STRUCTURE GUIDED IMAGE RESTORATION
A DEEP LEARNING APPROACH

by

Kamyar Nazeri Naeini

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Science

in

The Faculty of Science

Modelling and Computational Science

University of Ontario Institute of Technology

Oshawa, Ontario, Canada

May 2019

THESIS EXAMINATION INFORMATION

Submitted by: **Kamyar Nazeri Naeini**

Master of Science in **Modelling and Computational Science**

Thesis title: Structure Guided Image Restoration: A Deep Learning Approach

An oral defense of this thesis took place on May 29, 2019 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Lennaert van Veen
Research Supervisor	Dr. Mehran Ebrahimi
Examining Committee Member	Dr. Faisal Qureshi
External Examiner	Dr. Ken Pu

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

Abstract

Image restoration aims at recovery of degraded images and estimating the original. Over the past few years, computer vision research has been dominated by deep learning techniques in part due to advances in computing infrastructure, algorithms and image capturing devices. As a result, deep neural networks currently set the state-of-the-art in image restoration problems. However, many of these techniques fail to reconstruct reasonable structures as they are commonly over-smoothed and/or blurry.

In this dissertation, we develop models based on deep convolutional neural networks to address two image restoration problems: image inpainting and image super-resolution. We develop a new approach for image inpainting that does a better job of reproducing missing regions exhibiting fine details. Furthermore, we extend this method to image super-resolution by reformulating the problem as an in-between pixels inpainting task. We propose a two-stage adversarial model that comprises of an edge generator followed by an image completion network. The edge generator hallucinates edges of the missing region of the image, and the image completion network fills in the missing regions using hallucinated edges as a priori. We evaluate our model over the publicly available datasets and show that it outperforms current state-of-the-art techniques quantitatively and qualitatively.

Author's Declaration

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize University of Ontario Institute of Technology to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

Statement of Contribution

Part of the work described in Chapter 4 has been published as:

K. Nazeri, E. Ng, T. Joseph, F. Qureshi, and M. Ebrahimi, "Edgeconnect: Generative image inpainting with adversarial edge learning," *arXiv preprint arXiv:1901.00212*, 2019.

As the first author of the paper, I performed the majority of the experiments, design of the model, and writing of the manuscript.

Acknowledgements

For me, grad school has been an ample opportunity to interact with so many dedicated and talented people. I have been fortunate to work with amazing mentors, teachers, and friends at Ontario Tech University and I would like to thank all of them for contributing to the two wonderful years of my experience as a Master student.

I would especially like to thank my supervisor Dr. Mehran Ebrahimi for his help and support throughout this work. He helped me entering the world of research and through countless meetings over two years taught me not just aspects of mathematical image processing but how to be a researcher and above all how to think. I very much appreciate the freedom he gave me to explore my own research interests and work on an array of projects which eventually led to this work. He was not only a great supervisor but also a great friend to me and I am forever grateful for all I have learned from him.

My sincere gratitude goes to Dr. Faisal Z. Qureshi, member of my graduate committee, who fueled so much excitement and progress in my research and for his help and advice throughout the time I spent at Ontario Tech. I have benefited a lot from discussions with him and his advice regarding this work.

I would like to acknowledge my fellow colleagues and friends in the Modelling and Computational Science program, especially those in Imaging Lab for their friendship and so many good memories throughout the time I spent at Ontario Tech. Thanks for all the support and the adventures.

I will always be grateful to my mother, for raising me to value education, for the encouragement she gave me and for teaching me to be strong.

Contents

Abstract iii

Acknowledgements vi

List of Tables x

List of Figures xii

List of Symbols xix

1 Introduction 1

1.1 Overview 1

1.2 Contribution of This Work 5

1.3 Thesis Outline 6

1.4 Software, Open Data & Source Code 7

2 Deep Learning Background 9

2.1 Supervised/Unsupervised Learning 10

2.2 Optimization 14

2.2.1 Backpropagation 18

2.3 Neural Networks 20

2.3.1 Feedforward Neural Networks 20

2.3.2 Convolutional Neural Networks 22

CNN Building Blocks 24

Improved CNN Architectures for Image Generation 27

2.3.3 Generative Adversarial Networks 34

GANs Objective 39

Deep Convolutional GANs . . . 43
Improved GANs . . . 45
2.4 Summary . . . 50

3 Image Structures & Evaluations 51
3.1 Image Structures . . . 52
3.2 Edge Detection . . . 54
3.2.1 Gradient-Based Edge Detections . . . 56
3.2.2 Canny Edge Detection . . . 62
3.2.3 Learning-Based Edge Detections . . . 65
3.3 Image Quality Assessments and Similarity Metrics . . . 68
3.3.1 Mean Absolute Error . . . 69
3.3.2 Peak Signal to Noise Ratio (PSNR) . . . 71
3.3.3 Structured Similarity (SSIM) . . . 72
3.3.4 Deep Features as Perceptual Metric . . . 74
Perceptual Losses . . . 75
Fréchet Inception Distance . . . 78
3.3.5 Human Study & Psychophysical Similarity Measurements . . . 80
Two-Alternative Forced Choice (2AFC) . . . 81
Just Noticeable Differences (JND) . . . 81
3.4 Summary . . . 83

4 Image Inpainting 85
4.1 Introduction . . . 86
4.2 Related Work . . . 88
4.2.1 Diffusion-Based Inpainting . . . 88
4.2.2 Patch-Based Inpainting . . . 88
4.2.3 Learning-Based Inpainting . . . 89
4.2.4 Image-to-Edges vs. Edges-to-Image . . . 90
4.3 Model . . . 91
4.3.1 Edge Generation . . . 92
4.3.2 Image Completion . . . 94
4.3.3 Network Architecture . . . 96

4.3.4 Training . . . 97
Edge Information and Image Masks . . . 97
Training Setup and Strategy . . . 97
4.4 Experiments . . . 98
4.4.1 Datasets . . . 98
4.4.2 Qualitative Evaluation . . . 99
4.4.3 Quantitative Evaluation . . . 102
Inpainting Numerical Metrics . . . 102
Visual Turing Tests . . . 107
Accuracy of Edge Generator . . . 108
4.4.4 Ablative Study . . . 110
Quantity of Edges versus Inpainting Quality . . . 110
Alternative Edge Detection Systems . . . 111
4.4.5 Applications . . . 115
4.5 Summary . . . 117

5 Single Image Super-Resolution 118
5.1 Introduction . . . 119
5.2 Related Work . . . 120
5.3 Model . . . 122
5.4 Experiments . . . 124
5.4.1 Qualitative Evaluation . . . 125
5.4.2 Quantitative Evaluation . . . 125
Accuracy of Edge Generator . . . 129
5.5 Summary . . . 130

6 Conclusions 131

A Inpainting Results 134

Bibliography 134

List of Tables

4.1 Comparison of different approaches for image inpainting. Diffusion-based methods propagate background data into the missing region by following a diffusive process. Patch-based methods fill in missing regions with patches from a collection of source images that maximize patch similarity and provide better inpainting quality. Learning-based methods fill the missing pixels using learned data distribution and are superior to classical methods in every aspect. 87

4.2 Comparison of quantitative results (256×256) over **Places2** with CA [1], GLCIC [2], PConv [3], Ours (end-to-end). The best result of each row is boldfaced. †Lower is better. ⋆Higher is better. 103

4.3 Comparison of quantitative results (256×256) over **CelebA** with CA [1], GLCIC [2], PConv [3], Ours (end-to-end). The best result of each row is boldfaced. †Lower is better. ⋆Higher is better. 104

4.4 Comparison of quantitative results (256×256) over **Paris StreetView** with CA [1], GLCIC [2], PConv [3], Ours (end-to-end). The best result of each row is boldfaced. †Lower is better. ⋆Higher is better. 105

4.5 Comparison of Y-N and JND scores for various mask sizes on **Places2** with CA [1], GLCIC [2], PConv [3], and Ours. Y-N score for ground truth images is 94.6%.
108

4.6 Quantitative performance of edge generator for inpainting trained on Canny edges with $\sigma = 2$ for 256×256 images. Statistics are calculated over the standard test sets of each dataset 109

