UC Berkeley
UC Berkeley Electronic Theses and Dissertations

**Title**
Image Synthesis for Self-Supervised Visual Representation Learning

**Permalink**
https://escholarship.org/uc/item/7wp6r5mx

**Author**
Zhang, Richard

**Publication Date**
2018

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

# Image Synthesis for Self-Supervised Visual Representation Learning

By

Richard Zhang

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering – Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Alexei A. Efros, Chair
Professor Trevor Darrell
Professor Michael DeWeese

Spring 2018

# Image Synthesis for Self-Supervised Visual Representation Learning

Copyright 2018
by
Richard Zhang

Abstract

# Image Synthesis for Self-Supervised Visual Representation Learning

by

Richard Zhang

Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Alexei A. Efros, Chair

Deep networks are extremely adept at mapping a noisy, high-dimensional signal to a clean, low-dimensional target output (e.g., image classification). By solving this heavy compression task, the network also learns about natural image priors. However, this process requires the curation of large, labeled datasets. Meanwhile, the world provides massive amounts of raw, unlabeled pixels for free. This thesis investigates learning representations of high-dimensional input signals by mapping them to *high-dimensional* output targets. While more difficult, it is not only possible to learn a strong feature representation, but also to synthesize realistic images.

Part I describes the use of deep networks for conditional image synthesis. The section begins by exploring the problem of image colorization, proposing both automatic and user-guided approaches. This section then proposes a system for general image-to-image translation problems, BicycleGAN, with the specific aim of capturing the multimodal nature of the output space.

Part II explores the visual representations learned within deep networks. Colorization, as well as cross-channel prediction in general, is a simple but powerful pretext task for self-supervised learning. The representations from cross-channel prediction networks transfer strongly to high-level semantic tasks, such as image classification, and to low-level human perceptual similarity judgments. For the latter, a large-scale dataset of human perceptual similarity judgments is collected. The proposed cross-channel network method outperforms traditional metrics such as PSNR and SSIM. In fact, many unsupervised and self-supervised methods transfer strongly, even comparably to fully-supervised methods.

To my parents for their love and support

# Contents

# List of Figures

# List of Tables

# Acknowledgments

I have benefited greatly from the extraordinary research environment at Berkeley and have many individuals to thank. I would first like to thank my wonderful advisor, Alexei (Alyosha) Efros, for his support, leadership, and positive spirit. I was very fortunate that Alyosha joined Berkeley (after I had already started), and even more fortunate that he agreed to serve as my advisor. Not only does Alyosha provide inspirational guidance, he is also uncommonly selfless towards his students. He spent countless hours with me patiently discussing ideas, reviewing results pixel-by-pixel, and refining presentations and papers to minute detail (this thesis is no exception). Most of all, Alyosha places his students' success and happiness as top priority, and his actions everyday speak to this.

I have had the special privilege of interacting with many Berkeley faculty members, including my thesis and qualifying committee members – Trevor Darrell, Michael DeWeese, and Jitendra Malik – as well as Sergey Levine, Pieter Abbeel, Kai Vetter, Anca Dragan, and Kurt Keutzer. I also thank Avideh Zakhor, who helped launch my research career as a member of her lab. I also thank the ECE faculty at Cornell University, especially my undergraduate and masters advisors – Alyosha (Al) Molnar and David Hysell – as well as Tsuhan Chen and Ehsan Afshari, for laying down my educational foundation and helping me get admitted to Berkeley. Many thanks to Al Molnar for making the time to introduce a young undergraduate student to research.

I especially thank Phillip Isola, who was effectively a second advisor to me during his time as a postdoctoral scholar. His mentorship has shaped how I approach problems and think about research. In particular, Phil is extremely thorough and places a heavy emphasis on reaching the most parsimonious solution. Phil also spent a great deal of effort helping me, even spending all-nighters running evaluations, arranging figures, and polishing text. We also had fun post-conference travels in Europe, Patagonia, and Kauai, narrowly escaping Brexiters, airline strikes, and drowning in the Pacific to make it back safely.

Thanks to all the members of the Efros group, including Jun-Yan Zhu, Deepak Pathak, Tinghui Zhou, Mathieu Aubry, Andrew Owens, Angjoo Kanazawa, David Fouhey, Taesung Park, Carl Doersch, and Shiry Ginosar, for being incredible labmates

and friends. Jun-Yan and I spent a very nice summer in Seattle and also wrote two papers together, both of which were mostly finished the last week and a half before the deadline. I would strongly discourage anyone from such a submission strategy in the future. I am grateful to Tinghui for helping me transition into the Efros group by inviting me to reading groups and group activities.

Thanks to my collaborators at Adobe Research, including Eli Shechtman and Oliver Wang. I spent a wonderful summer internship in Seattle and look forward to beginning my industrial research career at Adobe.

I thank Angie Abbatecola and Shirley Salanio for helping me navigate nontrivial Berkeley logistics. I thank many members, visitors, and friends of Berkeley AI Research (BAIR) for facilitating such a collaborative, open, and friendly environment. Thanks to Saurabh Gupta, Shubham Tulsiani, Abhishek Kar, Pulkit Agrawal, Christian Häne, Dinesh Jayaraman, Bharath Hariharan, Lisa Anne Hendricks, Jeff Donahue, Eric Tzeng, Evan Shelhamer, Parsa Mahmoudieh, Judy Hoffman, Philipp Krähenbühl, Yang Gao, Alex X. Lee, Chelsea Finn, Sandy Huang, Frederik Ebert, Hemang Jangle, Angela S. Lin, Xinyang Geng, Tianhe Yu, Stefan Candra, Xin Qin, Forrest Iandola, Allan Jabri, Shizhan Zhu, Kate Rakelly, Panna Felsen, Katerina Fragkiadaki, Weicheng Kuo, Ke Li, Jacob Huh, Mohammad Rastegari, Stella Yu, Fisher Yu, Amir Zamir, Nicholas Corso, Eric Lee Turner, Ricardo Garcia, John Kua, Bryan Russell, and many others.

Finally, many thanks to my parents and family for their continued support, encouragement, and unconditional love. I owe the wonderful opportunities and experiences in my life to their dedication and hard work.

# Chapter 1

# Introduction

Recent progress in computer vision has largely been driven by (i) large, labeled datasets and (ii) convolutional neural networks (CNNs), or deep learning. In particular, deep networks are proficient at the task of classification – mapping a high-dimensional input (e.g., images) to a low-dimensional output (e.g., labels), as shown in Figure 1.1(left). An important side benefit is that the emergent representation in intermediate network layers can be easily transferred to other tasks [1–3]. For example, a classification network [4] can be repurposed for predicting depth and normals [5], segmenting images [6], detecting objects [7], and even correlate extremely well with IT neural responses in the macaque visual cortex [8]. Internally, the network appears to learn a feature hierarchy, bridging low-level pixels to high-level semantics [9, 10]. However, there is a strong drawback to this paradigm – the necessity of large, labeled datasets. Dataset labeling is expensive, can lead to bias [11], and ultimately restricts the quantity of data that can be used.

The world is full of unlabeled, free data. Is it possible for a deep network to learn from this raw signal instead? Can a network map from a high-dimensional input to a *high-dimensional output*, as shown in Figure 1.1(right)? If so, the benefits are potentially two-fold. First, if successful, networks can potentially be used to solve graphics tasks. Secondly, using raw data as a supervisory signal, known as self-supervised training [12], unlocks a much larger scale of data to learn from, which can be used to learn a representation of the visual world. In this dissertation, we explore both of these aspects – graphics and representation learning.

**Part I: Image synthesis using deep networks** We first study the use of deep networks for image generation, or synthesis. Image generation is challenging for two fundamental reasons. First, because the output is structured, the answer can look "wrong" in many ways. However, there is no simple, closed-form function which

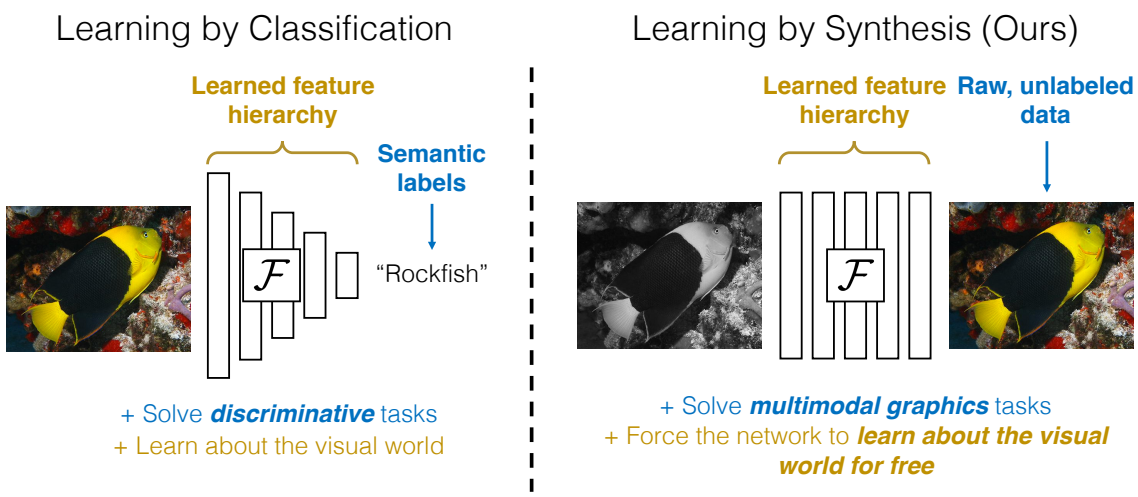



Figure 1.1: **Learning by classification vs learning by synthesis** Deep networks have been extremely effective at solving classification tasks (left). For example, a network can successfully associate an image to the label "rockfish". An emergent benefit is by learning to solve this task, the network learns a feature hierarchy which captures patterns in the visual world. In our work, we explore using deep networks for conditional image synthesis (right). The potential benefits are twofold – solving graphics tasks, which are typically multimodal in nature (Part I) and a free, emergent visual representation (Part II).

can describe the relative ordering of these bad answers. Thus, it is unclear what objective function should be used to optimize a deep network. Secondly, in many problems, there is often more than one "right" answer. Without accounting for the *multimodal* nature to the problem, a naive objective function will simply average between possible answers. For graphical results to look realistic and visually pleasing, these factors have to be accounted for. In Figure 1.2, we provide an overview of the image synthesis problems we tackle in this dissertation.

In Chapter 2, we first study the problem of automatic image colorization. To account for the multimodal nature of the problem, given a grayscale image, we train a network to predict a *distribution* of possible colors for each individual pixel, and then provide a single "best" guess. We test the resulting colorizations by running a Visual Turing test [13] – good colorizations should fool human judges into thinking that they are real.

The automatic colorization method was one of the first applications which produced images using deep networks. While the results were sometimes very good, the method could make mistakes. A more fundamental issue was the method could

only produce a single answer. Motivated by these problems, in Chapter 3, we propose a user-guided system for image colorization. The interface incorporates user inputs in the form of point-wise colors or global color histograms, and colorizes a grayscale image according to these constraints. In this user-guided paradigm, the multimodal nature of the colorization problem is effectively resolved by the user.

Having explored the problem of colorization in great detail, we move onto the more general problem of *image-to-image translation*. Many of these problems are by nature multimodal (for example, colorization). In Chapter 4, we propose the BicycleGAN system, which specifically aims to resolve the multimodality of the output space. We demonstrate its application it to a variety of problems such as generating images from edge maps, imagining a day image given a night webcam image, and synthesizing a satellite image from a Google map. We show that our general method produces results which are both realistic and diverse.

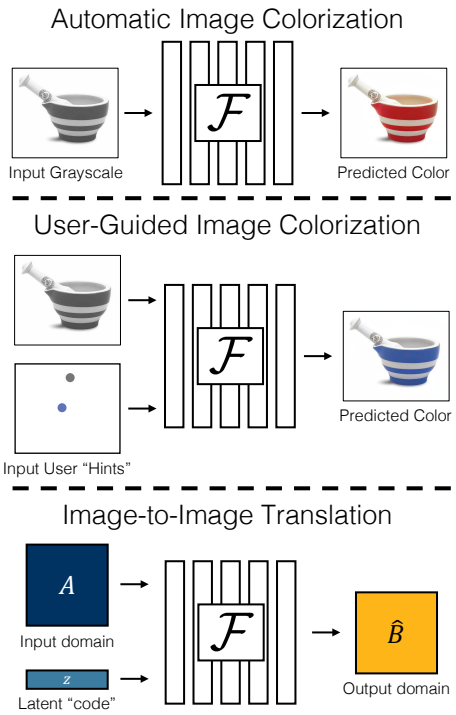



Figure 1.2: **Image synthesis overview**. In Part I, we explore the tasks of automatic image colorization (top), user-guided image colorization (middle), and the general problem of image-to-image translation (bottom). A primary challenge in image synthesis is the multimodal nature of the problem.

**Part II: Self-supervised visual representation learning** Next, we focus on studying the visual representations induced within networks. In Chapter 5, we find that the automatic colorization network proposed in Chapter 2 provides a representation which transfers surprisingly well to high-level semantic tasks, such as image classification or object detection. Intuitively, by being forced to associate different visual patterns to appropriate color distributions, the network is able to find higher-level abstractions. Motivated by these results, we propose Split-Brain Autoencoders, a generalization of colorization to cross-channel visual prediction. We note that self-supervised learning is not unlike classic unsupervised learning, where the goal is to learn patterns of the data without the use of labels. In fact, our system is similar to classic autoencoders [14], the de facto method for unsupervised learning with deep networks. We find that split-brain autoencoders learn strong features, comparable

and state-of-the-art relative to other previous and concurrent unsupervised and self-supervised methods. We test them by investigating how well they transfer to high-level tasks, such as object detection and segmentation [7].

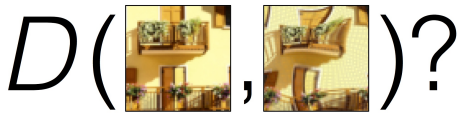



Figure 1.3: **Perceptual similarity**. In Part II, we explore how well the representation learned by deep networks transfer to high-level tasks, as well as low-level perceptual similarity. For example, deep networks can provide a "distance" between these two patches, as well as other pairs of patches, which have a consistent relative ordering to human judges.

Finding a function which can compare the "distance" between two images has been a long-standing, open problem in the graphics community [15–18]. In Chapter 6, we investigate how well visual representations transfer to low-level human perceptual similarity judgments. We collect a large-scale dataset of human perceptual similarity judgments by taking natural image patches and distorting them in a variety of ways. An example distorted image is shown in Figure 1.3. Distortions include hand-coded perturbations, such as blurring, warping, photometric changes, etc. as well as CNN-based corruptions. We then provide two random distorted images, and ask humans on Amazon Mechanical Turk (AMT) as to which distortion is less perceptually noticeable. We find that our method outperforms traditional metrics such as PSNR and SSIM [15]. We make the surprising discovery that networks across architectures and supervisory signals (supervised, self-supervised, and unsupervised!) transfer strongly. This indicates a common shared, emergent structure across networks.

Finally, we conclude in Chapter 7 and discuss possible directions for future work.

# Part I

# Image Synthesis using Deep Networks

# Chapter 2

# Automatic Image Colorization

Given a grayscale photograph as input, this chapter attacks the problem of hallucinating a *plausible* color version of the photograph. This is a specific instance of an image-to-image translation problem, and one which has some properties which make it easier than the general problem. Specifically, the structure of an image is mostly provided by the grayscale input, and humans are less sensitive to the color components of an image. However, there are significant challenges to this problem. First, the problem is clearly underconstrained, so previous approaches have either relied on significant user interaction or resulted in desaturated colorizations. We propose a fully automatic approach that produces vibrant and realistic colorizations. We embrace the underlying uncertainty, or *multimodality*, of the problem by posing it as a classification task and use class-rebalancing at training time to increase the diversity of colors in the result. The system is implemented as a feed-forward pass in a CNN at test time and is trained on over a million color images. We evaluate our algorithm using a "colorization Turing test," asking human participants to choose between a generated and ground truth color image. Our method successfully fools humans on 32% of the trials, significantly higher than previous methods. Moreover, we show that colorization can be a powerful pretext task for self-supervised feature learning, acting as a *cross-channel encoder*. At time of publication[1], this approach resulted in state-of-the-art performance on several feature learning benchmarks. We explore this idea further in Part II, Chapter 5.

[1]This work was published as *Colorful Image Colorization* in ECCV, 2016 [19].

## 2.1 Motivation and Background

Consider the grayscale photographs in Figure 2.1. At first glance, hallucinating their colors seems daunting, since so much of the information (two out of the three dimensions) has been lost. Looking more closely, however, one notices that in many cases, the semantics of the scene and its surface texture provide ample cues for many regions in each image: the grass is typically green, the sky is typically blue, and the ladybug is most definitely red. Of course, these kinds of semantic priors do not work for everything, e.g., the croquet balls on the grass might not, in reality, be red, yellow, and purple (though it's a pretty good guess). However, our goal is not necessarily to recover the actual ground truth color, but rather to produce a *plausible* colorization that could potentially fool a human observer. Therefore, our task becomes much more achievable: to model enough of the statistical dependencies between the semantics and the textures of grayscale images and their color versions in order to produce visually compelling results.

Given the lightness channel $L$, our system predicts the corresponding $a$ and $b$ color channels of the image in the CIE $Lab$ colorspace. To solve this problem, we leverage large-scale data. Predicting color has the nice property that training data is practically free: any color photo can be used as a training example, simply by taking the image's $L$ channel as input and its $ab$ channels as the supervisory signal. Others have noted the easy availability of training data, and previous works have trained convolutional neural networks (CNNs) to predict color on large datasets [20, 21]. However, the results from these previous attempts tend to look desaturated. One explanation is that [20, 21] use loss functions that encourage conservative predictions. These losses are inherited from standard regression problems, where the goal is to minimize Euclidean error between an estimate and the ground truth.

We instead utilize a loss tailored to the colorization problem. As pointed out by [22], color prediction is inherently multimodal – many objects can take on several plausible colorizations. For example, an apple is typically red, green, or yellow, but unlikely to be blue or orange. To appropriately model the multimodal nature of the problem, we predict a distribution of possible colors for each pixel. Furthermore, we re-weight the loss at training time to emphasize rare colors. This encourages our model to exploit the full diversity of the large-scale data on which it is trained. Lastly, we produce a final colorization by taking the *annealed-mean* of the distribution. The end result is colorizations that are more vibrant and perceptually realistic than those of previous approaches.

Evaluating synthesized images is notoriously difficult [23]. Since our ultimate goal is to make results that are compelling to a human observer, we introduce a novel way of evaluating colorization results, directly testing their perceptual realism. We set up

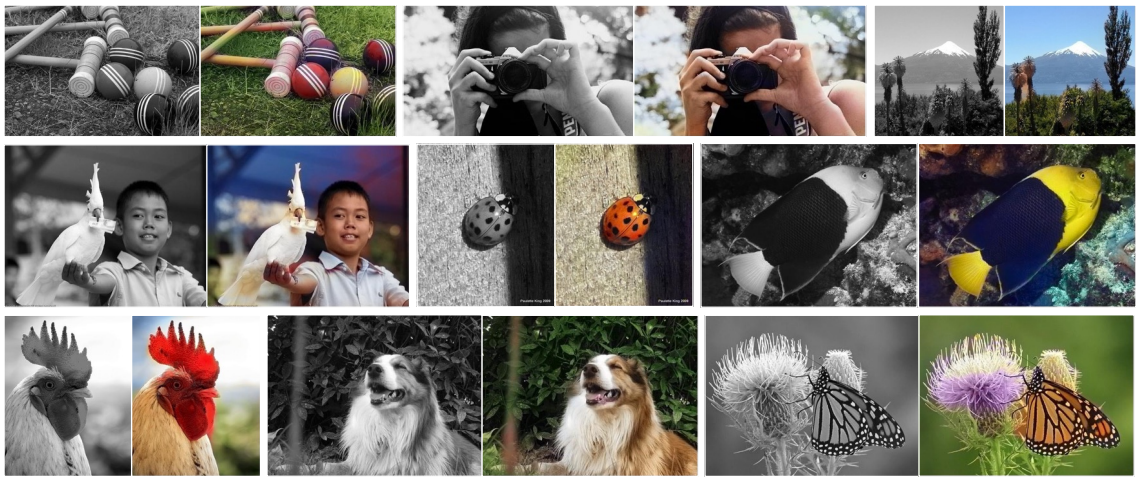


Figure 2.1: Example input grayscale photos and output colorizations from our algorithm. These examples are cases where our model works especially well. Please visit `http://richzhang.github.io/colorization/` to see the full range of results and to try our model and code. Best viewed in color (obviously).

a "colorization Turing test," in which we show participants real and synthesized colors for an image, and ask them to identify the fake. In this quite difficult paradigm, we are able to fool participants on 32% of the instances (ground truth colorizations would achieve 50% on this metric), significantly higher than prior work [21]. This test demonstrates that in many cases, our algorithm is producing nearly photorealistic results (see Figure 4.1 for selected successful examples from our algorithm). We also show that our system's colorizations are realistic enough to be useful for downstream tasks, in particular object classification, using an off-the-shelf VGG network [24].

We make progress on the graphics problem of automatic image colorization by (a) designing an appropriate objective function that handles the multimodal uncertainty of the colorization problem and captures a wide diversity of colors, (b) introducing a novel framework for testing colorization algorithms, potentially applicable to other image synthesis tasks, and (c) setting a new high-water mark on the task by training on a million color photos.

We also introduce the colorization task as a competitive and straightforward method for self-supervised representation learning, where raw data is used as its own source of supervision, and explore this further in Chapter 5. The idea of learning feature representations in this way goes back at least to autoencoders [25]. More recent works have explored feature learning via data imputation, where a held-out subset of the complete data is predicted (e.g., [26–32]). Our method follows in this

line, and can be termed a *cross-channel encoder*.

**Prior work on colorization** Colorization algorithms mostly differ in the ways they obtain and treat the data for modeling the correspondence between grayscale and color. Non-parametric methods, given an input grayscale image, first define one or more color reference images (provided by a user or retrieved automatically) to be used as source data. Then, following the Image Analogies framework [33], color is transferred onto the input image from analogous regions of the reference image(s) [34–37]. Parametric methods, on the other hand, learn prediction functions from large datasets of color images at training time, posing the problem as either regression onto continuous color space [20, 21, 38] or classification of quantized color values [22]. Our method also learns to classify colors, but does so with a larger model, trained on more data, and with several innovations in the loss function and mapping to a final continuous output.

**Concurrent work on colorization** Concurrently with our work, Larsson et al. [39] and Iizuka et al. [40] have developed similar systems, which leverage large-scale data and CNNs. The methods differ in their CNN architectures and loss functions. While we use a classification loss, with rebalanced rare classes, Larsson et al. use an un-rebalanced classification loss, and Iizuka et al. use a regression loss. In Section 5.5.1, we compare the effect of each of these types of loss function in conjunction with our architecture. The CNN architectures are also somewhat different: Larsson et al. use hypercolumns [41] on a VGG network [24], Iizuka et al. use a two-stream architecture in which they fuse global and local features, and we use a single-stream, VGG-styled network with added depth and dilated convolutions [42, 43]. In addition, while we and Larsson et al. train our models on ImageNet [4], Iizuka et al. train their model on Places [44]. In Section 2.3.1, we provide quantitative comparisons to Larsson et al.

## 2.2 Approach

We train a CNN to map from a grayscale input to a distribution over quantized color value outputs using the architecture shown in Figure 2.2. Architectural details are described in the supplementary materials on the project webpage[2], and the model is publicly available. In the following, we focus on the design of the objective function, and our technique for inferring point estimates of color from the predicted color distribution.

[2] `http://richzhang.github.io/colorization/`

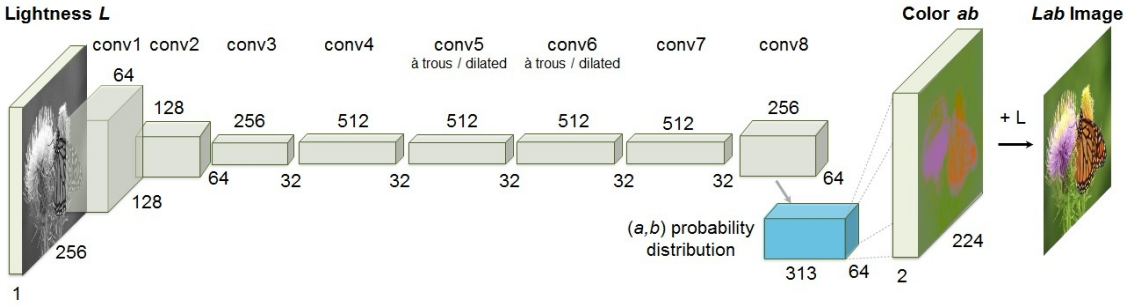



Figure 2.2: Our network architecture. Each `conv` layer refers to a block of 2 or 3 repeated `conv` and `ReLU` layers, followed by a `BatchNorm` [45] layer. The net has no `pool` layers. All changes in resolution are achieved through spatial downsampling or upsampling between `conv` blocks.

### 2.2.1 Objective Function

Given an input lightness channel $\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$, our objective is to learn a mapping $\widehat{\mathbf{Y}} = \mathcal{F}(\mathbf{X})$ to the two associated color channels $\mathbf{Y} \in \mathbb{R}^{H \times W \times 2}$, where $H, W$ are image dimensions. We denote predictions with a $\widehat{\cdot}$ symbol and ground truth without. We perform this task in CIE *Lab* color space. Because distances in this space model perceptual distance, a natural objective function, as used in [20, 21], is the Euclidean loss $\mathrm{L}_2(\cdot, \cdot)$ between predicted and ground truth colors:

$$\mathrm{L}_2(\widehat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{2} \sum_{h,w} \|\mathbf{Y}_{h,w} - \widehat{\mathbf{Y}}_{h,w}\|_2^2 \tag{2.1}$$

However, this loss is not robust to the inherent ambiguity and multimodal nature of the colorization problem. If an object can take on a set of distinct $ab$ values, the optimal solution to the Euclidean loss will be the mean of the set. In color prediction, this averaging effect favors grayish, desaturated results. Additionally, if the set of plausible colorizations is non-convex, the solution will in fact be out of the set, giving implausible results.

Instead, we treat the problem as multinomial classification. We quantize the $ab$ output space into bins with grid size 10 and keep the $Q = 313$ values which are in-gamut, as shown in Figure 2.3(a). For a given input $\mathbf{X}$, we learn a mapping $\widehat{\mathbf{Z}} = \mathcal{G}(\mathbf{X})$ to a probability distribution over possible colors $\widehat{\mathbf{Z}} \in [0, 1]^{H \times W \times Q}$, where $Q$ is the number of quantized $ab$ values.

To compare predicted $\widehat{\mathbf{Z}}$ against ground truth, we define function $\mathbf{Z} = \mathcal{H}_{gt}^{-1}(\mathbf{Y})$,

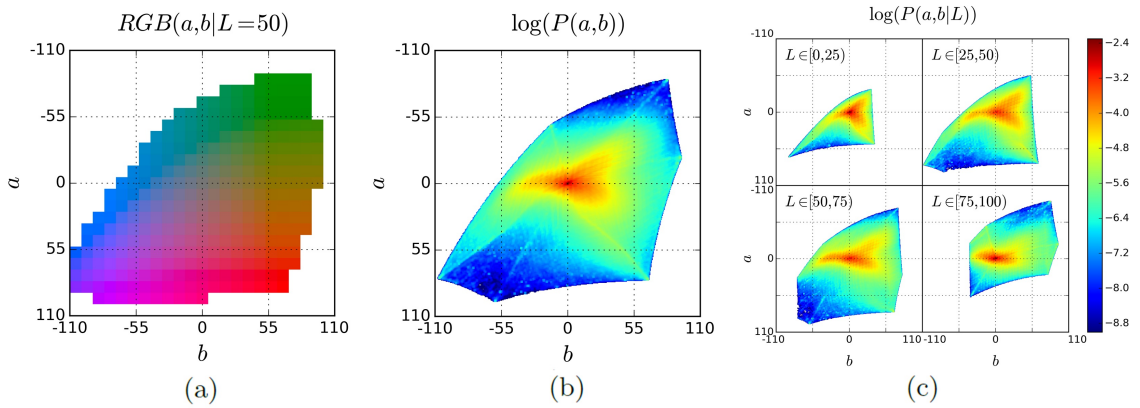



Figure 2.3: (a) Quantized $ab$ color space with a grid size of 10. A total of 313 $ab$ pairs are in gamut. (b) Empirical probability distribution of $ab$ values, shown in log scale. (c) Empirical probability distribution of $ab$ values, conditioned on $L$, shown in log scale.

which converts ground truth color $\mathbf{Y}$ to vector $\mathbf{Z}$, using a soft-encoding scheme[3]. We then use multinomial cross entropy loss $\mathrm{L}_{cl}(\cdot,\cdot)$, defined as:

$$\mathrm{L}_{cl}(\widehat{\mathbf{Z}},\mathbf{Z}) = -\sum_{h,w} v(\mathbf{Z}_{h,w}) \sum_{q} \mathbf{Z}_{h,w,q} \log(\widehat{\mathbf{Z}}_{h,w,q}) \tag{2.2}$$

where $v(\cdot)$ is a weighting term that can be used to rebalance the loss based on color-class rarity, as defined in Section 2.2.2 below. Finally, we map probability distribution $\widehat{\mathbf{Z}}$ to color values $\widehat{\mathbf{Y}}$ with function $\widehat{\mathbf{Y}} = \mathcal{H}(\widehat{\mathbf{Z}})$, which will be further discussed in Section 2.2.3.

### 2.2.2 Class Rebalancing

The distribution of $ab$ values in natural images is strongly biased towards values with low $ab$ values, due to the appearance of backgrounds such as clouds, pavement, dirt, and walls. Figure 2.3(b) shows the empirical distribution of pixels in $ab$ space, gathered from 1.3M training images in ImageNet [4]. Observe that the number of pixels in natural images at desaturated values are orders of magnitude higher than

[3]Each ground truth value $\mathbf{Y}_{h,w}$ can be encoded as a 1-hot vector $\mathbf{Z}_{h,w}$ by searching for the nearest quantized $ab$ bin. However, we found that soft-encoding worked well for training, and allowed the network to quickly learn the relationship between elements in the output space [46]. We find the 5-nearest neighbors to $\mathbf{Y}_{h,w}$ in the output space and weight them proportionally to their distance from the ground truth using a Gaussian kernel with $\sigma = 5$.

for saturated values. Without accounting for this, the loss function is dominated by desaturated *ab* values. We account for the class-imbalance problem by reweighting the loss of each pixel at train time based on the pixel color rarity. This is asymptotically equivalent to the typical approach of resampling the training space [47]. Each pixel is weighed by factor $\mathbf{w} \in \mathbb{R}^Q$, based on its closest *ab* bin.

$$v(\mathbf{Z}_{h,w}) = \mathbf{w}_{q^*}, \text{ where } q^* = \arg\max_q \mathbf{Z}_{h,w,q} \tag{2.3}$$

$$\mathbf{w} \propto \left((1-\lambda)\widetilde{\mathbf{p}} + \frac{\lambda}{Q}\right)^{-1}, \quad \mathbb{E}[\mathbf{w}] = \sum_q \widetilde{\mathbf{p}}_q \mathbf{w}_q = 1 \tag{2.4}$$

To obtain smoothed empirical distribution $\widetilde{\mathbf{p}} \in \Delta^Q$, we estimate the empirical probability of colors in the quantized *ab* space $\mathbf{p} \in \Delta^Q$ from the full ImageNet training set and smooth the distribution with a Gaussian kernel $\mathbf{G}_\sigma$. We then mix the distribution with a uniform distribution with weight $\lambda \in [0, 1]$, take the reciprocal, and normalize so the weighting factor is 1 on expectation. We found that values of $\lambda = \frac{1}{2}$ and $\sigma = 5$ worked well. We compare results with and without class rebalancing in Section 2.3.1.

### 2.2.3 Class Probabilities to Point Estimates

Finally, we define $\mathcal{H}$, which maps the predicted distribution $\widehat{\mathbf{Z}}$ to point estimate $\widehat{\mathbf{Y}}$ in *ab* space. One choice is to take the mode of the predicted distribution for each pixel, as shown in the right-most column of Figure 2.4 for two example images. This provides a vibrant but sometimes spatially inconsistent result, e.g., the red splotches on the bus. On the other hand, taking the mean of the predicted distribution produces spatially consistent but desaturated results (left-most column of Figure 2.4), exhibiting an unnatural sepia tone. This is unsurprising, as taking the mean after performing classification suffers from some of the same issues as optimizing for a Euclidean loss in a regression framework. To try to get the best of both worlds, we *interpolate* by re-adjusting the temperature $T$ of the softmax distribution, and taking the mean of the result. We draw inspiration from the simulated annealing technique [48], and thus refer to the operation as taking the *annealed-mean* of the distribution:

$$\mathcal{H}(\mathbf{Z}_{h,w}) = \mathbb{E}\big[f_T(\mathbf{Z}_{h,w})\big], \quad f_T(\mathbf{z}) = \frac{\exp(\log(\mathbf{z})/T)}{\sum_q \exp(\log(\mathbf{z}_q)/T)} \tag{2.5}$$

Setting $T = 1$ leaves the distribution unchanged, lowering the temperature $T$ produces a more strongly peaked distribution, and setting $T \to 0$ results in a 1-hot


Figure 2.4: The effect of temperature parameter $T$ on the *annealed-mean* output (Equation 2.5). The left-most images show the means of the predicted color distributions and the right-most show the modes. We use $T = 0.38$ in our system.

encoding at the distribution mode. We found that temperature $T = 0.38$, shown in the middle column of Figure 2.4, captures the vibrancy of the mode while maintaining the spatial coherence of the mean.

Our final system $\mathcal{F}$ is the composition of CNN $\mathcal{G}$, which produces a predicted distribution over all pixels, and the annealed-mean operation $\mathcal{H}$, which produces a final prediction. The system is not quite end-to-end trainable, but note that the mapping $\mathcal{H}$ operates on each pixel independently, with a single parameter, and can be implemented as part of a feed-forward pass of the CNN.

## 2.3 Experiments

In Section 2.3.1, we assess the graphics aspect of our algorithm, evaluating the perceptual realism of our colorizations, along with other measures of accuracy. We compare our full algorithm to several variants, along with recent [21] and concurrent work [39]. In Section 2.3.2, we show qualitative examples on legacy black and white images. In Section 2.3.3, we test how semantically interpretable our results are to a downstream classifier. In Section 2.3.4, we qualitatively investigate whether some low-level cues may be inordinately helping the network. In Section 2.3.5, we observe the multimodal distribution predicted by the network. We evaluate how well colorization works as a *pretext task* for self-supervised learning in Chapter 5.

| Colorization Results on ImageNet | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Method** | **Params (MB)** | **Model Feats (MB)** | **Runtime (ms)** | **AuC non-rebal (%)** | **AuC rebal (%)** | **VGG Top-1 Class Acc (%)** | **AMT Labeled Real (%)** |
| Ground Truth | – | – | – | 100 | 100 | 68.3 | 50 |
| Gray | – | – | – | 89.1 | 58.0 | 52.7 | – |
| Random | – | – | – | 84.2 | 57.3 | 41.0 | 13.0±4.4 |
| Dahl [21] | – | – | – | 90.4 | 58.9 | 48.7 | 18.3±2.8 |
| Larsson et al. [39] | 588 | 495 | 122.1 | **91.7** | 65.9 | **59.4** | **27.2±2.7** |
| Ours (L2) | 129 | 127 | 17.8 | 91.2 | 64.4 | 54.9 | 21.2±2.5 |
| Ours (L2, ft) | 129 | 127 | 17.8 | 91.5 | 66.2 | 56.5 | 23.9±2.8 |
| Ours (class) | 129 | 142 | 22.1 | 91.6 | 65.1 | 56.6 | 25.2±2.7 |
| Ours (full) | 129 | 142 | 22.1 | 89.5 | **67.3** | 56.0 | **32.3±2.2** |

Table 2.1: Colorization results on 10k images in the ImageNet validation set [4], as used in [39]. AuC refers to the area under the curve of the cumulative error distribution over *ab* space [38]. Results column 2 shows the class-balanced variant of this metric. Column 3 is the classification accuracy after colorization using the VGG-16 [24] network. Column 4 shows results from our AMT *real vs. fake* test (with mean and standard error reported, estimated by bootstrap [49]). Note that an algorithm that produces ground truth images would achieve 50% performance in expectation. Higher is better for all metrics. Rows refer to different algorithms; see text for a description of each. Parameter and feature memory, and runtime, were measured on a Titan X GPU using *Caffe* [50].

