects to the image. In turn, this means that the method can appear to remove information from the rain-free image by adding information to the rain effects map. We build the decomposer using auto-encoders for rain-free images and rain effect maps. Pre-training our decomposer with auto-encoders helps break the ambiguity of splitting a single image into two. The rain effects auto-encoder is trained on synthetic data, and that same data is paired with real rain-free images to train the rain-free to rainy to rain-free cycle.

A novel feature of our method is that both composer and decomposer are learned. However, this opens the door to significant problems with nuisance solutions produced by methods working in concert. For example, a learned composer may shift all colors towards green on the understanding that the decomposer shifts them all back (GAN losses seem to be insufficiently precise to stop this, as there are green, resp. brown, images; see figure 5.1). We manage this difficulty by using a relatively low capacity model for the composer and by introducing a loss that discourages the composer from changing images when there is no signal in the rain effects map.
5.3 APPROACH

For all images, our approaches use the YCgCo color space, a decorrelated color space. Details of data can be found in section 5.4.

Unlike [84] we explicitly define two different network architectures built to handle decomposition and composition rather than using a generic network architecture. This allows us to more carefully structure the architecture for the task at hand, use meaningful pre-training, and introduce constraints. For decomposition, this allows for the accounting of any rainy information that is “removed”. On the composition side, this allows for the construction of a specific rainy image. The composition setup can be thought of as a parameter trick. We wish to have a composition model that can generate all valid rain images from a single clean image. Either, the composition model has to sample from some complex distribution, a difficult problem, or we need to provide a sample for the composition model to condition on.

For the following, let $R$ be a rainy image, $I$ be a rain-free image, and $M$ be an rain-effect map (REM).

5.3.1 Decomposer

Rather than regress $I$ and $M$ against $R$ directly, we choose to build decoders $G_I(z_I)$ that produce $I$ (resp $M$) from compact codes $z_I$ (resp $z_M$). These codes are predicted from $R$ using transformation encoders, (eg. $T_I : R \rightarrow z_M$). This approach has advantages: decoders can be trained to produce only plausible images (resp REM) and the regression does not have to predict correlated variables. The full decomposer for the rain-free image is $D_I(R) = G_I(T_I(R))$. Finally, a “fixer” network $F : (R, I, M)) \rightarrow I'$ is applied which recovers high spatial frequency information which is commonly lost by encoder-decoder networks.

Pre-trained Decoders: We use a fully convolutional Variational Auto Encoder (conv-VAE) to train the decoders for rain-free images and REMs. The conv-VAE is a variant of the VAE described in [41] however, rather than learning a whole image prior, conv-VAEs learn a patch prior similar to [68]. A conv-VAE can be thought of as two networks. First, the encoder $E(x)$ which produces a produces a mean $\mu(x)$ and variance $\sigma(x)$ of a distribution on code $z$. Second, the decoder $G(z)$ which attempts to produce the input $x$. The loss is a linear combination of a data term, $\|x - G(E(x))\|$ and a distribution matching term, $D_{KL}(\mathcal{N}(\mu(x), \sigma(x)), \mathcal{N}(0, 1))$. After training, we discard the encoder, and use the decoder for our decomposer as described above.

We train our rain-free image decoder on images from the SUN dataset. We train our REM decoder on the rain-streak images from [66]. While this biases our REM towards rain-streaks, it does not prevent the composer from learning other rain phenomena. Creating an REM model that
does not require any artificial data remains an interesting line of future work.

**Transformation Encoders:** The transformation encoders $T_M : R \rightarrow z_M$ and $T_I : R \rightarrow z_I$ are learned to predict a code from a rainy image. We use the same architecture for the transformation encoders as the auto-encoders for the conv-VAE. These transformation encoders are combined with the pre-trained decoders from the conv-VAE to produce rain-free images or REMs. The transformation encoders are learned together with the composer, using a cycle-consistent loss.

**Fixer Network:** The fixer network $F : (R, I, M) \rightarrow I'$ allows us to recover the high spatial frequencies that the encoder-decoder architecture removes. It is modeled on the dynamic filter network of [102]. Specifically, we predict a filter at each pixel from the input rainy image, predicted rain-free image, and predicted rain-effect map. We then apply this filter to the luminance values of the predicted rain-free image and the input rainy image, to get a new luminance image. We use this luminance image with the color image channels from the predicted rain-free image. We train the network with an l2 reconstruction loss on artificial data after the composer and decomposer have been learned. The network learns to recover image texture information from the rainy image, without reintroducing the rain signal. As a result, the output images look qualitatively better.

5.3.2 Composer

Our composer $C : (I, M) \rightarrow R$ learns to compose a rainy image from an REM and rain-free image. Thinking about this as a composer, rather than a generator, is important. Information from the REM should be composed with the rain-free image to produce a specific rainy image. Think of the REM as an estimate of a set of latent parameters which determines which specific rainy image our composer should construct.

We explore two composer architectures.

The first is an **additive** composition model. This model is not learned. It is a standard model in the rain-removal literature, $R = I + M$. This limits our model so that it only learns the translation encoder. The representation can handle rain streaks, but will not be able to handle color-correction, dehazing, or puddles.

The second is a **dynamic filter network** model. This model is learned and the architecture is similar to the work of [102]. This model allows for local affine transformations, which are sufficient for most image editing tasks, without being overly permissive to allow for non photorealistic images ([103]). The network predicts a convolution and bias term at each pixel. We initialize the filters to represent an additive composition model, and the network parameters are learned jointly with the decomposer using a cycle-consistent loss.
5.3.3 Losses

We use the cycle-consistent loss from [84]. Cycle-consistent losses make sure that the composer and decomposer interact properly. Specifically, there are two cycles that should close. First rainy images composed from a rain-free image and an REM should decompose back to that rain-free image and REM. Any rainy image decomposes into a rain-free image and REM that should compose back to that original rainy image. Furthermore, the composition model should create images that look like real rainy images, enforced adversarially. Finally, we avoid nuisance effects with a prior that discourages the composer from deviating from the rain-free image when the REM contains no signal.

Let $I$ be a rain-free image, $M$ be an REM, and $R$ be a rainy image. A decomposer, $D : R \rightarrow (I, M)$ takes in a rainy image and returns a rain-free image and an REM. A composer $C : (I, M) \rightarrow R$ takes in a rain-free image and an REM and produces a rainy image.

Our full loss is a combination of a few terms. First the cycle-consistent terms

$$L_c(I, M, R) = \|D(C(I, M)) - (I, M)\| + \|C(D(R)) - R\|. \quad (5.1)$$

Second, we want our composer to produce pictures that are “like” real rain images. So that, for example our composer is forced to mute colors, add haze, etc rather than just insert streaks. We use an adversarial loss to enforce this requirement. Let $f$ be a fully convolutional sigmoid classification network learned to distinguish between patches of real rainy images and composed rain-free images and REMs with

$$\max_f \min_C \left[ \log(f(R)) + \log(1 - f(C(I, M))) \right]. \quad (5.2)$$

The loss for training the composer is

$$L_a(I, M) = -\log(f(C(I, M))). \quad (5.3)$$

We have found that adversarial loss on the rain-free images or REM does not help with decomposition, likely because the pre-trained decoder is effective at producing images (resp. REMs).

Finally, we handle nuisance effects, such as objects changing color unnecessarily by discouraging the composer from deviating from the rain-free image when the REM contains no signal. This takes the form of an $l_2$ distance, weighted by the amount of signal in the REM (the default value is .5) in a small Gaussian window around the pixel,

$$L_I(I, M) = \left\| \frac{I - C(I, M)}{(G \ast M - .5)^2 + \epsilon} \right\|. \quad (5.4)$$
<table>
<thead>
<tr>
<th></th>
<th>Real rain images in training</th>
<th>Only artificial data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Additive</td>
<td>Fixed</td>
</tr>
<tr>
<td>psnr</td>
<td>27.3716</td>
<td>23.6860</td>
</tr>
<tr>
<td>ssim</td>
<td>.8873</td>
<td>.81</td>
</tr>
</tbody>
</table>

Table 5.1: Results on generated artificial data for which we know the ground truth. The composition model for the artificial rain data is to add a rain-free image and a rain streak image from [84]. We train Im-2-Im, [23], on artificial data and note that it performs quite well. We train CycleGan, [84], and note that even in the artificial task, it performs poorly. Our methods are trained with real rainy images and the additive model still performs well. These quantitative results are ultimately suspect. Our fixer network does not improve quantitative results even though it clearly improves real rain qualitative results (figure 5.3). While we understand the appeal of having quantitative evaluation, quantitative results on artificial rain images will be biased. Composition models for artificial rainy data must be simplistic otherwise creating data will be difficult. Methods that perform well on the artificial data may have overfit to artificial data and may not perform well on real rainy data.

Our full objective is a linear combination of these losses. \( \min \lambda_I L_I(I, M) + \lambda_a L_a(I, M) + \lambda_c L_c(I, M, R) \). We set \( \lambda_I = .5, \lambda_a = .1 \) and \( \lambda_c = 1 \) for our experiments.

5.4 DATA

Previous works have outlined methods for creating artificial rainy data. We believe this is misguided. No matter how competent the rain rendering model, certain phenomena of rainy images will be difficult to accurately represent. We outlined a better process for building a cycle-consistent decomposer/composer architecture which allows the usage of real images for exemplar rainy and rain-free images. We use the rain streak images from [66] for our REMs, and create artificial rainy data by adding an REM and a rain-free image.

Due to the small number of rain-streak images, 12 total, 6 for training and 6 for testing, we augment the training streak images by scaling and rotating them for a total of 360 REMs. We perform the same augmentation for rainy images, creating 832 images from 36. We do not have to augment rain free images because plenty are available, we select 3 images per class from the SUN 397 image dataset, removing small and grayscale images. We resize the images such that they are all roughly the same size, maintaining the original aspect ratio. This gives us 1076 clean images for training. We also form a heldout set of 1050 clean images for testing.
Figure 5.3: We show results on images using our method with and without the final fixer network. Notice that while this adjustment hurts performance on artificial rain images in terms of PSNR and SSIM. The results with the fixer network are clearly better. Note that our initial results are smooth, and the final results restore almost all fine detail information while not reintroducing rain signal. Note also that the colors resulting from our method are quite pleasing as desaturated colors become much more vibrant, without the introduction of unrealistic coloring. See that our REM accurately finds rain signal in the rainy image. Full images are shown on the left, and cutouts are shown on the right. Best viewed in high resolution in color.

5.5 EXPERIMENTS

5.5.1 Quantitative Results: Removing fake Rain

Quantitative evaluation (table 5.1) for rain removal is somewhat untrustworthy because the artificial datasets are necessarily biased. Im-2-Im, [23] does well at removing fake rain from images but is not adapted to the relevant training required for real rain images. Even when trained and tested on artificial data, Cycle-gan [84] fails. This is likely because it won’t destroy rain information. Our additive model performs tolerably well on the artificial task. The artificial nature of this task is highlighted by the results for the fixer network. Because PSNR and SSIM scores do not strongly penalize smoothed images, our fixer appears to make things worse, even though qualitatively result are better (figure 5.3). Real rain is not like artificial data because rainy images cannot be recovered by simply adding rain streaks to rain-free images. We do not compare to [66] or [85] on our artificial data because code was not available at time of submission.
5.5.2 Qualitative Evaluation: Rain removal in real images

Removing rain from artificial images is neither here nor there. Furthermore, removing rain effects is not just removal of rain streaks; all effects should be handled. We are the first to try to remove all effects of rain in a single shot. As such, our method balances correcting the various effects and may handle some effects worse than methods specifically trained for a single effect. This is especially apparent when looking at rain streaks, an effect that previous methods have focused on. While rain streaks are a major effect in rainy images, they are not the only effect. Training to remove rain streaks, which are bright, biases previous methods towards producing overly dark images (figure 5.4). In comparison, our method brightens the dark portions of the image and saturates colors in a pleasing manner.

Removal of rain streaks also biases towards methods that identify a major orientation of rain streaks and then removes high frequency information that matches the orientation. This leads to oversmooth results with missing texture. In contrast our method errs on the side of leaving behind larger rain streaks, but maintains image texture. Finally, large rain streaks present a significant problem for any method. In the midle row of figure 5.5, the large streak on the left side of the image is replaced with a strange texture.

![Images of input and processed images with captions](image)

Figure 5.4: Comparison of our model to previous methods on real images. All methods struggle to remove strong or particularly bright rain signals (middle row). Other methods (DSC) [73], (LP) [66], and (DR) [85] tend to create dark images that are smooth, while our approach maintains image sharpness, brightens the image and saturates colors in a pleasing manner. Look at the bushes and plant leafs in the bottom two images and the bird feeder in the bottom image. We notice that DSC, LP and our method have trouble removing strong rain streaks. We use the rain streak data from LP while DR creates a new more extreme rain streak set. It is likely that using the rain streaks from DR could improve our results. Regardless, our method has clear strengths. Best viewed in high resolution in color.
Figure 5.5: Additional comparisons to [85]. Our method struggles to remove long and strong rain signals because they don’t match the signals that our rain perturbation model was trained on. However, our method handles other rain phenomena such as splashes on the concrete (top), and does not remove oriented edges such as those large fan leaves (middle) or long grass (bottom). Our method also more faithfully reproduces colors and doesn’t shift the image towards black, notice how dark the road (top), fan leaves (middle), and background (bottom) become in comparison to our method. Best viewed in high resolution in color.