4.7 Comparison of inpainting results with edge information (our full model) and without edge information (G_2 only, trained without edges). Statistics are based on $10,000$ random masks with size 40-50% of the entire image. . 110
4.8 Comparison of quantitative results between Hybrid (HED$\odot$Canny) and Canny edges over CelebA. Statistics are shown for generated edges (G_1) and ground truth edges (GT). †Lower is better. *Higher is better. 114

5.1 Comparison of PSNR and SSIM for $\times 2$, $\times 4$, and $\times 8$ factor SISR over **Set5**, **Set14**, **BSD100**, and **Celeb-HQ** datasets with Bicubic interpolation, ENet [4], EDSR [5], and baseline (without edge-data). The best result of each row is boldfaced. 128
5.2 Quantitative performance of edge enhancer for Single Image Super-Resolution trained on Canny edges with $\sigma = 2$ for 512×512 images. Statistics are calculated over the standard test sets of each dataset. 129

List of Figures

2.1 Schematic overview of backpropagation in a simple computational graph. During forward pass, vectors $\mathbf{x}$ and $\mathbf{y}$ are inputs to a node that performs some fixed computation on its inputs producing vector $\mathbf{z}$. Note that the we can compute the Jacobian matrices $\frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ and $\frac{\partial \mathbf{z}}{\partial \mathbf{y}}$ for the node at this stage. The output $\mathbf{z}$ flows further to the graph where at the end we calculate a loss using a differentiable scalar-valued function $\mathcal{L}$. The backward pass proceeds in the reverse order, effectively calculating the gradient of the loss with respect to all the elements in the graph using the chain rule. The gradient of the loss with respect to the vector $\mathbf{z}$ is calculated $\frac{\partial \mathcal{L}}{\partial \mathbf{z}}$ and gets multiplied with the local gradients calculated during the forward pass to find the global gradient of the loss with respect to the inputs $\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \frac{\partial \mathcal{L}}{\partial \mathbf{z}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{y}} = \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \frac{\partial \mathcal{L}}{\partial \mathbf{z}}$. In a neural network, each node contains parameters, where the gradient with respect to each parameter tells us how they should be changed to minimize the loss. 19

2.2 **Left:** schematic view of one neuron in a neural network. The neuron computes the weighted sum of its inputs followed by a non-linear function. **Right:** an example of a 3-layer neural network(multi-layer perceptron). Neurons in each layer are connected to all neurons in the previous layer. At each layer, the output activations are efficiently evaluated using matrix multiplication between the input and the weights matrices. 21

2.3 Schematic view of a convolutional neural network for an image classification task. An image is an input to the network. At each layer, the input is convolved with the convolutional kernels to create activation maps for the next layer. The output of the network is a categorical probability distribution and a loss function is being used during training to find the values of the convolutional kernels. 23

2.4 Convolutional layers with local receptive field extract local features in a hierarchical order. 24

2.5 Illustration of convolving four $5 \times 5 \times 3$ filter over a $32 \times 32 \times 3$ input image with stride 1 and no input padding. Each convolution filter is expanded with the same number of channels as the input. In total four convolutional filters exists, each expanded with 3 channels producing a four channel $28 \times 28 \times 4$ activation map. 26

2.6 Receptive field expansion with dilated convolution, after applying dilated factors of 1, 2, and 4. **a)** Normal 3×3 1-dilated convolution filter has a receptive field of 3×3. **b)** 2-dilated convolution filter applied on (a) increases the receptive field to 7×7. **c)** 4-dilated convolution filter applied on (b) increases the receptive field to 15×15. All convolutional filters have identical number of parameters. Figure ©Yu *et al*. [6] *Multi-Scale Context Aggregation by Dilated Convolutions.* 30

2.7 A building block of residual learning. Figure ©He *et al*. [7] *Deep Residual Learning for Image Recognition.* 32

2.8 Comparison of different normalization schemes. In each case, N represents the mini-batch axis, C is the channel axis and (H, W) are the spatial axes. The pixels colored in blue are normalized by computing the statistics (mean and variance) of these pixels. Figure ©Wu *et al*. [8] *Deep Residual Learning for Image Recognition.* 34

2.9 GAN framework overview. The generator network (G) directly produces samples by sampling from a prior distribution p_z. The discriminator (D) attempts to distinguish between real samples from a training set, which are labeled as 1, and samples generated by the generator which are labeled az 0. During learning, each network attempts to maximize its own performance and undo the other in a zero-sum game. 37

2.10 Example of the minimax objective in a two-player game. P1 tries to minimize the possible loss for a worst case scenario in three steps: Step 1) P1 can make three moves and predicts three countermoves for each of them, the score associated to each countermove is also calculated. Step 2) Maximum score (P1 loss) for each move is calculated. Step 3) P1 makes the move that has minimum value among all the values calculated in Step 2. . . 38

2.11 The original loss function for generator, shown in dashed red, provides small gradient for the samples that are not good ($D(G(z)) \approx 0$) and large gradient for the good samples ($D(G(z)) \approx 1$). The heuristic non-saturating loss function for generator, shown in green, is proposed by Goodfellow *et al*. [9], changes the generator loss from minimizing the log probability of the correct answer ($\log(1 - D(G(z)))$) to maximize the log probability of the wrong answer ($-\log(D(G(z)))$). 41

2.12 Visualization of the Sigmoid function. The function gets saturated with very large or small values where the gradient becomes very small. 42

3.1 Different visual components in a color image. From left to right: 1) Original color image, 2) Edge map as discontinuities in the image brightness, 3) Color as the visible spectral distribution, and 4) Texture that describes properties such as smoothness, coarseness, and regularity of the surface. . . 53

3.2 Different edge detection algorithms. From left to right: a) Original color image, b) Contour map extracted using Suzuki *et al*. [10] method, c) Edge map retrieved using Canny edge detection [11], d) Gradient magnitude after applying Sobel operator [12]. 55

3.3 Convolution kernels to approximate first order partial derivatives in an image. 56

3.4 Visual comparison of first order partial derivatives of an image using convolution filters. (a) original image, (b) derivative along x axis captures vertical edges, (c) derivative along y axis captures horizontal edges. 57

3.5 Finite difference filters used to approximate derivative. Prewitt operator [13] de-emphasizes values near the center. The Sobel operator [12] gives more emphasis to changes around the center pixel. Note that these filters sum to zero. 58

3.6 Visual comparison of different derivative-based edge detectors. Edges are acquired by applying convolution operator on the original image using various linear difference filters. 60

3.7 Non-maximum suppression procedure: a pixel is checked if it is a local maximum in its neighborhood along the direction of edge normal. p_5 is maximum in all cases. 64

3.8 Edge-maps generated by Canny edge detector for different values of Gaussian width σ. Increasing σ smooths the image and reduces the amount of edge. 65

3.9 Choosing the patch which is more "similar" to the reference in the middle. In each case, classical similarity metrics (L1/L2, PSNR, SSIM) disagree with human perceptual judgment. Features extracted from deep networks trained for different tasks and level of supervision (supervised, unsupervised, and self-supervised) agree with human visual perception. Figure ©Zang *et al*. [14] *The unreasonable effectiveness of deep features as a perceptual metric.* 75

3.10 Left to right: FID evaluated for Gaussian noise, Gaussian blur, and implanted black rectangles on images from CelebA dataset. The degradations level starts from zero and increased to highest value. In each case, FID shows a monotonically increasing behavior and captures the distortions very well. Figure ©Heusel *et al*. [15] *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium.* 79

3.11 Two-alternative forced choice example for comparing two deblurring algorithms. Image1 (left) and Image2 (right) are the results of two different deblurring methods and the reference image (center) is the ground truth. Participants are asked to choose the image that looks more similar to the reference. In each test, the position of Image1 and Image2 are selected randomly to prevent bias. 82

3.12 Just noticeable differences example to evaluate a deblurring algorithm. For each test, the result of a deblurring algorithm (left) and a reference image (right) are randomly positioned and the participants are asked to choose the image that looks more real. 83

4.1 (Left) Input images with missing regions. The missing regions are depicted in white. (Center) Computed edge masks. Edges drawn in black are computed (for the available regions) using Canny edge detector; whereas edges shown in blue are hallucinated (for the missing regions) by the edge generator network. (Right) Image inpainting results of the proposed approach. . . 86