### 2.3.1 Evaluating colorization quality

We train our network on the 1.3M images from the ImageNet training set [4], validate on the first 10k images in the ImageNet validation set, and test on a separate 10k images in the validation set, same as in [39]. We show quantitative results in Table 1 on three metrics. A qualitative comparison for selected success and failure cases is shown in Figure 2.5. For a comparison on a full selection of random images, please see our project webpage.

To specifically test the effect of different loss functions, we train our CNN with various losses. We also compare to previous [21] and concurrent methods [39], which both use CNNs trained on ImageNet, along with naive baselines:

1. **Ours (full)** Our full method, with classification loss, defined in Equation 2.2, and class rebalancing, as described in Section 2.2.2. The network was trained from scratch with k-means initialization [51], using the ADAM solver for

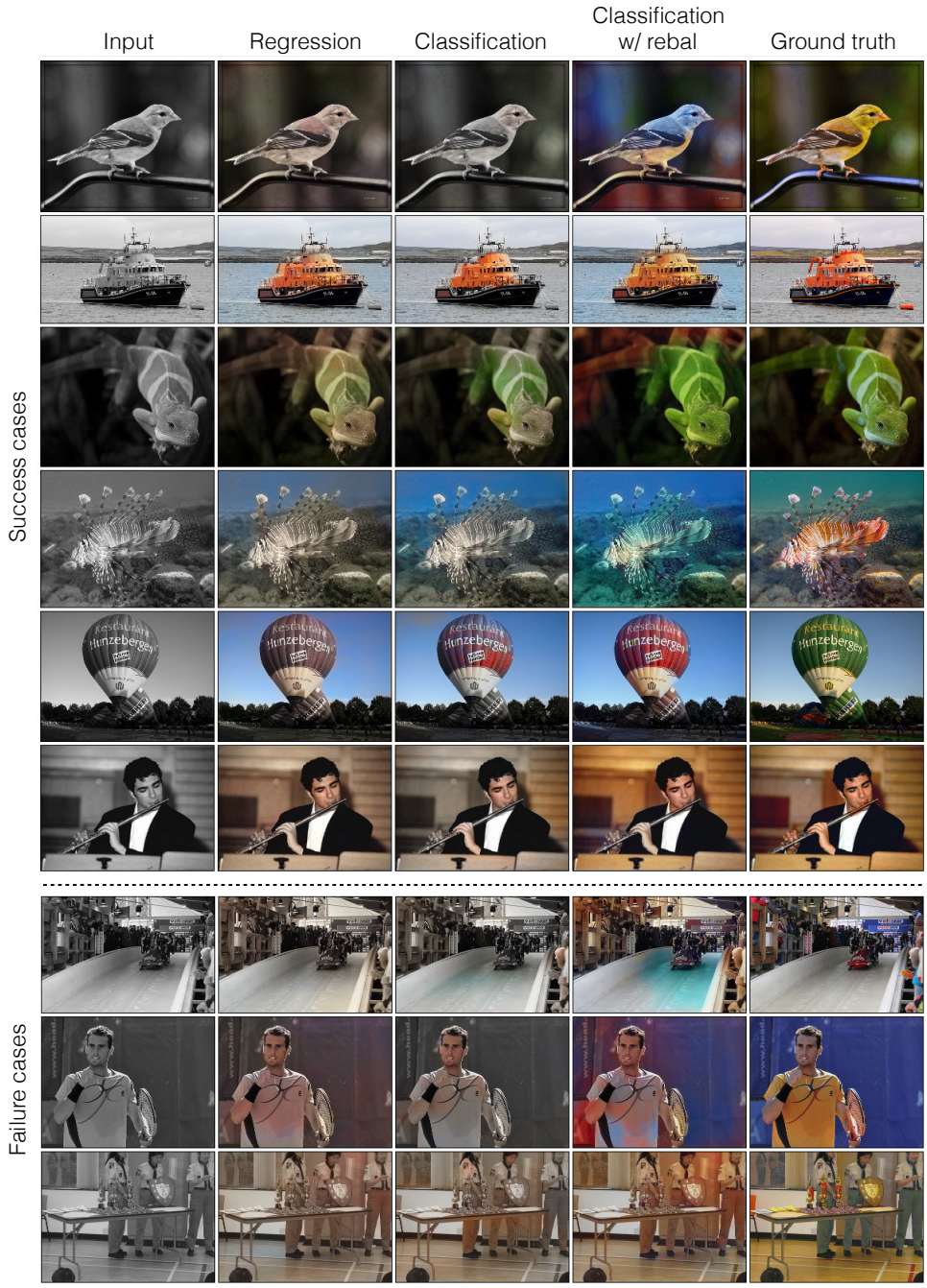



Figure 2.5: Example results from our ImageNet test set. Our classification loss with rebalancing produces more accurate and vibrant results than a regression loss or a classification loss without rebalancing. Successful colorizations are above the dotted line. Common failures are below. These include failure to capture long-range consistency, frequent confusions between red and blue, and a default sepia tone on complex indoor scenes. Please visit `http://richzhang.github.io/colorization/` to see the full range of results.

approximately 450k iterations[4].

2. **Ours (class)** Our network on classification loss but no class rebalancing ($\lambda = 1$ in Equation 2.4).

3. **Ours (L2)** Our network trained from scratch, with L2 regression loss, described in Equation 2.1, following the same training protocol.

4. **Ours (L2, ft)** Our network trained with L2 regression loss, fine-tuned from our full classification with rebalancing network.

5. **Larsson et al. [39]** A CNN method that also appears in these proceedings.

6. **Dahl [21]** A previous model using a Laplacian pyramid on VGG features, trained with L2 regression loss.

7. **Gray** Colors every pixel gray, with $(a, b) = 0$.

8. **Random** Copies the colors from a random image from the training set.

Evaluating the quality of synthesized images is well-known to be a difficult task, as simple quantitative metrics, like RMS error on pixel values, often fail to capture visual realism. To address the shortcomings of any individual evaluation, we test three that measure different senses of quality, shown in Table 1.

**1. Perceptual realism (AMT):** For many applications, such as those in graphics, the ultimate test of colorization is how compelling the colors look to a human observer. To test this, we ran a *real vs. fake* two-alternative forced choice experiment on Amazon Mechanical Turk (AMT). Participants in the experiment were shown a series of pairs of images. Each pair consisted of a color photo next to a re-colorized version, produced by either our algorithm or a baseline. Participants were asked to click on the photo they believed contained *fake* colors generated by a computer program. Individual images of resolution $256 \times 256$ were shown for one second each, and after each pair, participants were given unlimited time to respond. Each experimental session consisted of 10 practice trials (excluded from subsequent analysis), followed by 40 test pairs. On the practice trials, participants were given feedback as to whether or not their answer was correct. No feedback was given during the 40 test pairs. Each session tested only a single algorithm at a time, and participants were only allowed to complete at most one session. A total of 40

[4] $\beta_1 = .9$, $\beta_2 = .99$, and weight decay $= 10^{-3}$. Initial learning rate was $3 \times 10^{-5}$ and dropped to $10^{-5}$ and $3 \times 10^{-6}$ when loss plateaued, at 200k and 375k iterations, respectively. Other models trained from scratch followed similar training protocol.

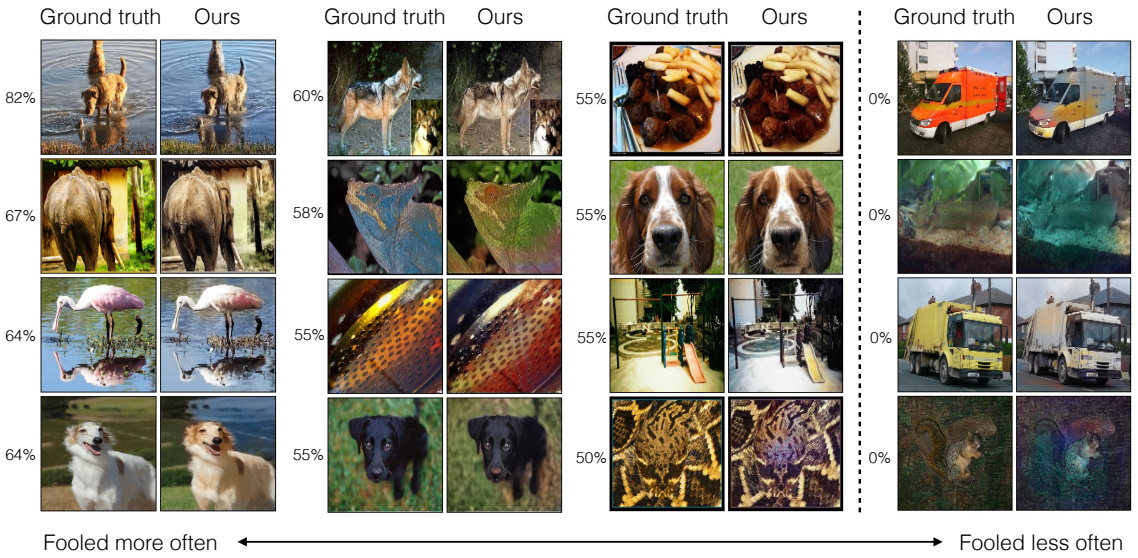



Figure 2.6: Images sorted by how often AMT participants chose our algorithm's colorization over the ground truth. In all pairs to the left of the dotted line, participants believed our colorizations to be more real than the ground truth on $\geq 50\%$ of the trials. In some cases, this may be due to poor white balancing in the ground truth image, corrected by our algorithm, which predicts a more prototypical appearance. Right of the dotted line are examples where participants were never fooled.

participants evaluated each algorithm. To ensure that all algorithms were tested in equivalent conditions (i.e. time of day, demographics, etc.), all experiment sessions were posted simultaneously and distributed to Turkers in an i.i.d. fashion.

To check that participants were competent at this task, 10% of the trials pitted the ground truth image against the Random baseline described above. Participants successfully identified these random colorizations as fake 87% of the time, indicating that they understood the task and were paying attention.

Figure 2.6 gives a better sense of the participants' competency at detecting subtle errors made by our algorithm. The far right column shows example pairs where participants identified the fake image successfully in 100% of the trials. Each of these pairs was scored by at least 10 participants. Close inspection reveals that on these images, our colorizations tend to have giveaway artifacts, such as the yellow blotches on the two trucks, which ruin otherwise decent results.

Nonetheless, our full algorithm fooled participants on 32% of trials, as shown in Table 1. This number is significantly higher than all compared algorithms ($p < 0.05$

in each case) except for Larsson et al., against which the difference was not significant ($p = 0.10$; all statistics estimated by bootstrap [49]). These results validate the effectiveness of using both a classification loss and class-rebalancing.

Note that if our algorithm exactly reproduced the ground truth colors, the forced choice would be between two identical images, and participants would be fooled 50% of the time on expectation. Interestingly, we can identify cases where participants were fooled *more* often than 50% of the time, indicating our results were deemed more realistic than the ground truth. Some examples are shown in the first three columns of Figure 2.6. In many case, the ground truth image is poorly white balanced or has unusual colors, whereas our system produces a more prototypical appearance.

**2. Semantic interpretability (VGG classification):** Does our method produce realistic enough colorizations to be interpretable to an off-the-shelf object classifier? We tested this by feeding our *fake* colorized images to a VGG network [24] that was trained to predict ImageNet classes from *real* color photos. If the classifier performs well, that means the colorizations are accurate enough to be informative about object class. Using an off-the-shelf classifier to assess the realism of synthesized data has been previously suggested by [31].

The results are shown in the second column from the right of Table 1. Classifier performance drops from 68.3% to 52.7% after ablating colors from the input. After re-colorizing using our full method, the performance is improved to 56.0% (other variants of our method achieve slightly higher results). The Larsson et al. [39] method achieves the highest performance on this metric, reaching 59.4%. For reference, a VGG classification network fine-tuned on grayscale inputs reaches a performance of 63.5%.

In addition to serving as a perceptual metric, this analysis demonstrates a practical use for our algorithm: without any additional training or fine-tuning, we can improve performance on grayscale image classification, simply by colorizing images with our algorithm and passing them to an off-the-shelf classifier.

**3. Raw accuracy (AuC):** As a low-level test, we compute the percentage of predicted pixel colors within a thresholded L2 distance of the ground truth in *ab* color space. We then sweep across thresholds from 0 to 150 to produce a cumulative mass function, as introduced in [38], integrate the area under the curve (AuC), and normalize. Note that this AuC metric measures *raw prediction accuracy*, whereas our method aims for *plausibility*.

Our network, trained on classification without rebalancing, outperforms our L2 variant (when trained from scratch). When the L2 net is instead fine-tuned from a color classification network, it matches the performance of the classification network. This indicates that the L2 metric can achieve accurate colorizations, but has difficulty in optimization from scratch. The Larsson et al. [39] method achieves

slightly higher accuracy. Note that this metric is dominated by desaturated pixels, due to the distribution of $ab$ values in natural images (Figure 2.3(b)). As a result, even predicting gray for every pixel does quite well, and our full method with class rebalancing achieves approximately the same score.

Perceptually interesting regions of images, on the other hand, tend to have a distribution of $ab$ values with higher values of saturation. As such, we compute a class-balanced variant of the AuC metric by re-weighting the pixels inversely by color class probability (Equation 2.4, setting $\lambda = 0$). Under this metric, our full method outperforms all variants and compared algorithms, indicating that class-rebalancing in the training objective achieved its desired effect.

### 2.3.2 Legacy Black and White Photos

Since our model was trained using "fake" grayscale images generated by stripping $ab$ channels from color photos, we also ran our method on real legacy black and white photographs, as shown in Figure 2.7 (additional results can be viewed on our project webpage). Figures 2.8, 2.9, and 2.10 show examples including work of renowned photographers, such as Ansel Adams and Henri Cartier-Bresson, photographs of politicians and celebrities, and old family photos. One can see that our model is often able to produce good colorizations, even though the low-level image statistics of old legacy photographs are quite different from those of modern-day photos. One can see that our model is still able to produce good colorizations, even though the low-level image statistics of the legacy photographs are quite different from those of the modern-day photos on which it was trained.

### 2.3.3 Semantic Interpretability of Colorizations

In Section 2.3.1, we investigated using the VGG classifier to evaluate the semantic interpretability of our colorization results. We show the categories which perform well and the ones which perform poorly, using this metric. We also show commonly confused categories after recolorization.

**Category Performance** In Figure 2.11, we show a selection of classes that have the most improvement in VGG classification with respect to grayscale, along with the classes for which our colorizations hurt the most. Interestingly, many of the top classes actually have a color in their name, such as the green snake, orange, and goldfinch. The bottom classes show some common errors of our system, such as coloring clothing incorrectly and inconsistently and coloring an animal with a plausible but incorrect color. This analysis was performed using 48k images from the

Figure 2.7: Applying our method to legacy black and white photos. Left to right: photo by David Fleay of a Thylacine, now extinct, 1936; photo by Ansel Adams of Yosemite; amateur family photo from 1956; *Migrant Mother* by Dorothea Lange, 1936.

ImageNet validation set, and images in the top and bottom 10 classes are provided on the website.

Our process for sorting categories and images is described below. For each category, we compute the top-5 classification performance on grayscale and recolorized images, $\mathbf{a}_{gray}, \mathbf{a}_{recolor} \in [0,1]^C$, where $C = 1000$ categories. We sort the categories by $\mathbf{a}_{recolor} - \mathbf{a}_{gray}$. The re-colored vs grayscale performance per category is shown in Figure 2.12(a), with top and bottom 50 categories highlighted. For the top example categories, the individual images are sorted by ascending rank of the correct classification of the recolorizeed image, with tiebreakers on descending rank of the correct classification of the grayscale image. For the bottom example categories, the images are sorted in reverse, in order to highlight the instances when recolorization results in an errant classification relative to the grayscale image.

**Common Confusions** To further investigate the biases in our system, we look at the common classification confusions that often occur after image recolorization, but not with the original ground truth image. Examples for some top confusions are shown in Figure 2.13. An image of a “minibus" is often colored yellow, leading to a misclassification as “school bus". Animal classes are sometimes colored differently than ground truth, leading to misclassification to related species. Note that the colorizations are often visually realistic, even though they lead to a misclassification.

To find common confusions, we compute the rate of top-5 confusion $\mathbf{C}_{orig}, \mathbf{C}_{recolor} \in$

Figure 2.8: Applying our method to black and white photographs by Ansel Adams.

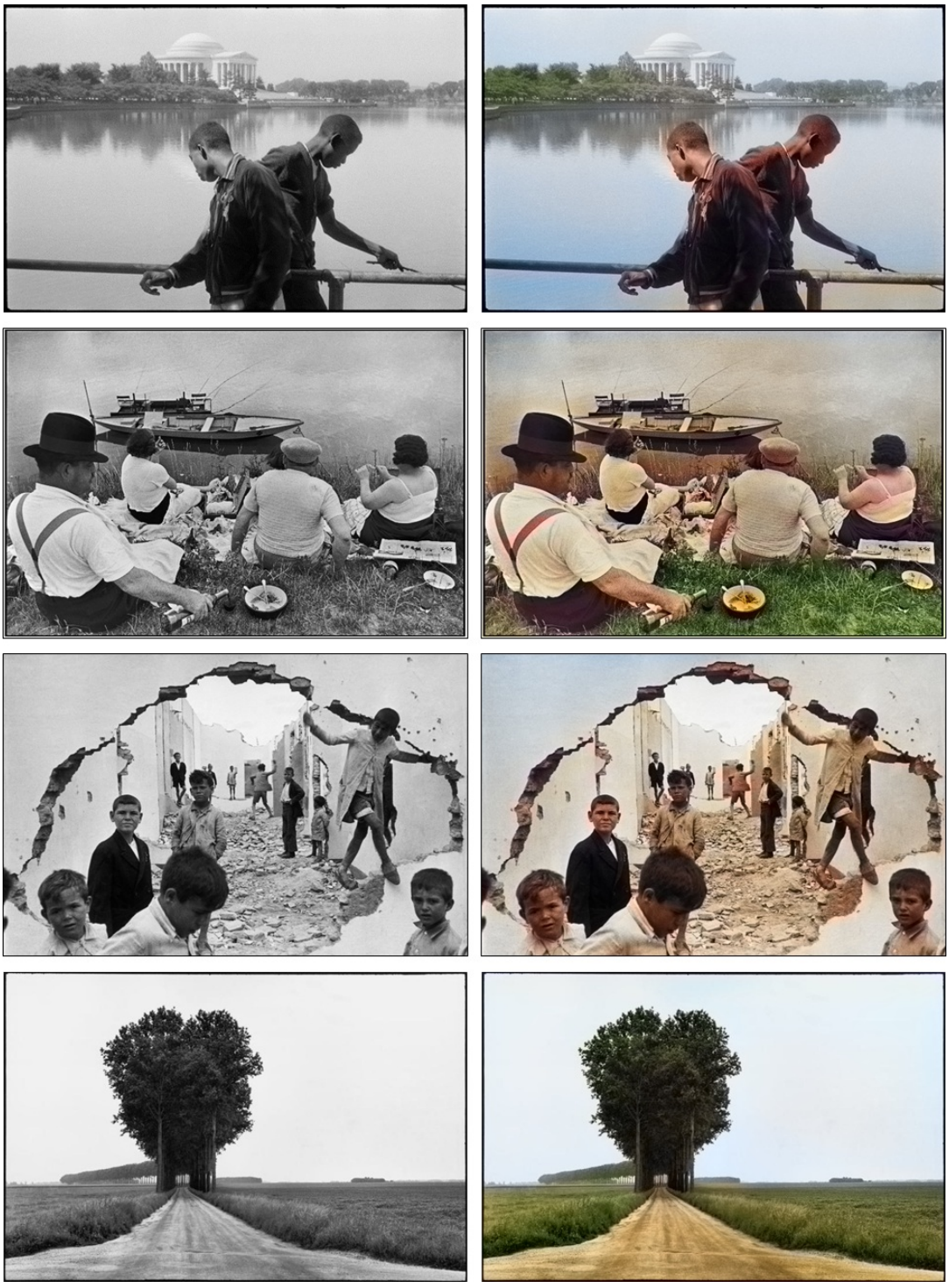


Figure 2.9: Applying our method to black and white photographs by Henri Cartier-Bresson.

Figure 2.10: Applying our method to legacy black and white photographs. Top to bottom, left to right: photo of Elvis Presley, photo of *Migrant Mother* by Dorothea Lange, photo of Marilyn Monroe, an amateur family photo, photo by Henri Cartier-Bresson, photo by Dr. David Fleay of *Benjamin*, the last captive thylacine which went extinct in 1936.


Figure 2.11: Images colorized by our algorithm from selected categories. Categories are sorted by VGG object classification accuracy of our colorized images relative to accuracy on gracyscale images. Top: example categories where our colorization *helps* the most. Bottom: example categories where our colorization *hurts* the most. Number in parentheses indicates category rank amongst all 1000. Notice that the categories most affected by colorization are those for which color information is highly diagnostic, such as birds and fruits. The bottom examples show several kinds of failures: 1) artificial objects such as modems and clothes have ambiguous colors; color is not very informative for classification, and moreover, our algorithm tends to predict an incoherent distribution of red and blue, 2) for certain categories, like the gray fox, our algorithm systematically predicts the wrong color, confusing the species.

$[0,1]^{C\times C}$, with ground truth colors and after recolorization. A value of $\mathbf{C}_{c,d} = 1$ means that every image in category $c$ was classified as category $d$ in the top-5. We find the class-confusion added after recolorization by computing $\mathbf{A} = \mathbf{C}_{recolor} - \mathbf{C}_{orig}$, and sort the off-diagonal entries. Figure 2.12(b) shows all $C \times (C-1)$ off-diagonal entries of $\mathbf{C}_{recolor}$ vs $\mathbf{C}_{orig}$, with the top 100 entries from $\mathbf{A}$ highlighted. For each category

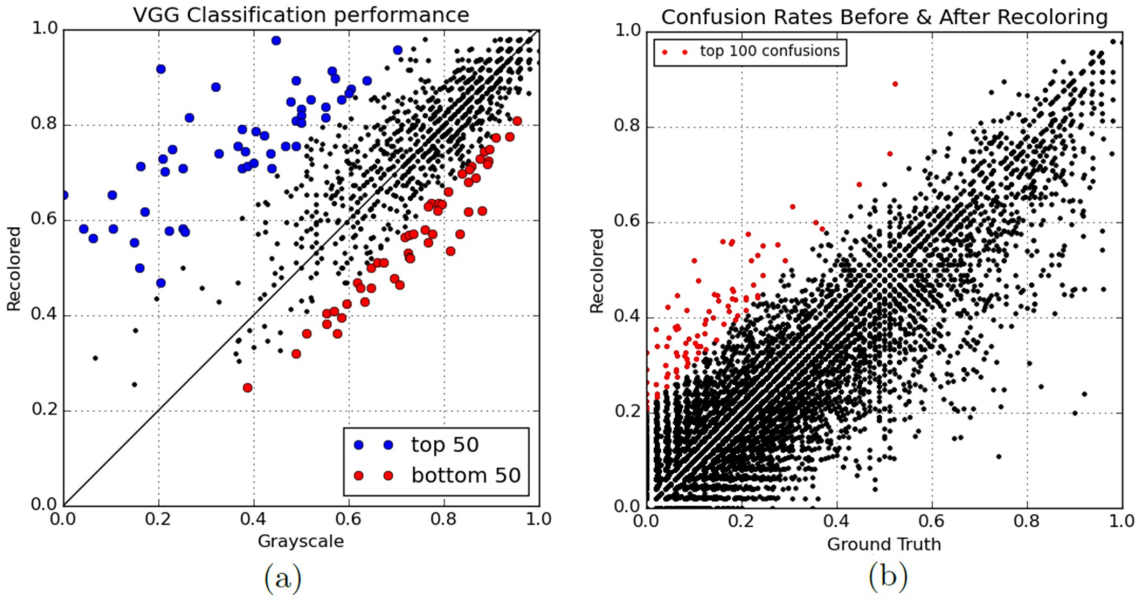



Figure 2.12: (a) Performance of VGG top-5 classification on recolorized images vs grayscale images per category (b) Top-5 confusion rates with recolorizations and original colors. Test was done on last 48,000 images in ImageNet validation set.

pair $(c, d)$, we extract the images that contained the confusion after recolorization, but not with the original colorization. We then sort the images in descending order of the classification score of the confused category.

### 2.3.4 Is the network exploiting low-level cues?

Unlike many computer vision tasks that can be roughly categorized as low, mid or high-level vision, color prediction requires understanding an image at both the pixel and the semantic-level. Studies of natural image statistics have shown that the lightness value of a single pixel can highly constrain the likely color of that pixel: darker lightness values tend to be correlated with more saturated colors [52].

Could our network be exploiting a simple, low-level relationship like this, in order to predict color?[5] We tested this hypothesis with the simple demonstration in Figure 2.14. Given a grayscale Macbeth color chart as input, our network was unable to recover its colors. This is true, despite the fact that the lightness values vary considerably for the different color patches in this image. On the other hand,

[5]E.g., previous work showed that CNNs can learn to use chromatic aberration cues to predict, given an image patch, its (x,y) location within an image [53].

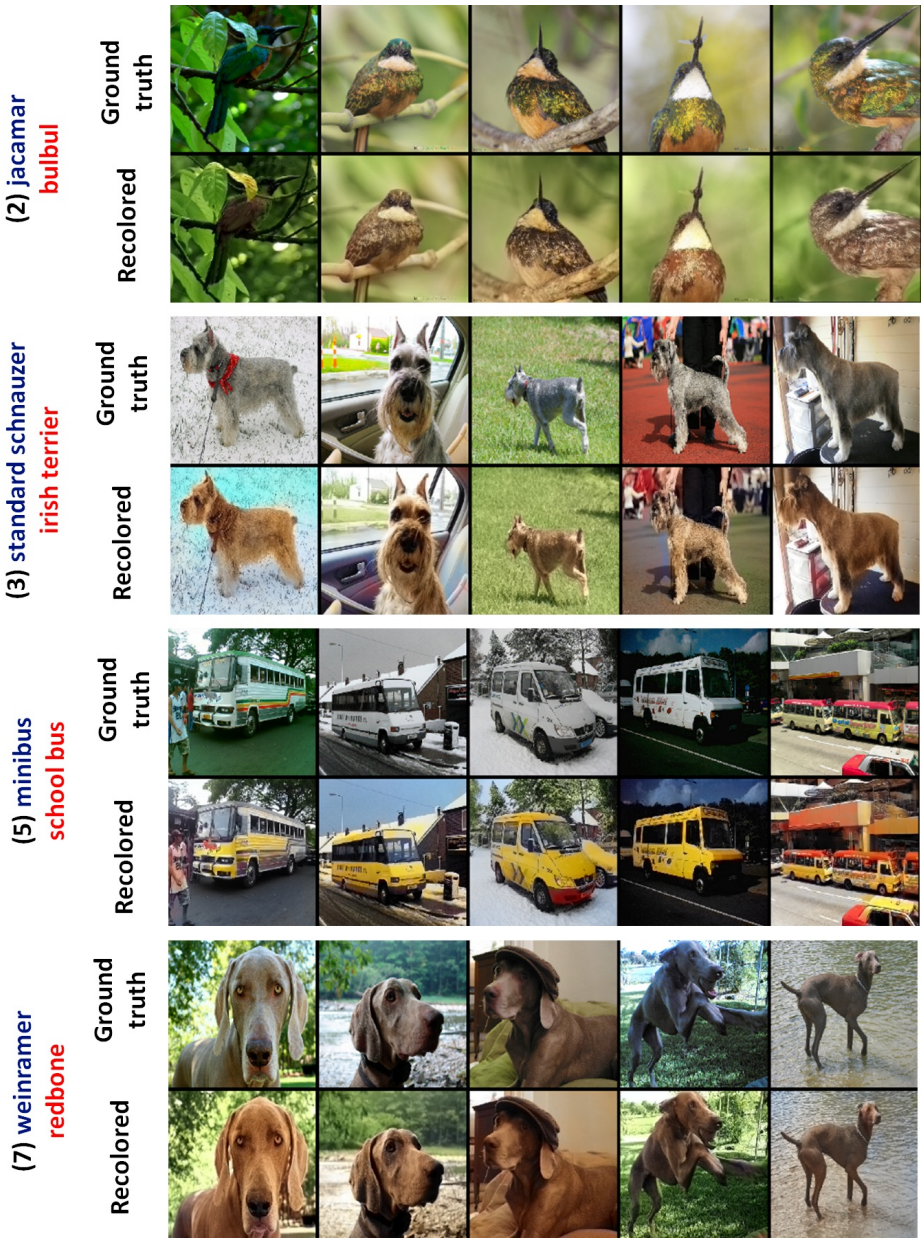



Figure 2.13: Examples of some most-confused categories. Top rows show ground truth image. Bottom rows show recolorized images. Rank of common confusion in parentheses. **Ground truth** and **confused** categories after recolorization are labeled.

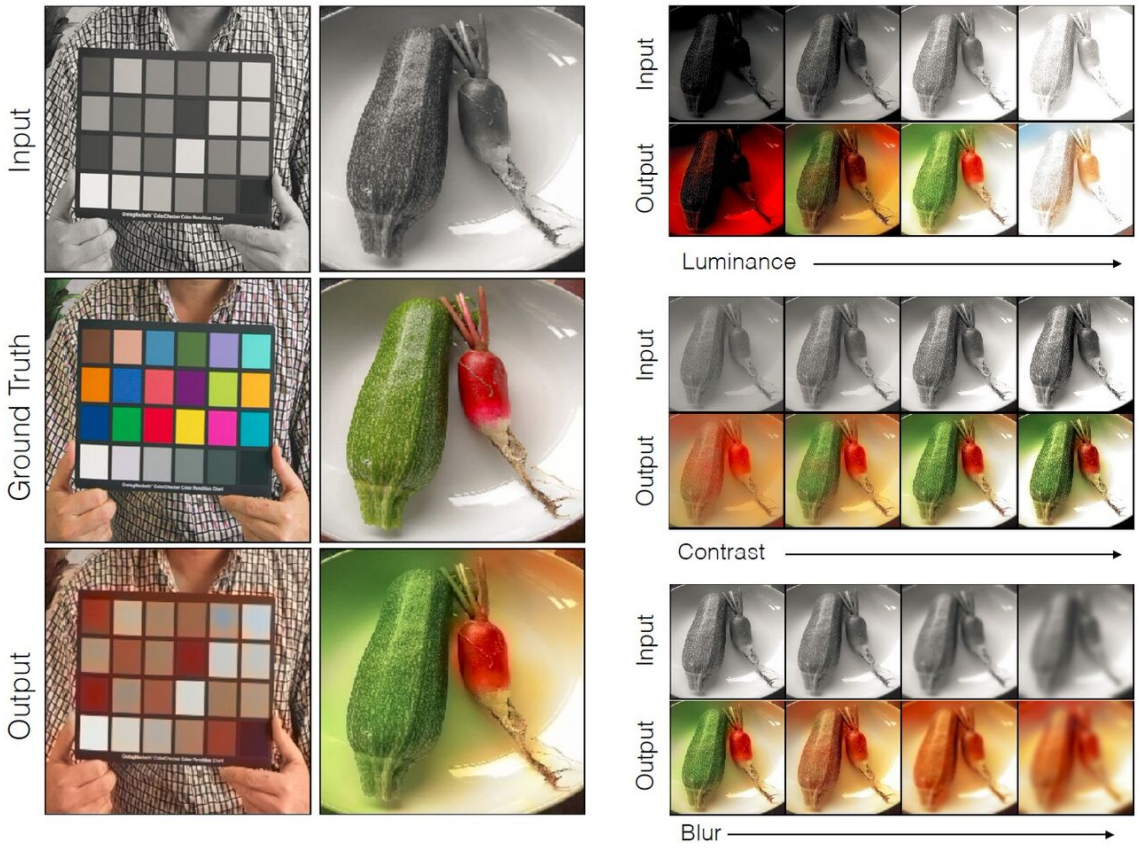



Figure 2.14: Left: pixel lightness on its own does not reveal color, as shown by the color chart. In contrast, two vegetables that are nearly isoluminant are recognized as having different colors. Right: stability of the network predictions with respect to low-level image transformations.

given two recognizable vegetables that are roughly isoluminant, the system is able to recover their color.

In Figure 2.14, we also demonstrate that the prediction is somewhat stable with respect to low-level lightness and contrast changes. Blurring, on the other hand, has a bigger effect on the predictions in this example, possibly because the operation removes the diagnostic texture pattern of the zucchini.

### 2.3.5 Does our model learn multimodal color distributions?

As discussed in Section 2.2.1, formulating color prediction as a multinomial classification problem allows the system to predict multimodal distributions, and

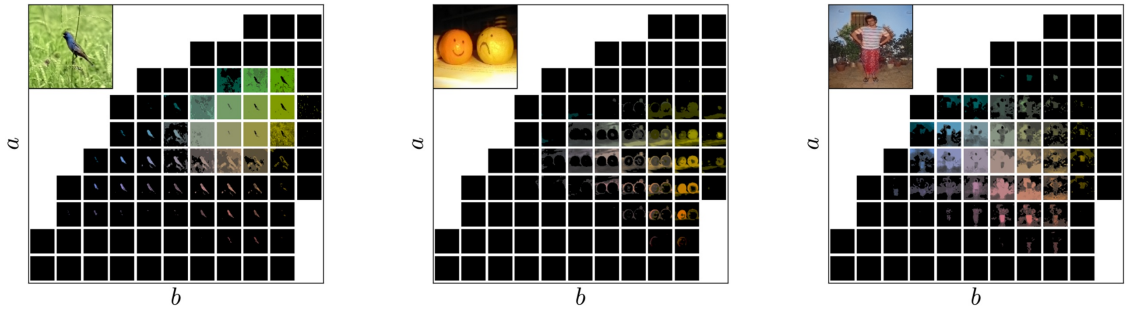



Figure 2.15: The output probability distributions per image. The top-left image is final prediction of our system. The black sub-images are quantized blocks of the $ab$ gamut. High probabilities are shown as higher luminance and are quantized for clarity. (a) Background of bird is predicted to be green or brown. Foreground bird has distribution across blue and red colors. (b) Oranges are predicted to be different colors. (c) The person's shirt and sarong has uncertainty across turqoise/cyan/orange and red/pink/purple colors, respectively. Note that despite the multimodality of the per-pixel distributions, the results after taking the annealed-mean are typically spatially consistent.

can capture the inherent ambiguity in the color of natural objects. In Figure 2.15, we illustrate the probability outputs $\widehat{\mathbf{Z}}$ and demonstrate that the network does indeed learn multimodal distributions. The system output $\widehat{\mathbf{Y}}$ is shown in the top-left of Figure 2.15. Each block illustrates the probability map $\widehat{\mathbf{Z}}_q \in [0,1]^{H,W}$ given $ab$ bin $q$ in the output space. For clarity, we show a subsampling of the $Q$ total output bins and coarsely quantize the probability values. In Figure 2.15(a), the system clearly predicts a different distribution for the background vegetation and the foreground bird. The background is predicted to be green, yellow, or brown, while the foreground bird is predicted to be red or blue. Figure 2.15(b) shows that oranges can be predicted to be different colors. Lastly, in Figure 2.15(c), the man's sarong is predicted to be either red, pink, or purple, while his shirt is classified as turquoise, cyan or light orange. Note that despite the multi-modality of the prediction, taking the annealed-mean of the distribution produces a spatially consistent prediction.

## 2.4 Discussion

While image colorization is a boutique computer graphics task, it is also an instance of a difficult pixel prediction problem in computer vision. Our method not

only provides a useful graphics output, but can also be viewed as a pretext task for representation learning. In Chapter 5, we show that although only trained to color, our network learns a representation that is surprisingly useful for high-level semantic tasks, such as object classification, detection, and segmentation.

Here we have shown that colorization with a deep CNN and a well-chosen objective function can come closer to producing results indistinguishable from real color photos. However, the method can make mistakes. In addition, in this chapter, we proposed a method which attempts to produce a *single, plausible* answer. However, a user may have wanted a different colorization. Motivated by these problems, in Chapter 3, we investigate integrating user inputs into the colorization pipeline.