5.6 DISCUSSION

This was an attempt to combine [84] with our model from chapter 4. The main focus of the paper was to allow cycle-consistent networks to store latent representational information. The goal of this work was to leverage the cycle-consistent losses along with conv-VAE representation to allow for the decomposition and then recombination in a cycle-loss. While it is interesting that our model could nearly matching image-to-image performance without ever seeing paired training data, the proposed model was difficult to generalize and the performance on the rain removal task on real data was underwhelming.
CHAPTER 6: FACE EDITING WITH TRANSFORMATION OPERATORS

6.1 INTRODUCTION

Modern image generation technologies have enabled impressive face editing to be learned from data. However, the state of the art still lacks very important properties. A good face editing system should have the following properties:

- **Identity**: It should be capable of doing nothing in the event a face already contains the attribute.
- **Source Agnostic**: The editing process should work on all faces, generalizing to faces not seen during training.
- **Conservatism**: Given a face image, the output should always be a face image.
- **Flexibility**: Editing multiple or a single attribute should be possible simply by setting knobs on an editor with minimal parameter reconfiguration.
- **Generality**: The edits that a user may want to apply should not need to be declared at the beginning of the training the model.

While current methods provide identity, source agnosticism, and conservatism, they have issues with flexibility and generality. A single fader network \[104\] can only adjust a small number of attributes declared in advance because finding an attribute invariant latent variable is not possible for more than about 5 attributes. StarGAN \[105\] has some flexibility, because it surfaces all attributes in a dataset, however all the attributes of interest must be declared at the beginning of a long training procedure, and adding another attribute requires retraining the entire architecture. In this paper we introduce our Face-TO-Face model which has all five properties by construction, and we evaluate it on a set of challenging face editing tasks.

Figure 6.1 shows an outline of our editing pipeline. An encoder produces a face latent variable. We construct a transformation operator (TO) to map face latent variables to other face latent variables. The family of TO parameterizations represents all possible actions on face latent variables. We apply a specific TO instantiation to the latent variable, then decode the result to produce an output face image. The editing workflow is: declare a list of required attributes for the transformed face; determine settings of the TO parameters to obtain that list; and then apply the TO to any face latent variable to obtain an edited result.

We construct TOs using a parametric family of transformations that can be applied to a latent code for face images and will always produce a latent code for edited face images. TOs are
TO($\cdot; \theta$) is a transformation operator. The encoder $E$ and decoder $D$ have a fixed parameterization, but the parameters of the TO are different for any particular attribute transformation. Note that TOS for attributes are independent in this formulation, as such our Face-TO-Face model can represent an arbitrary number of attribute transformations without any negative impact.

Identity: The architecture of our TO (section 6.3.2) ensures that the identity transformation can be produced. Source agnostic: As our experiments in section 6.4 show, once trained our method works on a variety of faces not seen during training. Conservatism: Our TOs are trained to map face latent variables to face latent variables, and so are conservative; figure 6.2 shows that outputs from the TO are similar or indistinguishable from the outputs from the auto encoder alone which provides strong evidence that TOs produce valid face latent variables. Flexibility: The architecture of our TO allows transformations that can erase, replace or update sections of a latent variable code; what needs to be done to produce a particular set of attributes is determined by descent on a relatively small amount of data. Furthermore, describing a TO for a new attribute is done without impacting any previously trained TOs. Generality: In our experiments (section 6.4) we show that our TO construction works on single attribute editing, multi-attribute editing, and identity editing. We use the same latent representation for all of our experiments.

---

Images that were not edited are (a), (d), (e), and (h). Attributes edited in the other images are (b)-arched eyebrows, (c)-mustache, (f)-glasses, (g)-lipstick, and (i)-smiling.
6.2 BACKGROUND

Face Editing We explore face attribute editing because it is widely studied (e.g. [106] [45] [104] [105] [107] [44]). Face attribute editing techniques typically require declaring the attributes in advance because the face representation is conditioned on the declared attributes. Instead, we present a system that combines an Auto Encoder with a novel Transformation Operation. We acknowledge that the previous works, particularly Fader Networks [104] or StarGAN [105] can produce higher quality attribute editing if the edits are declared in advance, and if data takes the form of images with labels. In particular, in order to achieve a specific edit, data that has that edit must be available. For rare attribute combinations it is possible that the attribute labeled image approaches cannot function. In comparison, our approach works by seeking a certification that an image has an attribute or attributes. For nominally independent attributes, this allows us to construct images that have uncommon attribute combinations such as women+no-arched-eyebrows+big-nose, as shown in section 6.4.2.

We are not the first to explore adjusting latent representations. Interpolating latent variables from neural networks has a long history in the literature. In particular, [108], shows that through the smart choice of offsets, one can create a compelling family of face attribute vectors. Our approach is more general than the attribute vector method in part because our model can decide whether or not an attribute is already apparent in the image and can then chose not to edit.

Image to Image Prediction Pix-2-Pix [23] learns a transformation from one image to another using pairs of images. UNIT [109] poses paired Image to Image transformation as learning a shared latent space between the two image distributions. CycleGan [84] and DiscoGan [110] propose using cycle-consistency to learn transformations without paired data. Either method is unsuited to attribute editing. The paired case requires collecting aligned images of attribute changes which is impossible for many attributes. The unpaired case is unaware of semantics, and would need to learn how to represent faces at the same time as it learns the attribute that is changing between face images. The key factor that distinguishes our method from others is that learning proceeds in two phases: first, the method learns how to represent face images; second, the method identifies what needs to be done to transform a given face to have desired properties. The first phase requires many varied face images. While the second phase requires only a way to certify that an output has desired properties (i.e. one or more classifiers). This is advantageous because it allows us to identify unusual attribute combinations (e.g. female+no-arched-eyebrows+big-nose, section 6.4.2), which may be hard to find in data.

Generative Adversarial Networks Proposed by [42] as a way to generate data which matches a data distribution using a min-max optimization with a real/fake classifier and a data generator. A number of improved GANs have been proposed, including WGAN [111], BEGAN [112], and
LSGAN [113]. GANs have also been used to augment standard image losses (e.g. [46]). We take advantage of the min-max training technique used extensively in the GAN literature.

**Perceptual Losses** Perceptual losses have been used to invert feature representations [40], perform style transfer [38], and improve the results of VAEs [114]. In comparison to previous works which use a fixed network to compute perceptual loss, we take an adversarial approach to learning a perceptual witness function concurrently with the autoencoder.

**Autoencoders** Autoregressive models [115] which can be trained to denoise data as in [116], have been used to build representations of various forms of image data. Essentially all face attribute editing and image to image methods use convolutional autoencoder (ie. encoder-decoder) architectures, with a latent variable bottleneck. We use an autoencoder to create a latent variable representation for images, which we then adjust using our TO.

We note that a variational autoencoder (VAE) [41] can be thought of as a form of TO. Specifically, adding a small offset to a VAE latent variable by design results in no change in the decoded result. Our TO’s (a) encode a richer family of transformation and (b) can be explicitly linked to desired semantic outcomes.

**Adversarial Examples** Adversarial examples [117] are images that have been modified such that classifiers incorrectly identify their content. Recent work constrains the modification to spatial transformations.[118] Our losses for training a TO are similar to adversarial losses, in particular, we wish to produce an image that has a certain attribute without changing the image “too much”. Because TOs are conservative we see perceptible changes rather than adversarial changes and our method can be thought of as probing the perceptual representations of a face attribute classifier.

### 6.3 FACE-TO-FACE MODEL

Our approach has two phases, 1) training a representation for face images based on autoencoders and 2) selecting a transformation. We describe these two phases using different terms because even though they are both trained by descent, there is an order of magnitude difference in the amount of data required to learn. For the face representation, we describe losses while for the transformation selection we describe costs. We show that this two phase process allows us to learn arbitrary attributes and demonstrate that our attribute model can extend to multiple attribute adjustment.

#### 6.3.1 Autoencoder

We want a latent representation of images that makes attributes explicit. For this we use an autoencoder trained with three losses. The first is a standard L2 reconstruction loss the second is
our innovative feature loss which creates better reconstructions, and the third is an image attribute loss which surfaces attributes in the latent space. We use a simple convolutional encoder-decoder architecture modeled on Fader Networks [104]. The encoder $E(\cdot)$ maps images to a latent variable $z$ with dimension $d$. The decoder $D(\cdot)$ maps the latent variable back to an image.

**Reconstruction Loss:** An autoencoder must reconstruct the data it sees. We use the L2 loss. For an image $I$ and an encoder-decoder $E$ and $D$ respectively, the reconstruction loss is

$$L_{\text{recon}}(I) = \|D(E(I)) - I\|^2_2. \quad (6.1)$$

**Perceptual Witness Feature Loss:** Using a perceptual witness function can improve image quality. Perceptual losses [40], rely on networks (such as VGG) which might not be suited to a specific task. Witness functions [119] are the most violated function from some function class. In particular, a witness function $f(\cdot; \theta_f)$, is the parameterization for which $f(\text{fake}; \theta_f)$ is most different from $f(\text{real}; \theta_f)$. A perceptual loss family is a simple convolutional neural network architecture (i.e. a few layers) that outputs a feature representation. Let $\Theta$ be the set of valid parameters. The perceptual witness feature loss is defined as

$$L_{\text{perceptual}}(I) = \max_{\theta_f \in \Theta} \|f(D(E(I); \theta_f)) - f(I; \theta_f)\|^2_2. \quad (6.2)$$

Solving this maximization during each gradient step is prohibitively expensive, so instead we solve an alternating minimization and maximization problem (i.e. a GAN). We limit our parameters $\theta_f$ to be within $[-\alpha, \alpha]$ as in Wasserstein GAN.[111] We make no claims about the theoretical correctness or convergence properties of this loss, but find that it improves convergence and in practice allows us to achieve a lower L2 reconstruction loss.

**Attribute Loss:** Autoencoders are under constrained and can formulate arbitrarily complex representations. We would like a latent representation that makes attributes explicit, because doing so will make the attributes easier to edit. As a practical matter, we say that an attribute is explicit if it is easy to predict from the latent representation. Let $I$ be an image and $y$ be a vector of independent binary attribute labels. Define $A(z; \theta_g)$ as a simple classifier that takes an auto-encoder latent variable and predicts $y$. Since we assume that the attributes are independent, our loss is the Binary Cross Entropy loss

$$L_{\text{latent}}(I, y) = \min_{\theta_g} \sum_i [-y_i \log(g(E(I); \theta_g)_i) - (1 - y_i) \log(1 - g(E(I); \theta_g)_i)]. \quad (6.3)$$

In practice, we use MOON reweighting from [120], but exclude it for simplicity from the above equation. Similar to the perceptual witness feature loss, explicitly solving this minimization re-
quires finding \( \theta_g \) over all images and is prohibitively expensive. Since this is a min-min problem, we add the parameters \( \theta_g \) to the autoencoder optimizer and solve jointly using gradient descent. It is not necessary to enumerate all possible attributes since many attributes are implicitly tied to others. For example emotions, like surprise or anger are adjustments of eyes and mouth attributes, and identities are an instantiation of attribute labels (e.g. Albert Einstein has a large nose, a squared chin, large eyebrows, and wavy gray hair).

**Full Loss:** Our full loss is a weighted combination of the three losses

\[
L_{AE}(x, y) = L_{recon} + \lambda_{perceptual} L_{perceptual} + \lambda_{latent} L_{latent}.
\]  

(6.4)

For all our experiments we train the autoencoder with \( \lambda_{perceptual} = .1 \) and \( \lambda_{latent} = 1.0 \) on the CelebA dataset with the 40 CelebA attributes.[121]

### 6.3.2 Transformation Operation

Our face representation consists of a family of transformations that can be applied to an autoencoder latent variable. We want the transformations to be defined with a small set of parameters; to ensure a transformation always takes a valid face latent variable to another valid face latent variable; but we also want to be sure any face latent variable can be reached by a transformation. We trained the autoencoder so that a relatively simple network could extract attributes from a latent variable. This suggests that important transformations are: clear a section of the code (so as to force down an attribute score); overwrite a section of the code (so as to add an attribute); and add some delta to the code (so as to modify an extant attribute). However, we do not know which portions of the code correspond to various attributes, and the type of action to take likely depends on the input code. This suggests a transformation function \( T \) that takes the following form

\[
Z'_l = T(Z_l; u, f, c) = Z_l f(Z_l; \theta_f, \phi_f) + u(Z_l; \theta_u, \phi_u) c(Z_l; \theta_c, \phi_c),
\]  

(6.5)

where \( f(\cdot) \), \( c(\cdot) \), and \( u(\cdot) \) are component functions. Using this transformation function, we can clear components of the latent variable when \( f \) and \( u \) return 0; overwrite when \( f \) returns 0; and add an offset when \( f \) returns 1. Note the similarity to LSTM or GRU models which also adjust latent variables in a conditional manner. Since we do not have sequence data, we do not apply \( T(\cdot) \) multiple times, and therefore do not have hidden or cell states. Removing these state parameters and applying \( T(\cdot) \) only once reduces the capacity of the module, so we compensate by using a more complex construction for the component functions (e.g. \( f(\cdot) \)) than is typical.