4.2 Summary of our proposed method. Incomplete grayscale image and edge map, and mask are the inputs of G_1 to predict the full edge map. Predicted edge map and incomplete color image are passed to G_2 to perform the inpainting task.
92

4.3 Comparison of qualitative results of 512×512 image inpainting with existing models. From left to right: Ground Truth, Masked Image, Iizuka *et al*. [2] (Globally and Locally Image Completion), Yu *et al*. [1] (Contextual Attention), Liu *et al*. (Partial Convolution) [3], Baseline (no edge data, G_2 only), Ours (Full Model). 99

4.4 Qualitative results of 512×512 image inpainting. (Left to Right) Original image, input image, generated edges, inpainted results without any post-processing. 100

4.5 Effect of mask sizes on ℓ_1, SSIM, PSNR, and FID for Places2 dataset. . . . 106

4.6 Effect of relative mask sizes on ℓ_1, SSIM, PSNR, and FID for CelebA dataset. 106

4.7 Effect of relative mask sizes on ℓ_1, SSIM, PSNR, and FID for Paris StreetView. 107

4.8 Effect of σ in Canny detector on PSNR and FID. 111

4.9 Effect of σ in Canny edge detector on inpainting results. Top to bottom: $\sigma = 1, 3, 5$, no edge data. 112

4.10 (a) Image. (b) Canny. (c) HED. (d) Canny$\odot$HED. 113

4.11 Generated edges by G_1 trained using hybrid (HED$\odot$Canny) edges. Images are best viewed in color. (a) Original Image. (b) Image with Masked Region. (c) Ground Truth Edges. (d) Generated Edges. 113

4.12 Edge-map (c) generated using the left-half of (a) (black edges) and right-half of (b) (red edge). Input is (a) with the right-half removed, producing the output (d). 115

4.13 Examples of object removal and image editing using our EdgeConnect model. (Left) Original image. (Center) Unwanted object removed with optional edge information to guide inpainting. (Right) Generated image. . . 116

4.14 Inpainting results where the edge generator fails to produce relevant edges. . 117

5.1 Schematic illustration of super-resolution problem. (a) The ground truth image, (b) The image downsampled by a factor of two. Each four-pixel information on the left turn into one pixel in the middle, as a result, the structure and orientation of edges are not distinguished anymore showing the problem is ill-posed. (c) The reconstruction of a high-resolution image from one-pixel information using bilinear interpolation. Most distinctive features in the original image are lost and the result is blurry around the edges. 120

5.2 An illustration of the proposed inpainting-based method for SISR problem. (a) The original LR image. (b) Upsampling by a factor of two corresponds to interpolating one pixel between every two adjacent pixels. We add an extra empty row and column for every rows and columns in the ground truth image (shown in gray) which we fill by an inpainting process. (c) Upsampling by a factor of four corresponds to interpolating three pixels between every two adjacent pixels where we can add three extra empty rows and columns for every rows and columns in the ground truth image to be inpainted. 121
5.3 Summary of our proposed edge enhancement network for $\times 4$ SISR. Low-resolution grayscale image and edge map are the inputs of G_1 to predict the high-resolution edge map. Predicted edge map will be used in an inpainting network to perform SISR. 123
5.4 Fixed fractionally strided convolution kernels to offset the pixels of the LR image and create an incomplete HR image for $\times 2$ and $\times 4$ SISR factors. . . 124
5.5 Comparison of qualitative results of images for $\times 4$ scale factor SISR cropped at 512×512. Left to right: Ground Truth HR, LR image upscaled using nearest-neighbor interpolation, SISR using Bicubic interpolation, Baseline (no edge data), Ours (Full Model) 126
5.6 Comparison of qualitative results of images for $\times 8$ scale factor SISR cropped at 512×512. Left to right: Ground Truth HR, LR image upscaled using nearest-neighbor interpolation, SISR using Bicubic interpolation, Baseline (no edge data), Ours (Full Model) 127

A.1 Sample of results with CelebA dataset (512×512). Images are best viewed in color. From left to right: Original Image. Input Image, Generated Result. 135
A.2 Sample of results with Places2 dataset (512×512). Images are best viewed in color. From left to right: Original Image. Input Image, Generated Result. 136
A.3 Sample of results with Places2 dataset (512×512). Images are best viewed in color. From left to right: Original Image. Input Image, Generated Result. 137
A.4 Sample of results with Places2 dataset (512×512). Images are best viewed in color. From left to right: Original Image. Input Image, Generated Result. 138

List of Symbols

$\mathcal{L}$, $\mathcal{H}$, $\mathcal{F}$	(Uppercase, stylized) Functions
x, y, z, f	(Lowercase, italicized) Variables, Functions
X, Y, Z	(Uppercase, italicized) Random variables, Sets
$\mathbf{I}$, $\mathbf{M}$, $\mathbf{X}$, $\mathbf{Y}$	(Uppercase, fixed-width) Matrices, Tensors
$\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$	(Lowercase, fixed-width) Vectors
$\mathbb{R}$	The set of all real numbers
$\mathbb{E}[X]$	Expected value of random variable X
$X \sim F$	Random variable X has distribution F
p_{data}, p_{model}	Probability distributions of data and model
$\Pr(A\|B)$	Conditional probability of event A given event B
$f : A \mapsto B$	Function f maps from set A to set B
$a \in A$	Element a is in set A
$\|\mathbf{v}\|_2$	2-norm of the vector $\mathbf{v}$
$\|\mathbf{M}\|_p$	p-norm of the matrix $\mathbf{M}$
$\mathbf{M}^{-1}$	Inverse of the matrix $\mathbf{M}$
$\mathbf{M}^T$	Transpose of the matrix $\mathbf{M}$
$\mathrm{Tr}(\mathbf{M})$	Trace of the matrix $\mathbf{M}$
$\approx$	Approximately equal to
$\gg$	Much bigger than
$\nabla\mathbf{x}$	Gradient of $\mathbf{x}$
I_x	Partial derivative of I with respect to x
$\mathbf{H}(f)$	Hessian matrix of second-order partial derivatives of f
$\langle\mathbf{u}, \mathbf{v}\rangle$	Inner product between vectors $\mathbf{u}$ and $\mathbf{v}$
$f * g$	Convolution of functions f and g

1. Introduction

1.1 Overview

The world of digital imaging has come so far since Steven Sasson made the first digital camera in 1975 [16]. According to the market research firm InfoTrends, over 1.2 trillion digital images were captured globally by digital cameras and smartphones in 2017 [17]. The number is expected to grow to over 1.4 billion units by 2020. This does not include digital medical images, satellite images, astrophotography, or images captured by surveillance cameras. Comparing that with film photography in its glory days and the difference is stunning where in the year 2000, Kodak announced that consumers around the world took 80 billion photos. This exponential growth in the number of images taken each year and the rapid proliferation of image capturing devices have also had a fundamental impact on other disciplines such as physics, biology, medicine, forensics, meteorology, space science, agriculture and of course computer vision [18].

While the performance and efficiency of image capturing devices have significantly advanced on several fronts in recent years, it is no secret that the best image quality is not always guaranteed due to the imperfect imaging conditions. What makes it more challenging is that simply retaking the image is not always an option due to various constraints such as budget, time, and resources. For example in satellite imagery or aerial photography, the cost of an image is calculated per square meter and is generally very expensive. Medi-

cal images such as Magnetic Resonance Imaging (MRI), on the other hand, are very time consuming and oftentimes inconvenient for patients to undergo. In other situations such as surveillance, nature photography, or photojournalism that rely on non-recurring events, retaking the image is not even possible. Finally, for cases such as space imaging and astrophotography, the quality of an image is constrained by the current technical limitation of the image acquisition devices. The solution to many of these problems in digital image processing is *image restoration* and *enhancement*.

Image restoration and image enhancement are techniques to improve the quality of digital images. They are however distinct concepts; While the former is objective in nature and aims at recovery of a degraded digital image and estimating the original, the latter is more subjective and focuses on making the image more pleasing to the observer [18]. For example, red-eye removal or transformation from grayscale to pseudo-color are forms of image enhancement and are normally difficult to evaluate or quantify. Image restoration techniques, on the other hand, have a clear objective and can be evaluated using mathematical models. Examples of image restoration include noise reduction (*denoising*), removing blurring artifacts from an image (*deblurring*), increasing an image resolution (*super-resolution*), or reconstructing lost parts of images (*inpainting*). The main focus of this dissertation is on image inpainting; we will provide a new model to address the problem, discuss quantitative and qualitative measures to evaluate the model and compare our results against different state of the art methods, and finally show how to extend this model to solve single image super-resolution problem.