# Chapter 3

# User-Guided Image Colorization

In this chapter, we propose a deep learning approach for *user-guided* image colorization. The system directly maps a grayscale image, along with sparse, local user "hints" to an output colorization with a CNN. Rather than using hand-defined rules, the network propagates user edits by fusing low-level cues along with high-level semantic information, *learned from large-scale data*. In the previous chapter, we resolved multimodality by posing the problem as a classification. Here, we actually rely on the user at test time to resolve the ambiguity. We train on a million images, with simulated user inputs. To guide the user towards efficient input selection, the system recommends likely colors based on the input image and current user inputs. The colorization is performed in a single feed-forward pass, enabling real-time use. Even with randomly simulated user inputs, we show that the proposed system helps novice users quickly create realistic colorizations, and offers large improvements in colorization quality with just a minute of use. In addition, we demonstrate that the framework can incorporate other user "hints" to the desired colorization, showing an application to color histogram transfer. Our code and models are available at https://richzhang.github.io/ideepcolor. [1]

## 3.1 Motivation

There is something uniquely and powerfully satisfying about the simple act of adding color to black and white imagery. Whether as a way of rekindling old, dormant memories or expressing artistic creativity, people continue to be fascinated

[1] This work was originally published as *Real-Time User-Guided Image Colorization with Learned Deep Priors* in SIGGRAPH, 2017 [54]. We often refer to the Chapter 2 automatic colorization method as Zhang et al. [19].

Figure 3.1: Our proposed method colorizes a grayscale image (left), guided by sparse user inputs (second), in real-time, providing the capability for quickly generating multiple plausible colorizations (middle to right). Photograph of *Migrant Mother* by Dorothea Lange, 1936 (Public Domain).

by colorization. From remastering classic black and white films, to the enduring popularity of coloring books for all ages, to the surprising enthusiasm for various (often not very good) automatic colorization bots online[2], this topic continues to fascinate the public.

In computer graphics, two broad approaches to image colorization exist: user-guided edit propagation and data-driven automatic colorization. In the first paradigm, popularized by the seminal work of Levin et al. [55], a user draws colored strokes over a grayscale image. An optimization procedure then generates a colorized image that matches the user's scribbles, while also adhering to hand-defined image priors, such as piecewise smoothness. These methods can achieve impressive results but often require intensive user interaction (sometimes over fifty strokes), as each differently colored image region must be explicitly indicated by the user. Because the system purely relies on user inputs for colors, even regions with little color uncertainty, such as green vegetation, need to be specified. Less obviously, even if a user knows what general color an object should take on, it can be surprisingly difficult to select the exact desired natural chrominance.

To address these limitations, researchers have also explored more data-driven colorization methods. These methods colorize a grayscale photo in one of two ways: either by matching it to an exemplar color image in a database and non-parametrically "stealing" colors from that photo, an idea going back to Image Analogies [33], or by learning parametric mappings from grayscale to color from large-scale image data. The recent methods in this paradigm proposed by Iizuka et al. [40], Larsson et al. [39], and our own method from Chapter 2 [19], use deep networks and are fully automatic. Although this makes colorizing a new photo cheap and easy, the

[2]e.g., http://demos.algorithmia.com/colorize-photos/

results often contain incorrect colors and obvious artifacts. More fundamentally, the color of an object, such as a t-shirt, is often inherently ambiguous – it could be blue, red, or green. Current automatic methods aim to choose a single colorization, and do not allow a user to specify their preference for a plausible, or perhaps artistic, alternative.

Might we be able to get the best of both worlds, leveraging large-scale data to learn priors about natural color imagery, while at the same time incorporating user control from traditional edit propagation frameworks? We propose to train a CNN to directly map grayscale images, along with sparse user inputs, to an output colorization. During training, we randomly simulate user inputs, allowing us to bypass the difficulty of collecting user interactions. Though our network is trained with ground truth natural images, the network can colorize objects with different, or even unlikely colorizations, if desired.

Most traditional tools in interactive graphics are defined either procedurally – e.g., as a designed image filter – or as constraints applied in a hand-engineered optimization framework. The behavior of the tool is therefore fully specified by human fiat. This approach is fundamentally limited by the skill of engineers to design complex operations/constraints that actually accomplish what is intended of them. Our approach differs in that the effect of interaction is *learned*. Through learning, the algorithm may come up with a more powerful procedure for translating user edits to colorized results than would be feasible by human design.

Our contribution are as follows: (1) We *end-to-end learn* how to propagate sparse user points from large-scale data, by training a deep network to directly predict the mapping from grayscale image and user points to full color image. (2) To guide the user toward making informed decisions, we provide a data-driven color palette, which suggests the most probable colors at any given location. (3) We run a study, showing that even given minimal training with our interface and limited time to colorize an image (1 min), novice users can quickly learn to produce colorizations that can often fool real human judges in a real vs. fake test. (4) Though our system is trained on natural images, it can also generate unusual colorizations. (5) We demonstrate that this framework is not limited to user points, and can, in principle, be trained with any statistic of the output, for example, global color distribution or average image saturation.

## 3.2 Background

**User-guided colorization** Prior interactive colorization work focused on local control, such as user strokes [55, 56]. Because the strokes are propagated using

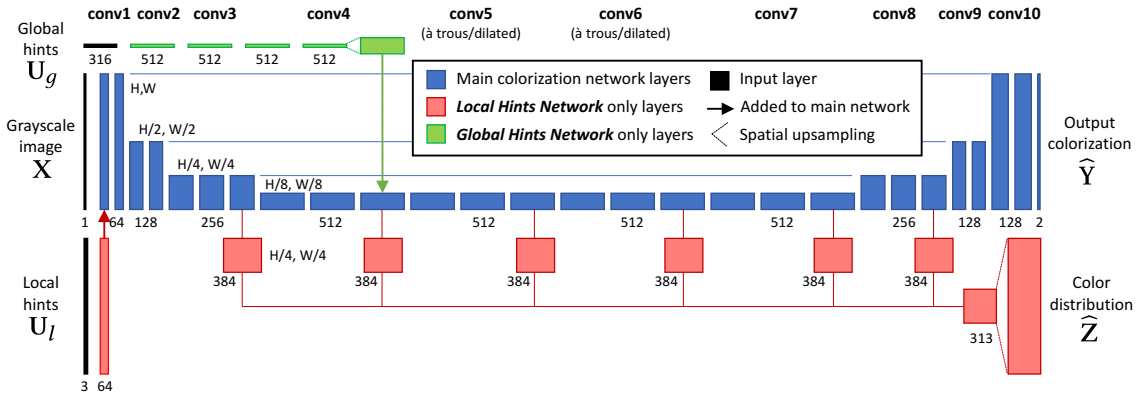



Figure 3.2: **Network architecture** We train two variants of the user interaction colorization network. Both variants use the blue layers for predicting a colorization. The **Local Hints Network** also uses red layers to (a) incorporate user points $\mathbf{U}_l$ and (b) predict a color distribution $\widehat{\mathbf{Z}}$. The **Global Hints Network** uses the green layers, which transforms global input $\mathbf{U}_g$ by $1 \times 1$ `conv` layers, and adds the result into the main colorization network. Each box represents a `conv` layer, with vertical dimension indicating feature map spatial resolution, and horizontal dimension indicating number of channels. Changes in resolution are achieved through subsampling and upsampling operations. In the main network, when resolution is decreased, the number of feature channels are doubled. Shortcut connections are added to upsampling convolution layers.

low-level similarity metrics, such as spatial offset and intensity difference, numerous user edits are typically required to achieve realistic results. To reduce user efforts, later methods focused on designing better similarity metrics [57, 58] and utilizing long-range connections [59, 60]. Learning machinery, such as boosting [61], local linear embeddings [62], feature discrimination [63], and more recently, neural networks [64], have been proposed to automatically learn similarity between pixels given user strokes and input images. In addition to local control, varying the color theme [65, 66] and color palette [67] are popular methods of expressive global control. We show that we can integrate global hints to our network and control colorization results by altering the color distribution and average saturation (see Section 3.3.3). Concurrently, Sangkloy et al. [68] developed a system to translate sketches to real images, with support for user color strokes, while PaintsChainer [69] and Frans [70] have developed open-source interactive online applications for line-drawing colorization.

**Automatic colorization** Early semi-automatic methods [34–37, 71] utilize an example-based approach that transfers color statistics from a reference image or

multiple images [72, 73] to the input grayscale image with techniques such as color transfer [74] and image analogies [33]. These methods work remarkably well when the input and the reference share similar content. However, finding reference images is time-consuming and can be challenging for rare objects or complex scenes, even when using semi-automatic retrieval methods [37]. In addition, some algorithms [37, 71] involve tedious manual efforts on defining corresponding regions between images.

Recently, fully automatic methods [20, 38–40, 75] have been proposed, including our own [19]. The recent methods from train CNNs [76] on large-scale image collections [4, 44] to directly map grayscale images to output colors. The networks can learn to combine low and high-level cues to perform colorization, and have been shown to produce realistic results, as determined by human judgments [19]. However, these approaches aim to produce a *single* plausible result, even though colorization is intrinsically an ill-posed problem with multi-modal uncertainty [22]. Larsson et al. [39] provide some post-hoc control through globally biasing the hue, or by matching global statistics to a target histogram. Our work addresses this problem by learning to integrate input hints in an end-to-end manner.

**Deep semantic image editing** Deep neural networks [77] excel at extracting rich semantics from images, from middle-level concepts like material [78, 79] and segmentation [80], to high-level knowledge such as objects [81] and scene categories [44]. All of this information could potentially benefit semantic image editing, i.e. changing the high-level visual content with minimal user interaction. Recently, neural networks have shown impressive results for various image processing tasks, such as photo enhancement [82], sketch simplification [83], style transfer [84, 85], inpainting [29], image blending [86] and denoising [87]. Most of these works built image filtering pipelines and trained networks that produce a filtered version of the input image with different low-level local details. However, none of these methods allowed dramatic, high-level modification of the visual appearance, nor do they provide diverse outputs in a user controllable fashion. On the contrary, we train a network that takes an input image as well as minimal user guidance and produces global changes in the image with a few clicks. Barnes et al. [88] emphasize that control and interactivity are key to image editing, because user intervention not only can correct errors, but can also help explore the vast design space of creative image manipulation. We incorporate this concept into an intuitive interface that provides expressive controls as well as real-time feedback. Zhu et al. [89] provided an interactive deep image synthesis interface that builds on an image prior learned by a deep generative network. Xu et al. [90] train a deep network for interactive object segmentation. Isola et al. [75] and Sangkloy et al. [68] train networks to generate images from sketches, using synthetic

sketches generated by edge detection algorithms for training data.

## 3.3 Approach

We train a deep network to predict the color of an image, given the grayscale version and user inputs. In Section 3.3.1, we describe the objective of the network. We then describe the two variants of our system (i) the **Local Hints Network** in Section 3.3.2, which uses sparse user points, and (ii) the **Global Hints Network** in Section 3.3.3, which uses global statistics. In Section 3.3.4, we define our network architecture.

### 3.3.1 Learning to Colorize

The inputs to our system are a grayscale image $\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$, along with an input user tensor $\mathbf{U}$. The grayscale image is the $L$, or lightness in the CIE $Lab$ color space, channel. The output of the system is $\widehat{\mathbf{Y}} \in \mathbb{R}^{H \times W \times 2}$, the estimate of the $ab$ color channels of the image. The mapping is learned with a CNN $\mathcal{F}$, parameterized by $\theta$, with the network architecture specified in Section 3.3.4 and shown in Figure 6.2. We train the network to minimize the objective function in Equation 3.1, across $\mathcal{D}$, which represents a dataset of grayscale images, user inputs, and desired output colorizations. Loss function $\mathcal{L}$ describes how close the network output is to the ground truth.

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{\mathbf{X},\mathbf{U},\mathbf{Y}\sim\mathcal{D}}[\mathcal{L}(\mathcal{F}(\mathbf{X},\mathbf{U};\theta),\mathbf{Y})] \tag{3.1}$$

We train two variants of our network, with local user hints $\mathbf{U}_l$ and global user hints $\mathbf{U}_g$. During training, the hints are generated by giving the network a "peek", or projection, of the ground truth color $\mathbf{Y}$ using functions $\mathcal{P}_l$ and $\mathcal{P}_g$, respectively.

$$\mathbf{U}_l = \mathcal{P}_l(\mathbf{Y}) \qquad \mathbf{U}_g = \mathcal{P}_g(\mathbf{Y}) \tag{3.2}$$

The minimization problems for the Local and Global Hints Networks are then described below in Equation 3.3. Because we are using functions $\mathcal{P}_l, \mathcal{P}_g$ to synethtically generate user inputs, our dataset only needs to contain grayscale and color images. We use the 1.3M ImageNet dataset [4].

$$\begin{aligned} \theta_l^* &= \arg\min_{\theta_l} \mathbb{E}_{\mathbf{X},\mathbf{Y}\sim\mathcal{D}}[\mathcal{L}(\mathcal{F}_l(\mathbf{X},\mathbf{U}_l;\theta_l),\mathbf{Y})] \\ \theta_g^* &= \arg\min_{\theta_g} \mathbb{E}_{\mathbf{X},\mathbf{Y}\sim\mathcal{D}}[\mathcal{L}(\mathcal{F}_g(\mathbf{X},\mathbf{U}_g;\theta_g),\mathbf{Y})] \end{aligned} \tag{3.3}$$

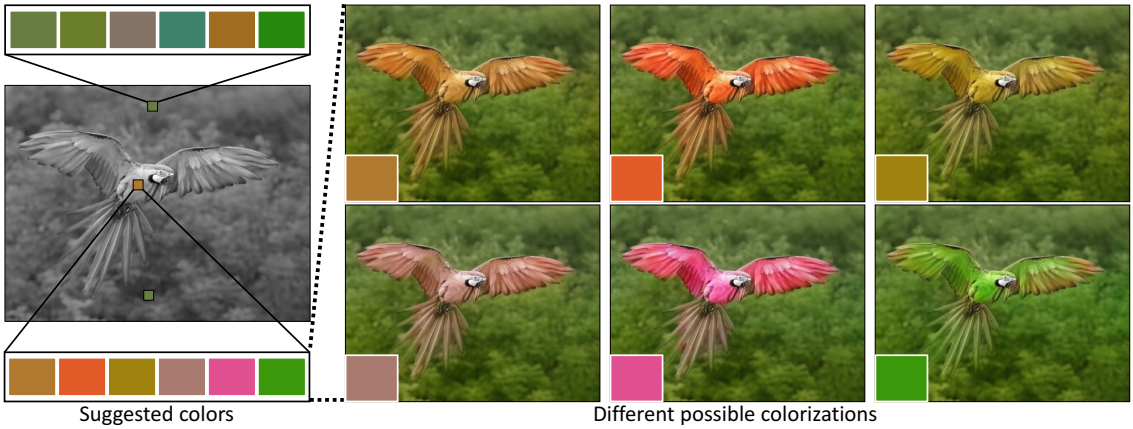



Figure 3.3: **Suggested Palette** Our interface provides suggested colors for any pixel, sorted by likelihood, based on the predicted color distribution given by our network. In this example, we show first suggested colors on the background vegetation (top palette), sorted by decreasing likelihood. The suggested colors are common colors for vegetation. We also show the top six suggested colors (bottom palette) of a pixel on the image of the bird. On the right, we show the resulting colorizations, based on the user selecting these top six suggested colors. Photograph of blue-and-yellow macaw by Luc Viatour, 2009.

**Loss Function** The choice of an appropriate loss function $\mathcal{L}$, which measures network performance and guides learning, requires some consideration. Iizuka et al. [40] use an $\ell_2$ loss. As discussed in Chapter 2 and by previous work [22, 39], this loss is not robust to the inherent multi-modal nature of the problem, and instead use a classification loss, followed by a fixed inference step. Another challenge is the large imbalance in natural image statistics, with many more pixels in desaturated regions of the color gamut. This can often lead to desaturated and dull colorizations. In Chapter 2, we use a class-rebalancing step to oversample more colorful portions of the gamut during training. This results in more colorizations which are vibrant and able to fool humans, but at the expense of images which are over-aggressively colorized. In the pix2pix framework, Isola et al. [75] use an $\ell_1$ regression loss with a Generative Adversarial Network (GAN) [91] term, which can help generate exciting, higher frequency patterns.

However, in this chapter, we forgo the use of class rebalancing from Chapter 2 and GAN term from [75] and use a smooth-$\ell_1$ (or Huber) loss, described in Equation 3.4. In the Local Hints Network, from a user experience standpoint, we found it more pleasing to start with a conservative colorization and allow the user to inject desired colors, rather than starting with a more vibrant but artifact-prone setting and having

the user fix mistakes. Much of the multi-modal ambiguity of the problem is quickly resolved by a few user clicks. In cases where there is ambiguity, the smooth-$\ell_1$ is also a robust estimator [92], which can help avoid the averaging problem. In addition, using a regression loss, described in Equation 3.4 with $\delta = 1$, enables us to perform end-to-end learning without a fixed inference step.

$$\ell_\delta(x, y) = \tfrac{1}{2}(x - y)^2 \mathbb{1}_{\left\{|x-y|<\delta\right\}} + \delta(|x - y| - \tfrac{1}{2}\delta)\mathbb{1}_{\left\{|x-y|\geq\delta\right\}} \tag{3.4}$$

The loss function $\ell_\delta$ is evaluated at each pixel and summed together to evaluate the loss $\mathcal{L}$ for a whole image.

$$\mathcal{L}(\mathcal{F}(\mathbf{X}, \mathbf{U}; \theta), \mathbf{Y}) = \sum_{h,w} \sum_{q} \ell_\delta\big(\mathcal{F}(\mathbf{X}, \mathbf{U}; \theta)_{h,w,q}, \mathbf{Y}_{h,w,q}\big) \tag{3.5}$$

Next, we describe the specifics of the local and global variants.

### 3.3.2 Local Hints Network

The Local Hints Network uses sparse user points as input. We describe the input, how we simulate user points, and features of our user interface.

**System Input** The user points are parameterized as $\mathbf{X}_{ab} \in \mathbb{R}^{H\times W\times 2}$, a sparse tensor with $ab$ values for the points provided by the user and $\mathbf{B}_{ab} \in \mathbb{B}^{H\times W\times 1}$, a binary mask indicating which points are provided by the user. The mask differentiates unspecified points from user-specified gray points with $(a, b) = 0$. Together, the tensors form input tensor $\mathbf{U}_l = \{\mathbf{X}_{ab}, \mathbf{B}_{ab}\} \in \mathbb{R}^{H\times W\times 3}$.

**Simulating User Interactions** One challenge in training deep networks is collecting training data. While data for automatic colorization is readily available – any color image can be broken up into its color and grayscale components – an appropriate mechanism for acquiring user interaction data is far less obvious. Gathering this on a large scale is not only expensive, but also comes with a chicken and egg problem, as user interaction behavior will be dependent on the system performance itself. We bypass this issue by training with synthetically generated user interactions. A concern with this approach is the potential domain gap between the generated data and test-time usage. However, we found that even through *randomly sampling*, we are able to cover the input space adequately and train an effective system.

We sample small patches and reveal the average patch color to the network. For each image, the number of points are drawn from a geometric distribution with $p = \frac{1}{8}$. Each point location is sampled from a 2-D Gaussian with $\mu = \frac{1}{2}[H, W]^T, \Sigma =$

$diag\big(\big[\big(\frac{H}{4}\big)^2, \big(\frac{W}{4}\big)^2\big]\big)$, as we expect users to more often click on points in the center of the image. The revealed patch size is drawn uniformly from size $1 \times 1$ to $9 \times 9$, with the average $ab$ within the patch revealed to the network. Lastly, we desire the correct limiting characteristic – given all of the points by the user, the network should simply copy the colors from the input to the output. To encourage this, we provide the full ground truth color to the image for 1% of the training instances. Though the network should implicitly learn to copy any provided user points to the output, there is no explicit constraint for the network to do so exactly. Note that these design decisions for projection function $\mathcal{P}_l(\mathbf{Y})$ were initially selected based on intuition, found to work well, but not finely tuned.

**User interface** Our interface consists of a drawing pad, showing user points overlaid on the grayscale input image, a display updating the colorization result in real-time, a data-driven color palette that suggests likely color for a given location (as shown in Figure 3.3), and a regular $ab$ gamut based on the lightness of the current point. A user is always free to add, move, delete, or change the color of any existing points. Please see our supplemental video for a detailed introduction of our interface, along with several demonstrations.

**Data-driven color palette** Picking a plausible color is an important step towards realistic colorization. Without the proper tools, selecting a color can be difficult for a novice user to intuit. For every pixel, we predict a probability distribution over output colors $\widehat{\mathbf{Z}} \in \mathbb{R}^{H \times W \times Q}$, where $Q$ is the number of quantized color bins. We use the parametrization of the CIE $Lab$ color space from Chapter 2 – the $ab$ space is divided into $10 \times 10$ bins, and the $Q = 313$ bins that are in-gamut are kept. The mapping from the input grayscale image and user points to predicted color distribution $\widehat{\mathbf{Z}}$ is learned with network $\mathcal{G}_l$, parametrized by $\psi_l$. Ground truth distribution $\mathbf{Z}$ is encoded from ground truth colors $\mathbf{Y}$ with the soft-encoding scheme from before – a real $ab$ color value is expressed as a convex combination of its 10 nearest bin centers, weighted by a Gaussian kernel with $\sigma = 5$. We use a cross-entropy loss function for every pixel to measure the distance between predicted and ground truth distributions, and sum over all pixels.

$$\mathcal{L}_{cl}(\mathcal{G}_l(\mathbf{X}, \mathbf{U}_l; \psi_l), \mathbf{Z}) = -\sum_{h,w} \sum_{q} \mathbf{Z}_{h,w,q} \log(\mathcal{G}_l(\mathbf{X}, \mathbf{U}_l; \psi_l)_{h,w,q}) \tag{3.6}$$

Network $\mathcal{G}_l$ is trained to minimize expected classification loss over the training set. We further describe the network architecture in Section 3.3.4.

$$\psi_l^* = \arg\min_{\psi_l} \mathbb{E}_{\mathbf{X},\mathbf{Y} \sim \mathcal{D}}[\mathcal{L}_{cl}(\mathcal{G}_l(\mathbf{X}, \mathbf{U}_l; \psi_l), \mathbf{Y})] \tag{3.7}$$

To provide discrete color suggestions, we soften the softmax distribution at the queried pixel, to make it less peaky, and perform weighted k-means clustering (with $K = 9$) to find modes of the distribution. For example, the system often recommends plausible colors based on the type of object, material and texture, for example, suggesting different shades of green the vegetation in Figure 3.3. For objects with diverse colors such as a parrot, our system will provide a wide range of suggestions. Once a user selects a suggested color, our system will produce the colorization result in real-time. In Figure 3.3, we show six possible colorizations based on the different choices for the parrot's feather. The color suggestions are continuously updated as the user adds additional points.

### 3.3.3 Global Hints Network

An advantage of the end-to-end learning framework is that it may be easily adapted to different types of user inputs. We show an additional use case, where the user provides global statistics, described by a global histogram $\mathbf{X}_{hist} \in \Delta^Q$ and average image saturation $\mathbf{X}_{sat} \in [0, 1]$. Whether or not the inputs are provided is indexed by indicator variables $\mathbf{B}_{hist}, \mathbf{B}_{sat} \in \mathbb{B}$, respectively. The user input to the system is then $\mathbf{U}_g = \{\mathbf{X}_{hist}, \mathbf{B}_{hist}, \mathbf{X}_{sat}, \mathbf{B}_{sat}\} \in \mathbb{R}^{1\times1\times(Q+3)}$.

We compute global histograms by resizing the color $\mathbf{Y}$ to quarter resolution using bilinear interpolation, encoding each pixel in quantized $ab$ space, and averaging spatially. Saturation is computed by converting the ground truth image to HSV colorspace and averaging over the S channel spatially. We randomly reveal the ground truth colorization distribution, ground truth saturation, both, or neither, to the network during training.

### 3.3.4 Network Architecture

We show our network architecture in Figure 6.2. The main colorization branch is used by both Local Hints and Global Hints networks. We then describe the layers which are only used for the Local Hints Network, namely processing the sparse user input $\mathbf{U}_l$ and the color distribution prediction branch, both shown in red. Finally, we describe the Global Hints Network-specific input branch, shown in green, as well as its integration in the main network.

#### Main colorization network

The main branch of our network, $\mathcal{F}$, uses a U-Net architecture [93], which has been shown to work well for a variety of conditional generation tasks [75]. We also utilize

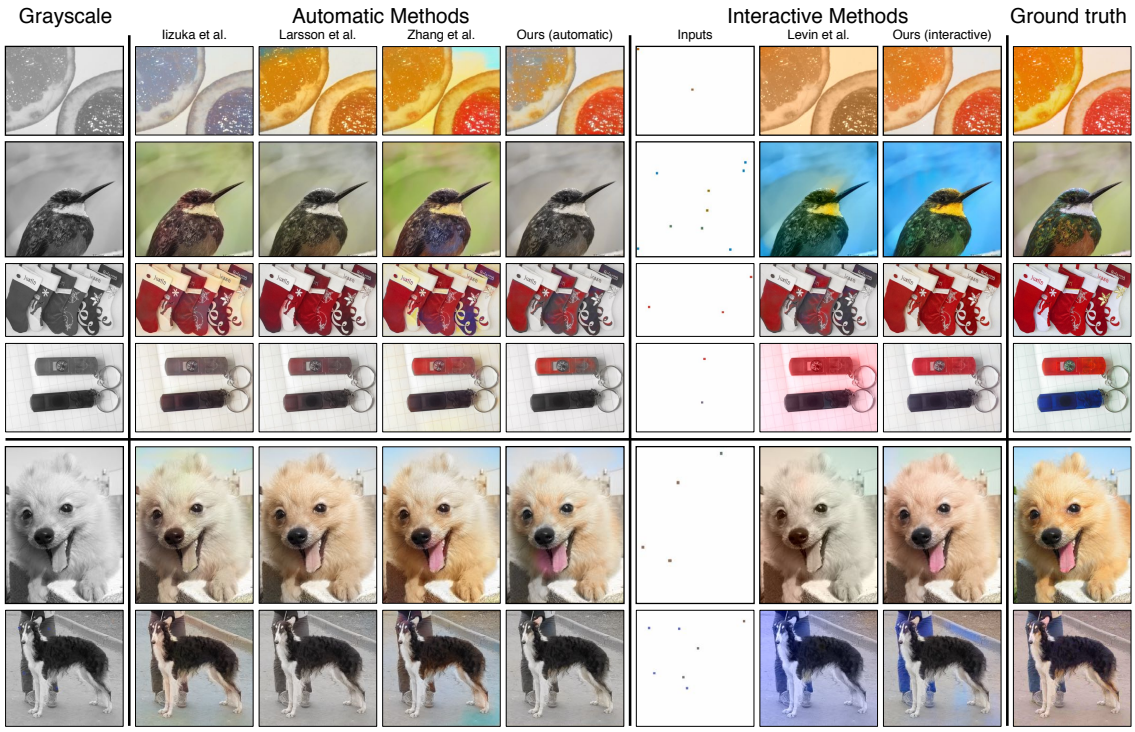



Figure 3.4: **User study results** These results are collected from novice users using our Local Hints Network system for 1 minute for each image, with minimal training. Users were not given the ground truth image, and were instructed to create a "realistic colorization". The first column shows the grayscale input image. Columns 2-5 show automatic results from previous methods, as well as our system without user points. Column 6 shows input points from a user, collected in 1 minute of time from a novice user. Columns 7-8 show the results from the seminal method of [55] and our model, incorporating user points, on the right. The final column shows the ground truth image (which was not provided to the user). In the selected examples in rows 1-4, our system produces higher quality colorizations given sparse inputs than [55], and produce nearly photorealistic results given little user interaction. Rows 5-6 show some failures of our system. In row 5, the green color by the user on the top-right is not successfully propagated to the top-left of the image. In row 6, the colors selected on the jeans are propagated to the background, demonstrating undesired non-local effects. All of the user study results are publicly available on https://richzhang.github.io/ideepcolor/. Images are from the ImageNet dataset [4].

design principles from [24] and [43]. The network is formed by 10 convolutional

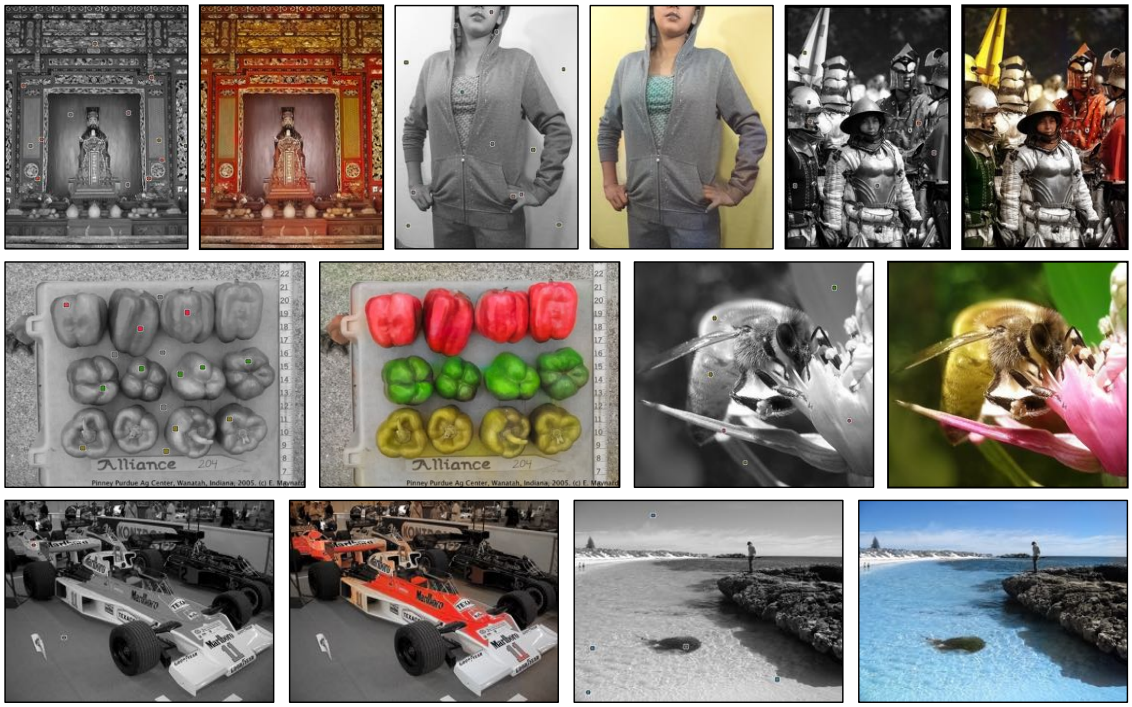


Figure 3.5: **Selected User Study Results** We show grayscale images with user inputs, alongside the output from our algorithm. Each image was colorized in only 1 minute of time by a novice user. All of the user study results are publicly available on https://richzhang.github.io/ideepcolor/. Images are from the Imagenet dataset [4].

blocks, `conv1-10`. In `conv1-4`, in every block, feature tensors are progressively halved spatially, while doubling in the feature dimension. Each block contains 2-3 `conv-relu` pairs. In the second half, `conv7-10`, spatial resolution is recovered, while feature dimensions are halved. In block `conv5-6`, instead of halving the spatial resolution, dilated convolutions with factor 2 is used. This has an equal effect on the receptive field of each unit with respect to the input pixels, but allows the network to keep additional information in the bottleneck. Symmetric shortcut connections are added to help the network recover spatial information [93]. For example, the `conv2` and `conv3` blocks are connected to the `conv8` and `conv9` blocks, respectively. This also enables easy accessibility to important low-level information for later layers; for example, the lightness value will limit the extent of the *ab* gamut. Changes in spatial resolution are achieved using subsampling or upsampling operations, and each convolution uses a $3 \times 3$ kernel. `BatchNorm` layers are added after each convolutional block, which has been shown to help training.

A subset of our network architecture, namely `conv1-8` without the shortcut connections, was used in our automatic colorization work [19]. For these layers, we fine-tune from these pre-trained weights. The added `conv9, conv10` layers and shortcut connections are trained from scratch. A last `conv` layer, which is a $1 \times 1$ kernel, maps between `conv10` and the output color. Because the *ab* gamut is bounded, we add a final `tanh` layer on the output, as is common practice when generating images [89, 91].

**Local Hints Network** The layers specific to the Local Hints Network are shown in red in Figure 6.2. Sparse user points are integrated by concatenation with the input grayscale image. As a side task, we also predict a color *distribution* at each pixel (conditioned on the grayscale and user points) to recommend to the user. The task of predicting a color distribution is undoubtedly related to the task of predicting a single colorization, so we reuse features from the main branch. We use a hypercolumn approach [39, 41] by concatenating features from multiple layers of the main branch, and learning a two-layer classifier on top. Network $\mathcal{G}_l$ is composed of the main branch, up to `conv8`, along with this side branch. The side task should not affect the main task's representation, so we do not back-propagate the gradients from the side task into the main branch. To save computation, we predict the distribution at a quarter resolution, and apply bilinear upsampling to predict at full resolution.

**Global Hints Network** Because the global inputs have no spatial information, we choose to integrate the information into the middle of the main colorization network. As shown in the top green branch in Figure 6.2, the inputs are processed through 4 `conv-relu` layers, with kernel size $1 \times 1$ and 512 channels each. This feature map is repeated spatially to match the size of the `conv4` feature in the main branch, $\mathbb{R}^{H/8 \times W/8 \times 512}$, and merged by summation, a similar strategy to the one used by Iizuka et al. [40].

## 3.4 Experiments

We detail qualitative and quantitative experiments with our system. In Section 3.4.1, we first automatically test the Local Hints Network. We then describe our user study in Section 3.4.2. The results suggest that even with little training and just 1 minute to work with an image, novice users can quickly create realistic colorizations. In Section 3.4.3, we show qualitative examples on unusual colorizations. In Section 3.4.4 we evaluate our Global Hints Network. In Section 3.4.5, we investigate how the

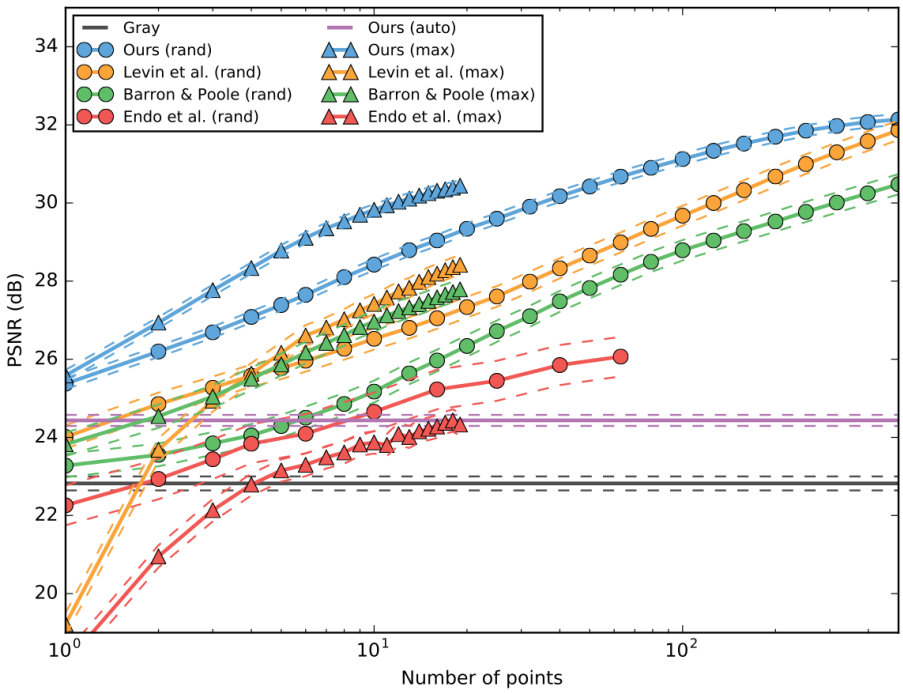



Figure 3.6: **Average PSNR vs Number of Revealed Points** We measure the average PSNR from our ImageNet test set across different algorithms. Points are revealed to each algorithm by **random** or **max**-error sampling. Max-error sampling selects the point with maximum $\ell_2$ error in $ab$ space between predicted and ground truth. Random sampling uses a uniformly drawn random point. The average color on a $7 \times 7$ patch is revealed to the algorithm. The x-axis is on a logarithmic scale. Baselines [55, 64, 94] are computed with publicly available code from the authors. Because our algorithm is learned on a large-scale corpus of data, our system provides more accurate colorizations given little user supervision. With large amounts of input points (approximately 500 for random sampling), [55] begins to achieve equal accuracy to ours. For reference, we show our network without user inputs, **Ours (auto)**, and predicting **Gray** for every pixel.

Local Hints Network reconciles two colors within a single segment. Finally, we show qualitative examples on legacy grayscale images in Section 3.4.6.