We now describe the details of the component functions \( f(\cdot) \), \( u(\cdot) \), and \( c(\cdot) \). We use a shared
low dimensional projection parameterized by \( \phi = \{\phi_f, \phi_u, \phi_c\} \). It makes sense to use a shared low dimensional representation for our attribute transformations because we expect that a transformation operator does not need to edit the entire latent variable. Furthermore, a shared representation helps our method identify which parts of the code are important (i.e. on the face) and which are not (i.e. background). Our transformation function has attribute specific parameters \( \theta_a = \{\theta_f, \theta_u, \theta_c\}_a \), which are determined independently for each attribute. A TO for an attribute is low dimensional, on the order of hundreds or thousands of parameters, and therefore easy to learn.

6.3.2.1 Selecting a Transformation Operator

Assume that we have already trained an autoencoder as described in Section 6.3.1 such that encoding an image \( I \) gives, \( E(I) = Z_I \) and decoding the latent variable \( Z_I \) nearly produces the input, \( D(Z_I) \approx I \). Assume also that we have an attribute classifier \( F(\cdot) \) which takes an image and produces the likelihoods for a number of attributes (e.g. male, bald, smiling, glasses). Let \( T_i(\cdot) \) be the transformation for the \( i \)th attribute. Note that, as described above, \( T_i \) and \( T_j \) share low dimensional projection parameters \( \phi \), but not their attribute parameters \( \theta_i, \theta_j \). We define two costs. First, the attribute cost, which seeks a certification that the transformed image has a particular attribute (or attributes). Second, a regularization cost which makes sure that the transformation operator has not found a trivial solution (e.g. producing the same output regardless of input).

Attribute Cost: An attribute cost is cheap if the selected transformation creates an image with the desired attributes and expensive otherwise. For the CelebA attributes, we use the binary cross entropy error for the specific attribute (or attributes) as the attribute cost. This cost assumes that attributes are independent and uncorrelated which is not correct (e.g. large noses are correlated with masculine features and arched eyebrows are correlated with feminine features), but other assumptions lead to intractable costs. For mutually exclusive attributes (e.g. identities) we use the softmax and negative log likelihood loss as the attribute cost.

Regularization cost: The transformer is capable of clearing the input latent variable and returning the same latent variable for any input. A simple robust regularization cost that verifies that the output image looks like the input image at most pixels prevents this collapse. We use the L1 loss in image space as a simple yet robust regularization cost

\[
C_{\text{reg}}(I) = |I - D(T(E(I))))|.
\]  

We weight the regularization with \( \lambda_{\text{reg}} \), a hyperparameter that we set experimentally.
6.3.3 Attribute Classifiers

In order to build our attribute cost, we need attribute classifiers. We use two distinct attribute types, CelebA attributes and Microsoft 1M identities.

**CelebA:** We use a standard LightCNN-9 architecture [122] trained with MOON loss weighted Binary Cross Entropy. [120] While the LightCNN was originally introduced for face recognition we found it performed well for attribute classification.

**Microsoft 1M:** For identity classification we use the pretrained LightCNN-9 trained on 79077 identities from the Microsoft 1M dataset. While our results are shown in color, this classifier was trained in grayscale. To deal with this, we transform our output images into grayscale before feeding them into the identity classifier. We choose not to train our own identity classifier to demonstrate the versatility of our method. We use the 40 identities with the most images according to the clean label set. These identities include politicians (e.g. JFK), generic identities (e.g. babies), historic figures (e.g. Einstein, Marilyn Monroe), and contemporary people (e.g. Paul McCartney, and Justin Bieber).

6.4 EXPERIMENTS

For our experiments, we first train an autoencoder using the losses from section 6.3.1 on the CelebA. [121]. We use the training set from Fader Networks. [104] For all our experiments we use the same autoencoder, trained with $\lambda_{perceptual} = .1$ and $\lambda_{latent} = 1.0$ on the 40 attributes from the celebA dataset. This allows a fair comparison to previous attribute editing work which used attributes from CelebA throughout training. We note in particular that we do not split the representation into attribute and appearance as in Fader Networks or enforce any particular notion of attributes as in StarGan. Instead our latent loss encourages that attributes are easily read off of the latent representation, which is a looser constraint. We train the autoencoder for 500 epochs of 10,000 images using a batch size of 64, and the Adam optimizer. Our latent variable dimension is 2048.

6.4.1 CelebA Single Attributes

We train the transformation operations on the CelebA validation set. We use a different set than in training the autoencoder because we want to make sure that our representation is not taking advantage of having “seen” an image in multiple stages of training. To select the parameters of TOs, we train for 50 epochs of 1000 images with a batch size of 2 using the Adam optimizer.

---

1. [https://github.com/AlfredXiangWu/LightCNN](https://github.com/AlfredXiangWu/LightCNN)
Figure 6.3: Attribute adjustment. We train transformation operators to adjust a single attribute at a time. Our image representation is a shared autoencoder. The first column is input image, the second column is the reconstruction using the autoencoder (i.e. without adjusting any attributes). Third through nine are images adjusted to have the following attributes Arched Eyebrows, Male, Big Nose, Glasses, High Cheek Bones, Gray Hair, and Young in that order. Notice that some attributes are clearly correlated. Arched Eyebrows adjusts the images to look feminine, while Big Nose adjusts the images to look masculine. We include Gray Hair and Eyeglasses as examples of our failure mode. For Gray Hair, the images have been adjusted to look like elderly men without changing the hair color, since we know that our results fool the attribute classifier, this is indicative of a classifier that has failed to find the correct representation. Eyeglasses on the other hand are difficult to encode and generate. In the bottom row, the original latent variable has captured the glasses, but some of the attribute adjustments remove the glasses. Notice also that the glasses column produces images that have some of the features of glasses such as the edge on the bridge of the nose, but the glasses are not complete.

We show single attribute transformations on images from the test set in figure 6.3. We notice that our attributes are clearly adjusting the image in interesting ways, however they don’t always adjust the attribute in the right way. For example, “Big Nose” adjusts the nose, but it also changes the age and gender, “Arched Eyebrows” also adjusts the makeup and gender. This is not surprising since these attributes co-occur strongly in the celebA dataset, and are therefore highly predictive of each other. To get an estimate for how accurate our results are, we report how often the attribute classifier certifies a transformed image. We report success if the probability that the attribute has the correct label is greater than 50%. We note that while the transformation operator is trained against the attribute classifier, it was not trained on the test images, therefore this is a fair and sensible evaluation. We fool the single-attribute classifier an average of 98.2% (sd 1.0%) of the time.
6.4.2 CelebA Multi-Attributes

We propose a minor variant of our method to handle correlated but non-overlapping features. Instead of training our transformations for each attribute independently, we train a transformer that can handle multiple attribute labels at the same time. We demonstrate this for three attributes, Arched Eyebrows, Big Nose, and Gender which were shown to be correlated in our single-attribute experiments. For our attributes, we have two settings on and off, we therefore have $3^2$ transformers. Using our multi-attribute transformers, we get very interesting results which we show in figure 6.4. We note that we are able to generate plausible images that have rare attribute combinations which fool all three attribute classifiers simultaneously 97.4% (sd 2.3%) of the time.

Figure 6.4: Multi-attribute adjustment. The first column is the input image, our results columns are labeled with the multi-attribute target: g for women, G for men; a for non-arched eyebrows, A for arched eyebrows; and n for small nose, N for large nose. We handle the issue of correlated attributes by training an attribute transformation to change attributes simultaneously. In this way, we are able to produce images of faces with rare attribute combinations such as men with arched eyebrows and small noses. We note that our method is adjusting other attributes that we did not constrain in order to get good results. For example, adding big noses makes the face look older. Setting the gender to feminine adds eyeshadow. Adding arched eyebrows makes the face look younger.
6.4.3 Identity Attributes

To demonstrate that our representation can extend to difficult attributes that were unseen during training, we train our transformer on the 40 most common identities from the Microsoft 1M dataset. We use the same autoencoder as in the CelebA attribute experiments, and a pre-trained identity classification network from [122]. We show identity transformations in figure 6.5. Overall, we have a top-1 fooling rate of 80.1% (sd 5.1%) for the identities on a 79077-way classifier. The perceptual quality of our identity transformations is not as good as the attribute transformations because identity transformation requires adjusting gross appearance (e.g. changing the shape of a face).

Figure 6.5: Identity adjustment. We train the transformation operator to change the identity of the face. We use the same encoder and decoder from our attribute adjustment task. Note that the encoder and decoder were unaware of face identity editing during training. It is clear that the columns share stertotypical features of the identity, for example, Albert Einstein’s large nose, Paul McCartney’s chin, or Marilyn Monroe’s lips.
6.5 DISCUSSION

Allowing for a generic editing function that doesn’t require prior knowledge of the transformation is interesting, however the tradeoff in this work was a reduction in the quality of the result. Other simultaneous papers improved the quality of face generation by taking advantage of skip layers [123]. However, combining our formulation of the problem with skip layers seemed to create problems with the pretraining not adequately constraining the transformation operator. Other contemporaneous work [124] showed that assuming the attributes to be transferred are known at training time skip connections can be used to produce high quality images.
CHAPTER 7: GRAVE OCR

7.1 INTRODUCTION

There are many problems in computer vision for which pre-trained detectors are available. Many of these models are black box detectors. The normal approach to improve a model for a different domain is transfer learning, but standard transfer learning does not apply on a black-box model. We instead frame the problem as adjusting the input data by constructing images that are more like the training data from the images from the new domain. However, learning a useful mapping with unpaired data is difficult. We can take advantage of domain knowledge to pose the problem as matching the new domain distribution using a generative model conditioned on images from the original domain. This lets us construct paired generated data and we can learn an image-to-image model. We construct our model to ensure that the output images have the right statistics by incorporating a weak detection model and that the transformation is applicable on real images using a maximum domain confusion loss.

Black-box detector models come in many forms. A model might only be available through a cloud service like those offered by Google, Microsoft, or Amazon. Other models are provided as gray-boxes. For example, state of the art object detectors incorporate box-extraction techniques that are not easily backpropable. While we could evaluate a numeric gradient doing so would be fragile, slow, and in the case of cloud services expensive. Many potential problems take this form. For example consider using a state of the detection model trained on Imagenet on images taken with a polaroid camera or images taken underwater. It’s relatively obvious that these types of images are different from Imagenet images, however if we could figure out a way to model the difference and make their appearance more like Imagenet images we would expect performance to improve.

We consider grave stone optical character recognition (OCR) because grave stones are prevalent, OCR is easy to evaluate, we know what images OCR tends to work well on, and state of the art OCR models (e.g. Google OCR) are provided in a black-box fashion. We also know that grave images represent a difficult domain for both humans and OCR models because most text in the world is formed by reflectance (color) changes while carved text is formed by shading effects. Consider the appearance of a carved curved line lit by a single directional light source. On the edge of the curve closer to the light there will be a shadow line and on the opposite edge there will be a bright line. When the curve bends enough relative to the light, the dark and light edge switch places. In order to recognize a glyph on a gravestone, an OCR model needs to be able to identify these light-dark patterns and connect them where they change orientation. Graves also
Figure 7.1: Image patches (top) and their corresponding predicted character images (bottom). The learned character image prediction removes surface detritus (left), stone discoloration (middle), and shadows (right), without eliminating character information.

have other effects that make them difficult to read. Stone textures often have higher contrast than the carved characters, meaning that in order to properly extract text from graves, a model cannot be purely bottom-up, it must also incorporate some top-down notion of what a character looks like to avoid boosting noise. However, a top-down approach can’t be too sensitive to specific characters because characters on graves include uncommon glyphs (e.g. the long/modal s which looks like an f) and carvings are often degraded due to weathering. In figure 7.1 we show graves and our corresponding character image prediction. The improvement our model provides over the image in terms of human legibility is obvious.

Similar to how a detection model trained on Imagenet won’t work well on underwater images, Google’s OCR model does not handle carving effects because it was trained to work well on a different type of image. We explore a method for recovering images that are better aligned with the data OCR would have been trained on. For graves, we lean on intrinsic images and builds a conditional generative model for generating fake grave images from clean character images using style transfer losses. We then use the paired fake grave images and clean character images to learn an image-to-image model. While this model is learned without directly accessing the OCR model, we can evaluate performance on OCR. This allows us to perform an extensive architecture and hyperparameter search that otherwise wouldn’t be possible. We show that our method clearly recovers character images that are easier for a human to read than the grave images and demonstrate improved performance on the black-box Google OCR model without directly training on OCR
7.2 RELATED WORKS

Image-to-Image [23] and Cycle-GAN [84] are effective models for learning a general purpose mapping between two distributions of images. A Cycle-GAN variant was shown to be able to transfer particular makeup appearance between faces in part through the use of plausible fake data. [125] We use a variant of the image-to-image model to create a grave-to-character image model using fake data.