Image inpainting is the process of reconstructing lost or deteriorated parts of images and videos. It is an important step in many image editing tasks. It can, for example, be used to fill in the holes left after removing unwanted objects from an image. It is also an important research field in medical imaging and Computer Aided Diagnosis (CAD). Many post-processing algorithms on medical images such as attenuation correction in PET/MRI or

radiation therapy planning require distortion-free images. However, various factors could lead to artifacts, noise or partial deteriorations in medical images [19]. For example, metallic orthopedic implants cause severe local artifacts on MR images [20] or bright spots in the Computed Tomography (CT) scans [21]. In these situations, image inpainting techniques are used to remove artifacts, conceal spots and even cracks in the image. Moreover, due to subtlety and importance of medical images, inpainting techniques typically require some form of expert human supervision [22]. An automated and reliable inpainting model capable of utilizing such expert knowledge not only reduces cost and time, it also opens the door to brand new use-cases that were not previously possible.

Image inpainting is an ill-posed inverse problem, which means that there is more than one solution to reconstruct the missing or deteriorated regions of the image and the goal is to fill those regions with most plausible prediction. Humans have an uncanny ability to zero in on visual inconsistencies. Consequently, the filled regions must be perceptually plausible. Among other things, the lack of fine structure in the filled region is a giveaway that something is amiss, especially when the rest of the image contain sharp details. The work presented in this dissertation is based by our observation that many existing image inpainting techniques generate over-smoothed and/or blurry regions, failing to reproduce fine details. Recently, deep learning approaches have found remarkable success at the task of image inpainting. These schemes fill the missing pixels using learned data distribution. They are able to generate coherent structures in the missing regions, a feat that was nearly impossible for traditional techniques [23, 1, 2, 24, 25, 26, 3, 27]. While these approaches are able to generate missing regions with meaningful structures, the generated regions are often blurry or suffer from artifacts, suggesting that these methods struggle to reconstruct high-frequency information accurately.

Then, how does one force an inpainting process to generate fine details? Since the scene structure is well represented in an image edge mask, we show that it is possible

to generate superior results by conditioning an image inpainting process on edges in the missing regions. Clearly, we do not have access to edges in the missing regions. Rather, we train a neural network that hallucinates edges in these areas. This is the first step in our two-stage proposed model. Next, we train another neural network, the image completion network, that uses the hallucinated edges and estimates RGB pixel intensities of the missing regions. Our proposed model of "lines first, color next" combines two different approaches to inpainting problem: *Structural Inpainting* [28, 29, 30] and *Textural Inpainting* [31, 32] as we simultaneously try to perform texture and structure filling in regions of missing image information. Our motivation for edge prediction is two-fold: Firstly, our approach is partly inspired by our understanding of how artists work [33]. "*In line drawing, the lines not only delineate and define spaces and shapes; they also play a vital role in the composition*", says Betty Edwards, highlights the importance of sketches from an artistic viewpoint [34]. The second motivation is dimensionality reduction. A color image in its most common format, RGB color-space using 8-bit color-depth, requires 24 bits to represent each pixel; that is $16,777,216$ variations in color, whereas, a binary mask only requires $\{0, 1\}$ for every pixel and the dimensionality of the problem can be reduced by almost seven orders of magnitude. Edge recovery, we suppose, is an easier task and our proposed model essentially decouples the recovery of high and low-frequency information of the inpainted region.

We evaluate our proposed model on standard datasets CelebA [35], CelebHQ [36], Places2 [37], and Paris StreetView [38]. We compare the performance of our model against current state-of-the-art schemes. Furthermore, we provide results of experiments carried out to study the effects of edge information on the image inpainting task. We finally show that our model can be used to solve single image super-resolution problem and common image editing applications, such as object removal and scene generation.

1.2 Contribution of This Work

In this dissertation, we present learning-based structure-driven models for two ill-posed image restoration problems: **image inpainting** and **single image super-resolution**. Inspired by artists work, we propose "lines first, color next" models to disentangle edge generation and color restoration. We show the effectiveness of this approach to preserve sharp details especially for challenging cases of image inpainting with large missing regions in a high-resolution image and big scale factor image super-resolution with unknown downsampling. In particular, we introduce convolutional neural network architectures, objective functions, and quality assessment techniques to address these problems.

This thesis makes the following contributions. Our work:

- Provides detailed analysis and review of metrics available for image restoration and image quality assessment techniques.
- Introduces a deep generative model for capturing image structures to hallucinate edges in the missing regions given the pixel intensities of the rest of the image.
- Proposes a structure-guided deep learning model for image inpainting that employs the structures to guide the inpainting and fills the missing regions with texture and color of the rest of the image.
- Proposes an alternative approach to single image super-resolution (SISR) by reformulating the problem as an in-between pixels inpainting task.
- Demonstrates the importance of edge information to image restoration and the effectiveness of proposed methods through comparative studies, qualitative and quantitative results, and visual Turing tests.
- Provides general machinery for applying the proposed framework to some common image editing applications, such as object removal and scene generation.

1.3 Thesis Outline

The remainder of this thesis is organized as follows.

In **Chapter 2** we review deep learning background and present formal definitions for supervised and unsupervised learning. We discuss the mathematical definition of neural networks, training procedures, optimization methods, and backpropagation algorithm. Convolutional neural network and generative adversarial networks are reviewed in detail and best practices and principles to improve these networks are presented.

In **Chapter 3** we review image structures in the form of edges. We discuss how to extract high-frequency information from an image and explain various edge detection methods. Different quality assessment techniques and subjective/objective methods to evaluate the performance of image restoration models which are used throughout this thesis are discussed in detail.

In **Chapter 4** we specifically address the problem of image inpainting. We develop a deep learning-based model for image inpainting. Our proposed "line first, color next" approach is based on image structure that explicitly disentangles structure inference and image completion. Quantitative and qualitative comparisons and user study are included to benchmark the proposed model against current state-of-the-art techniques. The content of this chapter is based on the image inpainting model of Nazeri *et al*. [39].

In **Chapter 5** a new approach to single image super-resolution (SISR) problem is presented by reformulating the problem as an in-between pixels inpainting task. Quantitative and qualitative experiments show the effectiveness of this approach.

In **Chapter 6** we identify the remaining challenges and direction for future research.

1.4 Software, Open Data & Source Code

Software

Python programming language was chosen for implementation. Python is well known for its readability and less complexity and by 2018 is the most popular language for scientific research in Machine Learning and Data Science [40]. Python is free and with a wide array of open source packages available, it can be used for mathematical applications, image processing, computer vision, and machine learning tasks.

In this project, we mainly work with large datasets of images and training of large neural networks. Both of these tasks require large computational power, hence an efficient use of computational resources is a top priority in this research. To that end, most of our computations are performed on GPUs (Graphics Processing Units) and we leverage CPU (Central Processing Unit) hyper-threading to pre-process and render datasets efficiently. The following open-sourced Python packages were used in this research.

- **PyTorch** is an open source deep learning platform Python package that provides support for tensor computation with strong GPU acceleration, and neural networks on a tape-based autograd system [41]. `https://github.com/pytorch/pytorch`
- **OpenCV** (Open Source Computer Vision Library) is an open source computer vision and machine learning software library that can take advantage of multi-core processing and hardware acceleration. `https://opencv.org`
- **Scikit-image** is a collection of algorithms for image processing written by an active community of volunteers. `https://scikit-image.org`
- **NumPy** is an open source Python package, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. `http://www.numpy.org`

Open Data

We evaluate our proposed models on the following publicly available standard datasets.