### 3.4.1 How well does the system incorporate inputs?

We test the system automatically by randomly revealing patches to the algorithm, and measuring PSNR, as shown in Figure 3.6. The pitfalls of using low-level or per-pixel metrics have been discussed in the automatic colorization regime [19]. A

| Method | Added Inputs | PSNR (dB) |
|---|---|---|
| Predict gray | – | 22.82±0.18 |
| Zhang et al. [19] | automatic | 22.04±0.11 |
| Zhang et al. [19] (no-rebal) | automatic | 24.51±0.15 |
| Larsson et al. [39] | automatic | 24.93±0.14 |
| Iizuka et al. [40] | automatic | 23.69±0.13 |
| Ours (Local) | automatic | 24.43±0.14 |
| Ours (Global) | + global hist | 27.85±0.13 |
| Ours (Global) | + global sat | 25.78±0.15 |
| Ours (Local) | + gt colors | 37.70±0.14 |
| Edit propagation | + gt colors | $\infty$ |

Table 3.1: **PSNR with added information.** Run on 1000 held-out test images, in the ILSVRC2012 [4] validation dataset. **Ours (Local)-automatic** is run completely automatically, with no user inputs. Methods [19, 39, 40] are recently automatic colorization methods. Even though our network is trained primarily for interactive colorization, it performs competitively for automatic colorization as well by this metric. **Ours (Global) +global hist** provides global distribution of colors in the *ab* gamut; **Ours (Global) +global sat** provides global saturation to the system. Our Global Hints Network learns to incorporate global statistics for more accurate colorizations.

system which chooses a plausible but different mode than the ground truth color will be overly penalized, and may even achieve a lower score than an implausible but neutral colorization, such as predicting gray for every pixel (PSNR 22.8). In this context, however, since ground truth colors are revealed to the algorithm, the problem is much more constrained, and PSNR is a more appropriate metric.

With no revealed information, edit propagation methods will default to gray for the whole image. Our system will perform automatic colorization, and provide its best estimate (PSNR 24.4), as described in Table 3.1. As points are revealed, PSNR incrementally increases across all methods. Our method achieves a higher PSNR than other methods, even up to 500 random points. As the number of points increases, edit propagation techniques such as [55] approach our method, and will inevitably surpass it. In the limiting case, where every point is revealed, edit propagation techniques such as [55, 64, 94] will correctly copy the inputs to the outputs (PSNR $\infty$). Our system is taught to do this, based on 1% of the training examples, but will not do so perfectly (PSNR 37.70). As the number of points increases to the

| Method | AMT Fooling Rate |
|---|---|
| Ours-automatic | 18.58% ± 1.09 |
| Ours-no recommendation | 26.98% ± 1.76 |
| Ours | **30.04% ± 1.80** |

Table 3.2: **Amazon Mechanical Turk real vs fake fooling rate** We test how often colorizations generated by novice users fool real humans. **Ours** is our full method, with color recommendations. **Ours-no recommendation** is our method, without the color recommendation system. **Ours-automatic** is our method with no user inputs. Note that the 95% confidence interval shown is not accounting for possible inter-subject variation (all subjects are assumed to be identical).

hundreds, knowledge of mid-to-high-level natural image statistics has diminishing importance, and the problem can be solved using low-level optimization.

We also run the same test, but with points sampled in a more intelligent manner. Given an oracle which provides the ground truth image, we compute the $\ell_2$ error between the current prediction and the ground truth, and average over a $25 \times 25$ window. We then select the point with the maximum error to reveal a $7 \times 7$ patch, excluding points which overlap with previously revealed patches. As expected, this sampling strategy typically achieves a higher PSNR, and the same trend holds – our method achieves higher accuracy than the current state-of-the-art method. Inferring the full colorization of an image given sparsely revealed points has been previously exploited in the image compression literature [95, 96]. An interesting extension of our network would be to optimally choose which points to reveal.

We also note that our method has been designed with point inputs, whereas previous work has been designed with stroke and point-based inputs in mind. In an interactive setting, the collection cost of strokes versus points is difficult to define, and will heavily depend on factors such as proper optimization of the user interface. However, the results strongly suggest that our method is able to accurately propagate sparse, point-based inputs.

### 3.4.2 Does our system aid the user in generating realistic colorizations?

We run a user study, with the goal of evaluating if novice users, given little training, can quickly produce realistic colorizations using our system. We provide minimal training for 28 test subjects, briefly walking them through our interface

for 2 minutes. The subjects are given the goal of producing “realistic colorizations" (without benefit of seeing the ground truth), and are provided 1 minute for each image. Images are randomly drawn from our ImageNet test set. Each subject is given 20 images – 10 images with our algorithm and full interface, including suggested colors, and 10 images with our algorithm but no color suggestions, for a total of 560 images (280 per test setting). We evaluate the resulting colorizations, along with automatic colorization, by running a real vs. fake test on Amazon Mechanical Turk (AMT), using the procedure proposed for automatic colorization [19]. AMT evaluators are shown two images in succession for 1 second each – one ground truth and one synthesized – and asked to identify the synthesized. We measure the “fooling rate" of each algorithm; one which produces ground truth colorizations every time would achieve 50% by this metric. The results are shown in Table 3.2. Note that the results may differ on an absolute scale from previous iterations of this test procedure, such as in the previous chapter or in [75], due to shifts or biases in the AMT population when the algorithm has been tested. Our network produces a fooling rate of 18.6% when run completely automatically (no user inputs). We test our interface without recommended colors, but with HSV sliders and 48 common colors. With this baseline interface, the fooling rate increases dramatically to 27.0%, indicating that users quickly acclimated to our network and made dramatic improvements with just 1 minute. When provided the data-driven color palette, the fooling rate further increases to 30.0%. This suggests that the color prediction feature can aid users in quickly selecting a desired color.

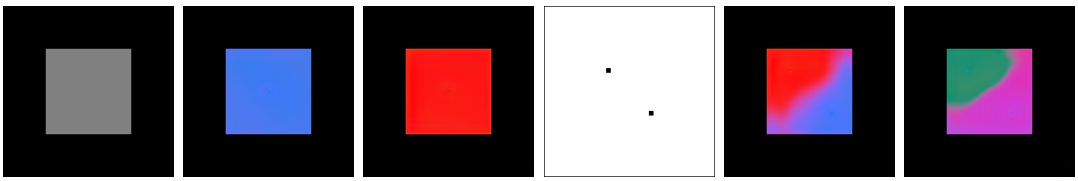
We show example results from our study in Figures 3.4 and 3.5. We compare the annotations to the seminal method proposed by Levin et al. [55], along with the automatic output from our network. Qualitatively, the added user points typically add (1) saturation when the automatic result is lacking and (2) accurate higher frequency detail, that automatic methods have difficulty producing. Comparing our method to Levin et al. [55], our method is more effective at finding segment boundaries given sparse user inputs. We do note that the user points are collected by running our system, which provides an advantage. However, collecting these points, with the right colors, is enabled by the interactive nature of our algorithm and our color recommendation system.

### 3.4.3 Does the network generalize to unusual colorizations?

During training, we use natural images, and reveal the *ground truth colors* to simulate user input. However, there are use cases where the user may intentionally desire an unusual colorization. Will the network be able to follow the inputs in these cases? In Figure 3.7, we show an unusual colorization guided by the user, giving the

(a) (b) (c)

Figure 3.7: **Unusual colorization (a)** User inputs with unusual colors **(b)** Output colorization using user points with unusual colors **(c)** Output colorization with user points using conventional colors. Photograph by Corporal Michael Guinto, 2014 (Public Domain).



(a) (b) (c) (d) (e) (f)

Figure 3.8: **Multiple user colors within a segment. (a)** Input grayscale image. **(b,c)** Output colorization conditioned on a single centered user point colored (b-blue, c-red). **(d)** Locations used for user points for (e) and (f). **(e,f)** Outputs given different user input colors (e-blue&red, f-green&pink).

actor a green face with three user points on the face. These results suggest that in the absence of nearby user inputs, the network will attempt to find an appropriate colorization for the object, based on the training corpus. However, once an input is provided by the user, the system fills in the segment with the desired color.

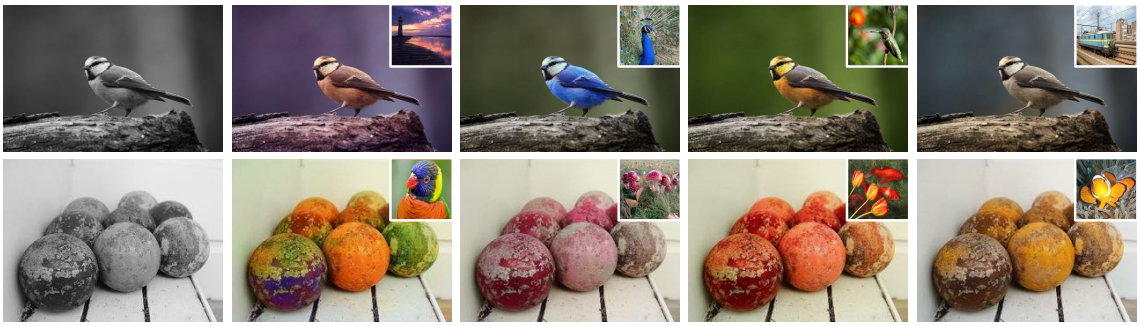


Figure 3.9: **Global histogram transfer** Using our Global Hints Network, we colorize the grayscale version of the image on the left using global histograms from the top-right inset images. Images are from the Imagenet dataset [4].

Figure 3.10: **Legacy black and white photographs** Our method applied to legacy black and white photographs. Top left: *The Tetons and Snake River*, Ansel Adams, 1942; Bottom left: Photo by John Rooney of Muhammad Ali versus Sonny Liston, 1965 (c.f. color photo by Neil Leifer at almost exactly the same moment); Right: *V-J Day in Times Square*, Alfred Eisenstaedt, 1945.

### 3.4.4 Is the system able to incorporate global statistics?

We train a variant of our system, taking global statistics as inputs, instead of local points. As described in Table 3.1, when given the ground truth statistics, such as the global histogram of colors or average saturation, the network achieves a higher PSNR scores, 27.9 and 25.6, respectively, than when performing automatic colorization (24.4), indicating that the network has learned how to fuse global statistics. We also test on the SUN-6 dataset, shown in Table 3.3, proposed by Deshpande et al. [38]. We show higher performance than Despande et al. [38] and almost equal performance

| Method | Added Inputs | PSNR (dB) |
|---|---|---|
| Deshpande et al. [38] | automatic | 23.18 ± 0.20 |
| Larsson et al. [39] | automatic | 25.60 ± 0.23 |
| Ours | automatic | 25.65 ± 0.23 |
| Deshpande et al. [38] | + global hist | 23.85 ± 0.23 |
| Larsson et al. [39] | + global hist | 28.62 ± 0.23 |
| Ours | + global hist | 28.57 ± 0.21 |

Table 3.3: **Global Histogram** We test our Global Hints Network at incorporating the global truth histogram on 240 images from SUN used by [38].

with Larsson et al. [39], which fuses the predictions from an automatic colorization network with a ground truth histogram using an energy minimization procedure.

The network has only been trained on images with its own ground truth histogram. In Figure 3.9, we qualitatively the network's generalization ability by computing the global histogram on separate reference images, and apply them to a photograph. The bird is an interesting test case, as it can be plausibly colorized in many different ways. We observe that that the color distributions of the reference input image is successfully transferred to the target grayscale image. Furthermore, the colorizations are realistic and diverse.

### 3.4.5 How does the system respond to multiple colors within an equiluminant segment?

In natural images, chrominance changes almost never appear without a lightness change. In Figure 3.8(a), we show a toy example of an image of a gray square on top of a black square. If given a $7 \times 7$ point in the center of the image, the system will successfully propagate the color to the center region, as shown in Figures 3.8(a)(b). However, how does the system respond if given two different colors within the same segment, as shown in Figure 3.8(d)? Given blue and red points, the system draws a seam between the two colors, as shown in Figure 3.8(e), where two points are placed symmetrically around the center of the image. Because our system is learned from data, it is difficult to characterize how the system will exactly behave in such a scenario. Qualitatively, we observe that the seam is not straight, and the shape as well as the sharpness of the transition is dependent on the colors. For example, in Figure 3.8(f), green and pink points produce a harder seam. We found similar behavior under similar scenarios in natural images as well.

### 3.4.6 Is the system able to colorize legacy photographs?

Our system was trained on “synthetic" grayscale images by removing the chrominance channels from color images. We qualitatively test our system on *legacy* grayscale images, and show some selected results in Figure 3.10.

## 3.5 Limitations and Discussion

A benefit of our system is that the network predicts user-intended actions based on learned semantic similarities. However, the network can also be over-optimistic and produce undesired non-local effects. For example, points added on a foreground object may cause an undesired change in the background, as shown on the last row in Figure 3.4. Qualitatively, we found that adding some control points can remedy this. In addition, the network can also fail to completely propagate a user point, as shown in the fifth row in Figure 3.4. In these instances, the user can fill in the region with additional input.

For scenes with difficult segmentation boundaries, the user sometimes needs to define boundaries explicitly by densely marking either side. Our system can continuously incorporate this information, even with hundreds of input points, as shown on Figure 3.6. Points can be added to fix color bleeding artifacts when the system has poor underlying segmentation. However, our interface is mainly designed for the “few seconds to couple minutes" interaction regime. For users wanting high-precision control and willing to spend hours per photograph, working in Photoshop is likely a better solution.

Our system is currently trained on points; we find that in this regime random sampling covers the low-dimensional workspace surprisingly well. However, a future step is to better simulate the user, and to effectively incorporate stroke-based inputs that traditional methods utilize. Integration between the local user points and global statistics inputs would be an interesting next step. Our interface code and models are publicly available at https://richzhang.github.io/ideepcolor, along with all images generated from the user study and random global histogram transfer results.

So far, in the last two chapters, we have investigated the specific problem of image colorization. In Chapter 2, we proposed a system for automatic colorization, where multimodality was resolved by predicting a distribution of possible colors for each pixel. In this chapter, the multimodality was resolved with the benefit of a user. The colorization problem can be seen as a special instance of *image-to-image translation*. Next, in Chapter 4, we propose a system for the general image-to-image translation problem, where multimodality is resolved with a learned, low-dimensional latent code.

# Chapter 4

# Toward Multimodal Image-to-Image Translation

Many image-to-image translation problems are ambiguous, as a single input image may correspond to multiple possible outputs. One example is the grayscale-to-color translation problem, which we explored in Chapters 2 and 3. In this chapter, we explore the *general* image-to-image translation problem. We aim to model a *distribution* of possible outputs in a conditional generative modeling setting. The ambiguity of the mapping is distilled into a low-dimensional latent vector, which can be randomly sampled at test time. A generator learns to map the given input, combined with this latent code, to the output. We explicitly encourage the connection between output and the latent code to be invertible. This helps prevent a many-to-one mapping from the latent code to the output during training, also known as the problem of mode collapse, and produces more diverse results. We explore several variants of this approach by employing different training objectives, network architectures, and methods of injecting the latent code. Our proposed method encourages bijective consistency between the latent encoding and output modes. We present a systematic comparison of our method and other variants on both perceptual realism and diversity.[1]

## 4.1 Motivation

Deep learning techniques have made rapid progress in conditional image generation. For example, networks have been used to inpaint missing image re-

[1]This work was originally published as *Toward Multimodal Image-to-Image Translation*. In NIPS, 2017 [97].

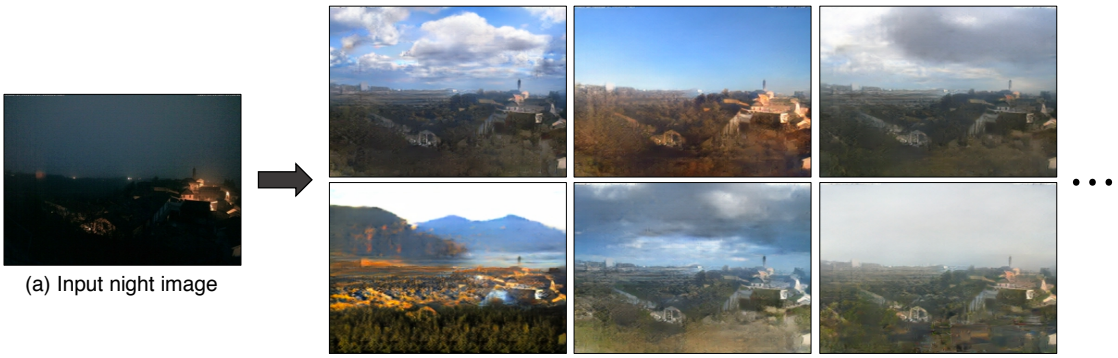



Figure 4.1: Multimodal image-to-image translation using our proposed method: given an input image from one domain (night image of a scene), we aim to model a *distribution* of potential outputs in the target domain (corresponding day images), producing both realistic and diverse results.

gions [29, 75, 98], add color to grayscale images [19, 39, 40, 75], and generate photorealistic images from sketches [68, 75]. However, most techniques in this space have focused on generating a *single* result.

In this work, we model a *distribution* of potential results, as many of these problems may be multimodal in nature. For example, as seen in Figure 4.1, an image captured at night may look very different in the day, depending on cloud patterns and lighting conditions. We pursue two main goals: producing results which are (1) perceptually realistic and (2) diverse, all while remaining faithful to the input.

Mapping from a high-dimensional input to a high-dimensional output distribution is challenging. A common approach to representing multimodality is learning a low-dimensional latent code, which should represent aspects of the possible outputs not contained in the input image. At inference time, a deterministic generator uses the input image, along with stochastically sampled latent codes, to produce randomly sampled outputs. A common problem in existing methods is *mode collapse* [99], where only a small number of real samples get represented in the output. We systematically study a family of solutions to this problem.

We start with the `pix2pix` framework [75], which has previously been shown to produce high-quality results for various image-to-image translation tasks. The method trains a generator network, conditioned on the input image, with two losses: (1) a regression loss to produce similar output to the known paired ground truth image and (2) a learned discriminator loss to encourage realism. The authors note that trivially appending a randomly drawn latent code did not produce diverse results. Instead, we propose encouraging a bijection between the output and latent

space. We not only perform the direct task of mapping the latent code (along with the input) to the output but also jointly learn an encoder from the output back to the latent space. This discourages two different latent codes from generating the same output (non-injective mapping). During training, the learned encoder attempts to pass enough information to the generator to resolve any ambiguities regarding the output mode. For example, when generating a day image from a night image, the latent vector may encode information about the sky color, lighting effects on the ground, and cloud patterns. Composing the encoder and generator sequentially should result in the same image being recovered. The opposite should produce the same latent code.

In this work, we instantiate this idea by exploring several objective functions, inspired by literature in unconditional generative modeling:

- `cVAE-GAN` **(Conditional Variational Autoencoder GAN)**: One approach is first encoding the ground truth image into the latent space, giving the generator a noisy "peek" into the desired output. Using this, along with the input image, the generator should be able to reconstruct the specific output image. To ensure that random sampling can be used during inference time, the latent distribution is regularized using KL-divergence to be close to a standard normal distribution. This approach has been popularized in the unconditional setting by VAEs [100] and VAE-GANs [101].

- `cLR-GAN` **(Conditional Latent Regressor GAN)**: Another approach is to first provide a randomly drawn latent vector to the generator. In this case, the produced output may not necessarily look like the ground truth image, but it should look realistic. An encoder then attempts to recover the latent vector from the output image. This method could be seen as a conditional formulation of the "latent regressor" model [102, 103] and also related to InfoGAN [104].

- `BicycleGAN`: Finally, we combine both these approaches to enforce the connection between latent encoding and output in both directions *jointly* and achieve improved performance. We show that our method can produce both diverse and visually appealing results across a wide range of image-to-image translation problems, significantly more diverse than other baselines, including naively adding noise in the `pix2pix` framework. In addition to the loss function, we study the performance with respect to several encoder networks, as well as different ways of injecting the latent code into the generator network.

We perform a systematic evaluation of these variants by using humans to judge photorealism and a perceptual distance metric [105] to assess output diversity. Code and data are available at https://github.com/junyanz/BicycleGAN.

## 4.2 Background

**Generative modeling** Parametric modeling of the natural image distribution is a challenging problem. Classically, this problem has been tackled using restricted Boltzmann machines [106] and autoencoders [14, 107]. Variational autoencoders [100] provide an effective approach for modeling stochasticity within the network by reparametrization of a latent distribution at training time. A different approach is autoregressive models [108–110], which are effective at modeling natural image statistics but are slow at inference time due to their sequential predictive nature. Generative adversarial networks [91] overcome this issue by mapping random values from an easy-to-sample distribution (e.g., a low-dimensional Gaussian) to output images in a single feedforward pass of a network. During training, the samples are judged using a discriminator network, which distinguishes between samples from the target distribution and the generator network. GANs have recently been very successful [89, 102–104, 111–116]. Our method builds on the conditional version of VAE [100] and InfoGAN [104] or latent regressor [102, 103] models by jointly optimizing their objectives. We revisit this connection in Section 4.3.4.

**Conditional image generation** All of the methods defined above can be easily conditioned. While conditional VAEs [117] and autoregressive models [109, 110] have shown promise [118–120], image-to-image conditional GANs have lead to a substantial boost in the quality of the results. However, the quality has been attained at the expense of multimodality, as the generator learns to largely ignore the random noise vector when conditioned on a relevant context [29, 68, 75, 98, 121, 122]. In fact, it has even been shown that ignoring the noise leads to more stable training [29, 75, 123].

**Explicitly-encoded multimodality** One way to express multiple modes is to explicitly encode them, and provide them as an additional input in addition to the input image. For example, color and shape scribbles and other interfaces were used as conditioning in iGAN [89], pix2pix [75], Scribbler [68] and interactive colorization [54]. An effective option explored by concurrent work [124–126] is to use a mixture of models. Though able to produce multiple discrete answers, these methods are unable to produce continuous changes. While there has been some degree of success for generating multimodal outputs in unconditional and text-conditional setups [91, 101, 113, 127, 128], conditional image-to-image generation is still far from achieving the same results, unless explicitly encoded as discussed above. In this work, we learn conditional image generation models for modeling multiple modes of output by enforcing tight connections between the latent and image spaces.

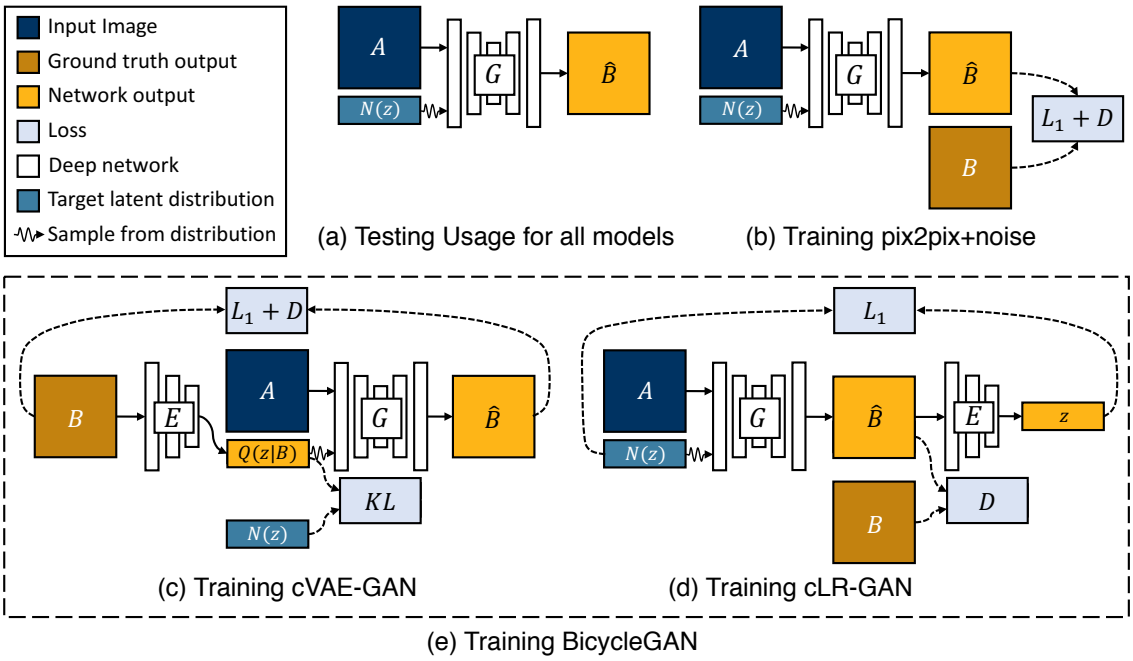



Figure 4.2: **Overview**: (a) Test time usage of all the methods. To produce a sample output, a latent code $\mathbf{z}$ is first randomly sampled from a known distribution (e.g., a standard normal distribution). A generator $G$ maps an input image $\mathbf{A}$ (blue) and the latent sample $\mathbf{z}$ to produce a output sample $\hat{\mathbf{B}}$ (yellow). (b) `pix2pix+noise` [75] baseline, with an additional ground truth image $\mathbf{B}$ (brown) that corresponds to $\mathbf{A}$. (c) `cVAE-GAN` (and `cAE-GAN`) starts from a ground truth target image $\mathbf{B}$ and encode it into the latent space. The generator then attempts to map the input image $\mathbf{A}$ along with a sampled $\mathbf{z}$ back into the original image $\mathbf{B}$. (d) `cLR-GAN` randomly samples a latent code from a known distribution, uses it to map $\mathbf{A}$ into the output $\hat{\mathbf{B}}$, and then tries to reconstruct the latent code from the output. (e) Our hybrid `BicycleGAN` method combines constraints in both directions.

## 4.3 Approach

Our goal is to learn a multi-modal mapping between two image domains, for example, edges and photographs, or night and day images, etc. Consider the input domain $\mathcal{A} \subset \mathbb{R}^{H \times W \times 3}$, which is to be mapped to an output domain $\mathcal{B} \subset \mathbb{R}^{H \times W \times 3}$. During training, we are given a dataset of paired instances from these domains, $\{(\mathbf{A} \in \mathcal{A}, \mathbf{B} \in \mathcal{B})\}$, which is representative of a joint distribution $p(\mathbf{A}, \mathbf{B})$. It is important to note that there could be multiple plausible paired instances $\mathbf{B}$ that would correspond to an input instance $\mathbf{A}$, but the training dataset usually contains only one such pair. However, given a new instance $\mathbf{A}$ during test time, our model should be able to generate a diverse set of output $\widehat{\mathbf{B}}$'s, corresponding to different modes in the distribution $p(\mathbf{B}|\mathbf{A})$.

While conditional GANs have achieved success in image-to-image translation tasks [29, 68, 75, 98, 121, 122], they are primarily limited to generating a deterministic output $\widehat{\mathbf{B}}$ given the input image $\mathbf{A}$. On the other hand, we would like to learn the mapping that could sample the output $\widehat{\mathbf{B}}$ from true conditional distribution given $\mathbf{A}$, and produce results which are both diverse and realistic. To do so, we learn a low-dimensional latent space $\mathbf{z} \in \mathbb{R}^Z$, which encapsulates the ambiguous aspects of the output mode which are not present in the input image. For example, a sketch of a shoe could map to a variety of colors and textures, which could get compressed in this latent code. We then learn a deterministic mapping $G : (\mathbf{A}, \mathbf{z}) \rightarrow \mathbf{B}$ to the output. To enable stochastic sampling, we desire the latent code vector $\mathbf{z}$ to be drawn from some prior distribution $p(\mathbf{z})$; we use a standard Gaussian distribution $\mathcal{N}(0, I)$ in this work.

We first discuss a simple extension of existing methods and discuss its strengths and weakness, motivating the development of our proposed approach in the subsequent subsections.

### 4.3.1 Baseline: `pix2pix+noise` ($\mathbf{z} \rightarrow \widehat{\mathbf{B}}$)

The recently proposed `pix2pix` model [75] has shown high quality results in the image-to-image translation setting. It uses conditional adversarial networks [91, 129] to help produce perceptually realistic results. GANs train a generator $G$ and discriminator $D$ by formulating their objective as an adversarial game. The discriminator attempts to differentiate between real images from the dataset and fake samples produced by the generator. Randomly drawn noise $\mathbf{z}$ is added to attempt to induce stochasticity. We illustrate the formulation in Figure 5.2(b) and describe it

below.

$$\mathcal{L}_{\text{GAN}}(G, D) = \mathbb{E}_{\mathbf{A},\mathbf{B}\sim p(\mathbf{A},\mathbf{B})}[\log(D(\mathbf{A}, \mathbf{B}))] + \mathbb{E}_{\mathbf{A}\sim p(\mathbf{A}),\mathbf{z}\sim p(\mathbf{z})}[\log(1 - D(\mathbf{A}, G(\mathbf{A}, \mathbf{z})))] \tag{4.1}$$

To encourage the output of the generator to match the input as well as stabilize the training, we use an $\ell_1$ loss between the output and the ground truth image.

$$\mathcal{L}_1^{\text{image}}(G) = \mathbb{E}_{\mathbf{A},\mathbf{B}\sim p(\mathbf{A},\mathbf{B}),\mathbf{z}\sim p(\mathbf{z})}||\mathbf{B} - G(\mathbf{A}, \mathbf{z})||_1 \tag{4.2}$$

The final loss function uses the GAN and $\ell_1$ terms, balanced by $\lambda$.

$$G^* = \arg\min_G \max_D \quad \mathcal{L}_{\text{GAN}}(G, D) + \lambda\mathcal{L}_1^{\text{image}}(G) \tag{4.3}$$

In this scenario, there is little incentive for the generator to make use of the noise vector which encodes random information. Isola et al. [75] note that the noise was ignored by the generator in preliminary experiments and was removed from the final experiments. This was consistent with observations made in the conditional settings by [29, 123], as well as the mode collapse phenomenon observed in unconditional cases [99, 130]. In this chapter, we explore different ways to explicitly enforce the latent coding to capture relevant information.

### 4.3.2 Conditional Variational Autoencoder GAN: cVAE-GAN ($\mathbf{B} \to \mathbf{z} \to \widehat{\mathbf{B}}$)

One way to force the latent code $\mathbf{z}$ to be "useful" is to directly map the ground truth $\mathbf{B}$ to it using an encoding function $E$. The generator $G$ then uses both the latent code and the input image $\mathbf{A}$ to synthesize the desired output $\widehat{\mathbf{B}}$. The overall model can be easily understood as the reconstruction of $\mathbf{B}$, with latent encoding $\mathbf{z}$ concatenated with the paired $\mathbf{A}$ in the middle – similar to an autoencoder [14]. This interpretation is better shown in Figure 5.2(c).

This approach has been successfully investigated in Variational Autoencoder [100] in the unconditional scenario without the adversarial objective. Extending it to conditional scenario, the distribution $Q(\mathbf{z}|\mathbf{B})$ of latent code $\mathbf{z}$ using the encoder $E$ with a Gaussian assumption, $Q(\mathbf{z}|\mathbf{B}) \triangleq E(\mathbf{B})$. To reflect this, Equation 4.1 is modified to sampling $\mathbf{z} \sim E(\mathbf{B})$ using the re-parameterization trick, allowing direct back-propagation [100].

$$\mathcal{L}_{\text{GAN}}^{\text{VAE}} = \mathbb{E}_{\mathbf{A},\mathbf{B}\sim p(\mathbf{A},\mathbf{B})}[\log(D(\mathbf{A}, \mathbf{B}))] + \mathbb{E}_{\mathbf{A},\mathbf{B}\sim p(\mathbf{A},\mathbf{B}),\mathbf{z}\sim E(\mathbf{B})}[\log(1 - D(\mathbf{A}, G(\mathbf{A}, \mathbf{z})))] \tag{4.4}$$

We make the corresponding change in the $\ell_1$ loss term in Equation 4.2 as well to obtain $\mathcal{L}_1^{\text{VAE}}(G) = \mathbb{E}_{\mathbf{A},\mathbf{B}\sim p(\mathbf{A},\mathbf{B}),\mathbf{z}\sim E(\mathbf{B})}||\mathbf{B}-G(\mathbf{A}, \mathbf{z})||_1$. Further, the latent distribution

encoded by $E(B)$ is encouraged to be close to a random Gaussian to enable sampling at inference time, when $\mathbf{B}$ is not known.

$$\mathcal{L}_{\text{KL}}(E) = \mathbb{E}_{\mathbf{B}\sim p(\mathbf{B})}[\mathcal{D}_{\text{KL}}(E(\mathbf{B})||\,\mathcal{N}(0, I))], \tag{4.5}$$

where $\mathcal{D}_{\text{KL}}(p||q) = -\int p(z)\log\frac{p(z)}{q(z)}dz$. This forms our `cVAE-GAN` objective, a conditional version of the VAE-GAN [101] as

$$G^*, E^* = \arg\min_{G,E}\max_{D} \quad \mathcal{L}^{\text{VAE}}_{\text{GAN}}(G, D, E) + \lambda\mathcal{L}^{\text{VAE}}_{1}(G, E) + \lambda_{\text{KL}}\mathcal{L}_{\text{KL}}(E). \tag{4.6}$$

As a baseline, we also consider the deterministic version of this approach, i.e., dropping KL-divergence and encoding $\mathbf{z} = E(\mathbf{B})$. We call it `cAE-GAN` and show a comparison in the experiments. There is no guarantee in `cAE-GAN` on the distribution of the latent space $\mathbf{z}$, which makes the test-time sampling of $\mathbf{z}$ difficult.

### 4.3.3 Conditional Latent Regressor GAN: `cLR-GAN` ($\mathbf{z} \to \widehat{\mathbf{B}} \to \widehat{\mathbf{z}}$)

We explore another method of enforcing the generator network to utilize the latent code embedding $\mathbf{z}$, while staying close to the actual test time distribution $p(\mathbf{z})$, but from the latent code's perspective. As shown in Figure 5.2(d), we start from a randomly drawn latent code $\mathbf{z}$ and attempt to recover it with $\widehat{\mathbf{z}} = E(G(\mathbf{A}, \mathbf{z}))$. Note that the encoder $E$ here is producing a point estimate for $\widehat{\mathbf{z}}$, whereas the encoder in the previous section was predicting a Gaussian distribution.

$$\mathcal{L}^{\text{latent}}_{1}(G, E) = \mathbb{E}_{\mathbf{A}\sim p(\mathbf{A}),\mathbf{z}\sim p(\mathbf{z})}||\mathbf{z} - E(G(\mathbf{A}, \mathbf{z}))||_1 \tag{4.7}$$

We also include the discriminator loss $L_{\text{GAN}}(G, D)$ (Equation 4.1) on $\widehat{\mathbf{B}}$ to encourage the network to generate realistic results, and the full loss can be written as:

$$G^*, E^* = \arg\min_{G,E}\max_{D} \quad \mathcal{L}_{\text{GAN}}(G, D) + \lambda_{\text{latent}}\mathcal{L}^{\text{latent}}_{1}(G, E) \tag{4.8}$$

The $\ell_1$ loss for the ground truth image $\mathbf{B}$ is not used. Since the noise vector is randomly drawn, the predicted $\widehat{\mathbf{B}}$ does not necessarily need to be close to the ground truth but does need to be realistic. The above objective bears similarity to the "latent regressor" model [102–104], where the generated sample $\widehat{\mathbf{B}}$ is encoded to generate a latent vector.

### 4.3.4 Our Hybrid Model: `BicycleGAN`

We combine the `cVAE-GAN` and `cLR-GAN` objectives in a hybrid model. For `cVAE-GAN`, the encoding is learned from real data, but a random latent code may not yield realistic images at test time – the KL loss may not be well optimized. Perhaps more importantly, the adversarial classifier $D$ does not have a chance to see results sampled from the prior during training. In `cLR-GAN`, the latent space is easily sampled from a simple distribution, but the generator is trained without the benefit of seeing ground truth input-output pairs. We propose to train with constraints in both directions, aiming to take advantage of both cycles ($\mathbf{B} \rightarrow \mathbf{z} \rightarrow \widehat{\mathbf{B}}$ and $\mathbf{z} \rightarrow \widehat{\mathbf{B}} \rightarrow \widehat{\mathbf{z}}$), hence the name `BicycleGAN`.

$$\begin{aligned} G^*, E^* = \arg \min_{G,E} \max_{D} \quad & \mathcal{L}_{\text{GAN}}^{\text{VAE}}(G, D, E) + \lambda \mathcal{L}_1^{\text{VAE}}(G, E) \\ & + \mathcal{L}_{\text{GAN}}(G, D) + \lambda_{\text{latent}} \mathcal{L}_1^{\text{latent}}(G, E) + \lambda_{\text{KL}} \mathcal{L}_{\text{KL}}(E), \end{aligned} \tag{4.9}$$

where the hyper-parameters $\lambda$, $\lambda_{\text{latent}}$, and $\lambda_{\text{KL}}$ control the relative importance of each term.

In the unconditional GAN setting, Larsen et al. [101] observe that using samples from both the prior $\mathcal{N}(0, I)$ and encoded $E(\mathbf{B})$ distributions further improves results. Hence, we also report one variant which is the full objective shown above (Equation 4.9), but without the reconstruction loss on the latent space $\mathcal{L}_1^{\text{latent}}$. We call it `cVAE-GAN++`, as it is based on `cVAE-GAN` with an additional loss $\mathcal{L}_{\text{GAN}}(G, D)$, which allows the discriminator to see randomly drawn samples from the prior.