To make our paired fake data, we use a conditional generative adversarial network [126], learning a transformation from ground truth character images to graves using style-transfer losses [34] to evaluate the transformation. In comparison to [125] who use a fixed and predetermined model for fake image construction, this allows us to learn the fake data construction.

Learning from fake data, and testing on real data can be thought of as transfer learning and [127] provide an overview of the literature. However, standard transfer learning approaches do not apply because they require access to the model. Fine tuning [128, 129] provides a way to pre-train a model on a large set of data, and then update it to better perform on a different data sample. Other methods for transfer learning include feature learning as in [130] or learning with minimal adjustment as in [131]. Our approach is most similar to [132], who use seek to maximize domain confusion, however unlike their model, we do not have labeled data in the real grave target domain and cannot update the detection model.

7.3 LEARNING WITHOUT TRAINING

We cannot train the black box OCR model because we don’t have access to the model, but we would still like to learn a mapping from grave images to something that humans and OCR models can use. We do this by treating the problem as an image-to-image problem. We have some idea of what sorts of images an OCR model would find easy to handle (e.g. a black and white character image) With this in mind, we wish to find an image-to-image [23] model that can turn grave images into character images. We choose the image-to-image model because it has been shown to work on a variety of problems that require both top-down and bottom-up knowledge. While it is easy to collect graves, and easy to generate character images, it is not easy to get a corresponding character image for a grave. Fortunately, we can generate plausible fake grave images. We use a conditional generative adversarial network [126] to construct a model that is capable of generating fake carving data from ground truth character images (section 7.3.1). We use style-transfer losses [34] to train
the parameters of the generative model to produce plausible fake data (section 7.3.2). Then we train an image-to-image model using a novel form of maximal domain confusion [132] based on EBGAN[133] and BEGAN[112] to encourage our model learned with fake data to generalize to real data (section 7.4). For a model overview refer to figure 7.2.

7.3.1 Generating Fake Data

We generate fake data using a conditional generative model. Given a ground truth character image $M$, we want to generate a grave that has carved characters that align with $M$. To make fake data generation tractable, we recall the image rendering model used in intrinsic image decomposition, $I = R \cdot S$. That is, an image $I$ is the multiplicative combination of reflectance $R$ and shading $S$. For graves, the reflectance is the stone texture and the shading is the carved characters plus some long scale shading effects. Ignoring the long scale shading effects, because they are minor due to the planar nature of graves, we can create a plausible fake grave by combining a stone texture and a shaded carving. While it might be concerning that we ignore the global shading effects, it is known that humans are bad at noticing inconsistent global shading effects and we expect that it is unlikely that a standard convolutional neural network would pick up on this cue, or be able to take advantage of it.

For the stone texture, we use a non-parametric model. We collect images of planar stone slabs of common grave materials, and crop them to remove any large non-stone portions. To generate a stone texture, we select a crop from one of the stone images at random. We subtract the mean color from the image crop and add back a color sampled from the color distribution of the true grave images. This non-parametric model is simple, but is essentially a powerful dictionary based texture generation technique (e.g. [134]). Let $R(D_r)$ be our reflectance function that returns a stone patch from a collection of stone images $D_r$.

For the character carving, we construct a large dataset of ground truth black-and-white character images with a variety of font styles. We use common names and epithets collected from the internet to generate a diverse collection. To construct the shading images, we “render” these black and white images by applying a parameterized convolutional filter. The parameters of the filter are hand selected using domain knowledge of shading effects for carved surfaces to capture the most prominent rendering effects. For example, a parameters represents the in plane location of the sun and another parameter represents the amount of environmental light (section 7.5.1 for full details). Let our shading function, $S(M; \theta)$ render a character image $M$ with the convolutional rendering parameters $\theta$. 

78
7.3.2 Generating plausible fake data

To generate a fake grave image we use \( J = R(D_r) \cdot S(M; \theta) \). This function takes a character image \( M \) and renders a shading image with parameters \( \theta \), and then multiplies the resulting shading image with a patch from a real stone image. While we don’t know how to set the parameters \( \theta \) apriori, we know that by construction any setting creates a fairly sensible image, and there are some settings that will be plausible graves. We search for these plausible settings using the style-transfer gram loss [34]. Since our image construction is heavily constrained, we do not worry about making sure the content is maintained and consider only improving the style reproduction.
Let $H^l$ be the operator that computes the gram matrix for the $l$th layer of VGG. Let $J$ be a set of fake grave image patches generated by our data generation function and let $I$ be a set of real grave image patches sampled uniformly at random. Our fake image style loss is

$$L_{fake}(J, I) = \sum_l \| \sum_{j \in J} H^l(j) - \sum_{i \in I} H^l(i) \|.$$  

We compute this loss with respect to mini-batches, generating a set of fake image patches and selecting a set of real grave patches. We perform this at a batch-level because unlike standard style transfer, we wish to match the distribution of grave images, rather than any particular image.

Learning the best rendering model takes the form of adjusting the sampling of $\theta$ to minimize $L_{fake}(J, I)$. We construct $S(\cdot)$ so that it is easy to compute $dS/d\theta$. To update $\theta$ sensibly, we perform a change of basis, replacing independent samples of $\theta$ from an unknown distribution with a multi-layer perceptron. This allows us to perform a deterministic sample $\theta(x)$ conditioned on a random variable $x \sim N(0, 1)$. This makes updating the distribution of $\theta$ as simple as updating the parameters of the $\theta(x)$. This construction can be thought of as a form of feed-forward style transfer as in [38]. However, instead of learning an image-to-image network, we estimate the parameters of a rendering network. Plausible fake grave data and component pieces generated from this model can be seen in figure 7.3.

![Figure 7.3: Example results from our trained fake data generation model. In generating the stone data for a particular batch, we use the mean color from the corresponding real image batch, this allows us to sample the colors from the true grave data in a simple way, and better aligns the the style transfer loss since it doesn’t have to try to correct a mean-color shift in a particular batch.](image)

7.4 CREATING A MODEL FROM FAKE DATA

While our generated fake data is plausible, it will never be perfect, and neural networks are adept at taking advantage of any signal to get good results. If we use our fake data to train an image-to-character image model, applying it to real images will not work as well as it could be-
because while the real images have some of the same effects as the fake images, the network isn’t punished for using signal that doesn’t occur in the real images. Instead of trying to improve our fake image generation further, a Sisyphean task, we incorporate an explicit generalization loss into the prediction model using a form of max domain confusion.[132] Assume that we have two encoders, one for real images and one for fake images. One way to guarantee that a model trained on fake data can be applied to real data is to make sure that the features that come from fake images are indistinguishable from the features that come from real images. While this isn’t a sufficient condition, a trivial solution of ignoring the input exists, it is a necessary condition for creating a model which will generalize to real data after being trained on fake data.

For now assume that we have an encoder $E_r(I) = z$ and decoder $D_r(z)$, such that for any real image $I$, $\|I - D_r(E_r(I))\|$ is small. If we fix the parameters of $D_r(\cdot)$ and train a new encoder $E_f(\cdot)$ such that any fake image, $J = R(D_r) \cdot S(M; \theta(x))$ is encoded to a code $E_f(J) = z_f$ that can be decoded by $D_r(z_f)$. As a loss,

$$L_{\text{confusion}}(J) = \| J - D_r(E_f(J)) \|. \quad (7.2)$$

We minimize this loss with respect to the parameters of $E_f$. While this loss appears to be an auto-encoder loss, it is not because $D_r$ fixed. Instead of learning an auto-encoder which searches for an encoder and decoder pairing that work together, we are searching for an encoder that can work with a pretrained decoder on a slightly different set of data. This loss serves as a strong regularization on the distribution of $E_f$’s output since any output has to be interpreted correctly by $D_r$.

We have produced input fake images that are similar to real images while being easy to ground truth and we’ve described how to use a pretrained auto encoder to create a representation that can’t tell between real and fake images. Now we will describe how to make an image-to-character image network. Since we have paired data, this is simply an application of image-to-image [23] on the paired data. Since we have already described an encoder, $E_f$, we will simply describe the decoder $D_f$ which takes the same inputs at $D_r$ and outputs a single channel character image. We train this directly using a binary cross entropy loss. Taking advantage of domain knowledge, instead of using a generic patchGAN, we use a character-based representation. We train a simple convolutional neural network to predict between 62 characters (a-z,A-Z,0-9), we then remove the final prediction layer, and use the second to last fully connected layer as a feature to compare the predicted character image to the ground truth character image. This lets us build in important knowledge about what errors are import to character construction (e.g. don’t remove the middle line of an 8), and which are less important (e.g. line width). Let BCE be pixel-wise binary cross entropy, $Q$ be the character-based feature construction, $M$ be a character image, and $J$ be the
generated fake image. Our prediction losses are

\[ L_{\text{pred}} = \text{BCE}(D_f(E_f(J)), M) \]  \hspace{1cm} (7.3)
\[ L_{\text{glyph}} = \|Q(D_f(E_f(J))) - Q(M)\|. \]  \hspace{1cm} (7.4)

We minimize these two losses jointly, updating the parameters of \( D_f \) and \( E_f \).

### 7.5 IMPLEMENTATION

#### 7.5.1 Character Rendering Function

Our character rendering function takes as input a 1-0 character mask and 5 parameters which are based on real effects that we can observe in grave images.

1. **The location of dark and light lines on a carved surface is dependent on the in-plane location of the sun**, which we capture by rotating the convolution with a parameter we call \( \alpha \).

2. **The contrast of the dark and light lines is dependent on the amount of environment light**, which we capture by scaling the convolution with a parameter we call \( \gamma \).

3. **The width of the dark and light lines is dependent on many factors including the depth and type of the carving and angle of the camera.** We consider these effects by adjusting the size of the convolution with a parameter we call \( w \).

4. **The out of plane position of the sun affects how deep the shadow or bright edges are.** We consider this effect by re-scaling the two pieces of the convolution filter with \( \lambda \).

5. **A factor in determining if a pixel is in shadow is how much of the in-plane hemisphere we should consider as potentially blocking the light.** We allow this to be adjusted with \( w_\alpha \).

Our filter is made up of two smooth filter parts, one which creates the shadows \( f_s \) and one which creates the brightness \( f_b \). These filters are both constructed by computing pixel distances in both angle and euclidean space. Assuming that 0,0 is the center pixel, let \( d(i,j) = (i^2 + j^2) / w \) and \( d_\alpha(i,j) = \text{dist}(\text{angle}(i,j), \alpha) / w_\alpha \), where \( \text{dist}(a, b) \) computes the angular distance (i.e. \( 0 = 2\pi \)), and \( \text{angle}(i,j) \) computes the angle between \( (i,j) \) and \( (1,0) \). The values of the filters are

\[ f_{i,j} = e^{-d(i,j)} \cdot e^{-d_\alpha(i,j)}. \]  \hspace{1cm} (7.5)

\[ d_\alpha(0,0) = 0 \]
We construct two filters $f_s$ and $f_b$, with the same parameters except that for $f_b$, $\alpha$ is incremented by $\pi$ radians. We form the final filter to be applied to the image as

$$\gamma (\lambda f_b - (1 - \lambda) f_s). \quad (7.6)$$

7.5.2 Training Details

Our completed network has four pieces, the fake data generation (parameterized by $\theta(x)$), the real image auto-encoder $(E_r, D_r)$, and the fake encoder/character image prediction $(E_f, D_f)$. We update these parts with four optimizers. The first optimizer, finds better parameterizations of $\theta(x)$ by descent on $L_{fake}$. The second, maintains the domain confusion property by descent on $L_{confusion}$ on $E_f$. The third, constructs the real image autoencoder $E_r, E_d$ by descent according to $L_{recon}$. The fourth, updates the encoder/prediction network $E_f, D_f$, by descent according to $L_{pred} + \lambda_{glyph} L_{glyph}$. All of these optimizers are Adam optimizers with a decaying learning rate, initially .1, decaying by .7 every epoch. We train for 10 epochs, with 5000 mini-batches of 8 images per epoch. We set $\lambda_{glyph} = .01$.

7.6 EXPERIMENTS

7.6.1 Data

**Graves:** We collect 111 grave images from the internet. We focus mainly on older graves which exhibit the phenomena that makes grave OCR difficult. We crop the images so that the ground truth text is unambiguous, and resize the image so that a character is about 35 to 64 pixels tall. We split this data into 85 images to be used for training and validation and 26 images to be used for testing.

**Character Images:** We generate approximately 127k black and white text images, created by rendering roughly five thousand epithets and names under 25 font types. We also generate 30k images specific to each glyph (a-z, A-Z, 0-9, and [space]) using the same 25 font types with random characters surrounding them. We use these 30k images-per-glyph to learn the character prediction model for the 63-way classification task.