- CelebA [35]. A large-scale face attributes dataset with 200K celebrity images.
  `http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html`
- Celeb-HQ [42]. High-quality version of the CelebA dataset with 30K images.
  `https://github.com/tkarras/progressive_growing_of_gans`
- Places2 [37]. More than 10 million images comprising 400+ unique scene categories.
  `http://places2.csail.mit.edu/`
- Paris StreetView [38] Geotagged imagery of Paris from Google Street View.
  `https://github.com/pathak22/context-encoder`
- Set5, Set14, BSDS100, Urban100 [43]. Standard SISR evaluation datasets.
  `http://vllab.ucmerced.edu/wlai24/LapSRN/`

Source Code

The Python implementation of our models, evaluation metrics and pre-trained models are licensed under a *Creative Commons Attribution-NonCommercial 4.0 International* and can be accessed through the following link.
`https://github.com/knazeri/edge-connect`

2. Deep Learning Background

Machine learning is a set of methods and technologies to allow computers to learn from experience. One solution to machine learning is to have machines understand the world in terms of a hierarchy of concepts. With this approach, the machine can learn complicated tasks by building them from simpler ones in a deep hierarchy of concepts. This approach to machine learning is called **deep learning** [44].

It is no secret that the performance of machine learning algorithms depends heavily on the representation of the data they are built upon. Since hierarchy of concepts can describe the world in multiple levels of representations, in principle they should make the learning algorithm's job easier. However, up until recently [45], the general understanding among Artificial Intelligence (AI) researchers was that this approach to machine learning was not practical due to its huge computational cost, difficulties it presents for the optimization algorithms [46, 47], and the lack of enough training data. It was only after we finally harnessed both the vast computational power and the enormous storehouses of data, and pioneered innovative optimization algorithms that deep learning started to take off and outperformed competing state-of-the-art machine learning technologies.

Today deep learning has demonstrated huge success in many application domains and has achieved state-of-the-art performance compared to traditional machine learning methods in image processing, computer vision, natural language processing (NLP), speech recognition, machine translation, medical imaging, robotics, and control [48]. In this chap-

ter, we provide a brief technical background on deep learning and neural networks. We discuss different categories of learning, including supervised, semi-supervised and unsupervised learning; and the optimization algorithms that are used to train these models. The chapter continues with discussions about different types of neural networks and how to improve the performance of these models. For a more thorough study of neural networks and deep learning please refer to *Neural Networks and Deep Learning* textbook by Charu C. Aggarwal [49] and the *Deep Learning* book by Goodfellow *et al*. [44].

2.1 Supervised/Unsupervised Learning

An agent is learning if it improves its performance on future tasks after making observations about the world [50]. This improvement depends on the prior knowledge of the agent, the representation in the data, the agent component to be improved, and the *feedback* agent uses to learn. There are three types of feedbacks commonly used by learning algorithms that determine the type of learning: *supervised learning*, *unsupervised learning*, and *reinforcement learning*. The *semi-supervised learning* is also a learning paradigm that falls between supervised and unsupervised learning. In this section, we briefly review the supervised and unsupervised learning paradigms.

Supervised Learning Supervised learning is a class of learning problems that can be formulated as a machine performing a mapping $f : X \rightarrow Y$, from a vector space of all possible inputs X to the vector space of all possible outputs Y where the output is known in advance and supplied by supervision. Given a training set of n examples of input-output pairs $\{(x_1, y_1), ...(x_n, y_n)\} \in X \times Y$, where y_i can be generated by a known function $y_i = f(x_i)$ the job of a learning algorithm is to approximate the true function f with a *hypothesis* function $h : X \rightarrow Y$. One example of supervised learning is *classification*

problems where the input needs to be mapped to a category of IDs using a learned function $h(X)$. For example in a binary classification task of face detection, X is set of input images and $Y = \{0, 1\}$ is a set of labels with 1 indicating a match and 0 otherwise. The output of the hypothesis function is a probability value in the interval $[0, 1]$ indicating the probability of a face matching the target. Another common supervised learning problem is a *regression* task where the output $Y = \mathbb{R}^m$ is a set of real-valued targets. For example, in a learning algorithm that estimates the age of a person from an image, the input is an image and the hypothesis function outputs a real-valued number estimating the age.

The learning procedure consists of finding a hypothesis function h from a **hypothesis space** $\mathcal{H}$ using a training set, where $\mathcal{H}$ is s space of functions $f : X \to Y$ the algorithm will search through. More precisely, let $\{(x_1, y_1), ...(x_n, y_n)\} \sim p_{\text{data}}$ be the training set of n independent and identically distributed (iid) examples taken from data distribution p_{data} and $f : X \to Y$ be the true mapping from an input set X to labels set Y. We consider a scalar-valued loss function $L(\hat{y_i}, y_i)$ that measures the disagreement between the true label y_i and the predicted value $\hat{y_i} = h(x_i)$ for some $h \in \mathcal{H}$. Our objective is to estimate h using

$$h^* = \arg\min_{h \in \mathcal{H}} \mathbb{E}_{(x,y)\sim p_{\text{data}}} \left[L(h(x), y) \right]. \tag{2.1}$$

In practice the expectation is taken over the training set meaning we seek to find a function h^* that minimizes the expected loss over the training set. Once the function h^* is learned we can use it to map samples from X to Y. We say the model can **generalize** if it performs accurately on novel unseen samples after being trained using the training data set.

One example of supervised learning algorithm is *logistic regression* which is used in binary classification problems. The hypothesis is defined as a *logistic function*, also known as the sigmoid function that measures the conditional probability of *true* label given an input X.

$$h_\theta(X) = \frac{1}{1 + e^{-\theta^T X}} = \text{Pr}(Y = 1|X; \theta), \tag{2.2}$$

where θ is a vector of model parameters and the sigmoid function outputs the probability of the model predicting 1. The probability of the model predicting 0 is then given by

$$\Pr(Y = 0|X; \theta) = 1 - h_\theta(X), \tag{2.3}$$

we can write the probability of $\Pr(Y|X; \theta), Y \in \{0, 1\}$ as a Bernoulli distribution

$$\Pr(Y|X; \theta) = h_\theta(X)^Y (1 - h_\theta(X))^{(1-Y)}. \tag{2.4}$$

The *maximum likelihood* is a common approach used to estimate the model where we define the likelihood function over all (x_i, y_i) samples in the training set as

$$\mathcal{L}(X; \theta) = \Pr(Y|X; \theta) = \prod_i h_\theta(x_i)^{y_i} (1 - h_\theta(x_i))^{(1-y_i)}. \tag{2.5}$$

It is a common practice to take the logarithm of the likelihood function. The loss is defined as minimizing the negative log likelihood of the above equation over the training set

$$\ell(X; \theta) = -\sum_i y_i \log h_\theta(x_i) + (1 - y_i) \log(1 - h_\theta(x_i)). \tag{2.6}$$

The minimization is performed by finding the gradient of the log-likelihood function with respect to model parameters in a gradient-based optimization algorithm. We will discuss model optimizations in Section 2.2.

Many supervised learning problems can be solved using the above formulation. A neural network classification, for example, can also use maximum likelihood to estimate model's parameters, we will discuss neural networks in detail in Section 2.3. The function space in which the hypothesis is defined is what makes models different; In general, there is a tradeoff between complex hypotheses that fits the training data well and simpler hypotheses that may generalize better [50]; this is known as *bias–variance tradeoff* in supervised learning. Once the hypothesis and the scalar-valued loss functions are selected, the problem of supervised learning reduces to an optimization problem to estimate the model parameters.

Unsupervised Learning The unsupervised learning problem is one where the learning algorithm learns patterns in data when no explicit feedback is supplied [50]. In other words, the algorithm is not provided with labels Y and the goal is to discover something about the structure of the input distribution. Most common unsupervised learning algorithms include clustering, anomaly detection, density estimation, neural networks, and latent variable learnings. In the context of deep learning, unsupervised learning is commonly used to learn an underlying probability distribution of a training dataset. Broadly speaking, the learning algorithm observes several examples from a training dataset X and attempts to implicitly or explicitly learn the probability distribution p_{data} that generated them, or extract meaningful properties of that distribution [44].