### 4.3.5 Implementation Details

The code and additional results are publicly available at https://github.com/junyanz/BicycleGAN. Please refer to our website for more details about the datasets, architectures, and training procedures.

**Network architecture** For generator $G$, we use the U-Net [93], which contains an encoder-decoder architecture, with symmetric skip connections. The architecture has been shown to produce strong results in the unimodal image prediction setting when there is a spatial correspondence between input and output pairs. For discriminator $D$, we use two PatchGAN discriminators [75] at different scales, which aims to predict real vs. fake for $70 \times 70$ and $140 \times 140$ overlapping image patches. For the encoder $E$, we experiment with two networks: (1) $\texttt{E}_{\texttt{CNN}}$: CNN with a few convolutional and downsampling layers and (2) $\texttt{E}_{\texttt{ResNet}}$: a classifier with several residual blocks [131].

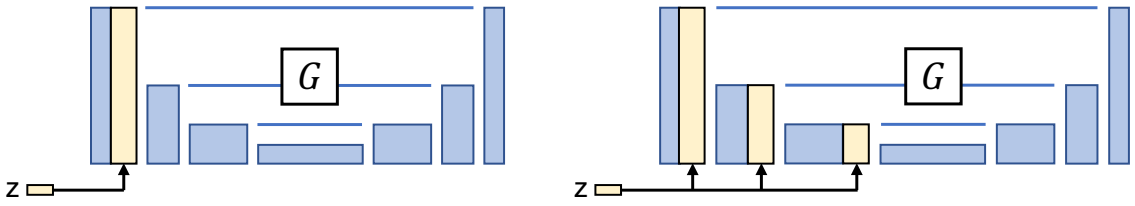



Figure 4.3: **Alternatives for injecting z into generator.** Latent code **z** is injected by spatial replication and concatenation into the generator network. We tried two alternatives, **(left)** injecting into the input layer and **(right)** every intermediate layer in the encoder.

**Training details** We build our model on the Least Squares GANs (LSGANs) variant [132], which uses a least-squares objective instead of a cross entropy loss. LSGANs produce high-quality results with stable training. We also find that not conditioning the discriminator $D$ on input **A** leads to better results (also discussed in [29]), and hence choose to do the same for all methods. We set the parameters $\lambda_{\text{image}} = 10$, $\lambda_{\text{latent}} = 0.5$ and $\lambda_{\text{KL}} = 0.01$ in all our experiments. We tie the weights for the generators and encoders in the `cVAE-GAN` and `cLR-GAN` models. For the encoder, only the predicted mean is used in `cLR-GAN`. We observe that using two separate discriminators yields slightly better visual results compared to sharing weights. We only update $G$ for the $\ell_1$ loss $\mathcal{L}_1^{\text{latent}}(G, E)$ on the latent code (Equation 4.7), while keeping $E$ fixed. We found optimizing $G$ and $E$ simultaneously for the loss would encourage $G$ and $E$ to hide the information of the latent code without learning meaningful modes. We train our networks from scratch using Adam [133] with a batch size of 1 and with a learning rate of 0.0002. We choose latent dimension $|\mathbf{z}| = 8$ across all the datasets.

**Injecting the latent code z to generator**. We explore two ways of propagating the latent code **z** to the output, as shown in Figure 4.3: (1) `add_to_input`: we spatially replicate a $Z$-dimensional latent code **z** to an $H \times W \times Z$ tensor and concatenate it with the $H \times W \times 3$ input image and (2) `add_to_all`: we add **z** to each intermediate layer of the network $G$, after spatial replication to the appropriate sizes.

## 4.4 Experiments

**Datasets** We test our method on several image-to-image translation problems from prior work, including edges → photos [89, 134], Google maps → satellite [75], labels → images [135], and outdoor night → day images [136]. These problems are all one-to-many mappings. We train all the models on $256 \times 256$ images.

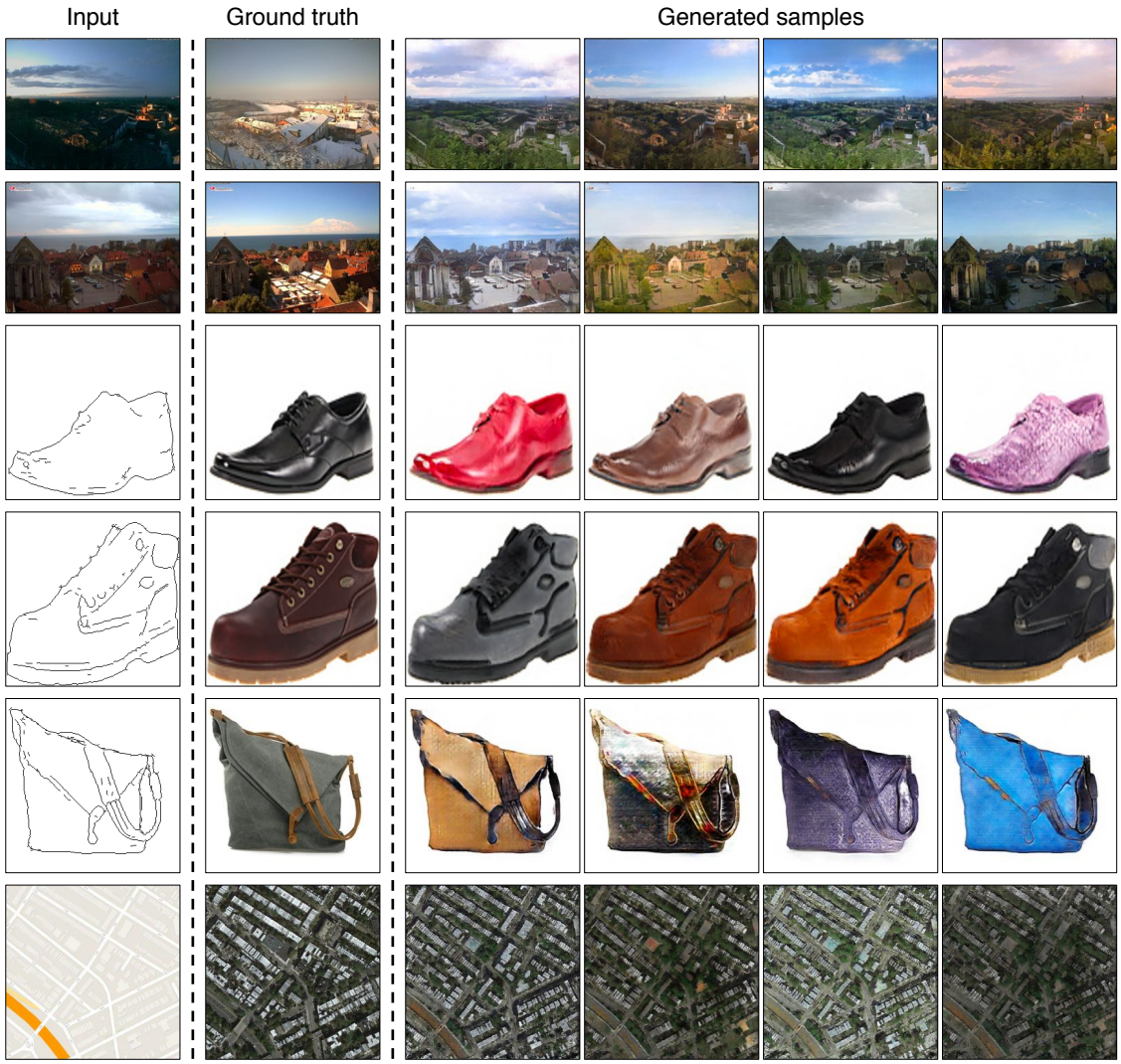



Figure 4.4: **Example Results** We show example results of our hybrid model `BicycleGAN`. The left column shows the input. The second shows the ground truth output. The final four columns show randomly generated samples. We show results of our method on night→day, edges→shoes, edges→handbags, and maps→satellites. Models and additional examples are available at https://junyanz.github.io/BicycleGAN.

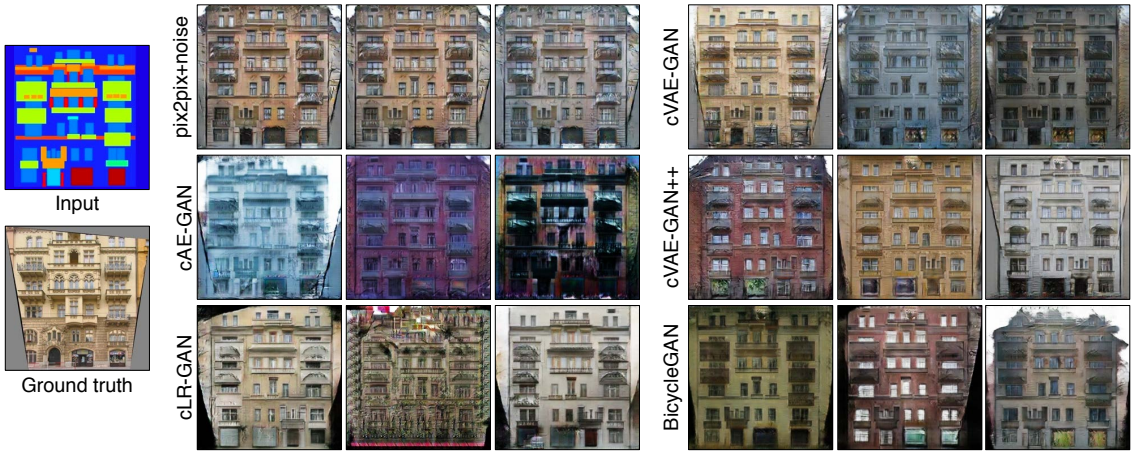



Figure 4.5: **Qualitative method comparison** We compare results on the labels → facades dataset across different methods. The `BicycleGAN` method produces results which are both realistic and diverse.

| Method | Realism: AMT Fooling Rate [%] | Diversity: LPIPS Distance |
|---|---|---|
| Random real images | 50.0% | .262 |
| `pix2pix+noise` [75] | 27.93±2.40 % | .013 |
| `cAE-GAN` | 13.64±1.80 % | .204 |
| `cVAE-GAN` | 24.93±2.27 % | .096 |
| `cVAE-GAN++` | 29.19±2.43 % | .098 |
| `cLR-GAN` | 29.23±2.48 % | *.090 |
| `BicycleGAN` | 34.33±2.69 % | .110 |

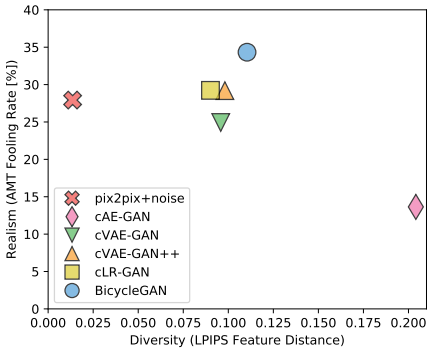



Figure 4.6: **Realism vs Diversity**. We measure diversity using average LPIPS distance [105], and realism using a real vs. fake Amazon Mechanical Turk test on the Google maps → satellites task. The LPIPS metric corresponds well to low-level human perceptual judgments, and we describe it in greater detail in Chapter 6. The `pix2pix+noise` baseline produces little diversity. Using only `cAE-GAN` method produces large artifacts during sampling. The hybrid `BicycleGAN` method, which combines `cVAE-GAN` and `cLR-GAN`, produces results which have higher realism while maintaining diversity. *We found that `cLR-GAN` resulted in severe mode collapse, resulting in ∼ 15% of the images producing the same result. Those images were omitted from this calculation.

**Methods** We evaluate the following models described in Section 6.2: `pix2pix+noise`, `cAE-GAN`, `cVAE-GAN`, `cVAE-GAN++`, `cLR-GAN`, and our hybrid model `BicycleGAN`.

### 4.4.1 Qualitative Evaluation

We show qualitative comparison results on Figure 4.5. We observe that `pix2pix+noise` typically produces a single realistic output, but does not produce any meaningful variation. `cAE-GAN` adds variation to the output, but typically at a large cost to result quality. An example on facades is shown on Figure 4.4.

We observe more variation in the `cVAE-GAN`, as the latent space is encouraged to encode information about ground truth outputs. However, the space is not densely populated, so drawing random samples may cause artifacts in the output. The `cLR-GAN` shows less variation in the output, and sometimes suffers from mode collapse. When combining these methods, however, in the hybrid method `BicycleGAN`, we observe results which are both diverse and realistic. Please see our website for a full set of results.

### 4.4.2 Quantitative Evaluation

We perform a quantitative analysis of the diversity, realism, and latent space distribution on our six variants and baselines. We quantitatively test the Google maps $\rightarrow$ satellites dataset.

**Diversity** We compute the average distance of random samples in deep feature space. Pretrained networks have been used as a "perceptual loss" in image generation applications [84, 137, 138], as well as a held-out "validation" score in generative modeling, for example, assessing the semantic quality and diversity of a generative model [130] or the semantic accuracy of a grayscale colorization [19].

In Figure 4.6, we show the diversity-score using the LPIPS metric proposed by [105][2]. For each method, we compute the average distance between 1900 pairs of randomly generated output $\widehat{\mathbf{B}}$ images (sampled from 100 input $\mathbf{A}$ images). Random pairs of ground truth real images in the $\mathbf{B} \in \mathcal{B}$ domain produce an average variation of .265. As we are measuring samples $\widehat{\mathbf{B}}$ which correspond to a specific input $\mathbf{A}$, a system which stays faithful to the input should definitely not exceed this score.

The `pix2pix` system [75] produces a single point estimate. Adding noise to the system `pix2pix+noise` produces a small diversity score, confirming the finding in [75] that adding noise does not produce large variation. Using the `cAE-GAN` model

[2]Learned Perceptual Image Patch Similarity (LPIPS) metric computes distance in AlexNet [139] feature space (`conv1-5`, pretrained on Imagenet [4]), with linear weights to better match human perceptual judgments.

| Encoder | $\texttt{E}_{\texttt{ResNet}}$ | $\texttt{E}_{\texttt{ResNet}}$ | $\texttt{E}_{\texttt{CNN}}$ | $\texttt{E}_{\texttt{CNN}}$ |
|---|---|---|---|---|
| Injecting $\mathbf{z}$ | `add_to_all` | `add_to_input` | `add_to_all` | `add_to_input` |
| label→photo | $0.292 \pm 0.058$ | $0.292 \pm 0.054$ | $0.326 \pm 0.066$ | $0.339 \pm 0.069$ |
| map → satellite | $0.268 \pm 0.070$ | $0.266 \pm 0.068$ | $0.287 \pm 0.067$ | $0.272 \pm 0.069$ |

Table 4.1: The encoding performance with respect to the different encoder architectures and methods of injecting $\mathbf{z}$. Here we report the reconstruction loss $||\mathbf{B} - G(\mathbf{A}, E(B))||_1$

.

to encode a ground truth image $\mathbf{B}$ into a latent code $\mathbf{z}$ does increase the variation. The `cVAE-GAN`, `cVAE-GAN++`, and `BicycleGAN` models all place explicit constraints on the latent space, and the `cLR-GAN` model places an implicit constraint through sampling. These four methods all produce similar diversity scores. We note that high diversity scores may also indicate that unnatural images are being generated, causing meaningless variations. Next, we investigate the visual realism of our samples.

**Perceptual Realism** To judge the visual realism of our results, we use human judgments, as proposed in [19] and later used in [75, 122]. The test sequentially presents a real and generated image to a human for 1 second each, in a random order, asks them to identify the fake, and measures the "fooling" rate. Figure 4.6(left) shows the realism across methods. The `pix2pix+noise` model achieves high realism score, but without large diversity, as discussed in the previous section. The `cAE-GAN` helps produce diversity, but this comes at a large cost to the visual realism. Because the distribution of the learned latent space is unclear, random samples may be from unpopulated regions of the space. Adding the KL-divergence loss in the latent space, used in the `cVAE-GAN` model, recovers the visual realism. Furthermore, as expected, checking randomly drawn $\mathbf{z}$ vectors in the `cVAE-GAN++` model slightly increases realism. The `cLR-GAN`, which draws $\mathbf{z}$ vectors from the predefined distribution randomly, produces similar realism and diversity scores. However, the `cLR-GAN` model resulted in large mode collapse - approximately 15% of the outputs produced the same result, independent of the input image. The full hybrid `BicycleGAN` gets the best of both worlds, as it does not suffer from mode collapse and also has the highest realism score by a significant margin.

**Encoder architecture** In `pix2pix`, [75] conduct extensive ablation studies on discriminators and generators. Here we focus on the performance of two encoder architectures, $\texttt{E}_{\texttt{CNN}}$ and $\texttt{E}_{\texttt{ResNet}}$, for our applications on the maps and facades datasets. We find that $\texttt{E}_{\texttt{ResNet}}$ better encodes the output image, regarding the image reconstruction loss $||\mathbf{B} - G(\mathbf{A}, E(B))||_1$ on validation datasets as shown in Table 4.1. We use $\texttt{E}_{\texttt{ResNet}}$ in our final model.


Figure 4.7: Different label → facades results trained with varying length of the latent code $|\mathbf{z}| \in \{2, 8, 256\}$.

**Methods of injecting latent code** We evaluate two ways of injecting latent code $\mathbf{z}$: `add_to_input` and `add_to_all` (Section 4.3.5), regarding the same reconstruction loss $||\mathbf{B} - G(\mathbf{A}, E(B))||_1$. Table 4.1 shows that two methods give similar performance. This indicates that the `U_Net` [93] can already propagate the information well to the output without the additional skip connections from $\mathbf{z}$. We use `add_to_all` method to inject noise in our final model.

**Latent code length** We study the `BicycleGAN` model results with respect to the varying number of dimensions of latent codes $\{2, 8, 256\}$ in Figure 4.7. A very low-dimensional latent code may limit the amount of diversity that can be expressed. On the contrary, a very high-dimensional latent code can potentially encode more information about an output image, at the cost of making sampling difficult. The optimal length of $\mathbf{z}$ largely depends on individual datasets and applications, and how much ambiguity there is in the output.

## 4.5 Discussion

In conclusion, we have evaluated a few methods for combating the problem of mode collapse in the conditional image generation setting. We find that by combining multiple objectives for encouraging a bijective mapping between the latent and output spaces, we obtain results which are more realistic and diverse. We see many interesting avenues of future work, including directly enforcing a distribution in the latent space that encodes semantically meaningful attributes to allow for image-to-image transformations with user controllable parameters.

# Part II

# Self-Supervised Visual Representation Learning

# Chapter 5

# Representation Learning by Cross-Channel Prediction

The colorization network from Chapter 2 is a type of *cross-channel encoder*, as it performs the difficult task of predicting one subset of the data channels (color) from another (grayscale). While solving the task, the network learns to capture patterns in the visual world, using only raw data as supervision. By forcing the network to solve cross-channel prediction tasks, we induce a representation within the network which transfers well to other, unseen tasks. In this chapter, we further investigate how well colorization, and cross-channel encoding in general, serves as a pretext task for inducing abstract representations. One weakness of a cross-channel encoder is that the network can only extract a representation from one subset of the data. For example, the colorization network is colorblind. We show that combining multiple cross-channel encoders into a "split-brain autoencoder" can mitigate this problem and lead to further improvements. The method can also be thought of as a straightforward modification of the traditional autoencoder architecture. At time of publication[1], this method achieved state-of-the-art performance on several large-scale transfer learning benchmarks.

## 5.1 Motivation

A goal of unsupervised learning is to model raw data without the use of labels, in a manner which produces a useful representation. By "useful" we mean a representation

[1]This work was first published as *Split-Brain Autoencoders: Unsupervised Learning by Cross-Channel Prediction* in CVPR, 2017. This chapter will sometimes refer to the colorization method from Chapter 2 as Zhang et al. [19]

that should be easily adaptable for other tasks, unknown during training time. Unsupervised deep methods typically induce representations by training a network to solve an auxiliary or "pretext" task, such as the image *reconstruction* objective in a traditional autoencoder model, as shown on Figure 5.1(top). We instead force the network to solve complementary *prediction* tasks by adding a split in the architecture, shown in Figure 5.1 (bottom), dramatically improving transfer performance.

Despite their popularity, autoencoders have actually not been shown to produce strong representations for transfer tasks in practice [29, 107]. Why is this? One reason might be the mechanism for forcing model abstraction. To prevent a trivial identity mapping from being learned, a bottleneck is typically built into the autoencoder representation. However, an inherent tension is at play: the smaller the bottleneck, the greater the forced abstraction, but the smaller the information content that can be expressed.

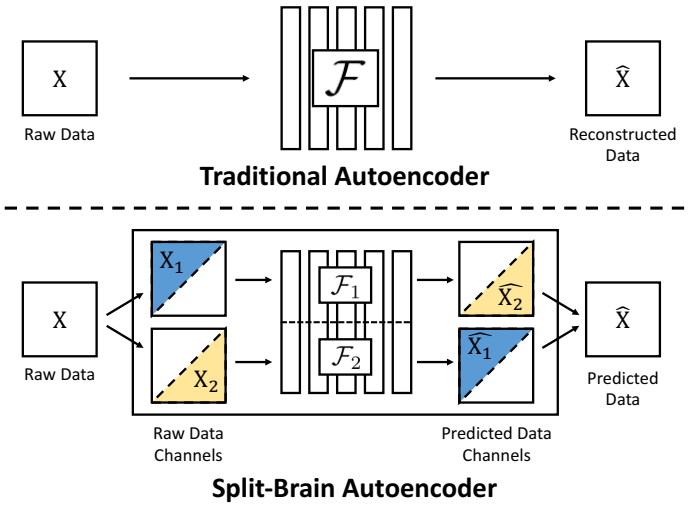



Figure 5.1: **Traditional vs Split-Brain Autoencoder architectures. (top)** Autoencoders learn feature representation $\mathcal{F}$ by learning to reconstruct input data $\mathbf{X}$. **(bottom)** The proposed split-brain autoencoder is composed of two disjoint sub-networks $\mathcal{F}_1, \mathcal{F}_2$, each trained to predict one data subset from another, changing the problem from reconstruction to prediction. The split-brain representation $\mathcal{F}$ is formed by concatenating the two sub-networks, and achieves strong transfer learning performance. The model is publicly available on https://richzhang.github.io/splitbrainauto.

Instead of forcing abstraction through compression, via a bottleneck in the network architecture, recent work has explored withholding parts of the input during training [19, 29, 107]. For example, Vincent et al. [107] propose denoising autoencoders, trained to remove *iid* noise added to the input. Pathak et al. [29] propose *context encoders*, which learn features by training to inpaint large, random contiguous blocks of pixels. Rather than dropping data in the spatial direction, several works have dropped data in the channel direction, e.g. predicting color channels from grayscale (the colorization task) [19, 39].

Context encoders, while an improvement over autoencoders, demonstrate lower

performance than competitors on large-scale semantic representation learning benchmarks [19]. This may be due to several reasons. First, image synthesis tasks are known to be notoriously difficult to evaluate [23] and the loss function used in [29] may not properly capture inpainting quality. Second, the model is trained on images with missing chunks, but applied, at test time, to full images. This causes a "domain gap" between training and deployment. Third, it could simply be that the inpainting task in [29] could be adequately solved without high-level reasoning, instead mostly just copying low and mid-level structure from the surround.

On the other hand, colorization turns out to be a surprisingly effective pretext task for inducing strong feature representations [19, 140]. Though colorization, like inpainting, is a synthesis task, the spatial correspondence between the input and output pairs may enable basic off-the-shelf loss functions to be effective. In addition, the *systematic*, rather than *stochastic* nature of the input corruption removes the pre-training and testing domain gap. Finally, while inpainting may admit reasoning mainly about textural structure, predicting accurate color, e.g., knowing to paint a schoolbus yellow, may more strictly require object-level reasoning and therefore induce stronger semantic representations. Colorization is an example of what we refer to as a *cross-channel encoding* objective, a task which directly predicts one subset of data channels from another.

In this chapter, we further explore the space of cross-channel encoders by systematically evaluating various channel translation problems and training objectives. Cross-channel encoders, however, face an inherent handicap: different channels of the input data are not treated equally, as part of the data is used for feature extraction and another as the prediction target. In the case of colorization, the network can only extract features from the grayscale image and is blind to color, leaving the color information unused. A qualitative comparison of the different methods, along with their inherent strengths and weaknesses, is summarized in Table 5.1.

Might there be a way to take advantage of the underlying principle of cross-channel encoders, while being able to extract features from the entire input signal? We propose an architectural modification to the autoencoder paradigm: adding a single split in the network, resulting in two disjoint, concatenated, sub-networks. Each sub-network is trained as a cross-channel encoder, predicting one subset of channels of the input from the other. A variety of auxiliary cross-channel prediction tasks may be used, such as colorization and depth prediction. For example, on RGB images, one sub-network can solve the problem of colorization (predicting $a$ and $b$ channels from the $L$ channel in $Lab$ colorspace), and the other can perform the opposite (synthesizing $L$ from $a, b$ channels). In the RGB-D domain, one sub-network may predict depth from images, while the other predicts images from depth. The architectural change induces the same forced abstraction as observed in cross-channel

| | auxiliary task type | domain gap | input handicap |
|---|---|---|---|
| Autoencoder [14] | reconstruction | no | no |
| Denoising autoencoder [107] | reconstruction | suffers | no |
| Context Encoder [29] | prediction | no | suffers |
| **Cross-Channel Encoder** [19, 140] | prediction | no | suffers |
| **Split-Brain Autoencoder** | **prediction** | **no** | **no** |

Table 5.1: **Qualitative Comparison** We summarize various qualitative aspects inherent in several representation learning techniques. **Auxiliary task type:** pretext task predicated on reconstruction or prediction. **Domain gap:** gap between the input data during unsupervised pre-training and testing time. **Input handicap:** input data is systematically dropped out during test time.

encoders, but is able to extract features from the full input tensor, leaving nothing on the table.

Our contributions are as follows:

- We propose the split-brain autoencoder, which is composed of concatenated cross-channel encoders, trained using *raw data* as its own supervisory signal.
- We demonstrate state-of-the-art performance on several semantic representation learning benchmarks in the RGB and RGB-D domains.
- To gain a better understanding, we perform extensive ablation studies by (i) investigating cross-channel prediction problems and loss functions and (ii) researching alternative aggregation methods for combining cross-channel encoders.

## 5.2 Background

Many unsupervised learning methods have focused on modeling raw data using a reconstruction objective. Autoencoders [14] train a network to reconstruct an input image, using a representation bottleneck to force abstraction. Denoising autoencoders [107] train a network to undo a random *iid* corruption. Techniques for modeling the probability distribution of images in deep frameworks have also been explored. For example, variational autoencoders (VAEs) [100] employ a variational Bayesian approach to modeling the data distribution. Other probabilistic models include restricted Boltzmann machines (RBMs) [106], deep Boltzmann machines (DBMs) [141], generative adversarial networks (GANs) [91], autoregressive models (Pixel-RNN [109] and Pixel-CNN [110]), bidirectional GANs (BiGANs) [102] and

Adversarially Learned Inference (ALI) [103], and real NVP [128]. Many of these methods [14, 102, 103, 107, 141] have been evaluated for representation learning.

Another form of unsupervised learning, sometimes referred to as "self-supervised" learning [12], has recently grown in popularity. Rather than predicting labels annotated by humans, these methods predict pseudo-labels computed from the raw data itself. For example, image colorization [19, 39] has been shown to be an effective pretext task. Other methods generate pseudo-labels from egomotion [27, 28], video [142, 143], inpainting [29], co-occurence [144], context [53, 145], and sound [12, 32, 146]. Concurrently, Pathak et al. [147] use motion masks extracted from video data. Concurrently, Larsson et al. [140] provide an in-depth analysis of colorization for self-supervision. These methods generally focus on a single supervisory signal and involve some engineering effort. In this work, we show that simply predicting raw data channels with standard loss functions is surprisingly effective, often outperforming previously proposed methods.

The idea of learning representations from multisensory signals also shows up in structure learning [148], co-training [149], and multi-view learning [150]. Our method is especially related to [12, 146, 151], which use bidirectional data prediction to learn representations from two sensory modalities.

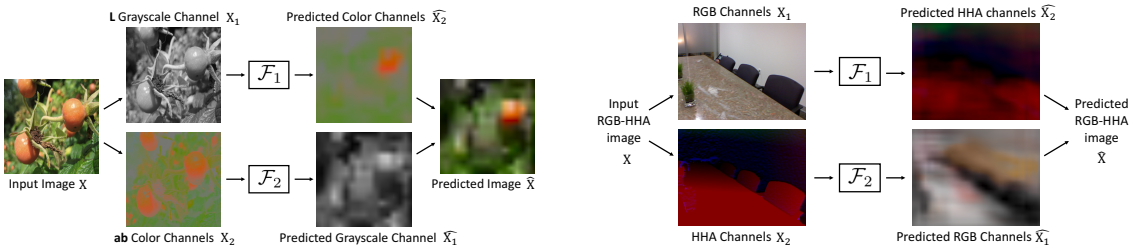
A large body of additional work in computer vision and graphics focuses on image channel prediction as an end in itself, such as colorization [19, 39, 40], depth prediction [152], and surface normal prediction [152, 153]. In contrast, rather than focusing on the graphics problem, we explore its utility for representation learning.

## 5.3 Approach

In Section 5.3.1, we define the paradigm of cross-channel encoding. In Section 5.3.2, we propose the split-brain autoencoder and explore alternatives methods for aggregating multiple cross-channel encoders into a single network.

### 5.3.1 Cross-Channel Encoders

We would like to learn a deep representation on input data tensor $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$, with $\mathbf{C}$ channels. We split the data into $\mathbf{X_1} \in \mathbb{R}^{H \times W \times C_1}$ and $\mathbf{X_2} \in \mathbb{R}^{H \times W \times C_2}$, where $C_1, C_2 \subseteq C$, and then train a deep representation to solve the prediction problem $\widehat{\mathbf{X_2}} = \mathcal{F}(\mathbf{X_1})$. Function $\mathcal{F}$ is learned with a CNN, which produces a layered representation of input $\mathbf{X_1}$, and we refer to each layer $l$ as $\mathcal{F}^l$. By performing this *pretext* task of predicting $\mathbf{X_2}$ from $\mathbf{X_1}$, we hope to achieve a representation $\mathcal{F}(\mathbf{X}_1)$ which contains high-level abstractions or semantics.




(a) ***Lab* Images** (b) **RGB-D Images**

Figure 5.2: **Split-Brain Autoencoders applied to various domains** (a) ***Lab* images** Input images are divided into the $L$ channel, which contains grayscale information, and the $a$ and $b$ channels, which contain color information. Network $\mathcal{F}_1$ performs automatic colorization, whereas network $\mathcal{F}_2$ performs grayscale prediction. (b) **RGB-D images** Input data $\mathbf{X}$ contains registered RGB and depth images. Depth images are encoded using the HHA encoding [154]. Image representation $\mathcal{F}_1$ is trained by predicting HHA channels. Representation $\mathcal{F}_2$ on HHA images is learned by predicting images in *Lab* space. Note that the goal of performing these synthesis tasks is to induce representations $\mathcal{F}_1, \mathcal{F}_2$ that transfer well to other tasks.

This prediction task can be trained with various loss functions, and we study whether the loss function affects the quality of the learned representation. To begin, we explore the use of $\ell_2$ regression, as shown in Equation 5.1.

$$\ell_2(\mathcal{F}(\mathbf{X_1}), \mathbf{X_2}) = \frac{1}{2} \sum_{h,w} \|\mathbf{X_2}_{h,w} - \mathcal{F}(\mathbf{X_1})_{h,w}\|_2^2 \tag{5.1}$$

We also study the use of a classification loss. Here, the target output $\mathbf{X_2} \in \mathbb{R}^{H \times W \times C_2}$ is encoded with function $\mathcal{H}$ into a *distribution* $\mathbf{Y_2} \in \mathbf{\Delta}^{H \times W \times Q}$, where $Q$ is the number of elements in the quantized output space. Network $\mathcal{F}$ is then trained to predict a distribution, $\widehat{\mathbf{Y_2}} = \mathcal{F}(\mathbf{X_1}) \in \mathbf{\Delta}^{H \times W \times Q}$. A standard cross-entropy loss between the predicted and ground truth distributions is used, as shown Equation 5.2.

$$\ell_{cl}(\mathcal{F}(\mathbf{X_1}), \mathbf{X_2}) = -\sum_{h,w} \sum_{q} \mathcal{H}(\mathbf{X_2})_{h,w,q} \log(\mathcal{F}(\mathbf{X_1})_{h,w,q}) \tag{5.2}$$

In Chapter 2, we discovered that the classification loss is more effective for the graphics task of automatic colorization than regression. We hypothesize that for some tasks, especially those with inherent uncertainty in the prediction, the classification loss may lead to better representations as well, as the network will be incentivized to match the whole distribution, and not only predict the first moment.

Note that with input and output sets $C_1, C_2 = C$, and an $\ell_2$ regression loss, the objective becomes identical to the autoencoder objective.

### 5.3.2 Split-Brain Autoencoders as Aggregated Cross-Channel Encoders

We can train multiple cross-channel encoders, $\mathcal{F}_1$, $\mathcal{F}_2$, on opposite prediction problems, with loss functions $L_1, L_2$, respectively, described in Equation 5.3.

$$\begin{aligned} \mathcal{F}_1^* &= \arg\min_{\mathcal{F}_1} L_1(\mathcal{F}_1(\mathbf{X_1}), \mathbf{X_2}) \\ \mathcal{F}_2^* &= \arg\min_{\mathcal{F}_2} L_2(\mathcal{F}_2(\mathbf{X_2}), \mathbf{X_1}) \end{aligned} \tag{5.3}$$

By concatenating the representations layer-wise, $\mathcal{F}^l = \{\mathcal{F}_1^l, \mathcal{F}_2^l\}$, we achieve a representation $\mathcal{F}$ which is pre-trained on full input tensor $\mathbf{X}$. Example split-brain autoencoders in the image and RGB-D domains are shown in Figures 5.2(a) and (b), respectively. If $\mathcal{F}$ is a CNN of a desired fixed size, e.g., AlexNet [77], we can design the sub-networks $\mathcal{F}_1, \mathcal{F}_2$ by splitting each layer of the network $\mathcal{F}$ in half, along the channel dimension. Concatenated representation $\mathcal{F}$ will then have the appropriate dimensionality, and can be simply implemented by setting the `group` parameter to 2 in most deep learning libraries. As each channel in the representation is only connected to half of the channels in the preceding layer, the number of parameters in the network is actually halved, relative to a full network.

Note that the input and the output to the network $\mathcal{F}$ is the full input $\mathbf{X}$, the same as an autoencoder. However, due to the split nature of the architecture, the network $\mathcal{F}$ is trained to *predict* $\mathbf{X} = \{\mathbf{X_1}, \mathbf{X_2}\}$, rather than simply *reconstruct* it from the input. In essence, an architectural change in the autoencoder framework induces the same forced abstraction achieved by cross-channel encoding.

**Alternative Aggregation Technique** We found the split-brain autoencoder, which aggregates cross-channel encoders through concatenation, to be more effective than several alternative strategies. As a baseline, we also explore an alternative: the same representation $\mathcal{F}$ can be trained to perform both mappings simultaneously. The loss function is described in Equation 5.4, with a slight abuse of notation: here, we redefine $\mathbf{X_1}$ to be the same shape as original input $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$, with channels in set $C \backslash C_1$ zeroed out (along with the analogous modification to $\mathbf{X_2}$).

$$\mathcal{F}^* = \arg\min_{\mathcal{F}} L_1(\mathcal{F}(\mathbf{X_1}), \mathbf{X_2}) + L_2(\mathbf{X_1}, \mathcal{F}(\mathbf{X_2})) \tag{5.4}$$

The network only sees data subsets but never full input $\mathbf{X}$. To alleviate this problem,

we mix in the autoencoder objective, as shown in Equation 5.5, with $\lambda \in [0, \frac{1}{2}]$.

$$\begin{aligned}\mathcal{F}^* = \arg\min_{\mathcal{F}} \lambda L_1(\mathcal{F}(\mathbf{X_1}), \mathbf{X_2}) + \lambda L_2(\mathcal{F}(\mathbf{X_2}), \mathbf{X_1}) \\ + (1 - 2\lambda) L_3(\mathbf{X}, \mathcal{F}(\mathbf{X}))\end{aligned} \tag{5.5}$$

Note that unlike the split-brain architecture, in these objectives, there is a domain gap between the distribution of pre-training data and the full input tensor $\mathbf{X}$.