**Stones:** We collect 58 mostly planar images of stones commonly used for graves; slate, sandstone, marble, and granite. We crop them to remove large non-stone portions.
Figure 7.4: Qualitative results on test grave images. While somewhat simple image processing can improve the legibility of graves for humans, our model is doing something different than the other methods. For the median filter, notice that texture on the same scale as characters is not suppressed (e.g. moss on second row). For retinex, notice that the contrast of the characters is enhanced, but the contrast of the stone texture is enhanced as well (e.g. dark vertical line in the first row). For gabor, the results depend strongly on having enough contrast in the original image to pick up edges and it will fail if there is not (e.g. dark output in the fourth row). In comparison, our method knows something about what characters should look like. This allows it to ignore the mossy texture in row 2, remove grave textures as in row 1, and recover characters even when they aren’t high contrast as in row 4.
Table 7.1: OCR mean character accuracy results for various models. Vanilla OCR, Google’s OCR meant to handle general text does not work on grave images, even after preprocessing. 

Document OCR performance is significantly better so we focus on it. Notice that sensible image preprocessing techniques such as median filters and retinex lead to performance degradation relative to the image input. This is because the task of improving a model without directly computing gradients on the model is very difficult. Many sensible adjustments to images lead to worse performance because they aren’t tuned to improve the particular model. Our 1.1% improvement is therefore indicative of a transformation that is generally useful, rather than fitting the particular model and data.

7.6.2 Evaluation

For evaluation, we compute an error by computing the Levenshtein distance to the gold text. We divide the Levenshtein distance by the length of the gold text to get a normalized distance measure which we call an error rate. To get an accuracy we subtract the error rate from 1. In most cases, the resulting value will be between 0 and 1, however for some short gold text strings or long predicted text it’s possible the value might be larger than 1 (eg Levensthtein(‘aa’, ‘b’)/len(‘b’) = 2.0). We correct this by clamping the maximum error value for any image to 1.0. This choice had no impact on our final results, with accuracy of 65%, none of the images had clamped normalized distances.

7.6.3 Architecture Search

We perform an architecture search using the 85 validation grave images as our real image set. We train our whole model, varying the depth of our Encoder-Decoder model, the dimensions of the interior code, dimensions of the skip layers, weighting of our losses, and vary the scaling of the character images. We find the best performing model, that is, the model with the highest character accuracy using the Google OCR. During this exploration we also noticed that the black-and-white character images were not particularly well read by document OCR. However, if we took a linear combination of the character image and the image we could get improvement. We think that this is because Google’s document OCR is expecting something that looks like a photograph.

Our best architecture consists of two encoders and two decoders. One encoder is for real images, and one is for fake images. One decoder produces 3-channel images, and the other decoder produces 1-channel character images. Our encoders and decoders all use skip-layers, except the first layer. Let cx-s be a convolution that returns x channels and has stride s. Our encoder are c32-1,c32-2,c64-2,c128-2,c256-2,c512-2,c64-2. Skip layers always pass 8 channel images com-
puted with one convolutinal layer except for the first and last layer. Our decoders use pixel-shuffle upsampling, producing an image at half size with 4x the number of channels as required, and then upsampling by 2x, interlacing the pixels. Let \(dx-s\) be a pixel-shuffle upsample convolution that produces an image \(s\) times larger than the input with \(x\) channels. Our decoders are \(d512-2, d256-2, d128-2, d64-2, d32-2, d32-2, c3-1\). Note that we connect our skip layers, for example the second layer in the decoder \(d256-2\) takes in an image which is 512+8 channels. All encode layers have leaky relu nonlinearities with .2 slope and decode layers have relu nonlinearities, except for the output layer.

Our MLP for \(\theta(x)\), takes as input a 64 dimensional normal vector, has a hidden layer of size 32 with sigmoid nonlinearity, and then has an output layer of 5 dimensions also with a sigmoid nonlinearity.

Our network for the character feature loss is \(c32-2, c64-2, c128-2, c128-1, f2048, f512\). To train the character feature loss, we have another fully connected layer that produces a softmax classification that we remove when using it as a loss.

### 7.6.4 Comparison Methods

To demonstrate that our deep learning approach is necessary for this task, we look at the performance of filtering based approaches. We compare to a median filter with various sizes, max angle gabor filter of various sizes, and retinex with a handful of standard parameterizations. We report only the (oracle) best result on the test data for each of these models in our results table.

### 7.7 RESULTS

We present a method for extracting a representation that is useful to humans and automatic OCR from grave images that requires no labeled training data or access to the OCR model during training. Our method demonstrates that it is possible to use carefully constructed fake data to train models that can be applied to real data. We present qualitative results in figure 7.4. Our model is a mean prediction from 12 models independently trained on the grave data. In comparison to the gabor filter method, our character image prediction is robust to the stone texture. In comparison to the median filter, our method highlights characters even if they don’t have significant contrast in the initial image, and doesn’t blur character edges. In comparison to retinex, our model does not accentuate albedo textures, instead focusing solely on text portions.

We motivated the paper by arguing that we need to think about how to improve and use pretrained black-box models. It is clear that using black-box model to perform architecture searches
resulted in an image-to-character image model that is useful for humans. We also see a slight increase in the OCR performance using our model in table 7.1. We note that improving the results of a model without directly evaluating the model is difficult. It is particularly interesting that sensible image filtering techniques do not improve the performance. Finally, in figure 7.5, we show a comparison of OCR results from our model.
Ly Memory of Mus?
MELIY UBEI TEDD ELL,
the amiable confort
s for a few fleeting years
the kind of prudent partner
the pleasing & confoling
companions of the
erar of this above
in hope of a better life
July 21786
At 44.
The fashow of this all bajleth away.
Selvour anifications on thing above
not on thingson tibi curth

In memori of
Sural Stearns.
Deitshter 01
Lich Eimethit
-Mrs. Sarah Stearns
Inno died June 16, 1900,

To the memory of
Miss CHERRY STONE,
who died
Oct, 0d D. 1806.
Et. 13.
From home I had a call to go
To help my friends when in distress,
Since here I meet u futal blow
I die in Jesus and am blest
Resigna in life in death I rest
Now I am free from care and pain;
Hope in Christ, & bi hirin blest,

Image

Figure 7.5: Results for the document OCR model. (Image 66.1% vs ours+image 67.2% accuracy) Our prepro-

Ours + Image

cessing helps clean up obvious gross errors such as the non-english text in the second image, but when the OCR already produces sensible results the improvement is rather minor. It is worth noting that the OCR occasionally produces rare glyphs e.g. AE and the modal s in the first image.
8.1 INTRODUCTION

The goal of intrinsic image decomposition is to recover a reflectance $R$ and shading $S$ from a single image $I$. An exploration of methods used to decompose single images as well as other formulations of the problem can be found in section 8.2.1. Methods for intrinsic image decomposition are widely adopted in computer graphics, for image editing, stylization, object insertion, and reshading [135, 136, 137]. Until recently the best performing method in general (Retinex) was not learned, and it remains a strong baseline. Learning an intrinsic image decomposition model is hard because all training datasets have practical disadvantages, section 8.2.2. To date, methods have been evaluated on L2 error or on a discriminative metric (IIW WHDR). Unfortunately, neither metric directly evaluates performance of intrinsic image decompositions in use. We describe a reshading metric, and show that existing evaluations do not appear to predict reshading performance. While the reshading error is not perfect, it is clearly a more direct evaluation of what intrinsic images are used for.

8.2 BACKGROUND

8.2.1 Intrinsic Image Models

Intrinsic image models can be broken down into two broad types. Optimization-based methods are characterized by describing and then minimizing a loss based on some function of the image. Direct-regression methods are characterized by learning a regressor of the shading or reflectance from the image.

The most long-lived optimization-based method is Retinex, described initially in [8]. We use the color-Retinex implementation from [2] for our experiments. The loss function is described in terms of image gradients, a reflectance image is constructed to maximize the agreement with image gradients greater than some threshold. After construction of the reflectance any unexplained image information is considered as shading. Alternatives to the Retinex threshold heuristic have been widely explored including; Adaboosted classification of gradients [138], accounting for texture [61], sparsity priors on reflectance [57], clustering priors [62], and L1 edge preservation [139]. Other optimization models incorporate hand defined priors to recover reflectance and shading [9, 20, 4]. Methods that predict the relationship between superpixels (brighter, equal, darker) and then recover the reflectance in an optimization framework using the predicted relationships are
explored in [140, 58, 141]. Incorporating smooth shading prediction into optimization is explored in [142].

The first direct regression method was [143] which predicted shading gradients from image-patch features using a mixture of experts. Direct regression methods were reintroduced using convolutional neural networks in [28]. Direct re-prediction of reflectance from optimization based methods is explored in [144] and learning direct regression models using various data and loss functions has been explored in [145, 146, 147].

8.2.2 Intrinsic Image Datasets

Learning models requires data, and for intrinsic image decomposition, collecting data is difficult. Existing intrinsic image datasets take three forms. Physically correct decomposition data provides pixel aligned images with corresponding reflectance and shading [2]. While useful, these datasets are difficult to collect in volume, and impossible to collect at a scene level. Rendered decomposition data also provides pixel aligned images with their corresponding reflectance and shading [3, 147]. Rendered data is easier to collect in volume, but modeling issues might mean that images aren’t physically correct. Human perception labels, [4, 142], ask people questions about what they perceive. While human perception is easy to collect at a large scale, collecting human perception rather than physical reflectance or shading raises evaluation and training questions. We need to consider whether we can build a model for human perception that agrees with true human perception and whether models for human perception evens agree with physical reflectance and shading.

8.2.3 Evaluation

Evaluation offers the same issues as data collection. In particular, physical accuracy of reflectance or shading is difficult to measure because it requires physically correct decomposition data and doesn’t appear to correlate well with performance anyway [136]. Discriminative evaluation using IIW presents special problems, see section 8.3. The best way to evaluate intrinsic image decomposition is to check if the image decomposition can be used for editing tasks like reshading or repainting reasonably. This requires a convention about how to construct an evaluation procedure and dataset for reshading and repainting. We describe such an evaluation procedure in section 8.4.1.
8.3 EVALUATING ON INTRINSIC IMAGES IN THE WILD

Described in [4], the Weighted Human Disagreement Ration (WHDR) is a measure of discriminative agreement with human annotators. Annotators were shown two points \((A, B)\) in an image and asked to determine whether \(A\) was a darker surface, \(B\) was darker surface, or the surfaces were about the same. The human judgments were aggregated into a judgment \(J_i\) and a confidence weight \(w_i\) for a pair of points \(i = (A, B)\). Given a function \(\hat{J}\) which transforms a reflectance \(R\) into a decision for points \(i\), the WHDR is computed as

\[
WHDR_\delta(J, R) = \frac{\sum_i w_i \cdot 1(J_i \neq \hat{J}_i(R; \delta))}{\sum_i w_i}.
\]  

(8.1)

Let \(R_A\) be the reflectance of the first point from the \(i\)th pair (resp. \(R_B\)), and \(\delta\) be the threshold. \(\hat{J}\) is defined as

\[
\hat{J}_i(R; \delta) = \begin{cases} 
1 & \text{if } R_B/R_A > 1 + \delta \\
2 & \text{if } R_A/R_B > 1 + \delta \\
E & \text{else}
\end{cases}
\]  

(8.2)

By convention \(\delta = .1\).

This evaluation can be misleading. For different choices of \(\delta\), methods which appear to have poor performance can be rather competitive, for example reporting the image as reflectance (table 8.1). This effect was partially explored in [144] who showed that a simple adjustment to the image can decrease the WHDR. This suggests that simple manipulations of the reflectance might manipulate the threshold.

We present a formal relationship between scaling the log reflectance and adjusting the WHDR \(\delta\) parameter. Let \(\delta = .1\) as is convention and let \(A\) and \(B\) be points in the reflectance image \(R\) such that \(A\) is brighter than \(B\), but they are similar enough that they are classified as equivalent. That is their ratio is \(1 + \epsilon\) with \(0 < \epsilon < \delta\).

\[
R_A/R_B = 1 + \epsilon < 1 + \delta.
\]  

(8.3)

By definition, \(\epsilon > 0\), therefore there is an \(\alpha\) such that,

\[
(1 + \epsilon)^\alpha > 1 + \delta.
\]  

(8.4)
And by substitution,

\[(R_A/R_B)\alpha > 1 + \delta \quad (8.5)\]
\[\exp(\alpha \log(R_A))/\exp(\alpha \log(R_B)) > 1 + \delta. \quad (8.6)\]

Therefore, scaling the log reflectance image by a scalar is equivalent to adjusting the WHDR parameter $\delta$.

While this cannot flip the direction of the relationship, it can change how many pairs are reported as equivalent. If we wish to increase the number of pairs we deem as “equivalent” (the same as increasing the WHDR threshold $\delta$) we set $\alpha < 1$, and if we wish to decrease the number of pairs that are deemed “equivalent” we set $\alpha > 1$ (the same as decreasing WHDR threshold $\delta$). In table 8.1 deep learning models do not benefit from a search over $\delta$. This relationship between $\delta$ and $\alpha$ explains why. Deep regression methods predict log-reflectance and are trained on WHDR data, therefore they estimate $\alpha$ during training. In comparison optimization-based methods such as Retinex improve because none of the parameters directly relate to $\delta$. While there has been real improvement on WHDR, color Retinex remains competitive with recent learned methods when this parameterization difference is accounted for.