For example, *autoencoders* are type of neural networks designed to learn an efficient representation (encoding) of data in an unsupervised manner to be used in many applications such as *dimensionality reduction* [51], denoising [52, 53], *semantic hashing* [54], and image retrieval [55]. The model consists of two parts: the encoder ϕ and the decoder ψ. The encoder takes an input $X \in \mathbb{R}^d$ and maps it to a new representation $Z \in \mathbb{R}^c$ where it is being called *code* and $c < d$. The decoder reverses the process by taking the representation Z and reconstructs the input. The learning is performed without specifying any labels by using some reconstruction loss that measures the disagreement between the input X and its reconstruction.

$$\begin{gathered} \phi, \psi = \arg\min_{\phi,\psi} \|X - (\psi \circ \phi)(X)\|_2^2, \\ \phi : \mathbb{R}^d \to \mathbb{R}^c \qquad \psi : \mathbb{R}^c \to \mathbb{R}^d \end{gathered} \tag{2.7}$$

where here ℓ_2 norm is being used as a reconstruction loss and an optimization algorithm can minimize this objective with respect to both ϕ and ψ to estimate the parameters of the encoder and decoder.

Another very popular unsupervised deep learning algorithm is *Generative Adversarial Networks* (GANs) [9]. GANs are class of neural network models that implicitly estimate a

high-dimensional distribution with an approximation. The learning is performed by introducing a *generator* network that directly produces new samples from a distribution and a *discriminator* network that validates those samples by measuring how realistic they look. For example, in an image inpainting problem, the job of a learning algorithm should not be to recreate the exact missing part of an image, but instead to fill the missing region with the most plausible content so that the final result looks realistic to a human eye. GANs are discussed in detail in Section 2.3.3; we will show how to model the inpainting problem using GANs in Chapter 4.3.

One strong benefit of unsupervised learning algorithms is that they can leverage practically unlimited amount of unlabeled data to train a model. Recent research has shown that supervised learning algorithms can also benefit from *unsupervised pre-training* [56]; where the pre-training procedure introduces a useful prior to the supervised fine-tuning training. This leads to significantly better performance than the standard initialization and the model generalize better even with a limited size of training dataset [56].

2.2 Optimization

Optimization is an essential part of every learning algorithm. It refers to either a task of minimization of maximization of some function $\ell(\theta) : A \rightarrow \mathbb{R}$ from some set A to the set of real numbers by altering θ. Normally, the optimization problems are phrased as minimizing $\ell(\theta)$ where we seek an element $\theta^* \in A$ that satisfies $\ell(\theta^*) \leq \ell(\theta)$ for all $\theta \in A$. In case of maximization we may alter the algorithm as minimizing $-\ell(\theta)$. The function $\ell(\theta)$ is called an **objective function**; when the optimization is performed by minimization, the function may also be referred to as the **cost function** or the **loss function**.

For example the maximum likelihood estimation for supervised learning discussed in previous section $\theta^* = \arg\max_{\theta \in \Theta} \mathcal{L}(X; \theta)$, with $\mathcal{L}$ being the likelihood function, can be

solved by defining an objective function given by the log likelihood (see Section 2.1)

$$\ell(\theta) = \log \mathcal{L}(X;\theta) = \mathbb{E}_{x\sim p_{\text{data}}} \log p_{\text{model}}(x;\theta), \tag{2.8}$$

where p_{model} and p_{data} are the model and data distributions respectively and we maximize $\ell(\theta)$ subject to $\theta \in \Theta$, or as we saw earlier minimize $-\ell(\theta)$. Sometimes we can obtain this analytically by solving $\nabla_\theta \ell(\theta) = 0$ for θ where ∇ is the gradient operator. However, this requires the closed-form solution for the equation which may not exist. Other times we can solve this using iterative *derivative-free* or *derivative-based* optimization methods.

Derivative-free optimization. These methods can be used to numerically optimize any function $\ell(\theta)$ by finding an input θ that effectively minimizes/maximizes the function through "guess-and-check". It is a common approach to iteratively improve the parameter guess by repeatedly making small perturbations to the input using *hill-climbing* methods in the function's landscape [50, 57]. Examples include the simplex algorithm for linear programming and binary search. Derivative-free optimization is used when the objective function is not differentiable, non-smooth, or expensive to evaluate. However, these methods are not very effective for neural networks where the parameter search space is extremely large and the process is computationally intractable.

Derivative-based optimization. These methods are based on an assumption that the objective function is smooth and differentiable. First order derivative-based optimization methods compute the gradient of the objective function $\nabla_\theta \ell(\theta)$ with respect to its parameters θ. The gradient is a vector of partial derivatives that gives the direction in which ℓ increases most rapidly along every dimension of θ. The gradient vector then can be used as a search direction. A very simple first order derivative-based optimization method is *gradient ascent*. The idea is to take small steps in the objective function landscape in the direction of its gradient using an iterative process.

$$\theta^{t+1} = \theta^t + \alpha \nabla_\theta \ell(\theta), \tag{2.9}$$

where α is a small positive **scalar** controlling the step-size, in the context of machine learnings also known as the *learning rate*. As mentioned before, normally optimization by minimization is preferred, where we take steps in the opposite direction of the gradient effectively performing *gradient descent*. During the optimization a training set $\{x_1, ...x_n\} \sim p_{\text{data}}$ is being used to approximate the model parameters and p_{data} is the training data distribution.

$$\theta^{t+1} = \theta^t - \alpha \nabla_\theta \, \mathbb{E}_{x \sim p_{\text{data}}} \left[\ell(x; \theta) \right], \tag{2.10}$$

where the expectation is taken over the entire training set. This can be computationally very expensive with a large dataset where we need to evaluate the loss for every training example in order to perform one step of gradient descent. To resolve this problem, *stochastic gradient descent* (SGD) [58] algorithm is proposed that calculates the gradient over a small subset of the training set

$$\theta^{t+1} = \theta^t - \alpha \nabla_\theta \left[\frac{1}{m} \sum_{x_i \in \mathbb{S}} \ell(x_i; \theta) \right], \tag{2.11}$$

where $\mathbb{S}$ is a subset of training examples $\{x_1, ...x_m\} \sim p_{\text{data}}$ randomly selected for each iteration of gradient descent and is called a *minibatch*. The typical size of a minibatch is between 1 and 128 [59]. The idea behind SGD is that we can perform many approximate updates instead of one exact gradient update. Each update only approximately takes a step toward the objective function's minimum, and this is why the algorithm is called "stochastic". However, this process can converge much faster than the regular gradient descent. This optimization algorithm is sometimes called *minibatch gradient descent* [60].

Optimization methods that only use gradients, such as gradient descent are called **first-order optimization algorithms**. In comparison, **second-order optimization methods** that leverage second derivatives information in an iterative updating optimization can reach the critical point much faster than first-order algorithms. For example, Newton's method in optimization is an iterative method to find the roots of a derivative of a twice-differentiable

function. It is based on a second-order Taylor series expansion to approximate a function $f(\mathbf{x})$ near a point $\mathbf{x}^{(0)}$ [44]

$$f(\mathbf{x}) \approx f(\mathbf{x}^{(0)}) + (\mathbf{x} - \mathbf{x}^{(0)})^T \nabla_{\mathbf{x}} f(\mathbf{x}^{(0)}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(0)})^T \mathbf{H}(f)(\mathbf{x}^{(0)})(\mathbf{x} - \mathbf{x}^{(0)}), \quad (2.12)$$

where $\mathbf{x}$ is a multi-dimensional input array and $\mathbf{H}(f)$ is a Hessian matrix of second-order partial derivatives of f with respect to every input dimension. Solving the above equation for the critical point $\mathbf{x}^*$ of the function we obtain.

$$\mathbf{x}^* = \mathbf{x}^{(0)} - \left[\mathbf{H}(f)(\mathbf{x}^{(0)})\right]^{-1} \nabla_{\mathbf{x}} f(\mathbf{x}^{(0)}). \quad (2.13)$$

We can solve the optimization problem recursively.

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \gamma \left[\mathbf{H}(f)(\mathbf{x}^t)\right]^{-1} \nabla_{\mathbf{x}} f(\mathbf{x}^t), \quad (2.14)$$

where γ is a small step size similar to the learning rate in the gradient descent algorithm. This approach, despite having a useful property of reaching the critical point much faster, may also converge to saddle points or local maximum which is a harmful property for minimization problems. Another problem with this method is that it requires to find an inverse of a Hessian matrix which can be computationally very expensive when the input dimension is large. Many second-order derivative methods are introduced in the literature, that fix converging to saddle points or problems with computing the Hessian matrix. However, second-order methods still remain difficult to scale to large neural networks [44].