## 5.4 Implementation Details

Here, we describe the pre-training and feature evaluation architectures. For pre-training, we use an AlexNet architecture [77], trained fully convolutionally [155]. The network is trained with 180×180 images, cropped from 256 × 256 resolution, and predicts values at a heavily downsampled 12×12 resolution. One can add upsampling-convolutional layers or use *a trous* [42]/dilated [43] convolutions to

| **Fully Convolutional AlexNet [77] Architecture** | | | | | | |
|---|---|---|---|---|---|---|
| **Layer** | **X** | **C** | **K** | **S** | **D** | **P** |
| **data** | 180 | * | – | – | – | – |
| **conv1** | 45 | 96 | 11 | 4 | 1 | 5 |
| **pool1** | 23 | 96 | 3 | 2 | 1 | 1 |
| **conv2** | 23 | 256 | 5 | 1 | 1 | 2 |
| **pool2** | 12 | 256 | 3 | 2 | 1 | 1 |
| **conv3** | 12 | 384 | 3 | 1 | 1 | 1 |
| **conv4** | 12 | 384 | 3 | 1 | 1 | 1 |
| **conv5** | 12 | 256 | 3 | 1 | 1 | 1 |
| **pool5** | 12 | 256 | 3 | 1 | 1 | 1 |
| **fc6** | 12 | 4096 | 6 | 1 | 2 | 6 |
| **fc7** | 12 | 4096 | 1 | 1 | 1 | 0 |
| **fc8** | 12 | * | 1 | 1 | 1 | 0 |

Table 5.2: **Fully Convolutional AlexNet architecture used for pre-training**. **X** spatial resolution of layer, **C** number of channels in layer; **K** `conv` or `pool` kernel size; **S** computation stride; **D** kernel dilation [42, 43]; **P** padding; * first and last layer channel sizes are dependent on the pre-text task, last layer is removed during transfer evaluation.

predict full resolution images at the expense of additional memory and run-time, but we found predicting at a lower resolution to be sufficient for representation learning. See Table 5.2 for feature map and parameter sizes during pre-training time. We remove `LRN` layers and add `BatchNorm` layers after every convolution layer. After pre-training, we remove `BatchNorm` layers by absorbing the parameters into the preceding `conv` layers. The pre-training network predicts a downsampled version of the desired output, which we found to be adequate for feature learning.

During feature evaluation time (such as the ImageNet [77], Places [44], and PASCAL [7] tests), the parameters are copied into an AlexNet classification architecture, shown in Table 5.3. During the linear classification tests, we downsample feature maps spatially, so that each layer has approximately the same number of features.

**Quantization procedure** Zhang et al. [19] use a class-rebalancing term, to oversample rare colors in the training set, and a soft-encoding scheme for $\mathcal{H}$. These choices were made from a graphics perspective, to produce more vibrant colorizations. In our classification colorization network, **L→ab(cl)**, our objective is more straightforward, as we do not use class-rebalancing. In addition, we use a 1-hot encoding representation

| **AlexNet Classification [77] Architecture** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Layer** | **X** | $\mathbf{X}_d$ | **C** | $\mathbf{F}_d$ | **K** | **S** | **D** | **P** |
| **data** | 227 | – | * | – | – | – | – | – |
| **conv1** | 55 | 10 | 96 | 9600 | 11 | 4 | 1 | 0 |
| **pool1** | 27 | 10 | 96 | 9600 | 3 | 2 | 1 | 0 |
| **conv2** | 27 | 6 | 256 | 9216 | 5 | 1 | 1 | 2 |
| **pool2** | 13 | 6 | 256 | 9216 | 3 | 2 | 1 | 0 |
| **conv3** | 13 | 5 | 384 | 9600 | 3 | 1 | 1 | 1 |
| **conv4** | 13 | 5 | 384 | 9600 | 3 | 1 | 1 | 1 |
| **conv5** | 13 | 6 | 256 | 9216 | 3 | 1 | 1 | 1 |
| **pool5** | 6 | 6 | 256 | 9216 | 3 | 2 | 1 | 0 |
| **fc6** | 1 | – | 4096 | – | 6 | 1 | 1 | 0 |
| **fc7** | 1 | – | 4096 | – | 1 | 1 | 1 | 0 |

Table 5.3: **AlexNet architecture used for feature evaluation**. **X** spatial resolution of layer, $\mathbf{X}_d$ downsampled spatial resolution for feature evaluation, **C** number of channels in layer; $\mathbf{F}_d = \mathbf{X}_d^2\mathbf{C}$ downsampled feature map size for feature evaluation (kept approximately constant throughout), **K** `conv` or `pool` kernel size; **S** computation stride; **D** kernel dilation [42, 43]; **P** padding; * first layer channel size is dependent on the pre-text task e.g., 3 for the split-brain autoencoder or 1 for the $L \rightarrow ab(cl)$ cross-channel encoder

of classes, rather than soft-encoding. The simplification in the objective function achieves higher performance on ImageNet and Places classification, as shown on Tables 5.4 and 5.5.

## 5.5 Experiments

In Section 5.5.1, we apply our proposed split-brain autoencoder architecture to learn unsupervised representations on large-scale image data from ImageNet [4]. We evaluate on established representation learning benchmarks and demonstrate state-of-the-art performance relative to previous unsupervised methods [29, 32, 51, 53, 102, 142, 143]. In Section 5.5.2, we apply the proposed method on the NYU-D dataset [5], and show performance above baseline methods.

### 5.5.1 Split-Brain Autoencoders on Images

We work with image data $\mathbf{X}$ in the *Lab* color space, and learn cross-channel encoders with $\mathbf{X}_1$ representing the $L$, or lightness channel, and $\mathbf{X}_2$ containing the $ab$ channels, or color information. This is a natural choice as (i) networks such as Alexnet, trained with grouping in their architecture, naturally separate into grayscale and color [77] even in a fully-supervised setting, and (ii) the individual cross-channel prediction problem of colorization, $L$ to $ab$, has produced strong representations [19, 39]. In preliminary experiments, we have also explored different cross-channel prediction problems in other color spaces, such as *RGB* and *YUV*. We found the $L$ and $ab$ to be most effective data split.

To enable comparisons to previous unsupervised techniques, all of our trained networks use AlexNet architectures [77]. Concurrent work from Larsson et al. [140] shows large performance improvements for the colorization task when using deeper networks, such as VGG-16 [24] and ResNet [156]. Because we are training for a pixel-prediction task, we run the network fully convolutionally [155]. Using the 1.3M ImageNet dataset [4] (without labels), we train the following aggregated cross-channel encoders:

- **Split-Brain Autoencoder (cl,cl) (Our full method)**: A split-brain autoencoder, with one half performing colorization, and the other half performing grayscale prediction. The top-level architecture is shown in Figure 5.2(a). Both sub-networks are trained for classification (cl), with a cross-entropy objective. (In Figure 5.2(a), the predicted output is a per-pixel probability distribution, but is visualized with a point estimate using the annealed-mean [19].)

- **Split-Brain Autoencoder (reg,reg)**: Same as above, with both sub-networks trained with an $\ell_2$ loss (reg).
- **Ensembled L→ab**: Two concatenated disjoint sub-networks, both performing colorization (predicting $ab$ from $L$). One subnetwork is trained with a classification objective, and the other with regression.
- **(L,ab)→(ab,L)**: A single network for both colorization and grayscale prediction, with regression loss, as described in Equation 5.4. This explores an alternative method for combining cross-channel encoders.
- **(L,ab,Lab)→(ab,L,Lab)**: $\lambda = \frac{1}{3}$ using Equation 5.5.

Single cross-channel encoders are ablations of our main method. We systematically study combinations of loss functions and cross-channel prediction problems.

- **L→ab(reg)**: Automatic colorization using an $\ell_2$ loss.
- **L→ab(cl)**: Automatic colorization using a classification loss. We follow the quantization procedure proposed in [19]: the output $ab$ space is binned into grid size $10 \times 10$, with a classification loss over the 313 bins that are within the $ab$ gamut.
- **ab→L(reg)**: Grayscale prediction using an $\ell_2$ loss.
- **ab→L(cl)**: Grayscale prediction using a classification loss. The $L$ channel, which has values between 0 and 100, is quantized into 50 bins of size 2 and encoded.
- **Lab→Lab**: Autoencoder objective, reconstructing $Lab$ from itself using an $\ell_2$ regression loss, with the same architecture as the cross-channel encoders.
- **Lab(drop50)→Lab**: Same as above, with 50% of the input randomly dropped out during pre-training. This is similar to denoising autoencoders [107].

We compare to the following methods, which all use variants of Alexnet [77]. For additional details, refer to Table 3 in [19]. Note that one of these modifications resulted in a large deviation in feature map size[2].

- **ImageNet-labels** [77]: Trained on ImageNet labels for the classification task in a fully supervised fashion.
- **Gaussian**: Random Gaussian initialization of weights.

[2]The method from [145] uses stride 2 instead of 4 in the `conv1` layer, resulting in 4× denser feature maps throughout all convolutional layers. While it is unclear how this change affects representational quality, experiments from Larsson et al. [140] indicate that changes in architecture can result in large changes in transfer performance, even given the same training task. The network uses the same number of parameters, but 5.6× the memory and 7.4× the run-time.

| Task Generalization on ImageNet Classification [4] | | | | | |
|---|---|---|---|---|---|
| **Method** | **conv1** | **conv2** | **conv3** | **conv4** | **conv5** |
| ImageNet-labels [77] | 19.3 | 36.3 | 44.2 | 48.3 | 50.5 |
| Gaussian | 11.6 | 17.1 | 16.9 | 16.3 | 14.1 |
| Krähenbühl et al. [51] | 17.5 | 23.0 | 24.5 | 23.2 | 20.6 |
| [1]Noroozi & Favaro [145] | 19.2 | 30.1 | 34.7 | 33.9 | 28.3 |
| Doersch et al. [53] | 16.2 | 23.3 | 30.2 | 31.7 | 29.6 |
| Donahue et al. [102] | **17.7** | 24.5 | 31.0 | 29.9 | 28.0 |
| Pathak et al. [29] | 14.1 | 20.7 | 21.0 | 19.8 | 15.5 |
| Zhang et al. [19] | 13.1 | 24.8 | 31.0 | 32.6 | 31.8 |
| Lab→Lab | 12.9 | 20.1 | 18.5 | 15.1 | 11.5 |
| Lab(drop50)→Lab | 12.1 | 20.4 | 19.7 | 16.1 | 12.3 |
| L→ab(cl) | 12.5 | 25.4 | 32.4 | 33.1 | 32.0 |
| L→ab(reg) | 12.3 | 23.5 | 29.6 | 31.1 | 30.1 |
| ab→L(cl) | 11.6 | 19.2 | 22.6 | 21.7 | 19.2 |
| ab→L(reg) | 11.5 | 19.4 | 23.5 | 23.9 | 21.7 |
| (L,ab)→(ab,L) | 15.1 | 22.6 | 24.4 | 23.2 | 21.1 |
| (L,ab,Lab)→(ab,L,Lab) | 15.4 | 22.9 | 24.0 | 22.0 | 18.9 |
| Ensembled L→ab | 11.7 | 23.7 | 30.9 | 32.2 | 31.3 |
| Split-Brain Auto (reg,reg) | 17.4 | 27.9 | 33.6 | 34.2 | 32.3 |
| Split-Brain Auto (cl,cl) | **17.7** | **29.3** | **35.4** | **35.2** | **32.8** |

Table 5.4: **Task Generalization on ImageNet Classification** To test unsupervised feature representations, we train linear logistic regression classifiers on top of each layer to perform 1000-way ImageNet classification, as proposed in [19]. All weights are frozen and feature maps spatially resized to be ∼9000 dimensions. All methods use AlexNet variants [77], and were pre-trained on ImageNet without labels, except for **ImageNet-labels**. Note that the proposed split-brain autoencoder achieves the best performance on all layers across unsupervised methods.

- **Krähenbühl et al.** [51]: A stacked k-means initialization method.
- **Doersch et al.** [53], **Noroozi & Favaro** [145], **Pathak et al.** [29], **Donahue et al.** [102], and **Zhang et al.** [19] all pre-train on the 1.3M ImageNet dataset [4].
- **Wang & Gupta** [142] and **Owens et al.** [32] pre-train on other large-scale data.

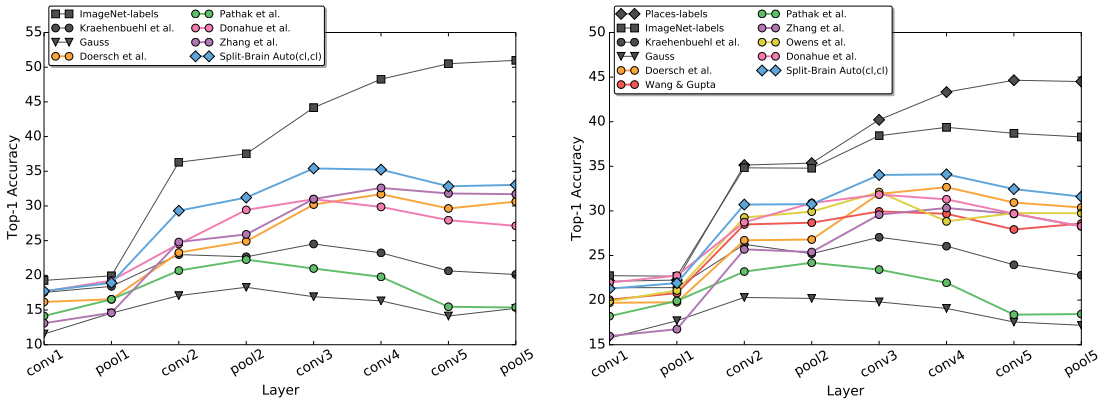
| Dataset & Task Generalization on Places Classification [44] | | | | | |
|---|---|---|---|---|---|
| **Method** | **conv1** | **conv2** | **conv3** | **conv4** | **conv5** |
| Places-labels [44] | 22.1 | 35.1 | 40.2 | 43.3 | 44.6 |
| ImageNet-labels [77] | 22.7 | 34.8 | 38.4 | 39.4 | 38.7 |
| Gaussian | 15.7 | 20.3 | 19.8 | 19.1 | 17.5 |
| Krähenbühl et al. [51] | 21.4 | 26.2 | 27.1 | 26.1 | 24.0 |
| [1]Noroozi & Favaro [145] | 23.0 | 32.1 | 35.5 | 34.8 | 31.3 |
| Doersch et al. [53] | 19.7 | 26.7 | 31.9 | 32.7 | 30.9 |
| Wang & Gupta [142] | 20.1 | 28.5 | 29.9 | 29.7 | 27.9 |
| Owens et al. [32] | 19.9 | 29.3 | 32.1 | 28.8 | 29.8 |
| Donahue et al. [102] | **22.0** | 28.7 | 31.8 | 31.3 | 29.7 |
| Pathak et al. [29] | 18.2 | 23.2 | 23.4 | 21.9 | 18.4 |
| Zhang et al. [19] | 16.0 | 25.7 | 29.6 | 30.3 | 29.7 |
| L→ab(cl) | 16.4 | 27.5 | 31.4 | 32.1 | 30.2 |
| L→ab(reg) | 16.2 | 26.5 | 30.0 | 30.5 | 29.4 |
| ab→L(cl) | 15.6 | 22.5 | 24.8 | 25.1 | 23.0 |
| ab→L(reg) | 15.9 | 22.8 | 25.6 | 26.2 | 24.9 |
| Split-Brain Auto (cl,cl) | 21.3 | **30.7** | **34.0** | **34.1** | **32.5** |

Table 5.5: **Dataset & Task Generalization on Places Classification** We train logistic regression classifiers on top of frozen pre-trained representations for 205-way Places classification. Note that our split-brain autoencoder achieves the best performance among unsupervised learning methods from `conv2-5` layers.

**Transfer Learning Tests** How well does the pre-text task of cross-channel prediction generalize to unseen tasks and data? We run various established large-scale representation learning benchmarks.

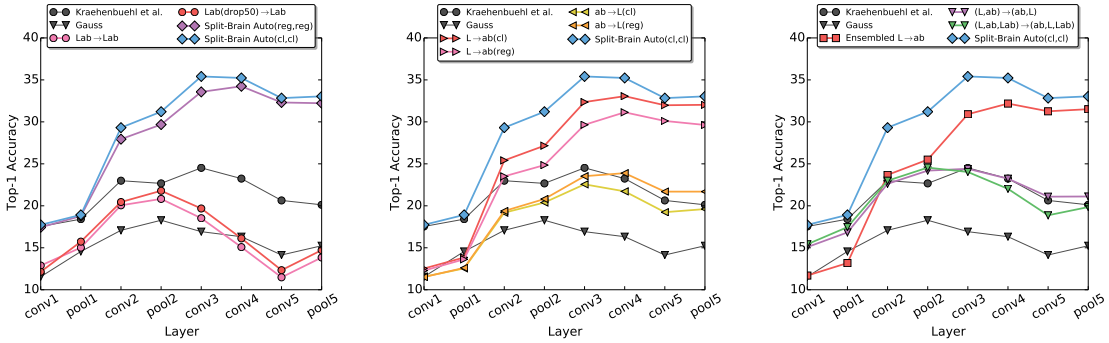
**ImageNet** [77] As proposed in [19], we test the *task* generalization of the representation by freezing the weights and training multinomial logistic regression classifiers on top of each layer to perform 1000-way ImageNet classification. Note that each classifier is a single learned linear layer, followed by a softmax. To reduce the effect of differences in feature map sizes, we spatially resize feature maps through bilinear interpolation, so that the flattened feature maps have approximately equal dimensionality (9600 for `conv1,3,4` and 9216 for `conv2,5`). The results are shown in Table 5.4 and Figures 5.3(a) and 5.4.

**Places** [44] In the previous test, we evaluated the representation on the same input training data, the ImageNet dataset, with a different task than the pretraining tasks. To see how well the network generalizes to new input *data* as well, we run




(a) **ImageNet Classification** (b) **Places Classification**

Figure 5.3: **Comparison to Previous Unsupervised Methods** We compare our proposed Split-Brain Autoencoder on the tasks of (a) ImageNet classification and (b) Places Classification. Note that our method outperforms other large-scale unsupervised methods [19, 29, 32, 53, 102, 142] on all layers in ImageNet and from `conv2-5` on Places.




(a) **Autoencoder Objectives** (b) **Cross-Channel Encoders** (c) **Aggregation Methods**

Figure 5.4: **Ablation Studies** We conduct various ablation studies on our proposed method, using the ImageNet classification benchmark proposed in [19]. Specifically, we compare (a) variations using an autoencoder objective (b) different cross-channel problems and loss functions (c) different methods for aggregating multiple cross-channel encoders.

the same linear classification task on the large-scale Places dataset [44]. The dataset contains 2.4M images for training and 20.5k for validation from 205 scene categories.

The results are shown in Table 5.5 and Figure 5.3(b).

**PASCAL** [7] To further test generalization, we fine-tune the learned representation on standard representation learning benchmarks on the PASCAL dataset, as shown in Table 5.6, using established testing frameworks in classification [51], detection [157], and segmentation [155]. Classification involves 20 binary classification decisions, regarding the presence or absence of 20 object classes. Detection involves drawing an accurately localized bounding box around any objects in the image, and is performed using the Fast R-CNN [157] framework. Segmentation is pixel-wise labeling of the object class, either one of the 20 objects of interest or background. Here, the representation is *fine-tuned* through multiple layers of the network, rather than frozen. Prior to fine-tuning, we follow common practice and use the rescaling method from [51], which rescales the weights so that the layers learn at the same "rate", using the ratio of expected gradient magnitude over feature activation magnitude as a heuristic.

**Split-Brain Autoencoder Performance** Our primary result is that the proposed method, **Split-Brain Auto (cl,cl)**, achieves state-of-the-art performance on almost all established self-supervision benchmarks, as seen in the last row on Tables 5.4, 5.5, 5.6, over previously proposed self-supervision methods, as well as our ablation baselines. Figures 5.3(a) and (b) shows our split brain autoencoder method compared to previous self-supervised methods [19, 29, 32, 53, 102, 142] on the ImageNet and Places classification tests, respectively. We especially note the straightforward nature of our proposed method: the network simply predicts raw data channels from other raw data channels, using a classification loss with a basic 1-hot encoding scheme.

As seen in Figure 5.4(a) and Table 5.4, the autoencoder objective by itself, **Lab→Lab**, does not lead to a strong representation. Performance is near Gaussian initialization through the initial layers, and actually falls below in the `conv5` layer. Dropping 50% of the data from the input randomly during training, **Lab(drop50)→Lab**, in the style of denoising autoencoders, adds a small performance boost of approximately 1%. A large performance boost is observed by adding a split in the architecture, **Split-Brain Auto (reg,reg)**, even with the same regression objective. This achieves 5% to 20% higher performance throughout the network, state-of-the-art compared to previous unsupervised methods. A further boost of approximately 1-2% throughout the network observed using a classification loss, **Split-Brain Auto (cl,cl)**, instead of regression.

**Cross-Channel Encoding Objectives** Figure 5.4(b) compares the performance of the different cross-channel objectives we tested on the ImageNet classification

| Task and Data Generalization on PASCAL VOC [7] | | | | | | | |
|---|---|---|---|---|---|---|---|
| | Classification [51] (%mAP) | | | Detection [157] (%mAP) | | Seg. [155] (%mIU) | |
| **frozen layers** | | **conv5** | **none** | | **none** | | **none** |
| **fine-tuned layers** | **Ref** | **fc6-8** | **all** | **Ref** | **all** | **Ref** | **all** |
| ImageNet labels [77] | [19] | 78.9 | 79.9 | [51] | 56.8 | [155] | 48.0 |
| Gaussian | [29] | – | 53.3 | [29] | 43.4 | [29] | 19.8 |
| Autoencoder | [102] | 16.0 | 53.8 | [29] | 41.9 | [29] | 25.2 |
| Krähenbühl et al. [51] | [102] | 39.2 | 56.6 | [51] | 45.6 | [102] | 32.6 |
| Jayaraman & Grauman [28] | – | – | – | [28] | 41.7 | – | – |
| Agrawal et al. [27] | [51] | – | 52.9 | [51] | 41.8 | – | – |
| Agrawal et al. [27]† | [102] | 31.0 | 54.2 | [51] | 43.9 | – | – |
| Wang & Gupta [142] | [51] | – | 62.8 | [51] | 47.4 | – | – |
| Wang & Gupta [142]† | [51] | – | 63.1 | [51] | 47.2 | – | – |
| Doersch et al. [53] | [51] | – | 55.3 | [51] | 46.6 | – | – |
| Doersch et al. [53]† | [102] | 55.1 | 65.3 | [51] | 51.1 | – | – |
| Pathak et al. [29] | [29] | – | 56.5 | [29] | 44.5 | [29] | 29.7 |
| Donahue et al. [102]† | [102] | 52.3 | 60.1 | [102] | 46.9 | [102] | 35.2 |
| Misra et al. [143] | – | – | – | [143] | 42.4 | – | – |
| Owens et al. [32] | ▷ | 54.6 | 54.4 | [32] | 44.0 | – | – |
| Owens et al. [32]† | ▷ | 52.3 | 61.3 | – | – | – | – |
| Zhang et al. [19]† | [19] | 61.5 | 65.9 | [19] | 46.9 | [19] | 35.6 |
| Larsson et al. [140]◇ | [140] | – | 65.9 | – | – | [140] | **38.4** |
| Pathak et al. [147]◇ | [147] | – | 61.0 | [147] | **52.2** | – | – |
| Split-Brain Auto (cl,cl)† | ▷ | **63.0** | **67.1** | ▷ | 46.7 | ▷ | 36.0 |

Table 5.6: **Task and Dataset Generalization on PASCAL VOC** Classification and detection on PASCAL VOC 2007 [158] and segmentation on PASCAL VOC 2012 [159], using mean average precision (mAP) and mean intersection over union (mIU) metrics for each task, with publicly available testing frameworks from [51], [157], [155]. Column **Ref** documents the source for a value obtained from a previous paper. Character ▷ indicates that value originates from this chapter. †indicates that network weights have been rescaled with [51] before fine-tuning, as is common practice. Character ◇ indicates concurrent work in these proceedings.

benchmark. As shown in [19] and further confirmed here, colorization, **L→ab(cl)**, leads to a strong representation on classification transfer tasks, with higher performance than other unsupervised representations pre-trained on ImageNet, using inpainting [29], relative context [53], and adversarial feature networks [102] from layers from `conv2` to `pool5`. We found that the classification loss produced stronger

representations than regression for colorization, consistent with the findings from concurrent work from Larsson et al. [140].

Interestingly, the task of predicting grayscale from color can also learn representations. Though colorization lends itself closely to a graphics problem, the application of grayscale prediction from color channels is less obvious. As seen in Tables 5.4 and 5.5 and Figure 5.4(b), grayscale prediction objectives **ab→L(cl)** and **ab→L(reg)** can learn representations above the **Gaussian** baseline. Though the learned representation by itself is weaker than other self-supervised methods, the representation is learned on $a$ and $b$ channels, which makes it complementary to the colorization network. For grayscale prediction, regression results in higher performance than classification. Choosing the appropriate loss function for a given channel prediction problem is an open problem. However, note that the performance difference is typically small, indicating that the cross-channel prediction problem is often times an effective method, even without careful engineering of the objective.

**Cross-Channel Encoder Aggregation Analysis** In Figure 5.4(c), we show variations on aggregated cross-channel encoders. To begin, we hypothesize that the performance improvement of split-brain autoencoders **Split-Brain Auto (cl,cl)** over single cross-channel encoders **L→ab** is due to the merging of complementary signals, as each sub-network in **Split-Brain Auto** has been trained on different portions of the input space. However, the improvement could be simply due to an ensembling effect. To test this, we train a split-brain autoencoder, comprising of two **L→ab** networks, **Ensemble L→ab**. As seen in Figure 5.4(c) and Table 5.4, the ensembled colorization network achieves lower performance than the split-brain autoencoder, suggesting that concatenating signals learned on complementary information is beneficial for representation learning.

We find that combining cross-channel encoders through concatenation is effective. We also test alternative aggregation techniques. As seen in Figure 5.4(c), training a single network to perform multiple cross-channel tasks **(L,ab)→(ab,L)** is not effective for representation learning on full *Lab* images. Adding in the autoencoder objective during training, **(L,ab,Lab)→(ab,L,Lab)**, in fact lowers performance in higher layers.

Our proposed methods outperform these alternatives, which indicates that (i) our choice of aggregating complementary signals improves performance (ii) concatenation is an appropriate choice of combining cross-channel encoders.

| Method | Data | Label | RGB | D | RGB-D |
|---|---|---|---|---|---|
| Gupta et al. [154] | 1M ImNet [4] | ✓ | 27.8 | 41.7 | 47.1 |
| Gupta et al. [160] | 1M ImNet [4] | ✓ | 27.8 | 34.2 | 44.4 |
| Gaussian | None | | – | 28.1 | – |
| Krähenbühl et al. [51] | 20 NYU-D [5] | | 12.5 | 32.2 | 34.5 |
| Split-Brain Autoencoder | 10k NYU-D [5] | | **18.9** | **33.2** | **38.1** |

Table 5.7: **Split-Brain Autoencoder Results on RGB-D images** We perform unsupervised training on 10k RGB-D keyframes from the NYU-D [5] dataset, extracted by [154]. We pre-train representations on RGB images using $\ell_2$ loss on depth images in *HHA* space. We pre-train *HHA* representations on $L$ and $ab$ channels using $\ell_2$ and classification loss, respectively. We show performance gains above Gaussian and Krähenbühl et al. [51] initialization baselines. The methods proposed by Gupta et al. [154, 160] use 1.3M labeled images for supervised pre-training. We use the test procedure from [154]: Fast R-CNN [157] networks are first trained individually in the RGB and D domains separately, and then ensembled together by averaging (RGB-D).

### 5.5.2 Split-Brain Autoencoders on RGB-D

We also test the split-brain autoencoder method on registered images and depth scans from NYU-D [5]. Because RGB and depth images are registered spatially, RGB-D data can be readily applied in our proposed framework. We split the data by modality, predicting RGB from D and vice-versa. Previous work in the video and audio domain [146] suggest that separating modalities, rather than mixing them, provides more effective splits. This choice also provides easy comparison to the test procedure introduced by [160].

**Dataset & Detection Testbed** The NYUD dataset contains 1449 RGB-D labeled images and over 400k unlabeled RGB-D video frames. We use 10k of these unlabeled frames to perform unsupervised pre-training, as extracted from [154]. We evaluate the representation on the 1449 labeled images for the detection task, using the framework proposed in [154]. The method first trains individual detectors on the RGB and D domains, using the Fast R-CNN framework [157] on an AlexNet architecture, and then late-fuses them together through ensembling.

**Unsupervised Pre-training** We represent depth images using the HHA encoding, introduced in [160]. To learn image representation $\mathcal{F}_{HHA}$, we train an Alexnet architecture to regress from RGB channels to HHA channels, using an $\ell_2$ regression loss.

To learn depth representations, we train an Alexnet on HHA encodings, using $\ell_2$

loss on $L$ and classification loss on $ab$ color channels. We chose this combination, as these objectives performed best for training individual cross-channel encoders in the image domain. The network extracts features up to the `conv5` layer, using an Alexnet architecture, and then splits off into specific branches for the $L$ and $ab$ channels. Each branch contains AlexNet-type `fc6-7` layers, but with 512 channels each, evaluated fully convolutionally for pixel prediction. The loss on the $ab$ term was weighted 200× with respect to the $L$ term, so the gradient magnitude on the `pool5` representation from channel-specific branches were approximately equal throughout training.

Across all methods, weights up to the `conv5` layer are copied over during fine-tuning time, and `fc6-7` layers are randomly initialized, following [160].

**Results** The results are shown in Table 5.7 for detectors learned in RGB and D domains separately, as well as the ensembled result. For a Gaussian initialization, the RGB detector did not train using default settings, while the depth detector achieved performance of 28.1%. Using the stacked k-means initialization scheme from Krähenbühl et al. [51], individual detectors on RGB and D perform at 12.5% and 32.2%, while achieving 34.5% after ensembling. Pre-training with our method reaches 18.9% and 33.2% on the individual domains, above the baselines. Our RGB-D ensembled performance was 38.1%, well above the Gaussian and Krähenbühl et al. [51] baselines. These results suggest that split-brain autoencoding is effective not just on *Lab* images, but also on RGB-D data.

## 5.6 Discussion

We present split-brain autoencoders, a method for unsupervised pre-training on large-scale data. The split-brain autoencoder contains two disjoint sub-networks, which are trained as cross-channel encoders. Each sub-network is trained to predict one subset of raw data from another. We test the proposed method on *Lab* images, and achieve state-of-the-art performance relative to previous self-supervised methods. We also demonstrate promising performance on RGB-D images. The proposed method solves some of the weaknesses of previous self-supervised methods. Specifically, the method (i) does not require a representational bottleneck for training, (ii) uses input dropout to help force abstraction in the representation, and (iii) is pre-trained on the full input data.

An interesting future direction of is exploring the concatenation of more than 2 cross-channel sub-networks. Given a fixed architecture size, e.g. AlexNet, dividing the network into $N$ disjoint sub-networks results in each sub-network becoming smaller, less expressive, and worse at its original task. To enable fair comparisons to

previous large-scale representation learning methods, we focused on learning weights for a fixed AlexNet architecture. It would also be interesting to explore the regime of fixing the sub-network size and allowing the full network to grow with additional cross-channel encoders.

In this chapter, we tested the ability for the representation to transfer to high-level semantic tasks. In the next chapter, we test the transferability to a low-level task – predicting human perceptual judgments.

# Chapter 6

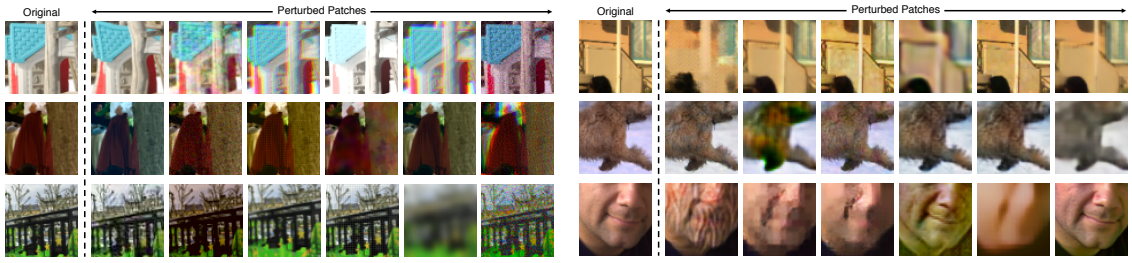
# Using Deep Representations as a Low-Level Perceptual Metric

While it is nearly effortless for humans to quickly assess the perceptual similarity between two images, the underlying processes are thought to be quite complex. Despite this, the most widely used perceptual metrics today, such as PSNR and SSIM, are simple, shallow functions, and fail to account for many nuances of human perception. Recently, the deep learning community has found that features of the VGG network trained on ImageNet classification has been remarkably useful as a training loss for image synthesis. But how perceptual are these so-called "perceptual losses"? What elements are critical for their success? To answer these questions, we introduce a new dataset of human perceptual similarity judgments. We systematically evaluate deep features across different architectures and tasks and compare them with classic metrics. We find that deep features outperform all previous metrics by large margins on our dataset. More surprisingly, this result is not restricted to ImageNet-trained VGG features, but holds across different deep architectures and levels of supervision (supervised, self-supervised, or even unsupervised). Our results suggest that perceptual similarity is an emergent property shared across deep visual representations.[1]

## 6.1 Motivation and Background

The ability to compare data items is perhaps the most fundamental operation underlying all of computing. In many areas of computer science it does not pose

[1]This work was originally published as *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric* in CVPR, 2018.




(a) **Traditional** (b) **CNN-based**

Figure 6.1: **Example distortions.** We show example distortions using our (a) traditional and (b) CNN-based methods.

much difficulty: one can use Hamming distance to compare binary patterns, edit distance to compare text files, Euclidean distance to compare vectors, etc. The unique challenge of computer vision is that even this seemingly simple task of comparing visual patterns remains a wide-open problem. Not only are visual patterns very high-dimensional and highly correlated, but, the very notion of visual similarity is often subjective, aiming to mimic human visual perception. For instance, in image compression, the goal is for the compressed image to be indistinguishable from the original by a human observer, irrespective of the fact that their pixel representations might be very different.

Classic per-pixel measures, such as $\ell_2$ Euclidean distance, commonly used for regression problems, or the related Peak Signal-to-Noise Ratio (PSNR), are insufficient for assessing structured outputs such as images, as they assume pixel-wise independence. A well-known example is that blurring causes large perceptual but small $\ell_2$ change.

What we would really like is a "perceptual distance," which measures how similar are two images in a way that coincides with human judgment. This problem has been a longstanding goal, and there have been numerous perceptually motivated distance metrics proposed, such as SSIM [15], MSSIM [16], FSIM [17], and HDR-VDP [18].

However, constructing a perceptual metric is challenging, because human judgments of similarity (1) depend on high-order image structure [15], (2) are context-dependent [161–163], and (3) may not actually constitute a distance metric [164]. The crux of (2) is that there are many different "senses of similarity" that we can simultaneously hold in mind: is a red circle more similar to a red square or to a blue circle? Directly fitting a function to human judgments may be intractable due the the context-dependent and pairwise nature of the judgments (which compare the similarity between *two* images). Indeed, we show in this chapter a negative result where this approach fails to generalize, even when trained on a large-scale dataset containing many distortion types.

| Dataset | # Input Imgs/Patches | Input Type | Num Distort. | Distort Types | # Levels | # Distorted Imgs/Patches | # Judgments | Judgment Type |
|---|---|---|---|---|---|---|---|---|
| LIVE [165] | 29 | images | 5 | traditional | cont. | .8k | 25k | MOS |
| CSIQ [166] | 30 | images | 6 | traditional | 5 | .8k | 25k | MOS |
| TID2008 [167] | 25 | images | 17 | traditional | 4 | 2.7k | 250k | MOS |
| TID2013 [168] | 25 | images | 24 | traditional | 5 | 3.0k | 500k | MOS |
| BAPPS (2AFC–Distort) | 160.8k | $64 \times 64$ patch | 425 | trad+CNN | cont. | 321.6k | 349.8k | 2AFC |
| BAPPS (2AFC–Real alg) | 26.9k | $64 \times 64$ patch | – | alg outputs | – | 53.8k | 134.5k | 2AFC |
| BAPPS (JND–Distort) | 9.6k | $64 \times 64$ patch | 425 | trad+CNN | cont. | 9.6k | 28.8k | Same? |

Table 6.1: **Dataset comparison.** A primary differentiator between our proposed Berkeley-Adobe Perceptual Patch Similarity (BAPPS) dataset and previous work is scale of distortion types. We provide human perceptual judgments on distortion set using uncompressed images from [169, 170]. Previous datasets have used a small number of distortions at discrete levels. We use a large number of distortions (created by sequentially composing atomic distortions together) and sample continuously. For each input patch, we corrupt it using two distortions and ask for a few human judgments (2 for train, 5 for test set) per pair. This enables us to obtain judgments on a large number of patches. Previous databases summarize their judgments into a mean opinion score (MOS); we simply report pairwise judgments (two alternative force choice). In addition, we provide judgments on outputs from *real algorithms*, as well as a same/not same Just Noticeable Difference (JND) perceptual test.