While this suggests that WHDR is a flawed metric, omitting WHDR evaluation isn’t sensible. IIW is useful for training and differences between human perception and physical reflectance can be taken into account (section 8.5.3). More importantly, evaluating WHDR is the easiest and best way to validate a model learned on IIW. If you train on IIW, poor performance on WHDR is a sign that a trained model is missing something or has failed to learn.

### 8.4 EVALUATING RESHADING AND REPAINTING

An evaluation that reflects actual application of intrinsic images is evaluating how well the decomposition can be used to adjust the image. However, it is hard to design an evaluation procedure for image adjustment that is easy to compute, informative, and impossible to game. If you know the evaluation procedure ahead of time it is surprisingly easy to trick the evaluation by smuggling information into either the shading or reflectance to get low scores. We describe possible evaluation procedures and their downsides. We leave details of data construction and collection for section 8.4.1.

Consider an evaluation for reshading. Two images, $I_{11}$ and $I_{12}$, are taken of the same scene under different lighting. A decomposition model computes shading and reflectance as $S(\cdot)$ and $R(\cdot)$ respectively. A reshading loss is computed as $\mathcal{L}(I_{11}, R(I_{12})S(I_{11}))$. $\mathcal{L}$ is some arbitrary choice of image-loss (e.g. L2 or Neural). This reshading evaluation with an L2 loss has been used
### Table 8.1: Oracle WHDR results found by searching the WHDR threshold at .05 intervals from 0 to 1. Direct regression methods have parameters that are directly optimized for the WHDR threshold, while optimization-based methods do not. Therefore, the largest improvements in performance is unsurprisingly in the initial methods tested in [4]. *Best result reported in CGIntrinsics was 14.8%. Which included a post processing step that was not applied to the images released by the authors.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Best $\delta$</th>
<th>Image type</th>
<th>WHDR at best $\delta$</th>
<th>WHDR at $\delta = .1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fan et al. [145]</td>
<td>0.10</td>
<td>linear</td>
<td>14.4%</td>
<td>14.4%</td>
</tr>
<tr>
<td>CGIntrinsics [146]</td>
<td>0.10</td>
<td>linear</td>
<td>15.5%</td>
<td>15.5%*</td>
</tr>
<tr>
<td>Bi et al. [139]</td>
<td>0.15</td>
<td>linear</td>
<td>17.4%</td>
<td>18.1%</td>
</tr>
<tr>
<td>Bell et al. [4]</td>
<td>0.40</td>
<td>srgb</td>
<td>18.2%</td>
<td>21.1%</td>
</tr>
<tr>
<td>Color Retinex [2]</td>
<td>0.65</td>
<td>srgb</td>
<td>19.1%</td>
<td>27.4%</td>
</tr>
<tr>
<td>Retinex Gray [2]</td>
<td>0.80</td>
<td>srgb</td>
<td>19.4%</td>
<td>27.3%</td>
</tr>
<tr>
<td>Image as Reflectance</td>
<td>0.60</td>
<td>linear</td>
<td>21.0%</td>
<td>51.6%</td>
</tr>
<tr>
<td>Zhao et al. [61]</td>
<td>0.15</td>
<td>srgb</td>
<td>22.1%</td>
<td>23.7%</td>
</tr>
<tr>
<td>Garces et al. [62]</td>
<td>0.30</td>
<td>srgb</td>
<td>22.1%</td>
<td>25.9%</td>
</tr>
<tr>
<td>Shen and Yeo [57]</td>
<td>0.30</td>
<td>srgb</td>
<td>22.8%</td>
<td>32.4%</td>
</tr>
<tr>
<td>Chroma as Reflectance</td>
<td>0.10</td>
<td>linear</td>
<td>36.2%</td>
<td>36.6%</td>
</tr>
<tr>
<td>Flat Reflectance</td>
<td>0.00</td>
<td>linear</td>
<td>36.5%</td>
<td>-</td>
</tr>
</tbody>
</table>

In [141], however, by simply smuggling the images into the shading $S(I) = I$, $R(I) = 1$, we can minimize the loss for any image-loss.

Consider an evaluation for repainting. Two images $I_{11}$ and $I_{21}$, are taken of a scene with that same lighting and different apparent reflectance. This apparent change can be achieved by physically adjusting objects in the image without adjusting their shape or by post-processing the image. Similar to reshading, repainting can be evaluated by image reconstruction as $\mathcal{L}(I_{11}, R(I_{12})S(I_{21}))$, $\mathcal{L}$ defined as above to be some choice of image-loss. We can defeat this evaluation, for any choice of image-loss, by smuggling the image into the reflectance, $R(I) = I$ and $S(I) = 1$.

While both reshading or repainting are easily fooled, they are fooled by smuggling the information in two different ways. This suggests that considering both reshading and repainting simultaneously can overcome the problem. We consider a 4 image set of $I_{11}, I_{12}, I_{21},$ and $I_{22}$, where $I_{ij}$ is an image taken under $i$th reflectance and $j$th shading condition. We simultaneously evaluate the contribution of reshading and repainting as $\mathcal{L}(I_{11}, R(I_{12})S(I_{21}))$. This can be computed similarly for the other 4 images. This construction of evaluation is much harder to fool. The best we can do is to assume that reflectance changes are only chroma changes. If this is the case a simple solution of, $S(I) =$ Luminance($I$) and $R(I) =$ Chroma($I$) can defeat the evaluation. However, this is easily avoided during data construction by changing the luminance of an object not just the hue.

While we describe an image reconstruction approach to evaluation, constructed datasets of multiple images with the same shading or reflectance can be evaluated against a consistency measure.
on their decompositions rather than with image reconstruction. In particular, images with the same reflectance can be evaluated by $\mathcal{L}(R(I_{11}), R(I_{12}))$ and images with the same shading can be evaluated with $\mathcal{L}(S(I_{11}), S(I_{12}))$. We do not use this construction because it can be misleading. Models that do not enforce $I_{11} = R(I_{11})S(I_{11})$, perform better by not reproducing the image fully and the assumption that reflectance and shading can be scaled arbitrarily makes meaningful evaluation difficult.

In practice, these evaluations are computed by taking the mean over images in a dataset such as those described in section 8.4.1.

8.4.1 Reshading and Repainting Datasets

We present three datasets for evaluation. Reshading data is used to train intrinsic image decomposition in [148], however we collect datasets with a relatively small number of scenes and since scene splits would leave us with insufficient data for testing and image based splits would bias testing accuracy, we only use our data during evaluation. Our contribution of these small-scale data collections is to demonstrate that 1) current datasets do not appear to capture important effects and 2) collecting large scale reshading and repainting dataset that can be used during training is a compelling future direction that would improve the quality of intrinsic image decomposition.

**Augmenting an Existing Relighting Dataset:** Any scene or object can be captured under varied lighting, as in [149]. To evaluate reshading we need to make a dataset of images that meet a convention of “reasonableness”, but the images in [149] do not meet this standard. First, the light source (and the person holding it) are often visible in the individual photographs. This means that the content of the scene between individual photos has changed. Second, the light source only illuminates a portion of the scene leaving much of the image dark, an effect which is not present in most “reasonable” images. Fortunately, there is an easy solution to these problems. For a fixed scene, any non-negative linear combination of images is also a real image. We can therefore create a dataset by taking a weighted combination of single illuminant images. To remove the effect of the light source being visible in the photos, we use the mask that identifies where the light source and person holding it are in the image so we can “erase” the effect of seeing the illuminant in any individual image. Let $N$ be the number of single light source images. For our dataset we create 10 images per scene, where each image is an average of $\sqrt{N}$ randomly selected images. We find that this setting created images that were different and met our standard for “reasonableness”.

To evaluate repainting, we have to augment the constructed relighting dataset. In particular, we assume that the shading is colorless, and apply a transformation to the image that won’t change the shading. That is, for an image $I = SR$, we describe a modification function $M(\cdot)$ that commutes
over multiplication and acts as identity on shading images

\[ M(I) = M(S)M(R) \]
\[ = SM(R). \]  

(8.7)

An example transformation that has this effect is to rearrange the red, green, and blue channels of the image. A more generic version of the transformation is a 1x1 convolution with a few constraints. Let \( M \) be a 3x3 matrix representing the 1x1 convolution and \( p \) be a 3x1 vector representing the \( r, g, b \) channels. For any colorless pixel (i.e. \( r = g = b \)), the resulting pixel should also be colorless, that is, \( M1 = 1 \). To avoid arbitrary scaling, the intensity of the resulting image and the intensity of the input image should not change, that is \( 1^T M = 1^T \). Finally, the resulting image should lie in the [0,1] gammut. The constraints are easy to apply, we have described a 3x3 matrix with 6 linear constraints, meaning that the matrix has 4 parameters. Constraining the gammut is much harder, and in practice, we do it by proposing a transformation and then checking if the transformation is valid, rather than constructing and minimizing the constrained optimization problem in the input image.

Images from our dataset and the raw images are shown in figure 8.1.

![Sample images from our reshading dataset (bottom 2 rows) and example single light image (top row). While the single light photographs have strong lighting changes, they also include effects that we do not want in a reshading dataset. All of the images have portions that are unlit, a person holding a light is visible in two images and lens flair is present in another.](image)

**Color Swatch Dataset:** For our color swatch dataset, we collected a set of images for 3 scenes.
We inserted 10 known color swatches (11 including no-swatch), and took photographs under 3 different lightings. This gave us a dataset of 99 images. For a specific scene we used all possible unique combinations. This gave us 55 color combinations and 3 lighting combinations for 165 total lighting and color pairs per scene and 495 total comparisons. Images from our dataset are shown in figure 8.2.

![Sample images from our color swatch dataset. The same color swatch is shown under a different lighting (column 1,2) and then the same lightings are shown with a different color swatch (column 3,4).](image)

**Color Gradient Dataset:** One downside of the color swatch dataset is that the color swatches were close to equiluminant. This means that a simple decomposition will work well on that dataset. In an attempt to identify that effect we created a dataset of color gradients by selecting 5 swatches with nearly the same chroma but different luminance, for example five swatches of black, forest green, green, mint green and white would be such a gradient. We insert the gradient from bright to dark, taking photos under 4 lighting conditions. We then insert the gradient from dark to bright retaking photos in the same 4 lighting conditions. For this dataset, we pair up the same color gradients for color-pairs, and have 6 possible lighting combinations across 7 different color gradients in 4 scenes. This gives us 168 total comparisons. Images from the color gradient dataset are shown in figure 8.3

![Images from the color gradient dataset](image)
Figure 8.3: Sample images from our color gradient dataset. The same gradient is shown under a different lighting (column 1,2) and then the same lightings are shown with the flipped gradient (column 3,4).

8.5 USING INTRINSIC IMAGES IN THE WILD

8.5.1 Current Practice

Current practice is to assume that human annotations are a direct representation of the physical reflectance. A formalization of the WHDR as a hinge-loss is first described by [144] and hinge-loss formulations of IIW are used in [145, 146] to get state of the art WHDR scores by training on the IIW data. In addition to IIW data, [146] incorporates a rendered reflectance and shading dataset based on [150] and the smooth shading annotations from [142]. While it makes sense to use a combination of data, when we use multiple sources of data we have to consider whether those sources of data agree on a solution. It seems natural to assume that humans and physics would agree, however human perception is not veridical.
8.4: IIW perception pairs are often labeled in ways that are inconsistent with physical intensity. The point annotated with a double circle was identified by human annotators as being brighter than the point in the single circle. 8.4a: Annotations are sensitive to grouping effects. Annotations across wall boundaries have lower annotator agreement than other annotations in the dataset, and there is a slight preference to label “foreground” objects as brighter than “background” objects even if they have the same physical reflectance. Low annotator agreement on equivalence relationships across object boundaries means that methods will not be punished for misrepresenting global equivalence (e.g. all walls in a room being white). 8.4b: Human annotations are sensitive to color. When the image is grayscale (center) the points appear to be the same intensity. When the saturation is increased (right) it is clear why certain objects were labeled as brighter. Since WHDR is computed as a ratio of reflectance intensity, to get these pairs correct, a method must get the physical reflectance wrong. Figure must be viewed in color.

8.5.2 Human Lightness Perception

In order to understand what human perception labels tells us about reflectance, we have to consider how human perception works. While humans are very good at discounting illumination when reasoning about scenes, there is significant evidence that human perception of lightness is not veridical. Simultaneous contrast effects change human perception of virtually inserted swatches on real images as shown in [151]. Complex grouping effects based on scene level information inform human lightness perception (eg. Koffka rings), anchoring suggests that perception of the relative brightness of two patches can be changed by adjusting the crop, and the scaling effect suggests that large dark areas will be perceived as brighter than equally dark smaller patches [152]. Other confounds are explored in [153] which describes when human lightness perception diverges from the Retinex theory. For a recent attempt to model the human perception of a grayscale image see [154]. In IIW we see a persistent grouping effect where annotator agreement for points across wall boundaries have lower annotator agreement than pairs on the same wall surface (figure 8.4a).

Human judgment of lightness is also affected by color. The Helmholtz-Kohlrausch effect states
that red and blue appear brighter than equiluminant yellow and green. This effect is documented in the Munsel Color system [155] and verified for standard color swatches [156] and natural images [157]. We see this effect in IIW annotations shown in figure 8.4b where patches that would be labeled as equivalent in grayscale are labeled as different when seen in color.