In practice, stochastic gradient descent remains the standard optimization method to train neural networks while some modifications to the computation of the update direction of SGD such as Momentum [61], RMSProp update [62], and Adam optimizer [63] are proposed in the literature, that make SGD algorithm converge faster. For a more thorough study on different gradient descent algorithms refer to *an overview of gradient descent optimization algorithms* by Sebastian Ruder [64].

2.2.1 Backpropagation

In the stochastic gradient descent algorithm discussed earlier, we need to compute the gradient of the loss with respect to model's parameters to minimize it. Computing the gradient using analytical expression is straightforward, however evaluating such expression for every parameter in a model that contains thousands or even millions of parameters is computationally expensive. Using chain rule of calculus, one can see that different elements of a gradient with respect to the model's parameters contain many common subexpressions. The **backpropagation** algorithm or simply **backprop** [65], is a recursive application of the chain rule that avoids re-computing these subexpressions to compute the gradient efficiently. The idea is based on formalizing the model as a function mapping from input to output in a directed acyclic graph (DAG) called the **computational graph**. In a computational graph, we use nodes to indicate differentiable transformations (operation) performed on some input variables (scalar, vector, matrix, or tensor). A node may contain its own variables and always produces one or more outputs which then flows to other nodes. The graph may be evaluated in a *forward pass* or *backward pass*.

In the forward pass, we take an input (batch of data in a neural network application) and forward the graph by evaluating each operation in the graph recursively. Each node in the graph has a known differentiable operation, and during the forward pass, the Jacobian of the output of the node with respect to its inputs (and its local variables) are evaluated and stored locally. In the backward pass, the gradient of the loss with respect to the output of the graph is calculated and gets passed to the nodes in the graph in reverse order. The gradient of the loss with respect to each node's inputs (and local variables) is evaluated using the chain rule of calculus by multiplying the gradient coming from the next node with the local gradients stored locally during the forward pass. The result is passed to previous nodes to recursively evaluate the gradient with respect to every variable in the graph.

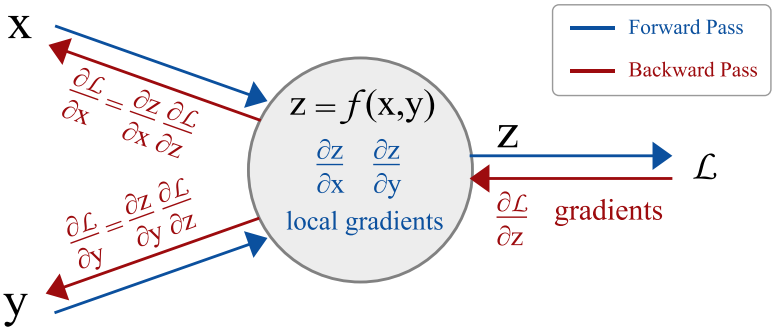


Figure 2.1: Schematic overview of backpropagation in a simple computational graph. During forward pass, vectors $\mathbf{x}$ and $\mathbf{y}$ are inputs to a node that performs some fixed computation on its inputs producing vector $\mathbf{z}$. Note that the we can compute the Jacobian matrices $\frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ and $\frac{\partial \mathbf{z}}{\partial \mathbf{y}}$ for the node at this stage. The output $\mathbf{z}$ flows further to the graph where at the end we calculate a loss using a differentiable scalar-valued function $\mathcal{L}$. The backward pass proceeds in the reverse order, effectively calculating the gradient of the loss with respect to all the elements in the graph using the chain rule. The gradient of the loss with respect to the vector $\mathbf{z}$ is calculated $\frac{\partial \mathcal{L}}{\partial \mathbf{z}}$ and gets multiplied with the local gradients calculated during the forward pass to find the global gradient of the loss with respect to the inputs $\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \frac{\partial \mathcal{L}}{\partial \mathbf{z}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{y}} = \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \frac{\partial \mathcal{L}}{\partial \mathbf{z}}$. In a neural network, each node contains parameters, where the gradient with respect to each parameter tells us how they should be changed to minimize the loss.

Figure 2.1 shows a schematic overview of backpropagation for a single node in a computational graph. In a neural network application, the inputs to each node are commonly tensors generated by transformations applied on the network input from previous nodes. The local variable of the nodes are the network parameters we try to find. The gradient with respect to each parameter tells us how they should be changed to minimize the loss. In practice, deep learning software frameworks, use backpropagation to evaluate the gradients where we design the graph and the intermediate operations and the backward pass is performed implicitly by the framework during optimization.

2.3 Neural Networks

Artificial Neural networks are popular machine learning techniques that simulate the mechanism of learning in biological organisms [49]. These networks are computing systems inspired by a biological mechanism which contain many computation units referred to as neurons. An artificial neural network in its simplest form is a differentiable function $\mathcal{F} : \mathbf{X} \rightarrow \mathbf{Y}$ that transforms an input set $\mathbf{X}$ to the desired output set $\mathbf{Y}$. The function $\mathcal{F}$, also called a model, is a composition of many simple functions known as neurons each doing a linear transformation on their input using their parameters known as weights, followed by a non-linearity. The search space of function $\mathcal{F}$ and the intermediate parameters of its neurons are determined by optimizing the model with respect to a differentiable loss function using some derivative-based optimization technique. The model optimization is called *training* where the model parameters are adjusted using a finite set of input-output pairs called *training set*. Once the model is trained, it can be used at *inference* where it maps any unseen input from set $\mathbf{X}$ to an output from set $\mathbf{Y}$. This ability to compute functions of unseen inputs by training over a finite training set is referred to as *model generalization*.

In this section, we briefly review the basic structure of neural networks and the most popular types of neural nets for both supervised and unsupervised learning.

2.3.1 Feedforward Neural Networks

A simple case of neural networks is a *feedforward neural network* also known as *multilayer perceptron* (MLP) that consists of at least three layers: an input layer, a hidden layer, and an output layer. Each layer in the network consists of multiple neurons, each applying a linear transformation on their inputs followed by a non-linearity, also known as *activation function*. Except for the first layer, neurons in each layer are connected to all neurons in the previous layer. A value known as *weight*, is associated with each connection, effectively

making a neuron performing the weighted sum of its inputs. Figure 2.2 on the right shows

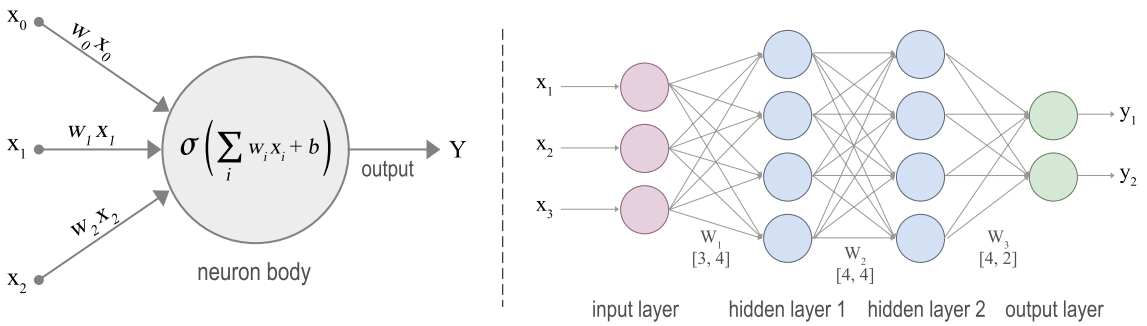


Figure 2.2: **Left:** schematic view of one neuron in a neural network. The neuron computes the weighted sum of its inputs followed by a non-linear function. **Right:** an example of a 3-layer neural network(multi-layer perceptron). Neurons in each layer are connected to all neurons in the previous layer. At each layer, the output activations are efficiently evaluated using matrix multiplication between the input and the weights matrices.

an example of a 3-layer neural network. The network can be seen as a function mapping from an input vector $\mathbf{X}$ to the output vector $\mathbf{Y}$ given by $\mathcal{F}(\mathbf{X}) = \sigma(W_3^T\sigma(W_2^T\sigma(W_1^T\mathbf{X})))$ where W_1, W_2, and W_3 are the *weight* matrices associated with each layer. $\sigma(.)$ is a non-linear function, also called the *activation function*. Common choices for the activation function are the sigmoid function $1/(1+e^{-x})$, $\tanh(x)$, and rectifier linear unit (ReLU) [66, 67]. The arrangement of weights in the matrix form allows us to efficiently evaluate the output using matrix multiplication. Figure 2.2 on the left, shows one neuron performing a weighted sum of its inputs followed by an activation function. It is a common practice to associate a bias term b to each neuron where it gets added to the weighted sum of inputs. The bias term allows us to apply affine transformation on data.