Instead, might there be a way to learn a notion of perceptual similarity without directly training for it? The computer vision community has discovered that internal activations of deep convolutional networks, though trained on a high-level image classification task, are often surprisingly useful as a representational space for a much wider variety of tasks. For example, features from the VGG architecture [24] have been used on tasks such as neural style transfer [84], image superresolution [137], and conditional image synthesis [125, 138]. These methods measure distance in VGG feature space as a "perceptual loss" for image regression problems [137, 138].

But how well do these "perceptual losses" actually correspond to human visual perception? How do they compare to traditional perceptual image evaluation metrics? Does the network architecture matter? Does it have to be trained on the ImageNet classification task, or would other tasks work just as well? Do the networks need to be trained at all?

In this chapter, we evaluate these questions on a new large-scale database of human judgments, and arrive at several surprising conclusions. We find that internal activations of networks trained for high-level classification tasks, even across network architectures [24, 77, 171] and no further calibration, do indeed correspond to human perceptual judgments. In fact, they correspond far better than the commonly used

metrics like SSIM and FSIM [15, 17], which were not designed to handle situations where spatial ambiguities are a factor [172]. Furthermore, the best performing self-supervised networks, including BiGANs [102], puzzle solving [145], and our split-brain autoencoder [19, 173] perform just as well at this task, even without the benefit of human-labeled training data. Even a simple unsupervised network initialization with stacked k-means [51] beats the classic metrics by a large margin! This illustrates an *emergent property* shared across networks, even across architectures and training signals. Importantly, however, having *some* training signal appears crucial – a randomly initialized network achieves much lower performance.

Our study is based on a newly collected perceptual similarity dataset, using a large set of distortions and real algorithm outputs. It contains both traditional distortions, such as contrast and saturation adjustments, noise patterns, filtering, and spatial warping operations, and CNN-based algorithm outputs, such as autoencoding, denoising, and colorization, produced by a variety of architectures and losses. Our dataset is richer and more varied than previous datasets of this kind [168]. We also collect judgments on outputs from real algorithms for the tasks of superresolution, frame interpolation, and image deblurring, which is especially important as these are the real-world use cases for a perceptual metric. We show that our data can be used to "calibrate" existing networks, by learning a simple linear scaling of layer activations, to better match low-level human judgments.

Our results are consistent with the hypothesis that perceptual similarity is not a special function all of its own, but rather a *consequence* of visual representations tuned to be predictive about important structure in the world. Representations that are effective at semantic prediction tasks are also representations in which Euclidean distance is highly predictive of perceptual similarity judgments.

Our contributions are as follows:

- We introduce a large-scale, highly varied, perceptual similarity dataset, containing 484k human judgments. Our dataset not only includes parameterized distortions, but also real algorithm outputs. We also collect judgments on a different perceptual test, just noticeable differences (JND).
- We show that deep features, trained on supervised, self-supervised, and unsupervised objectives alike, model low-level perceptual similarity surprisingly well, outperforming previous, widely-used metrics.
- We demonstrate that network architecture alone does not account for the performance: untrained nets achieve much lower performance.
- With our data, we can improve performance by "calibrating" feature responses from a pre-trained network.

**Prior work on datasets** In order to evaluate existing similarity measures, a number of datasets have been proposed. Some of the most popular are the LIVE [165], TID2008 [167], CSIQ [166], and TID2013 [168] datasets. These datasets are referred to Full-Reference Image Quality Assessment (FR-IQA) datasets and have served as the de-facto baselines for development and evaluation of similarity metrics. A related line of work is on No-Reference Image Quality Assessment (NR-IQA), such as AVA [174] and LIVE In the Wild [175]. These datasets investigate the "quality" of individual images by themselves, without a reference image. We collect a new dataset that is complementary to these: it contains a substantially larger number of distortions, including some from newer, deep network based outputs, as well as geometric distortions. Our dataset is focused on perceptual similarity, rather than quality assessment. Additionally, it is collected on patches as opposed to full images, in the wild, with a different experimental design (more details in Sec 6.2).

**Prior work on deep networks and human judgments** Recently, advances in DNNs have motivated investigation of applications in the context of visual similarity and image quality assessment. Kim and Lee [176] use a CNN to predict visual similarity by training on low-level differences. Concurrent work by Talebi and Milanfar [177,178] train a deep network in the context of NR-IQA for image aesthetics. Gao et al. [179] and Amirshahi et al. [180] propose techniques involving leveraging internal activations of deep networks (VGG and AlexNet, respectively) along with additional multiscale post-processing. In this work, we conduct a more in-depth study across different architectures, training signals, on a new, large scale, highly-varied dataset.

Recently, Berardino et al. [181] train networks on perceptual similarity, and importantly, assess the ability of deep networks to make predictions on a *separate* task – predicting most and least perceptually-noticeable directions of distortion. Similarly, we not only assess image patch similarity on parameterized distortions, but also test generalization to real algorithms, as well as generalization to a separate perceptual task – just noticeable differences.

## 6.2 Berkeley-Adobe Perceptual Patch Similarity (BAPPS) Dataset

To evaluate the performance of different perceptual metrics, we collect a large-scale highly diverse dataset of perceptual judgments using two approaches. Our main data collection employs a two alternative forced choice (2AFC) test, that asks which of two distortions is more similar to a reference. This is validated by a second

| Sub-type | Distortion type |
|---|---|
| Photometric | lightness shift, color shift, contrast, saturation |
| Noise | uniform white noise, Gaussian white, pink, & blue noise, Gaussian colored (between violet and brown) noise, checkerboard artifact |
| Blur | Gaussian, bilateral filtering |
| Spatial | shifting, affine warp, homography, linear warping, cubic warping, ghosting, chromatic aberration, |
| Compression | jpeg |

| Parameter type | Parameters |
|---|---|
| Input corruption | null, pink noise, white noise, color removal, downsampling |
| Generator network architecture | # layers, # skip connections, # layers with dropout, force skip connection at highest layer, upsampling method, normalization method, first layer stride # channels in $1^{st}$ layer, max # channels |
| Discriminator | number of layers |
| Loss/Learning | weighting on oixel-wise ($\ell_1$), VGG, discriminator losses, learning rate |

Table 6.2: **Our distortions.** Our traditional distortions (left) are performed by basic low-level image editing operations. We also sequentially compose them to better explore the space. Our CNN-based distortions (right) are formed by randomly varying parameters such as task, network architecture, and learning parameters. The goal of the distortions is to mimic plausible distortions seen in real algorithm outputs.

experiment where we perform a just noticeable difference (JND) test, which asks whether two patches – one reference and one distorted – are the same or different. These judgments are collected over a wide space of distortions and real algorithm outputs.

### 6.2.1 Distortions

**Traditional distortions** We create a set of “traditional" distortions consisting of common operations performed on the input patches, listed in Table 6.2 (left). In general, we use photometric distortions, random noise, blurring, spatial shifts and corruptions, and compression artifacts. We show qualitative examples of our traditional distortions in Figure 6.1. The severity of each perturbation is parameterized - for example, for Gaussian blur, the kernel width determines the amount of corruption applied to the input image. We also compose pairs of distortions sequentially to increase the overall space of possible distortions. In total, we have 20 distortions and 308 sequentially composed distortions.

**CNN-based distortions** To more closely simulate the space of artifacts that can arise from deep-learning based methods, we create a set of distortions created by neural networks. We simulate possible algorithm outputs by exploring a variety of tasks, architectures, and losses, as shown in Table 6.2 (right). Such tasks include autoencoding, denoising, colorization, and superresolution. All of these tasks can be achieved by applying the appropriate corruption to the input. In total, we generated 96 “denoising autoencoders" and use these as CNN-based distortion functions. We train each of these networks on the 1.3M ImageNet dataset [4] for 1 epoch. The

goal of each network is not to solve the task per se, but rather to explore common artifacts that plague the outputs of deep learning based methods.

**Distorted image patches from real algorithms** The true test of an image assessment algorithm is on real problems and real algorithms. We gather perceptual judgments using such outputs. Data on real algorithms is more limited, as each application will have their own unique properties. For example, different colorization methods will not show much structural variation, but will be prone to effects such as color bleeding and color variation. On the other hand, superresolution will not have color ambiguity, but may see larger structural changes from algorithm to algorithm.

**Superresolution** We evaluate results from the 2017 NTIRE workshop [182]. We use 3 tracks from the workshop – $\times 2$, $\times 3$, $\times 4$ upsampling rates using "unknown" downsampling to create the input images. Each track had approximately 20 algorithm submissions. We also evaluate several additional methods, including bicubic upsampling, and four of the top performing deep superresolution methods [183–186]. A common qualitative way of presenting superresolution results is zooming into specific patches and comparing differences. As such, we sample random $64 \times 64$ triplets from random locations of images in the Div2K [182] dataset – the ground truth high-resolution image, along with two algorithm outputs.

**Frame interpolation** We sample patches from different frame interpolation algorithms, including three variants of flow-based interpolation [187], CNN-based interpolation [188], and phase-based interpolation [189] on the Davis Middleburry dataset [190]. Because artifacts arising from frame interpolation may occur at different scales, we randomly rescale the image before sampling a patch triplet.

**Video deblurring** We sample from the video deblurring dataset [191], along with deblurring outputs from Photoshop Shake Reduction, Weighted Fourier Aggregation [192], and three variants of a deep video deblurring method [191].

**Colorization** We sample patches using random scales on the colorization task, on images from the ImageNet dataset [4]. The algorithms are from pix2pix [144], Larsson et al. [39], and the variants from Chapter 2 [19].

### 6.2.2 Psychophysical Similarity Measurements

**2AFC similarity judgments** We randomly select an image patch $x$ and apply two distortions to produce patches $x_0, x_1$. We then ask a human which is closer to the original patch $x$, and record response $h \in \{0, 1\}$. On average, people spent approximately 3 seconds per judgment. Let $\mathcal{T}$ denote our dataset of patch triplets $(x, x_0, x_1, h)$.

A comparison between our dataset and previous datasets is shown in Table 6.1. Previous datasets have focused on collecting large numbers of human judgments for a few images and distortion types. For example, the largest dataset, TID2013 [168], has 500k judgments on 3000 distortions (from 25 input images with 24 distortions types, each sampled at 5 levels). We provide a complementary dataset that focuses instead on a large number of distortions types. In, addition, we collect judgments on a large number of $64 \times 64$ patches rather than a small number of images. There are three reasons for this. First, the space of full images is extremely large, which makes it much harder to cover a reasonable portion of the domain with judgments (even $64 \times 64$ color patches represent an intractable 12k-dimensional space). Second, by choosing a smaller patch size, we focus on lower-level aspects of similarity, to mitigate the effect of differing "respects of similarity" that may be influenced by high-level semantics [162]. Finally, modern methods for image synthesis train deep networks with patch-based losses (implemented as convolutions) [125, 193]. Our dataset consists of over 161k patches, derived from the MIT-Adobe 5k dataset [169] (5000 uncompressed images) for training, and the RAISE1k dataset [170] for validation.

To enable large-scale collection, our data is collected "in-the-wild" on Amazon Mechanical Turk, as opposed to a controlled lab setting. Crump et al. [194] show that AMT can be reliably used to replicate many psychophysics studies, despite the inability to control all environmental factors. We ask for 2 judgments per example in our "train" set and 5 judgments in our "val" sets. Asking for fewer judgments enables us to explore a larger set of image patches and distortions. We add sentinels which consist of pairs of patches with obvious deformations, e.g., a large amount of Gaussian noise vs a small amount of Gaussian noise. Approximately 90% of Turkers were able to correctly pass at least 93% of the sentinels (14 of 15), indicating that they understood the task and were paying attention. We choose to use a larger number of distortions than prior datasets.

**Just noticeable differences (JND)** A potential shortcoming of the 2AFC task is that it is "cognitively penetrable," in the sense that participants can consciously choose which respects of similarity they will choose to focus on in completing the task [162], which introduces subjectivity into the judgments. To validate that the

| Dataset | Data source | Train/ Val | # Examples | # Judge /Example |
|---|---|---|---|---|
| Traditional | [169] | Train | 56.6k | 2 |
| CNN-based | [169] | Train | 38.1k | 2 |
| Mixed | [169] | Train | 56.6k | 2 |
| **2AFC–Distort [Trn]** | – | Train | 151.4k | 2 |
| Traditional | [170] | Train | 4.7k | 5 |
| CNN-based | [170] | Train | 4.7k | 5 |
| **2AFC–Distort [Val]** | – | Val | 9.4k | 5 |
| Superres | [195] | Val | 10.9k | 5 |
| Frame Interp | [190] | Val | 1.9 | 5 |
| Video Deblur | [196] | Val | 9.4 | 5 |
| Colorization | [4] | Val | 4.7 | 5 |
| **2AFC–Real Alg [Val]** | – | Val | 26.9k | 5 |
| Traditional | [170] | Val | 4.8k | 3 |
| CNN-based | [170] | Val | 4.8k | 3 |
| **JND–Distort** | – | Val | 9.6k | 3 |

Table 6.3: **Our dataset breakdown**. We split our 2AFC dataset in to three main portions (1,2) training and test sets with our distortions. Our training and test sets contain patches sampled from the MIT5k [169] and RAISE1k [170] datasets, respectively (3) a test set containing real algorithm outputs, containing patches from a variety of applications. Our JND data is on traditional and CNN-based distortions.

judgments actually reflected something objective and meaningful, we also collected user judgments of "just noticeable differences" (JNDs). We show a reference image, followed by a randomly distorted image, and ask a human if the images are the same or different. The two image patches are shown for 1 second each, with a 250 ms gap in between. Two images which look similar may be easily confused, and a good perceptual metric will be able to order pairs from most to least confusable. JND tests like this may be considered less subjective, since there is a single correct answer for each judgment, and participants are presumed to be aware of what correct behavior entails. We gather 3 JND observations for each of the 4.8k patches in our traditional and CNN-based validation sets. Each subject is shown 160 pairs, along with 40 sentinels (32 identical and 8 with large Gaussian noise distortion applied). We also provide a short training period of 10 pairs which contain 4 "same" pairs, 1 obviously different pair, and 5 "different" pairs generated by our distortions. We chose to do this in order to prime the users towards expecting approximately 40%

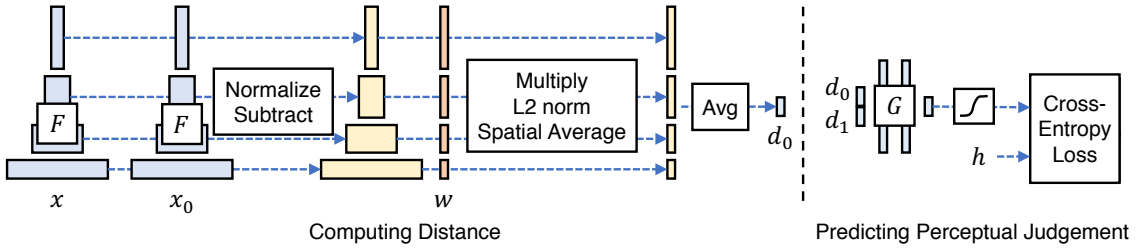



Figure 6.2: **Computing distance from a network** (Left) To compute a distance $d_0$ between two patches, $x$, $x_0$, given a network $\mathcal{F}$, we first compute deep embeddings, normalize the activations in the channel dimension, scale each channel by vector $w$, and take the $\ell_2$ distance. We then average across spatial dimension and across all layers. (Right) A small network $\mathcal{G}$ is trained to predict perceptual judgment $h$ from distance pair $(d_0, d_1)$.

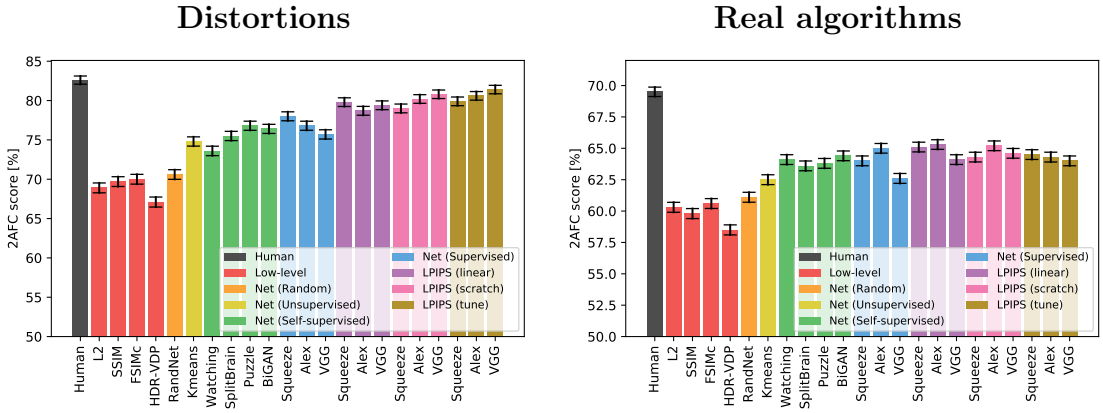



Figure 6.3: **Quantitative comparison.** We show a quantitative comparison across metrics on our test sets. (Left) Results averaged across our traditional and CNN-based distortions. (Right) Results averaged across our 4 real algorithm sets.

of the patch pairs to be identical. Indeed, 36.4% of the pairs were marked “same" (70.4% of sentinels and 27.9% of test pairs).

## 6.3 Deep Feature Spaces as a Perceptual Metric

We evaluate feature distances in different networks. For a given convolutional layer, we compute cosine distance (in the channel dimension) and average across spatial dimensions and layers of the network. We also discuss how to tune an existing network on our data.

**Network architectures** We evaluate the SqueezeNet [171], AlexNet [77], and VGG [24] architectures. We use 5 `conv` layers from the VGG network, which has become the de facto standard for image generation tasks [84, 125, 138]. We also compare against the shallower AlexNet network, which may more closely match the architecture of the human visual cortex [8]. We use the `conv1-conv5` layers from [139]. Finally, the SqueezeNet architecture was designed to be extremely lightweight (2.8 MB) in size, with similar classification performance to AlexNet. We use the first `conv` layer and some subsequent "`fire`" modules.

We additionally evaluate self-supervised methods, including puzzle-solving [145], cross-channel prediction [19, 173], learning from video [147], and generative modeling [102]. We use publicly available networks from these and other methods, which use variants of AlexNet [77].

**Network activations to distance** Figure 6.2 (left) and Equation 6.1 illustrate how we obtain the distance between reference and distorted patches $x, x_0$ with network $\mathcal{F}$. We extract feature stack from $L$ layers and unit-normalize in the channel dimension, which we designate as $\hat{y}^l, \hat{y}^l_0 \in \mathbb{R}^{H_l \times W_l \times C_l}$ for layer $l$. We scale the activations channel-wise by vector $w^l \in \mathbb{R}^{C_l}$ and compute the $\ell_2$ distance. Finally, we average spatially and sum channel-wise. Note that using $w_l = 1 \forall l$ is equivalent to computing cosine distance.

$$d(x, x_0) = \sum_l \frac{1}{H_l W_l} \sum_{h,w} ||w_l \odot (\hat{y}^l_{hw} - \hat{y}^l_{0hw})||_2^2 \tag{6.1}$$

**Training on our data** We consider a few variants for training with our perceptual judgments: ***lin***, ***tune***, and ***scratch***. For the ***lin*** configuration, we keep pre-trained network weights $\mathcal{F}$ fixed, and learn linear weights $w$ on top. This constitutes a "perceptual calibration" of a few parameters in an existing feature space. For example, for the VGG network, 1472 parameters are learned. For the ***tune*** configuration, we initialize from a pre-trained classification model, and allow all the weights for network $\mathcal{F}$ to be fine-tuned. Finally, for ***scratch***, we initialize the network from random Gaussian weights and train it entirely on our judgments. Overall, we refer to these as variants of our proposed **Learned Perceptual Image Patch Similarity (LPIPS)** metric. We illustrate the training loss function in Figure 6.2 (right). Given two distances, $(d_0, d_1)$, we train a small network $\mathcal{G}$ on top to map to a score $\hat{h} \in (0, 1)$. The architecture uses two 32-channel `FC-ReLU` layers, followed by a 1-channel `FC`

layer and a sigmoid. Our final loss function is shown in Equation 6.2.

$$\begin{aligned}\mathcal{L}(x, x_0, x_1, h) = &-h \log \mathcal{G}(d(x, x_0), d(x, x_1)) \\ &-(1-h) \log(1 - \mathcal{G}(d(x, x_0), d(x, x_1)))\end{aligned} \tag{6.2}$$

In preliminary experiments, we also tried a ranking loss, which attempts to force a constant margin between patch pairs $d(x, x_0)$ and $d(x, x_1)$. We found that using a learned network, rather than enforcing the same margin in all cases, worked better.

We train with 5 epochs at initial learning rate $10^{-4}$, 5 epochs with linear decay, and batch size 50. Each training patch pair is judged 2 times, and the judgments are grouped together. If, for example, the two judges are split, then the classification target ($h$ in Figure 3) will be set at 0.5. We enforce non-negative weightings on the linear layer $w$, since larger distances in a certain feature should not result in two patches becoming closer in the distance metric. This is done by projecting the weights into the constraint set at every iteration. In other words, we check for any negative weights, and force them to be 0. The project was implemented using `PyTorch` [197].

## 6.4 Experiments

### 6.4.1 Evaluation on our dataset

Results on our validation sets are shown in Figure 6.3. We first evaluate how well our metrics and networks work. All validation sets contain 5 pairwise judgments for each triplet. Because this is an inherently noisy process, we compute agreement of an algorithm with *all* of the judgments. For example, if there are 4 preferences for $x_0$ and 1 for $x_1$, an algorithm which predicts the more popular choice $x_0$ would receive 80% credit.

If humans chose patches $\{x_1,x_0\}$ with fraction $\{p,1-p\}$, the theoretical maximum for an oracle is $\max(p, 1-p)$. However, human performance is lower. If an agent chooses them with probability $\{q,1-q\}$, the agent will agree with $qp + (1-q)(1-p)$ humans on expectation. With a human agent, $q = p$, the expected human score is $p^2 + (1-p)^2$.

**How well do low-level metrics and classification networks perform?** Figure 6.3 shows the performance of various low-level metrics (in red), deep networks, and human ceiling (in black). The scores are averaged across the 2 distortion test sets (traditional+CNN-based) in Figure 6.3 (left), and 4 real algorithm benchmarks

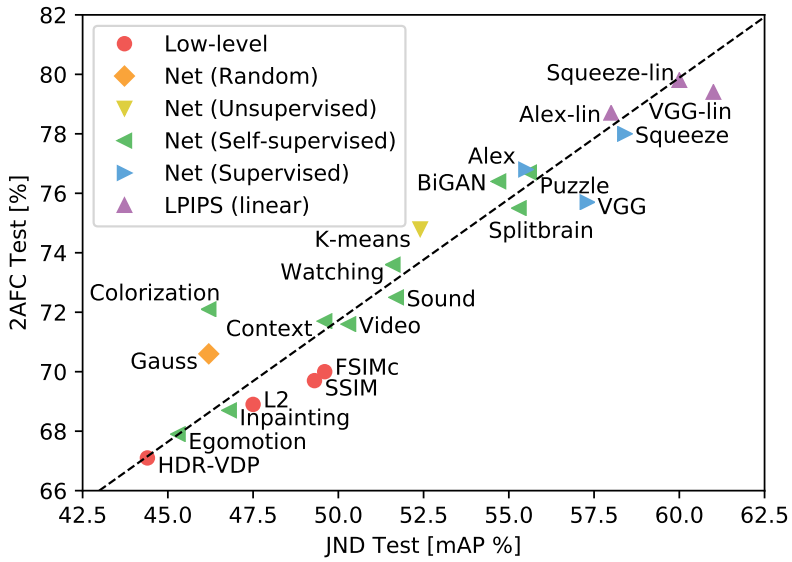



Figure 6.4: **Correlating Perceptual Tests.** We show performance across methods, including unsupervised [51], self-supervised [19, 27, 29, 31, 53, 102, 142, 145, 147, 173], supervised [24,139,171], and our perceptually-learned metrics (LPIPS). The scores are on our 2AFC and JND tests, averaged across traditional and CNN-based distortions.

(superresolution, frame interpolation, video deblurring, colorization) in Figure 6.3 (right). All scores within each test set are shown in the appendix. Averaged across all 6 test sets, humans are 73.9% consistent. Interestingly, the supervised networks perform at about the same level to each other, at 68.6%, 68.9%, and 67.0%, even across variation in model sizes – SqueezeNet (2.8 MB), AlexNet (9.1 MB), and VGG (58.9 MB) (only convolutional layers are counted). They all perform better than traditional metrics $\ell_2$, SSIM, and FSIM at 63.2%, 63.1%, 63.8%, respectively. Despite its common use, SSIM was not designed for situations where geometric distortion is a large factor [172].

**Does the network have to be trained on classification?** In Figure 6.3, we show model performance across a variety of unsupervised and self-supervised tasks, shown in green – generative modeling with BiGANs [102], solving puzzles [145], cross-channel prediction [173], and segmenting foreground objects from video [147]. These self-supervised tasks perform on par with classification networks. This indicates that tasks across a large spectrum can induce representations which transfer well to perceptual distances. Also, the performance of the stacked k-means method [51],

| | | 2AFC | JND | Class. | Det. | Avg |
|---|---|---|---|---|---|---|
| **Perceptual** | **2AFC** | – | .928 | .640 | .363 | .644 |
| **Perceptual** | **JND** | .928 | – | .612 | .232 | .591 |
| **PASCAL** | **Classification** | .640 | .612 | – | .429 | .560 |
| **PASCAL** | **Detection** | .363 | .232 | .429 | – | .341 |

Table 6.4: **Task correlation**. We correlate scores between our low-level perceptual tests along with high-level semantic tests across methods. Perceptual scores are averaged between traditional and CNN-based distortion sets. Correlation scores are computed for AlexNet-like architectures.

shown in yellow, outperforms low-level metrics. Random networks, shown in orange, with weights drawn from a Gaussian, do not yield much improvement. This indicates that the combination of network structure, along with orienting filters in directions where data is more dense, can better correlate to perceptual judgments.

In Table 6.4, we explore how well our perceptual task correlates to semantic tasks on the PASCAL dataset [158], using results summarized in [173], including additional self-supervised methods [19, 27, 29, 31, 53, 142]. We compute the correlation coefficient between each task (perceptual or semantic) across different methods. The correlation from our 2AFC distortion preference task to classification and detection is .640 and .363, respectively. Interestingly, this is similar to the correlation between the classification and detection tasks (.429), even though both are considered "high-level" semantic tasks, and our perceptual task is "low-level."

**Do metrics correlate across different perceptual tasks?** We test if training for the 2AFC distortion preference test corresponds with another perceptual task, the JND test. We order patch pairs by ascending order by a given metric, and compute precision-recall on our CNN-based distortions – for a good metric, patches which are close together are more likely to be confused for being the same. We compute area under the curve, known as mAP [158]. The 2AFC distortion preference test has high correlation to JND: $\rho = .928$ when averaging the results across distortion types. Figure 6.4 shows how different methods perform under each perceptual test. This indicates that 2AFC generalizes to another perceptual test and is giving us signal regarding human judgments.

**Where do deep metrics and low-level metrics disagree?** In Figure 6.5, we show a qualitative comparison across our traditional distortions for a deep method, BiGANs [102], and a representation traditional perceptual method, SSIM [15]. Pairs which BiGAN perceives to be far but SSIM to be close generally contain some blur.


Figure 6.5: **Qualitative comparisons on distortions.** We show qualitative comparison on traditional (top) and CNN-based (bottom) distortions, using the SSIM [15] metric and BiGAN network [102]. We show examples where the metrics agree and disagree. A primary difference is that deep embeddings appear to be more sensitive to blur. Please see the appendix for additional examples.

BiGAN tends to perceive correlated noise patterns to be a smaller distortion than SSIM.

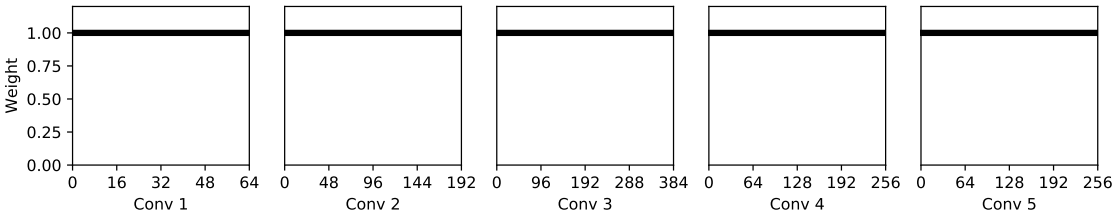
### 6.4.2 Training using our dataset

**Can we train a metric on traditional and CNN-based distortions?** In Figure 6.3, we show performance using our ***lin***, ***scratch***, and ***tune*** configurations, shown in purple, pink, and brown, respectively. When validating on the traditional and CNN-based distortions (Figure 6.3(a)), we see improvements. Allowing the network to tune all the way through (brown) achieves higher performance than simply learning linear weights (purple) or training from scratch (pink). The higher capacity network VGG also performs better than the lower capacity SqueezeNet and AlexNet architectures. These results verify that networks can indeed learn from perceptual judgments.

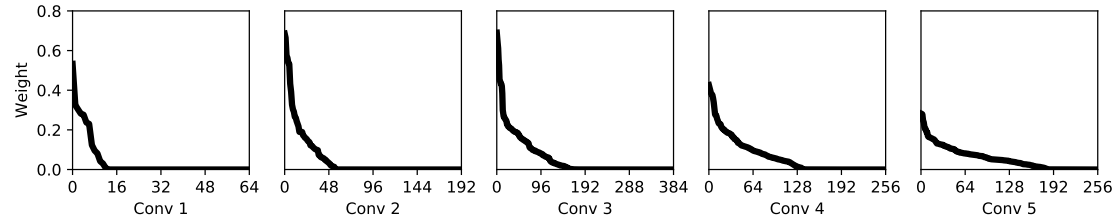
**Does training on traditional and CNN-based distortions transfer to real-world scenarios?** We are more interested in how performance generalizes to *real-world algorithms*, shown in Figure 6.3(b). The SqueezeNet, AlexNet, and VGG architectures start at 64.0%, 65.0%, and 62.6%, respectively. Learning a linear classifier (purple) improves performance for all networks. Across the 3 networks and 4 real-algorithm tasks, 11 of the 12 scores improved, indicating that "calibrating" activations on a pre-existing representation using our data is a safe way to achieve a small boost in performance (1.1%, 0.3%, and 1.5%, respectively). Training a network from scratch (pink) yields slightly lower performance for AlexNet, and slightly higher performance for VGG than linear calibration. However, these still outperform low-level metrics. This indicates that the distortions we have expressed do project onto our test-time tasks of judging real algorithms.

Interestingly, starting with a pre-trained network and tuning throughout *lowers* transfer performance. This is an interesting negative result, as training for a low-level perceptual task directly does not necessarily perform as well as transferring a representation trained for the high-level task.

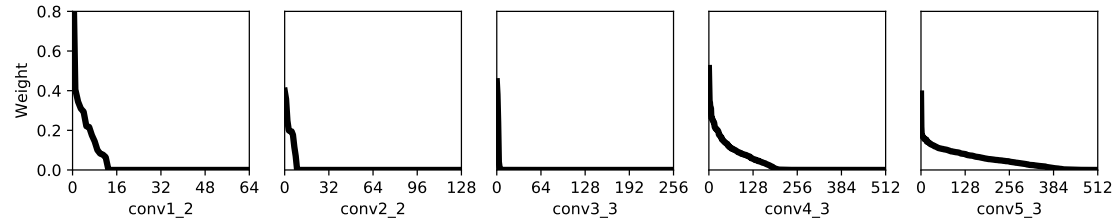
**What weights are learned when perceptually linearly calibrating networks?** Learning linear weights on top of the ***Alex*** model achieves state-of-the-art results on the real algorithms test set. The ***linear*** models have a learned linear layer on top of each channel, whereas the out-of-the-box versions weight each channel equally. In Figure 6.6b, we show the learned weights for the ***Alex –frozen*** model. The `conv1-5` layers contain 64, 192, 384, 256, and 256 channels, respectively, for a total of 1152 weights. For each layer, `conv1-5`, 79.7%, 71.4%, 56.8%, 46.5%, 27.7%,



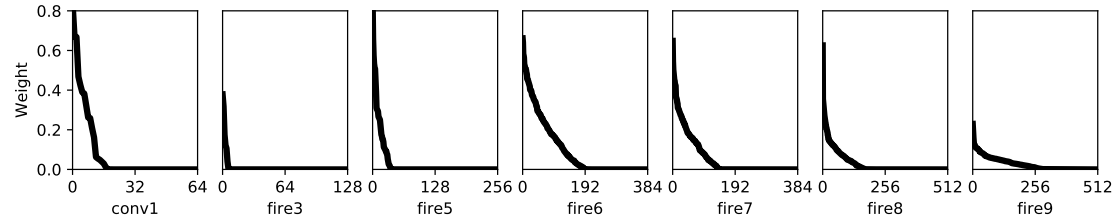

(a) **Unlearned weights for AlexNet model (cosine distance)**




(b) **Learned weights from *Alex–lin* model**




(c) **Learned weights from *VGG–lin* model**




(d) **Learned weights from *Squeeze–lin* model**

Figure 6.6: **Learned linear weights by layer.** (a) Unlearned weights correspond to using weighting 1 for each channel in each layer, which results in computing cosine distance. (b) We show the learned weights from each layer of our *Alex–lin* model. This is the $w$ term in Figure 6.2. Each subplot shows the channel weights from each layer, sorted in descending order. The x-axis shows the channel number, and y-axis shows the weight. Weights are restricted to be non-negative, as image patches should not have negative distance. (c,d) Same as (b), but with the *VGG–lin* and *Squeeze–lin* models.

respectively, of the weights are zero. This means that a majority of the `conv1` and `conv2` units are ignored, and almost all of the `conv5` units are used. Overall, about half of the units are ignored. Taking the cosine distance is equivalent to setting all weights to 1 (Figure 6.6a).

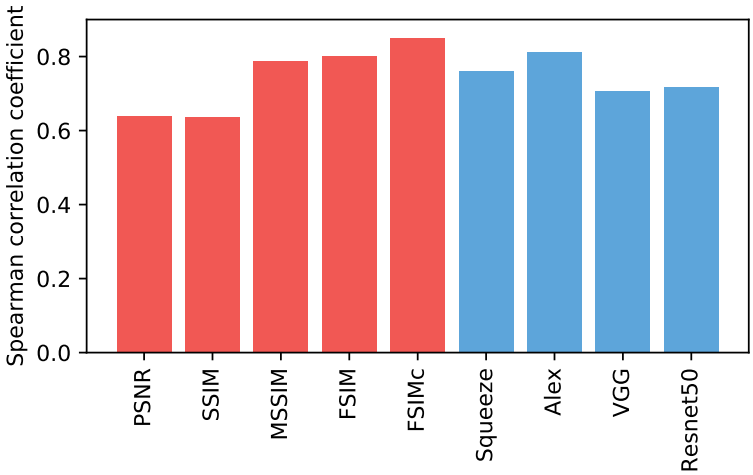



Figure 6.7: **TID Dataset** We show the Spearman correlation coefficient of various methods on the TID2013 Dataset [168]. Note that deep networks trained for classification perform well out of the box (blue).

### 6.4.3 TID2013 Dataset

In Figure 6.7, we compute scores on the TID2013 [168] dataset. We test the images at a different resolutions, using $\{128, 192, 256, 384, 512\}$ for the smaller dimension. We note that even averaging across all scales and layers, with no further calibration, the AlexNet [139] architecture gives scores near the highest metric, FSIMc [17]. On our traditional perturbations, the FSIMc metric achieves 61.4%, close to $\ell_2$ at 59.9%, while the deep classification networks we tested achieved 73.3%, 70.6%, and 70.1%, respectively. The difference is likely due to the inclusion of geometric distortions in our dataset. Despite their frequent use in such situations, metrics such as SSIM were not designed to handle geometric distortions [172].