8.5.3 Human Perception Model

While previous work on IIW focused on how to train a model to minimize WHDR, we focus on how to design a model that can handle the non-veridical nature of human perception labels with physical reflectance. Rather than computing the IIW WHDR by ratio of reflectance predictions, we use a classifier that takes the predicted reflectance image as input. This allows our method to handle situations where human judgments disagree with the physical reflectance ratios without forcing the physical reflectance to change.

We wish to incorporate knowledge about human perception of lightness into our model. We know that color and context around a point matters. We therefore build our human perception model on color reflectance rather than grayscale values and instead of using a single pixel value, we use a multiscale pyramid centered on the point of interest. This feature construction is fully backpropable and allows our human perception model to take into account color and context when making a decision about the relationship between two points.

**Implementation:** We adopt CGIntrinsics [146] for our physical reflectance and shading prediction. The human perception model takes in two feature vectors centered at the two points of interest. These features are first transformed using a shared (siamese) MLP. Then the two transformed features are combined using a bilinear layer to get a single feature. The output from the bilinear layer is fed to an MLP that outputs a 2-dimensional binary classifier. The first dimension of the output determines whether the points are equivalent, and the second determines which of the two points is brighter (conditioned on inequivalence). We found this construction to be better than a 3-way classifier because it more accurately represents the relationships between the classes and prevents a dataset bias towards equivalence from dominating the directionality prediction.

To train the 2-d binary classifier, we treat the equivalence labeled points differently from the greater-than/less-than labeled points. Let $R_i$ be the multiscale feature construction for the $i$th pixel of reflectance image $R$, $H_E(R_i, R_j)$ be the equivalence human perception binary classifier and $H_G(R_i, R_j)$ be the greater-than/less-than binary classifier as described (shared network, bilinear, MLP), and let $r_{ij}$ be the ground truth ordinal relationship from IIW (0 equal, -1 if $i$ is darker, 1 if $j$ is darker), and $w_{ij}$ be the confidence provided by IIW. Our loss, which replaces the ordinal reflectance loss (equation 7 in [146]) is the combination of two binary cross entropy losses.
\[ \mathcal{L}_{ord}(R) = \sum_{i,j} e_{i,j}(R), \text{ where} \]

\[
e_{i,j}(R) = w_{i,j}([r_{i,j} = 0] \log(\text{HE}(R_i, R_j)) + [r_{i,j} \neq 0] \log(1 - \text{HE}(R_i, R_j))+ [r_{i,j} = 1] \log(\text{HG}(R_i, R_j)) + [r_{i,j} = -1] \log(1 - \text{HG}(R_i, R_j))). \quad (8.8)
\]

8.6 EXPERIMENTS

8.6.1 Intrinsic Images in the Wild

While prior models outperform our model on IIW WHDR (table 8.2), our method performs as well as a CGIntrinsic model trained on the same data. We report both the WHDR at \( \delta = .1 \) as well as the WHDR at the best delta. For our method, this construction is not meaningful since we parameterize the human perception prediction as a classification. We therefore cannot adjust the WHDR \( \delta \) so we only report our model as WHDR at best \( \delta \). This choice of reporting allows for fair comparison of our model which is tuned for IIW against models which are not tuned on IIW (see section 8.5.2 and section 8.3). Figures 8.17, 8.18, 8.19 show qualitative decompositions of IIW data.

<table>
<thead>
<tr>
<th>Model</th>
<th>WHDR (( \delta = .1 ))</th>
<th>WHDR (best ( \delta ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color Retinex [2]</td>
<td>27.4%</td>
<td>19.1%</td>
</tr>
<tr>
<td>SIRFS [9]</td>
<td>29.9%</td>
<td>26.9%</td>
</tr>
<tr>
<td>Image as Reflectance</td>
<td>51.6%</td>
<td>21.0%</td>
</tr>
<tr>
<td>Image as Shading</td>
<td>36.6%</td>
<td>36.2%</td>
</tr>
<tr>
<td>CGIntrinsics (pretrained) [146]</td>
<td>16.8%</td>
<td>16.8%</td>
</tr>
<tr>
<td>CGIntrinsics +human perception</td>
<td>-</td>
<td>16.9% ± 0.5</td>
</tr>
<tr>
<td>CGIntrinsics Color</td>
<td>-</td>
<td>18.0% ± 0.6</td>
</tr>
<tr>
<td>CGIntrinsics +human perception</td>
<td>-</td>
<td>16.7% ± 0.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16.6% ± 0.18</td>
</tr>
</tbody>
</table>

Table 8.2: WHDR by method. We report two WHDRs for pre-existing methods. WHDR at \( \delta = .1 \), which is typically taken from the literature (exception SIRFS) and WHDR at best \( \delta \) which we recompute for the preexisting models. This allows to fairly compare to our human perception models which predict the human perception labels as a discrimination task, and therefore is at least as powerful as selecting \( \delta \). While the pretrained CGIntrinsics model performs better than our model, it is trained on the SAW dataset as well as the CGI and IIW data. When compared to the CGIntrinsics model trained on the same data, our model achieves slightly better performance when a color reflectance is considered.
8.6.2 Reshading and Repainting

Optimization-based decomposition methods, especially retinex, appear to be far better than deep learning based methods as can be seen in table 8.3 and table 8.4 (“Original” column). However, this appears to be due to a difference in how these two different models treat images. Regression models, such as deep learning models, are trained to reproduce the input image (in an L2 sense), but at test time shading and reflectance are predicted independently. This independence can be a positive, for example, independence lets a model ignore pieces of the image that don’t fit it’s representation of reflectance or shading. This choice can also be a positive if the evaluation procedure is sparse. However, for other evaluation criteria, including our reshading and repainting loss, not conditioning the reflectance on the shading hurts performance. As can be seen in figure 8.5 the reshading error of direct regression models is predicted by the reconstruction error.

This suggests that it is worth investigating **consistent** variants of the direct regression methods that we believe might improve performance. Let the original regression model produces $R(I), S(I)$ as reflectance and shading for an image $I$ independently. The shading scaled version of the model reports $R(I), \alpha S(I)$ with $\alpha = \arg\min_{\alpha} \| I - R(I)\alpha S(I) \|$. The shading residual method reports $R(I), I/R(I)$ as reflectance and shading. The reflectance residual method reports $I/S(I), S(I)$ as reflectance and shading. Each of these methods reduces the image reconstruction error by adjusting the interpretation of the resulting output. Remarkably, these consistent versions – which involve no retraining of the method, just a reinterpretation of the method outputs – produce significant improvements in reshading error (table 8.3 and table 8.4), and yield improvements over Retinex. Note that these improvements are not be predicted by WHDR, and in particular the scaled shading and shading residual methods do not adjust the WHDR because they do not change the reflectance. Qualitative results can be found in figures 8.8 8.9 8.10 8.14 8.15 8.16 8.11 8.12 8.13.

8.6.3 Combining Reshading and WHDR

Since WHDR and reshading performance for current models are not correlated, we need to consider how to use reshading evaluation alongside WHDR. Rather than relying on individual dataset evaluation, we propose adopting Pareto efficiency for evaluating models. A Pareto optimal model is a model which is not categorically dominated by other models. All Pareto optimal models are said to be on the Pareto frontier. Models on the Pareto frontier offer the most efficient tradeoff between the two evaluations, and lets us consider models which perform well with respect to both evaluation criteria rather than considering models that strictly push performance on a single evaluation. We show the results of plotting IIW WHDR vs Reshading in figures 8.6 and 8.7. Pareto optimality offers a simple way to combine performance on multiple datasets and can be adopted
### Table 8.3: Reshading/Repainting results for the [149] augmented dataset using an L2 Image error.

<table>
<thead>
<tr>
<th>Model</th>
<th>Original</th>
<th>Scale Sh</th>
<th>Residual Sh</th>
<th>Residual Refl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color Retinex [2]</td>
<td>0.079</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIW [4]</td>
<td>0.188</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIRFS [9]</td>
<td>0.175</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image as Reflectance</td>
<td>0.09</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image as Shading</td>
<td>0.016</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGIntrinsics [146]</td>
<td>0.142</td>
<td>0.077</td>
<td>0.052</td>
<td>0.068</td>
</tr>
<tr>
<td>CGIntrinsics</td>
<td>0.165 ± 0.039</td>
<td>0.114 ± 0.026</td>
<td>0.048 ± 0.002</td>
<td>0.095 ± 0.025</td>
</tr>
<tr>
<td>+Human Perception</td>
<td>0.159 ± 0.012</td>
<td>0.116 ± 0.009</td>
<td>0.064 ± 0.003</td>
<td>0.105 ± 0.009</td>
</tr>
<tr>
<td>CGIntrinsics Color</td>
<td>0.147 ± 0.011</td>
<td>0.103 ± 0.009</td>
<td>0.054 ± 0.001</td>
<td>0.087 ± 0.009</td>
</tr>
<tr>
<td>+Human Perception</td>
<td>0.155 ± 0.012</td>
<td>0.111 ± 0.011</td>
<td>0.05 ± 0.003</td>
<td>0.093 ± 0.009</td>
</tr>
</tbody>
</table>

### Table 8.3: Reshading/Repainting results for the Color Swatch Dataset using an L2 image error.

<table>
<thead>
<tr>
<th>Model</th>
<th>Original</th>
<th>Scale Sh</th>
<th>Residual Sh</th>
<th>Residual Refl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color Retinex [2]</td>
<td>0.111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIW [4]</td>
<td>0.333</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIRFS [9]</td>
<td>0.204</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image as Reflectance</td>
<td>0.195</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image as Shading</td>
<td>0.051</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGIntrinsics [146]</td>
<td>0.228</td>
<td>0.115</td>
<td>0.102</td>
<td>0.099</td>
</tr>
<tr>
<td>CGIntrinsics</td>
<td>0.263 ± 0.044</td>
<td>0.135 ± 0.011</td>
<td>0.1 ± 0.007</td>
<td>0.127 ± 0.02</td>
</tr>
<tr>
<td>+Human Perception</td>
<td>0.257 ± 0.037</td>
<td>0.16 ± 0.033</td>
<td>0.148 ± 0.027</td>
<td>0.147 ± 0.039</td>
</tr>
<tr>
<td>CGIntrinsics Color</td>
<td>0.25 ± 0.061</td>
<td>0.136 ± 0.028</td>
<td>0.102 ± 0.006</td>
<td>0.136 ± 0.056</td>
</tr>
<tr>
<td>+Human Perception</td>
<td>0.259 ± 0.041</td>
<td>0.14 ± 0.026</td>
<td>0.099 ± 0.008</td>
<td>0.144 ± 0.048</td>
</tr>
</tbody>
</table>

### Table 8.3: Reshading/Repainting results for the Color Gradient Dataset using an L2 image error.

<table>
<thead>
<tr>
<th>Model</th>
<th>Original</th>
<th>Scale Sh</th>
<th>Residual Sh</th>
<th>Residual Refl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color Retinex [2]</td>
<td>0.068</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIW [4]</td>
<td>0.174</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIRFS [9]</td>
<td>0.134</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image as Reflectance</td>
<td>0.113</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image as Shading</td>
<td>0.054</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGIntrinsics [146]</td>
<td>0.218</td>
<td>0.089</td>
<td>0.062</td>
<td>0.059</td>
</tr>
<tr>
<td>CGIntrinsics</td>
<td>0.26 ± 0.042</td>
<td>0.104 ± 0.014</td>
<td>0.069 ± 0.007</td>
<td>0.072 ± 0.005</td>
</tr>
<tr>
<td>+Human Perception</td>
<td>1.236 ± 0.04</td>
<td>0.111 ± 0.016</td>
<td>0.09 ± 0.011</td>
<td>0.083 ± 0.012</td>
</tr>
<tr>
<td>CGIntrinsics Color</td>
<td>0.224 ± 0.012</td>
<td>0.103 ± 0.014</td>
<td>0.074 ± 0.003</td>
<td>0.073 ± 0.01</td>
</tr>
<tr>
<td>+Human Perception</td>
<td>0.235 ± 0.02</td>
<td>0.106 ± 0.011</td>
<td>0.067 ± 0.005</td>
<td>0.077 ± 0.013</td>
</tr>
</tbody>
</table>

Table 8.3: We report multiple reshading/repainting errors for the deep learning regression models. “Original” uses the prediction from the neural networks directly, while scale shading (scale sh), residual shading (residual sh), and residual reflectance (residual refl) perform post processing on the prediction to decrease the image reconstruction error. Amazingly, the neural network models (including ours) have significantly better reshading/repainting performance when we ignore either the shading prediction or the reflectance prediction (Residual shading and residual reflectance respectively).
(a) Reshading/Repainting results for the [149] augmented dataset using L2 loss on a neural embedding.

<table>
<thead>
<tr>
<th>Model</th>
<th>Original</th>
<th>Scale Sh</th>
<th>Residual Sh</th>
<th>Residual Refl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color Retinex [2]</td>
<td>0.318</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIW [4]</td>
<td>0.489</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIRFS [9]</td>
<td>0.499</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image as Reflectance</td>
<td>0.389</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image as Shading</td>
<td>0.108</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGIntrinsics [146]</td>
<td>0.535</td>
<td>0.469</td>
<td>0.232</td>
<td>0.35</td>
</tr>
<tr>
<td>CGIntrinsics</td>
<td>0.583 ± 0.031</td>
<td>0.541 ± 0.024</td>
<td>0.213 ± 0.011</td>
<td>0.405 ± 0.031</td>
</tr>
<tr>
<td>+Human Perception</td>
<td>0.571 ± 0.007</td>
<td>0.545 ± 0.009</td>
<td>0.28 ± 0.005</td>
<td>0.425 ± 0.018</td>
</tr>
<tr>
<td>CGIntrinsics Color</td>
<td>0.558 ± 0.013</td>
<td>0.528 ± 0.017</td>
<td>0.266 ± 0.006</td>
<td>0.397 ± 0.013</td>
</tr>
<tr>
<td>+Human Perception</td>
<td>0.57 ± 0.011</td>
<td>0.537 ± 0.012</td>
<td>0.235 ± 0.009</td>
<td>0.405 ± 0.012</td>
</tr>
</tbody>
</table>

(b) Reshading/Repainting results the Color Swatch Dataset using L2 loss on a neural embedding.

<table>
<thead>
<tr>
<th>Model</th>
<th>Original</th>
<th>Scale Sh</th>
<th>Residual Sh</th>
<th>Residual Refl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color Retinex [2]</td>
<td>0.28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIW [4]</td>
<td>0.452</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIRFS [9]</td>
<td>0.386</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image as Reflectance</td>
<td>0.351</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image as Shading</td>
<td>0.184</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGIntrinsics [146]</td>
<td>0.508</td>
<td>0.473</td>
<td>0.28</td>
<td>0.333</td>
</tr>
<tr>
<td>CGIntrinsics</td>
<td>0.582 ± 0.041</td>
<td>0.533 ± 0.045</td>
<td>0.258 ± 0.011</td>
<td>0.436 ± 0.067</td>
</tr>
<tr>
<td>+Human Perception</td>
<td>0.614 ± 0.087</td>
<td>0.583 ± 0.083</td>
<td>0.384 ± 0.025</td>
<td>0.47 ± 0.114</td>
</tr>
<tr>
<td>CGIntrinsics Color</td>
<td>0.58 ± 0.056</td>
<td>0.533 ± 0.044</td>
<td>0.316 ± 0.015</td>
<td>0.429 ± 0.073</td>
</tr>
<tr>
<td>+Human Perception</td>
<td>0.594 ± 0.062</td>
<td>0.555 ± 0.045</td>
<td>0.306 ± 0.014</td>
<td>0.448 ± 0.073</td>
</tr>
</tbody>
</table>

(c) Reshading/Repainting results for the Color Gradient Dataset using L2 loss on a neural embedding.

<table>
<thead>
<tr>
<th>Model</th>
<th>Original</th>
<th>Scale Sh</th>
<th>Residual Sh</th>
<th>Residual Refl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color Retinex [2]</td>
<td>0.205</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIW [4]</td>
<td>0.317</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIRFS [9]</td>
<td>0.29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image as Reflectance</td>
<td>0.232</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image as Shading</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGIntrinsics</td>
<td>0.48</td>
<td>0.44</td>
<td>0.19</td>
<td>0.211</td>
</tr>
<tr>
<td>CGIntrinsics</td>
<td>0.532 ± 0.028</td>
<td>0.471 ± 0.035</td>
<td>0.18 ± 0.005</td>
<td>0.258 ± 0.026</td>
</tr>
<tr>
<td>+Human Perception</td>
<td>0.536 ± 0.029</td>
<td>0.486 ± 0.039</td>
<td>0.255 ± 0.012</td>
<td>0.29 ± 0.064</td>
</tr>
<tr>
<td>CGIntrinsics Color</td>
<td>0.514 ± 0.02</td>
<td>0.464 ± 0.028</td>
<td>0.226 ± 0.006</td>
<td>0.261 ± 0.032</td>
</tr>
<tr>
<td>+Human Perception</td>
<td>0.526 ± 0.018</td>
<td>0.483 ± 0.026</td>
<td>0.201 ± 0.007</td>
<td>0.277 ± 0.028</td>
</tr>
</tbody>
</table>

Table 8.4: We report multiple reshading/repainting errors for the deep learning regression models. “Original” uses the prediction from the neural networks directly, while scale shading (scale sh), residual shading (residual sh), and residual reflectance (residual refl) perform post processing on the prediction to decrease the image reconstruction error. Amazingly, the neural network models (including ours) have significantly better reshading/repainting performance when we ignore either the shading prediction or the reflectance prediction (Residual shading and residual reflectance respectively).
Figure 8.5: For the direct intrinsic methods we saw poor reshading results. However, the poor performance is because the input image isn’t reproduced by the predicted shading and reflectance. For direct intrinsic methods, reflectance and shading are predicted independently at test time so there might be a significant amount of the image that is not captured in either the reflectance or shading. Simple adjustments to adjust the shading or reflectance to fully represent the input image to produce “consistent” decomposition improves reshading results as shown in table 8.3 and table 8.4.

widely for tasks when evaluations appear to be contradictory.
Figure 8.6: We plot three graphs for our three different image reshading/repainting datasets for an L2 Image Error vs WHDR. We plot multiple versions of the CGIntrinsics model and our human perception model, trained at different initializations with the same hyperparameters. In 8.6a, notice that Retinex is on the Pareto frontier. It’s not until we consider post-processing (8.6b) that the CGIntrinsics model results push Retinex off the Pareto frontier. The biggest factor in performance for the deep learned models is which post-processing technique is used. Adjustments that change the shading do not impact the WHDR score, while residual reflectance processing decreases WHDR performance but increases reshading/repainting performance. The pretrained CGIntrinsics model appears on the Pareto frontier in all three datasets. While the simple baseline of reporting the image as the shading appears on the Pareto frontier due to its reshading/repainting performance, it’s low performance on WHDR makes it clear that it shouldn’t be considered a good model. There is strong agreement between the rankings of models across the datasets a- well as image reshading/repainting error (figure 8.7).
Figure 8.7: Similar to figure 8.6 we plot three graphs for our three different image reshading/repainting datasets, this time for L2 Neural Error vs WHDR. We plot multiple versions of the CGIntrinsics model and our human perception model, trained at different initializations with the same hyperparameters. In 8.7a Retinex is on the Pareto frontier. It’s not until we consider post-processing (8.7b) that the CGIntrinsics model results push Retinex off the Pareto frontier. As before, the biggest factor in performance for the deep learned models is which post-processing technique is used. The pretrained CGIntrinsics model appears on the pareto frontier in all three datasets. The simple baseline of reporting the image as the shading appears on the Pareto frontier due to its reshading/repainting performance, but it’s low performance on WHDR makes it clear that it shouldn’t be considered a “reasonable” model. There is strong agreement between the rankings of models across the datasets.
Figure 8.8: Relighting/Repainting qualitative results for one of the Boyadziev images.
Figure 8.9: Relighting/Repainting qualitative results for one of the Boyadziev images.
Figure 8.10: Relighting/Repainting qualitative results for one of the Boyadziev images.
<table>
<thead>
<tr>
<th>Image</th>
<th>Reflectance</th>
<th>Shading</th>
<th>Reflectance</th>
<th>Reshading</th>
<th>Image</th>
<th>Reflectance</th>
<th>Shading</th>
<th>Reflectance</th>
<th>Reshading</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color Retinex [2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIW [4]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIRFS [9]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGIntrinsics Pretrained</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGIntrinsics Pretrained (Residual Shading)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours (Residual Shading)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.11: Relighting/Repainting qualitative results for one of the Boyadziev images.
<table>
<thead>
<tr>
<th></th>
<th>Reflectance Shading</th>
<th>Reshading</th>
<th></th>
<th>Reflectance Shading</th>
<th>Reshading</th>
<th>Reflectance Shading</th>
<th>Reshading</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.121</td>
<td></td>
<td></td>
<td>0.049</td>
<td></td>
<td>0.084</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.192</td>
<td></td>
<td></td>
<td>0.356</td>
<td></td>
<td>0.167</td>
</tr>
<tr>
<td></td>
<td>CGIntrinsics Pretrained</td>
<td></td>
<td></td>
<td></td>
<td>0.153</td>
<td></td>
<td>0.084</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.221</td>
<td></td>
<td></td>
<td>0.231</td>
<td></td>
<td>0.167</td>
</tr>
<tr>
<td></td>
<td>CGIntrinsics Pretrained (Residual Shading)</td>
<td></td>
<td></td>
<td></td>
<td>0.071</td>
<td></td>
<td>0.092</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.088</td>
<td></td>
<td></td>
<td>0.243</td>
<td></td>
<td>0.261</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>0.219</td>
<td></td>
<td></td>
<td>0.215</td>
<td></td>
<td>0.071</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.243</td>
<td></td>
<td></td>
<td>0.068</td>
<td></td>
<td>0.067</td>
</tr>
</tbody>
</table>

Figure 8.12: Relighting/Repainting qualitative results for one of the Boyadziev images.
Figure 8.13: Relighting/Repainting qualitative results for one of the Boyadziev images.

<table>
<thead>
<tr>
<th>Method</th>
<th>Reflectance Error</th>
<th>Shading Error</th>
<th>Reshading Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color Retinex [2]</td>
<td>0.069</td>
<td>0.095</td>
<td>0.063</td>
</tr>
<tr>
<td>IIW [4]</td>
<td>0.151</td>
<td>0.264</td>
<td>0.159</td>
</tr>
<tr>
<td>SIRFS [9]</td>
<td>0.159</td>
<td>0.139</td>
<td>0.125</td>
</tr>
<tr>
<td>CGIntrinsics Pretrained</td>
<td>0.150</td>
<td>0.312</td>
<td>0.297</td>
</tr>
<tr>
<td>CGIntrinsics Pretrained (Residual Shading)</td>
<td>0.119</td>
<td>0.141</td>
<td>0.108</td>
</tr>
<tr>
<td>Ours</td>
<td>0.216</td>
<td>0.319</td>
<td>0.235</td>
</tr>
<tr>
<td>Ours (Residual Shading)</td>
<td>0.096</td>
<td>0.119</td>
<td>0.086</td>
</tr>
</tbody>
</table>
Figure 8.14: Relighting/Repainting qualitative results for one of the Boyadziev images.
<table>
<thead>
<tr>
<th>Image</th>
<th>Reflectance</th>
<th>Shading</th>
<th>Reshading</th>
<th>Image</th>
<th>Reflectance</th>
<th>Shading</th>
<th>Reshading</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIRFS [9]</td>
<td></td>
<td></td>
<td></td>
<td>CGIntrinsics Pretrained</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGIntrinsics Pretrained (Residual Shading)</td>
<td></td>
<td></td>
<td></td>
<td>Ours</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours (Residual Shading)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.15: Relighting/Repainting qualitative results for one of the Boyadziev images.
Figure 8.16: Relighting/Repainting qualitative results for one of the Boyadziev images.
Figure 8.17: Decompositions for IIW data. Individual image IIW WHDR for the method is shown in white text in the upper lefthand corner of the reflectance image.
Figure 8.18: Decompositions for IIW data. Individual image IIW WHDR for the method is shown in white text in the upper lefthand corner of the reflectance image.
<table>
<thead>
<tr>
<th>Image</th>
<th>Reflectance</th>
<th>Shading</th>
<th>Reflectance</th>
<th>Shading</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIRFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGIntrinsics</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.19: Decompositions for IIW data. Individual image IIW WHDR for the method is shown in white text in the upper lefthand corner of the reflectance image.
CHAPTER 9: CONCLUSIONS

This thesis presents a view of image representation prior to and during the early years of deep learning. As a result this thesis presents what I believe to be good ideas, with implementations that even as of writing seem quite dated. This thesis also demonstrates the improvement for image representation during this work. The attempt in chapter 1 to construct a generic image regression framework to learn image-to-image tasks was limited by the LEARCH framework, was difficult and slow to train, and the resulting outputs were fairly low quality. In comparison, image prediction using deep networks is quick to evaluate and much easier to train in part due to the removal of a hand selected representation. Authoring representations that can be used on real data (chapters 2, 3, 4, 5, 7) remains an interesting but frustrating endeavor. The promise of being able to use a model learned on fake or generated data on real data would drastically reduce the largest expense in creating high quality models, namely collecting high quality data. Finally, evaluating tasks fairly remains elusive (chapter 8). It is not sufficient to collect data, train a neural network model directly on the data, and declare success over existing methods. Deep learning models are willing to find tricks that avoid solving the actual problem and will happily solve an easier problem if a particular dataset or evaluation procedure allows. As a result we need to be more willing to run more exhaustive evaluations which attempt to close the gap between the goal tasks and our evaluation. Unfortunately evaluating many goal tasks requires humans to make decisions about which image is best after a model has been trained. In the short term and medium term chapter 8 suggests that it is sufficient to treat multiple imperfect datasets as a multitask learning problem with the expectation that issues in any particular dataset offset each other.
REFERENCES


[155] T. M. Cleland, A practical description of the Munsell color system, with suggestions for its use., 1921.