## 6.5 Discussion

Our results indicate that networks trained to solve challenging visual prediction and modeling tasks end up learning a representation of the world that correlates

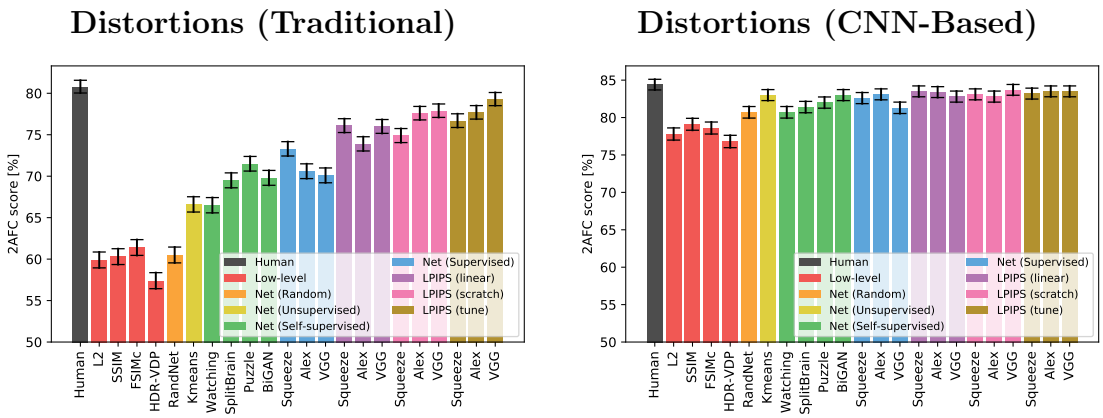



Figure 6.8: **Individual results** (left) traditional distortions (right) CNN-based distortions

well with perceptual judgments. A similar story has recently emerged in the representation learning literature: networks trained on self-supervised and unsupervised objectives end up learning a representation that is also effective at semantic tasks [53]. Interestingly, recent findings in neuroscience make much the same point: representations trained on computer vision tasks also end up being effective models of neural activity in macaque visual cortex [8]. Moreover (and roughly speaking), the stronger the representation is at the computer vision task, the stronger it is as a model of cortical activity. Our work makes a similar finding: the stronger a feature set is at classification and detection, the stronger it is as a model of perceptual similarity judgments, as suggested in Table 6.4. Together, these results suggest that a good feature is a good feature. Features that are good at semantic tasks are also good at self-supervised and unsupervised tasks, and also provide good models of both human perceptual behavior and macaque neural activity. This last point aligns with the "rational analysis" explanation of visual cognition [198], suggesting that the idiosyncrasies of biological perception arise as a consequence of a rational agent attempting to solve natural tasks. Further refining the degree to which this is true is an important question for future research.

In Table 6.5, we show full quantitative results across all validation sets and considered metrics, including low-level metrics, along with random, unsupervised, self-supervised, supervised, and perceptually-learned networks.

In Figures 6.8, 6.9, 6.10, we plot performance in individual validation sets. Figure 6.8 shows our traditional and CNN-based distortions, and Figures 6.9, 6.10 show results on real algorithm applications individually.

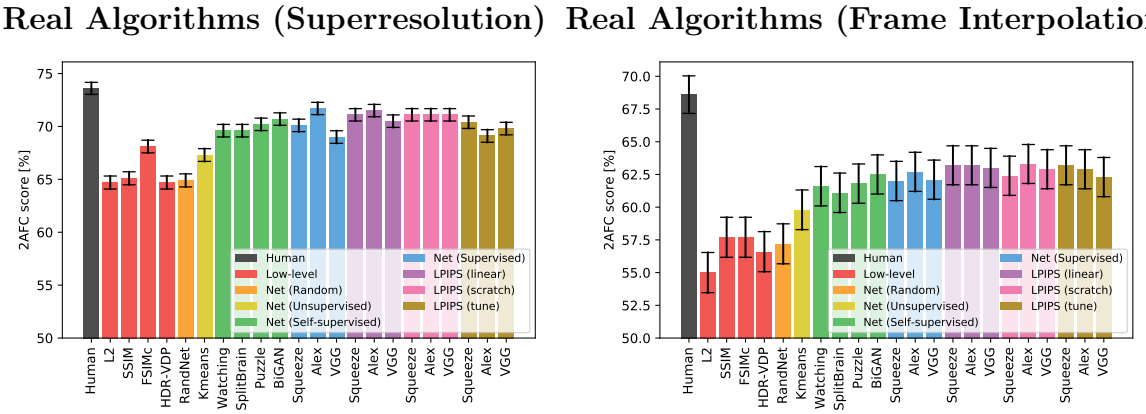



Figure 6.9: **Individual results** (left) superresolution (right) frame interpolation

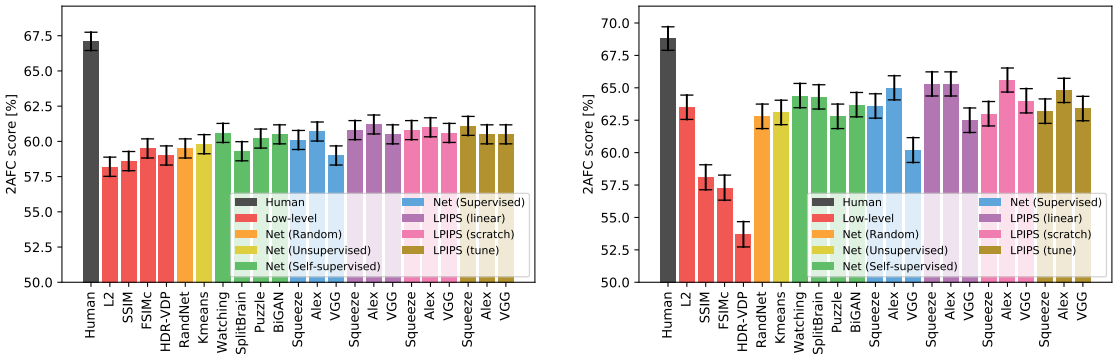



Figure 6.10: **Individual results** (left) video deblurring (right) colorization

**Data quantity for training models on distortions** The performance of the validation set on our distortions (80.6% and 81.4% for ***Alex*** – ***tune*** and ***VGG*** – ***tune***, respectively), is almost equal to human performance of 82.6%. This indicates that our training set size of 150k patch pairs and 300k judgments is nearly large enough to fully explore the traditional and CNN-based distortions which we defined. However, there is a small gap between the ***tune*** and ***scratch*** models (0.4% and 0.6% for ***Alex*** and ***VGG***, respectively).

| Subtype | Metric | Distortions | | | Real Algorithms | | | | | All |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Traditional | CNN-Based | All | Super-res | Video Deblur | Color-ization | Frame Interp | All | All |
| Oracle | Human | 80.8 | 84.4 | 82.6 | 73.4 | 67.1 | 68.8 | 68.6 | 69.5 | 73.9 |
| Low-level | L2 | 59.9 | 77.8 | 68.9 | 64.7 | 58.2 | 63.5 | 55.0 | 60.3 | 63.2 |
| | SSIM [15] | 60.3 | 79.1 | 69.7 | 65.1 | 58.6 | 58.1 | 57.7 | 59.8 | 63.1 |
| | FSIMc [17] | 61.4 | 78.6 | 70.0 | 68.1 | 59.5 | 57.3 | 57.7 | 60.6 | 63.8 |
| | HDR-VDP [18] | 57.4 | 76.8 | 67.1 | 64.7 | 59.0 | 53.7 | 56.6 | 58.5 | 61.4 |
| Net (Random) | Gaussian | 60.5 | 80.7 | 70.6 | 64.9 | 59.5 | 62.8 | 57.2 | 61.1 | 64.3 |
| Net (Unsupervised) | K-means [51] | 66.6 | ***83.0*** | 74.8 | 67.3 | 59.8 | 63.1 | 59.8 | 62.5 | 66.6 |
| Net (Self-supervised) | Watching [147] | 66.5 | 80.7 | 73.6 | 69.6 | 60.6 | 64.4 | 61.6 | 64.1 | 67.2 |
| | Split-Brain [173] | 69.5 | 81.4 | 75.5 | 69.6 | 59.3 | 64.3 | 61.1 | 63.6 | 67.5 |
| | Puzzle [145] | 71.5 | 82.0 | 76.8 | 70.2 | 60.2 | 62.8 | 61.8 | 63.8 | 68.1 |
| | BiGAN [102] | 69.8 | ***83.0*** | 76.4 | 70.7 | 60.5 | 63.7 | 62.5 | 64.4 | ***68.4*** |
| Net (Supervised) | SqueezeNet [171] | **73.3** | ***82.6*** | **78.0** | 70.1 | 60.1 | 63.6 | 62.0 | 64.0 | ***68.6*** |
| | AlexNet [139] | 70.6 | **83.1** | 76.8 | **71.7** | 60.7 | 65.0 | 62.7 | ***65.0*** | **68.9** |
| | VGG [24] | 70.1 | 81.3 | 75.7 | 69.0 | 59.0 | 60.2 | 62.1 | 62.6 | 67.0 |
| *LPIPS (Learned Perceptual Image Patch Similarity) | Squeeze – lin | 76.1 | 83.5 | 79.8 | 71.1 | ***60.8*** | ***65.3*** | ***63.2*** | ***65.1*** | 70.0 |
| | Alex – lin | 73.9 | 83.4 | 78.7 | ***71.5*** | **61.2** | ***65.3*** | ***63.2*** | **65.3** | 69.8 |
| | VGG – lin | 76.0 | 82.8 | 79.4 | 70.5 | 60.5 | 62.5 | ***63.0*** | 64.1 | 69.2 |
| | Squeeze – scratch | 74.9 | 83.1 | 79.0 | 71.1 | ***60.8*** | 63.0 | 62.4 | 64.3 | 69.2 |
| | Alex – scratch | 77.6 | 82.8 | 80.2 | 71.1 | ***61.0*** | **65.6** | **63.3** | ***65.2*** | **70.2** |
| | VGG – scratch | 77.9 | **83.7** | 80.8 | 71.1 | 60.6 | 64.0 | ***62.9*** | 64.6 | 70.0 |
| | Squeeze – tune | 76.7 | 83.2 | 79.9 | 70.4 | ***61.1*** | 63.2 | ***63.2*** | 64.5 | 69.6 |
| | Alex – tune | 77.7 | 83.5 | 80.6 | 69.1 | 60.5 | 64.8 | ***62.9*** | 64.3 | 69.7 |
| | VGG – tune | **79.3** | 83.5 | **81.4** | 69.8 | 60.5 | 63.4 | 62.3 | 64.0 | 69.8 |

Table 6.5: **Results**. We show 2AFC scores (higher is better) across a spectrum of methods and test sets. The **bolded & underlined** values are the highest performing. The ***bolded & italicized*** values are within 0.5% of highest. *LPIPS metrics are trained on the same traditional and CNN-based distortions, and as such have an advantage relative to other methods when testing on those same distortion types, even on unseen test images. These values are indicated by gray values. The best gray value per column is also **bolded**.

# Chapter 7

# Conclusions and Discussion

This thesis explored using deep networks for image synthesis, and investigated the visual representations learned from these tasks. Image synthesis is challenging, as the problem is often multimodal in nature. This thesis first descrxibed the problem of automatic and user-guided colorization in Chapters 2 [19] and 3 [54], respectively. In automatic colorization, multimodality was modeled by predicting the distribution of possible colors for each pixel. In user-guided colorization, the method incorporated intuitive ways for a human to provide input to the system, which helped resolve ambiguities. Chapter 4 [97] tackled the image-to-image translation problem in general. Ambiguity is resolved with a *learned* low-dimensional code, which described the possible variations of the output.

Chapter 5 investigated the representation learned by the colorization network. By solving the colorization task, the network learned a representation which transferred surprisingly well to high level semantic tasks. This approach could be generalized as a cross-channel encoder. Concatenating multiple cross-channel encoders into a split-brain autoencoder [173] boosted performance further. Distances in deep network embedding space corresponded well to low-level human perceptual judgments, as described in Chapter 6, even across supervisory signals.

Below, I describe some potential future directions, building on the insights from the work in this thesis.

**Finding the "Right" Self-Supervisory Task** This thesis found that bidirectional prediction in a cross-channel setting served as strong pretext task for self-supervised learning. While synthesizing pixels across channels works well, synthesizing a missing hole in an image, the inpainting task, does not perform as well [29]. However, context prediction [53, 145], foreground segmentation [147], and predicting rotations [199] do perform well. So why do some self-supervisory signals work well,

while others do not? One possible hypothesis is the mutual information shared between the two parts of the split-brain autoencoder, $L$ and $ab$ channels, is smaller than the information shared between the middle of an image and its surrounding, which forces a network to work harder and induces a more abstract representation. Formally investigating what makes certain signals or data splits form better representations is an interesting future direction.

Investigating the connections between self-supervised and unsupervised learning is a possible future direction as well. So far, self-supervised learning has involved coming up with clever tricks to induce labels. A few of these variants, for example our cross-channel prediction [173], as well as inpainting [29], model conditional probability $P(X_2|X_1)$ in some way. Similarly, Pixel-RNN/Pixel-CNN [109,110] model conditional probabilities $P(X_i|X_{0:i-1})$. By modeling the conditional probabilities for each pixel given all previous pixels, the model actually models the full joint probability $P(X)$ (as the conditionals can be composed into the joint by using chain rule). Generally, unsupervised learning involves maximizing the likelihood of the data $P(X)$. For example, VAEs maximize the data likelihood by maximizing the variational bound of the evidence lower bound [100]. The generative BiGAN [102]/ALI [103] model the full distribution and produce strong representations. It is currently unclear which leads to better representations – modeling the full joint distribution or a cleverly chosen subset, and may warrant further investigation.

**User Interactivity in Image-to-Image Translation** Chapter 3 defined a set of ways for a user to inject inputs to the system to customize a colorization, specifically sparse, pointwise color inputs. This can be thought of as a *fixed, sparse, interpretable* space which can express the possible outputs. However, this space was not easy to sample from, and the system relied on a user's guidance to generate possible outputs. Afterwards, Chapter 4 proposed the BicycleGAN system, which uses a *learned, dense, easy-to-sample* space. However, this space was not necessarily interpretable. What a user would really like is all the factors of variation to be accounted for, and easy "dials" to choose desired output. A longstanding goal in unsupervised learning is the idea of *disentanglement* [25], where the learned latent structure should decompose into separate explanatory factors of variation, freely discovered from data. A continuous latent space is currently used, parameterized as a latent isotropic Gaussian, partially due to the ease of the reparameterization trick, which allows us to backpropagate through a sampling step [100]. One possible avenue of exploration includes mixing discrete and continuous latent spaces, which may allow for easier clustering of discrete patterns. Another possible direction is to augment a dataset of real images using computer generated images to "guide" the learning, so that elements which should be factorized in the latent space are more easily able to do

so. The idea of learning from CG has been explored in the context of generating chairs and faces [200, 201]. For example, for night-to-day translation, two elements in the latent vector should correspond to the angular position of the sun relative to the scene. Given a graphics engine, one could simulate such variation, and enforce those variations to be completely captured with two latent elements, while leaving the other elements untouched. Additional steps to enabling user interactivity would make deep generative models more useful for practical applications.

# Bibliography

[1] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *ICML*, 2014.

[2] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *CVPR Workshop*, 2014.

[3] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *CVPR*, 2014.

[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *IJCV*, 2015.

[5] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[6] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *ICCV*, 2001.

[7] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *IJCV*, 2010.

[8] D. L. Yamins and J. J. DiCarlo, "Using goal-driven deep learning models to understand sensory cortex," *Nature neuroscience*, 2016.

[9] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *ECCV*, 2014.

[10] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Object detectors emerge in deep scene cnns," *ICLR*, 2015.

[11] A. Torralba and A. A. Efros, "Unbiased look at dataset bias," in *CVPR*, 2011.

[12] V. R. de Sa, "Learning classification with unlabeled data," *NIPS*, 1994.

[13] A. M. Turing, "Computing machinery and intelligence," in *Parsing the Turing Test*, 2009.

[14] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, 2006.

[15] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *TIP*, 2004.

[16] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *Signals, Systems and Computers*, 2004.

[17] L. Zhang, L. Zhang, X. Mou, and D. Zhang, "Fsim: A feature similarity index for image quality assessment," *TIP*, 2011.

[18] R. Mantiuk, K. J. Kim, A. G. Rempel, and W. Heidrich, "Hdr-vdp-2: A calibrated visual metric for visibility and quality predictions in all luminance conditions," in *TOG*, 2011.

[19] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," 2016.

[20] Z. Cheng, Q. Yang, and B. Sheng, "Deep colorization," in *ICCV*, 2015.

[21] R. Dahl, "Automatic colorization," in *http://tinyclouds.org/colorize/*, 2016.

[22] G. Charpiat, M. Hofmann, and B. Schölkopf, "Automatic image colorization via multimodal predictions," in *ECCV*, 2008.

[23] G. Ramanarayanan, J. Ferwerda, B. Walter, and K. Bala, "Visual equivalence: towards a new standard for image fidelity," *TOG*, 2007.

[24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv*, 2014.

[25] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *PAMI*, 2013.

[26] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *ICML*, 2011.

[27] P. Agrawal, J. Carreira, and J. Malik, "Learning to see by moving," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 37–45, 2015.

[28] D. Jayaraman and K. Grauman, "Learning image representations tied to ego-motion," in *ICCV*, 2015.

[29] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. Efros, "Context encoders: Feature learning by inpainting," in *CVPR*, 2016.

[30] W. Lotter, G. Kreiman, and D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," *ICLR*, 2017.

[31] A. Owens, P. Isola, J. McDermott, A. Torralba, E. H. Adelson, and W. T. Freeman, "Visually indicated sounds," *CVPR*, 2016.

[32] A. Owens, J. Wu, J. H. McDermott, W. T. Freeman, and A. Torralba, "Ambient sound provides supervision for visual learning," in *ECCV*, 2016.

[33] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *Computer Graphics and Interactive Techniques*, 2001.

[34] T. Welsh, M. Ashikhmin, and K. Mueller, "Transferring color to greyscale images," *TOG*, 2002.

[35] R. K. Gupta, A. Y.-S. Chia, D. Rajan, E. S. Ng, and H. Zhiyong, "Image colorization using similar images," in *ACM Conference on Multimedia*, 2012.

[36] X. Liu, L. Wan, Y. Qu, T.-T. Wong, S. Lin, C.-S. Leung, and P.-A. Heng, "Intrinsic colorization," *TOG*, 2008.

[37] A. Y.-S. Chia, S. Zhuo, R. K. Gupta, Y.-W. Tai, S.-Y. Cho, P. Tan, and S. Lin, "Semantic colorization with internet images," in *TOG*, 2011.

[38] A. Deshpande, J. Rock, and D. Forsyth, "Learning large-scale automatic image colorization," in *ICCV*, 2015.

[39] G. Larsson, M. Maire, and G. Shakhnarovich, "Learning representations for automatic colorization," in *ECCV*, 2016.

[40] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification," *TOG*, 2016.

[41] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Hypercolumns for object segmentation and fine-grained localization," in *CVPR*, 2015.

[42] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *arXiv*, 2016.

[43] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *ICLR*, 2016.

[44] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *NIPS*, 2014.

[45] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *ICMLR*, 2015.

[46] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *NIPS Workshop*, 2015.

[47] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *PAMI*, 2013.

[48] S. Kirkpatrick, M. P. Vecchi, *et al.*, "Optimization by simmulated annealing," *Science*, 1983.

[49] B. Efron, "Bootstrap methods: another look at the jackknife," in *Breakthroughs in Statistics*, pp. 569–593, Springer, 1992.

[50] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Conference on Multimedia*, 2014.

[51] P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell, "Data-dependent initializations of convolutional neural networks," in *ICLR*, 2016.

[52] A. Chakrabarti, "Color constancy by learning to predict chromaticity from luminance," in *NIPS*, 2015.

[53] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *ICCV*, 2015.

[54] R. Zhang, J.-Y. Zhu, P. Isola, X. Geng, A. S. Lin, T. Yu, and A. A. Efros, "Real-time user-guided image colorization with learned deep priors," *ACM Transactions on Graphics (TOG)*, 2017.

[55] A. Levin, D. Lischinski, and Y. Weiss, "Colorization using optimization," *TOG*, 2004.

[56] Y.-C. Huang, Y.-S. Tung, J.-C. Chen, S.-W. Wang, and J.-L. Wu, "An adaptive edge detection based colorization algorithm and its applications," in *ACM Conference on Multimedia*, 2005.

[57] Y. Qu, T.-T. Wong, and P.-A. Heng, "Manga colorization," *TOG*, 2006.

[58] Q. Luan, F. Wen, D. Cohen-Or, L. Liang, Y.-Q. Xu, and H.-Y. Shum, "Natural image colorization," in *Eurographics Conference on Rendering Techniques*, 2007.

[59] X. An and F. Pellacini, "Appprop: all-pairs appearance-space edit propagation," in *ACM Transactions on Graphics (TOG)*, 2008.

[60] K. Xu, Y. Li, T. Ju, S.-M. Hu, and T.-Q. Liu, "Efficient affinity-based edit propagation using kd tree," *ACM Transactions on Graphics (TOG)*, 2009.

[61] Y. Li, E. Adelson, and A. Agarwala, "Scribbleboost: Adding classification to edge-aware interpolation of local image and video adjustments," in *CGF*, 2008.

[62] X. Chen, D. Zou, Q. Zhao, and P. Tan, "Manifold preserving edit propagation," *ACM Transactions on Graphics (TOG)*, 2012.

[63] L. Xu, Q. Yan, and J. Jia, "A sparse control model for image and video editing," *ACM Transactions on Graphics (TOG)*, 2013.

[64] Y. Endo, S. Iizuka, Y. Kanamori, and J. Mitani, "Deepprop: Extracting deep features from a single image for edit propagation," in *Computer Graphics Forum*, 2016.

[65] B. Wang, Y. Yu, T.-T. Wong, C. Chen, and Y.-Q. Xu, "Data-driven image color theme enhancement," in *ACM Transactions on Graphics (TOG)*, 2010.

[66] X. Li, H. Zhao, G. Nie, and H. Huang, "Image recoloring using geodesic distance based color harmonization," *Computational Visual Media*, 2015.

[67] H. Chang, O. Fried, Y. Liu, S. DiVerdi, and A. Finkelstein, "Palette-based photo recoloring," *TOG*, 2015.

[68] P. Sangkloy, J. Lu, C. Fang, F. Yu, and J. Hays, "Scribbler: Controlling deep image synthesis with sketch and color," in *CVPR*, 2017.

[69] P. N. Inc, "Paints chainer," 2017.

[70] K. Frans, "Outline colorization through tandem adversarial networks," in *arXiv*, 2017.

[71] R. Irony, D. Cohen-Or, and D. Lischinski, "Colorization by example," in *Eurographics Symposium on Rendering*, 2005.

[72] Y. Morimoto, Y. Taguchi, and T. Naemura, "Automatic colorization of grayscale images using multiple images on the web," in *SIGGRAPH Posters*, 2009.

[73] Y. Liu, M. Cohen, M. Uyttendaele, and S. Rusinkiewicz, "Autostyle: automatic style transfer from image collections to users' images," in *Computer Graphics Forum*, 2014.

[74] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley, "Color transfer between images," *Computer Graphics and Applications*, 2001.

[75] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *CVPR*, 2016.

[76] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[77] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[78] S. Bell, P. Upchurch, N. Snavely, and K. Bala, "Material recognition in the wild with the materials in context database," in *CVPR*, 2015.

[79] T.-C. Wang, J.-Y. Zhu, E. Hiroaki, M. Chandraker, A. A. Efros, and R. Ramamoorthi, "A 4d light-field dataset and cnn architectures for material recognition," in *ECCV*, 2016.

[80] S. Xie and Z. Tu, "Holistically-nested edge detection," in *ICCV*, 2015.

[81] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, 2014.

[82] Z. Yan, H. Zhang, B. Wang, S. Paris, and Y. Yu, "Automatic photo adjustment using deep neural networks," *TOG*, 2016.

[83] E. Simo-Serra, S. Iizuka, K. Sasaki, and H. Ishikawa, "Learning to simplify: fully convolutional networks for rough sketch cleanup," *ACM Transactions on Graphics (TOG)*, 2016.

[84] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *CVPR*, 2016.

[85] A. Selim, M. Elgharib, and L. Doyle, "Painting style transfer for head portraits using convolutional neural networks," *ACM Transactions on Graphics (TOG)*, 2016.

[86] J.-Y. Zhu, P. Krahenbuhl, E. Shechtman, and A. A. Efros, "Learning a discriminative model for the perception of realism in composite images," in *CVPR*, 2015.

[87] M. Gharbi, G. Chaurasia, S. Paris, and F. Durand, "Deep joint demosaicking and denoising," *ACM Transactions on Graphics (TOG)*, 2016.

[88] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman, "Patchmatch: a randomized correspondence algorithm for structural image editing," *ACM Transactions on Graphics (TOG)*, 2009.

[89] J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros, "Generative visual manipulation on the natural image manifold," in *ECCV*, 2016.

[90] N. Xu, B. Price, S. Cohen, J. Yang, and T. Huang, "Deep interactive object selection," *CVPR*, 2016.

[91] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014.

[92] P. J. Huber, "Robust estimation of a location parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, 1964.

[93] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *MICCAI*, 2015.

[94] J. T. Barron and B. Poole, "The fast bilateral solver," *ECCV*, 2016.

[95] L. Cheng and S. Vishwanathan, "Learning to compress images and videos," in *ICML*, 2007.

[96] X. He, M. Ji, and H. Bao, "A unified active and semi-supervised learning framework for image compression," in *CVPR*, 2009.

[97] J.-Y. Zhu, R. Zhang, D. Pathak, T. Darrell, A. A. Efros, O. Wang, and E. Shechtman, "Toward multimodal image-to-image translation," in *NIPS*, 2017.

[98] C. Yang, X. Lu, Z. Lin, E. Shechtman, O. Wang, and H. Li, "High-resolution image inpainting using multi-scale neural patch synthesis," in *CVPR*, 2017.

[99] I. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," *arXiv*, 2016.

[100] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *ICLR*, 2014.

[101] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," in *ICML*, 2016.

[102] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial feature learning," in *ICLR*, 2017.

[103] V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. Courville, "Adversarially learned inference," in *ICLR*, 2017.

[104] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: interpretable representation learning by information maximizing generative adversarial nets," in *NIPS*, 2016.

[105] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *CVPR*, 2018.

[106] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," tech. rep., DTIC Document, 1986.

[107] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *ICML*, 2008.

[108] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling," in *ICCV*, 1999.

[109] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *ICML*, 2016.

[110] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional image generation with pixelcnn decoders," in *NIPS*, 2016.

[111] E. L. Denton, S. Chintala, A. Szlam, and R. Fergus, "Deep generative image models using a laplacian pyramid of adversarial networks," in *NIPS*, 2015.

[112] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *ICLR*, 2016.

[113] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text-to-image synthesis," in *ICML*, 2016.

[114] J. Zhao, M. Mathieu, and Y. LeCun, "Energy-based generative adversarial network," in *ICLR*, 2017.

[115] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," in *ICLR*, 2017.

[116] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," in *ICCV*, 2017.

[117] K. Sohn, X. Yan, and H. Lee, "Learning structured output representation using deep conditional generative models," in *NIPS*, 2015.

[118] J. Walker, C. Doersch, A. Gupta, and M. Hebert, "An uncertain future: Forecasting from static images using variational autoencoders," in *ECCV*, 2016.

[119] T. Xue, J. Wu, K. Bouman, and B. Freeman, "Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks," in *NIPS*, 2016.

[120] S. Guadarrama, R. Dahl, D. Bieber, M. Norouzi, J. Shlens, and K. Murphy, "Pixcolor: Pixel recursive colorization," in *BMVC*, 2017.

[121] W. Xian, P. Sangkloy, J. Lu, C. Fang, F. Yu, and J. Hays, "Texturegan: Controlling deep image synthesis with texture patches," in *CVPR*, 2018.

[122] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *ICCV*, 2017.

[123] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," in *ICLR*, 2016.

[124] A. Ghosh, V. Kulharia, V. Namboodiri, P. H. Torr, and P. K. Dokania, "Multi-agent diverse generative adversarial networks," *CVPR*, 2018.

[125] Q. Chen and V. Koltun, "Photographic image synthesis with cascaded refinement networks," in *ICCV*, 2017.

[126] A. Bansal, Y. Sheikh, and D. Ramanan, "Pixelnn: Example-based image synthesis," in *ICLR*, 2018.

[127] A. Nguyen, J. Yosinski, Y. Bengio, A. Dosovitskiy, and J. Clune, "Plug & play generative networks: Conditional iterative generation of images in latent space," in *CVPR*, 2017.

[128] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *ICLR*, 2017.

[129] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[130] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *arXiv preprint arXiv:1606.03498*, 2016.

[131] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[132] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley, "Least squares generative adversarial networks," in *ICCV*, 2017.

[133] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2014.

[134] A. Yu and K. Grauman, "Fine-grained visual comparisons with local learning," in *CVPR*, 2014.

[135] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *CVPR*, 2016.

[136] P.-Y. Laffont, Z. Ren, X. Tao, C. Qian, and J. Hays, "Transient attributes for high-level understanding and editing of outdoor scenes," *SIGGRAPH*, 2014.

[137] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," 2016.

[138] A. Dosovitskiy and T. Brox, "Generating images with perceptual similarity metrics based on deep networks," in *NIPS*, 2016.

[139] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," 2014.

[140] G. Larsson, M. Maire, and G. Shakhnarovich, "Colorization as a proxy task for visual understanding," *CVPR*, 2017.

[141] R. Salakhutdinov and G. E. Hinton, "Deep boltzmann machines," in *AISTATS*, 2009.

[142] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," in *ICCV*, 2015.

[143] I. Misra, C. L. Zitnick, and M. Hebert, "Shuffle and learn: unsupervised learning using temporal order verification," in *ECCV*, 2016.

[144] P. Isola, D. Zoran, D. Krishnan, and E. H. Adelson, "Learning visual groups from co-occurrences in space and time," *ICLR Workshop*, 2016.

[145] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," *ECCV*, 2016.

[146] V. R. De Sa, "Sensory modality segregation," in *NIPS*, 2003.

[147] D. Pathak, R. Girshick, P. Dollár, T. Darrell, and B. Hariharan, "Learning features by watching objects move," *CVPR*, 2017.

[148] R. K. Ando and T. Zhang, "A framework for learning predictive structures from multiple tasks and unlabeled data," *JMLR*, 2005.

[149] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Computational learning theory*, 1998.

[150] C. Xu, D. Tao, and C. Xu, "A survey on multi-view learning," *arXiv preprint arXiv:1304.5634*, 2013.

[151] K. Sohn, W. Shang, and H. Lee, "Improved multimodal deep learning with variation of information," in *NIPS*, 2014.

[152] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *ICCV*, 2015.

[153] X. Wang, D. Fouhey, and A. Gupta, "Designing deep networks for surface normal estimation," in *CVPR*, 2015.

[154] S. Gupta, J. Hoffman, and J. Malik, "Cross modal distillation for supervision transfer," in *CVPR*, June 2016.

[155] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, 2015.

[156] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CVPR*, 2016.

[157] R. Girshick, "Fast r-cnn," in *ICCV*, 2015.

[158] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results." http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[159] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results." http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[160] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik, "Learning rich features from rgb-d images for object detection and segmentation," in *ECCV*, pp. 345–360, Springer, 2014.

[161] N. Goodman, "Seven strictures on similarity," *Problems and Projects*, 1972.

[162] D. L. Medin, R. L. Goldstone, and D. Gentner, "Respects for similarity.," *Psychological review*, vol. 100, no. 2, p. 254, 1993.

[163] A. B. Markman and D. Gentner, "Nonintentional similarity processing," *The new unconscious*, pp. 107–137, 2005.

[164] A. Tversky, "Features of similarity.," *Psychological review*, 1977.

[165] H. R. Sheikh, M. F. Sabir, and A. C. Bovik, "A statistical evaluation of recent full reference image quality assessment algorithms," *TIP*, 2006.

[166] E. C. Larson and D. M. Chandler, "Most apparent distortion: full-reference image quality assessment and the role of strategy," *Journal of Electronic Imaging*, 2010.

[167] N. Ponomarenko, V. Lukin, A. Zelensky, K. Egiazarian, M. Carli, and F. Battisti, "Tid2008-a database for evaluation of full-reference visual quality assessment metrics," *Advances of Modern Radioelectronics*, 2009.

[168] N. Ponomarenko, L. Jin, O. Ieremeiev, V. Lukin, K. Egiazarian, J. Astola, B. Vozel, K. Chehdi, M. Carli, F. Battisti, *et al.*, "Image database tid2013: Peculiarities, results and perspectives," *Signal Processing: Image Communication*, 2015.

[169] V. Bychkovsky, S. Paris, E. Chan, and F. Durand, "Learning photographic global tonal adjustment with a database of input / output image pairs," in *CVPR*, 2011.

[170] D.-T. Dang-Nguyen, C. Pasquini, V. Conotter, and G. Boato, "Raise: a raw images dataset for digital image forensics," in *ACM Multimedia Systems Conference*, 2015.

[171] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5 mb model size," in *CVPR*, 2017.

[172] M. P. Sampat, Z. Wang, S. Gupta, A. C. Bovik, and M. K. Markey, "Complex wavelet structural similarity: A new image similarity index," *TIP*, 2009.

[173] R. Zhang, P. Isola, and A. A. Efros, "Split-brain autoencoders: Unsupervised learning by cross-channel prediction," in *CVPR*, 2017.

[174] N. Murray, L. Marchesotti, and F. Perronnin, "Ava: A large-scale database for aesthetic visual analysis," in *CVPR*, 2012.

[175] D. Ghadiyaram and A. C. Bovik, "Massive online crowdsourced study of subjective and objective picture quality," *TIP*, 2016.

[176] J. Kim and S. Lee, "Deep learning of human visual sensitivity in image quality assessment framework," in *CVPR*, 2017.

[177] H. Talebi and P. Milanfar, "Learned perceptual image enhancement," *ICCV*, 2017.

[178] H. Talebi and P. Milanfar, "Nima: Neural image assessment," *TIP*, 2018.

[179] F. Gao, Y. Wang, P. Li, M. Tan, J. Yu, and Y. Zhu, "Deepsim: Deep similarity for image quality assessment," *Neurocomputing*, 2017.

[180] S. Ali Amirshahi, M. Pedersen, and S. X. Yu, "Image quality assessment by comparing cnn features between images," *Electronic Imaging*, 2017.

[181] A. Berardino, V. Laparra, J. Ballé, and E. Simoncelli, "Eigen-distortions of hierarchical representations," in *NIPS*, 2017.

[182] E. Agustsson and R. Timofte, "Ntire 2017 challenge on single image super-resolution: Dataset and study," in *CVPR Workshops*, July 2017.

[183] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *CVPR*, 2016.

[184] Z. Wang, D. Liu, J. Yang, W. Han, and T. Huang, "Deep networks for image super-resolution with sparse prior," in *ICCV*, 2015.

[185] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *CVPR*, 2016.

[186] M. S. Sajjadi, B. Schölkopf, and M. Hirsch, "Enhancenet: Single image super-resolution through automated texture synthesis," *ICCV*, 2017.

[187] C. Liu *et al.*, *Beyond pixels: exploring new representations and applications for motion analysis*. PhD thesis, Massachusetts Institute of Technology, 2009.

[188] S. Niklaus, L. Mai, and F. Liu, "Video frame interpolation via adaptive separable convolution," in *ICCV*, 2017.

[189] S. Meyer, O. Wang, H. Zimmer, M. Grosse, and A. Sorkine-Hornung, "Phase-based frame interpolation for video," in *CVPR*, 2015.

[190] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *IJCV*, 2002.

[191] S. Su, M. Delbracio, J. Wang, G. Sapiro, W. Heidrich, and O. Wang, "Deep video deblurring for hand-held cameras," in *CVPR*, 2017.

[192] M. Delbracio and G. Sapiro, "Hand-held video deblurring via efficient fourier aggregation," *Transactions on Computational Imaging*, 2015.

[193] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *CVPR*, 2017.

[194] M. J. Crump, J. V. McDonnell, and T. M. Gureckis, "Evaluating amazon's mechanical turk as a tool for experimental behavioral research," *PloS one*, 2013.

[195] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," in *CVPR Workshops*, 2017.

[196] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *IJCV*, 2011.

[197] A. Paszke, S. Chintala, R. Collobert, K. Kavukcuoglu, C. Farabet, S. Bengio, I. Melvin, J. Weston, and J. Mariethoz, "Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration, may 2017."

[198] J. R. Anderson, *The adaptive character of thought*. Psychology Press, 1990.

[199] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," *ICLR*, 2018.

[200] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum, "Deep convolutional inverse graphics network," in *NIPS*, 2015.

[201] A. Dosovitskiy, J. T. Springenberg, and T. Brox, "Learning to generate chairs with convolutional neural networks," in *CVPR*, 2015.