The neural networks architecture discussed here is not an efficient way to handle high-dimensional input data such as images. Next, we will discuss another family of neural networks, *Convolutional Neural Networks*, designed to operate on these types of inputs.

2.3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class neural network architectures designed for data with spatial structure *e.g*. sequence of characters in the text, sound signals, images, videos, and 3D voxel data. In each case, input is a high dimensional tensor with highly correlated features. For example, in case of a color image, the input is a $H \times W \times 3$ array where H and W are image height and width respectively and each index represents a pixel color in a spatial structure. These pixels are highly correlated which means their values and locations in a spatial neighborhood form structural information. The fully connected network architecture in the preceding section, while very effective in some cases, does not scale well to these very high dimension data as it suffers from the curse of dimensionality. Due to the full connectivity between nodes, the number of parameters in this architecture grows exponentially with the input dimension and as a result, training time increases significantly. The high dimensionality and the spatial structure of these data require a special architecture of neural networks that can leverage the spatial arrangement and correlation between features, use local connectivity, and sensible parameter sharing schemes [68].

Convolutional networks have been used in image recognition since the 1980s. LeNet was one of the pioneering works on convolutional networks introduced in 1988 for character recognition tasks such as reading zip codes, checks, *etc*. [69]. In the last few years, the hardware and algorithms for training deep nets using CNNs have seen extensive improvements and as a result, CNNs have managed to achieve human-level performance on complex visual tasks [45, 7, 70, 71].

As the name suggests, "convolutional neural network" employs a mathematical operation known as **convolution**. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers [44]. The network consists of different layers, and each layer consists of small convolutional kernels.

At each layer, convolution operators are being performed on the output from the previous layer to form a new input (feature map) for the next layer. These kernels can be seen as local feature extractors that encode the input in hierarchical order as it is being passed through the network. The values of the convolutional kernels are the network parameters (or weights) that need to be determined in training. Moreover, since we use the same kernel to convolve the input at every location, we effectively introduce parameter sharing scheme. Figure 2.3 shows a schematic view of a CNN network for an image classification task. An image is passed to a network and at each layer, convolutional filters convolve the input to create activation maps for the next layer. The output of the network is categorical class labels each representing the probability of the input being one of the categories. A loss function is being used during training to update the values of the convolutional kernels using backpropagation.

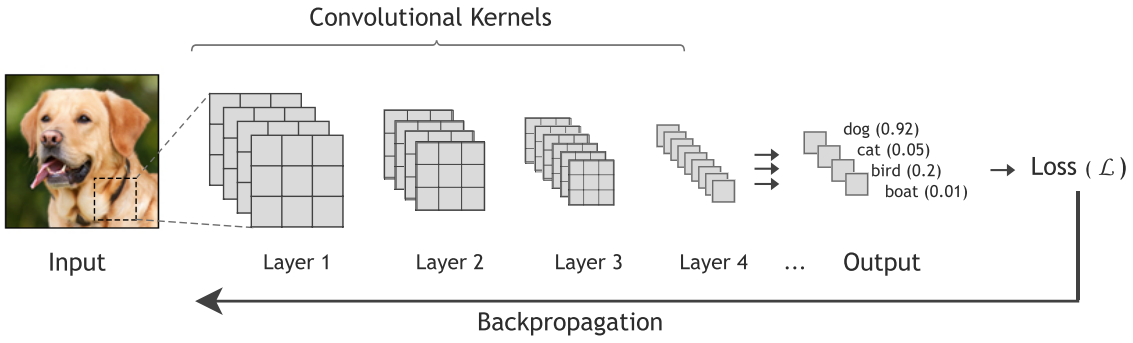


Figure 2.3: Schematic view of a convolutional neural network for an image classification task. An image is an input to the network. At each layer, the input is convolved with the convolutional kernels to create activation maps for the next layer. The output of the network is a categorical probability distribution and a loss function is being used during training to find the values of the convolutional kernels.

In the next section, we review the core building blocks of convolutional neural networks and describe variants of these modules that are widely used in practice for neural networks.

CNN Building Blocks

Convolutional Layers The most important building block of a CNN is a *convolutional layer*. Unlike fully connected layers, neurons in the convolutional network are only connected to every pixel in their receptive field [72]. For example, neurons in the first layer of a CNN only see a small region of the input image with the size of the first convolutional filter (see Figure 2.4). Consecutively, each neuron in the second layer is connected only to a small region from the output of the first layer. This architecture allows the network to extract local low-level features in the early layers and then assemble them into higher-level features in the consecutive layers.

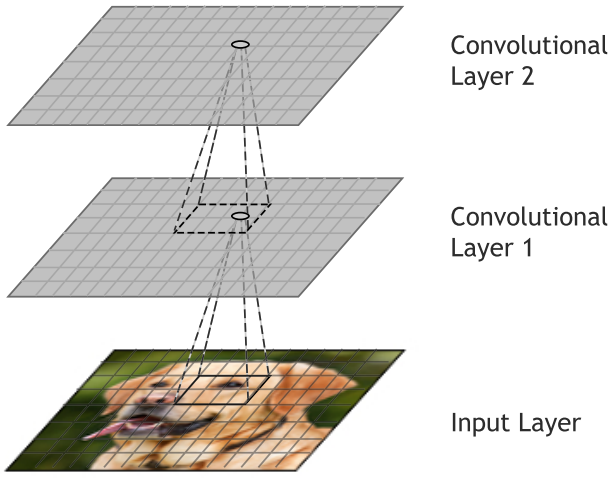


Figure 2.4: Convolutional layers with local receptive field extract local features in a hierarchical order.

Mathematically, a convolutional operator on a 1D signal x using the filter w is defined as

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{2.15}$$

In most machine learning applications, the input is a multidimensional array where we normally apply convolution over more than one dimension. For example, in case of a two-

dimensional image $\mathbf{I}$, a two-dimensional kernel $\mathbf{K}$ is also used where it is defined as

$$S(i,j) = (\mathbf{I} * \mathbf{K})(i,j) = \sum_m \sum_n \mathbf{I}(m,n)\mathbf{K}(i-m, j-n). \tag{2.16}$$

Using the commutative property of the convolution we can equivalently write

$$S(i,j) = (\mathbf{K} * \mathbf{I})(i,j) = \sum_m \sum_n \mathbf{I}(i-m, j-n)\mathbf{K}(m,n). \tag{2.17}$$

Note that with convolution operator, the kernel is flipped relative to the input. This preserves the commutative property of the convolution and is useful for writing proofs, however, many machine learning frameworks implement convolution without kernel flipping. In signal processing this function is called **cross-correlation**, however, one can argue that since the values of these kernels are being learned, the learning algorithm will learn a kernel that is flipped relative to the kernel learned by an algorithm with the flipping [44]. The convolution operator on a two-dimensional image is then defined as

$$S(i,j) = (\mathbf{K} * \mathbf{I})(i,j) = \sum_m \sum_n \mathbf{I}(i+m, j+n)\mathbf{K}(m,n), \tag{2.18}$$

where the kernel $\mathbf{K}$ is slided over the input image and the dot product between the kernel and input is calculated to produce the output which is known as *activation map*. The amount by which the filter shifts at each convolutional step is known as the *stride*. In a convolutional neural network, each layer normally contains multiple convolutional filters where each filter convolves every channel from the input. The resulting convolutions are added element-wise, and a bias term is added to each element. This follows by a non-linearity function $\sigma(.)$ to create an activation map for the next layer:

$$\begin{aligned} S(i,j,C) &= \sigma\left(\boldsymbol{b}^{(C)} + \sum_{m=-w}^{w} \sum_{n=-h}^{h} \sum_{k=1}^{C_{in}} \mathbf{I}(i+m, j+n, k)\mathbf{K}^{(C)}(m,n,k)\right), \\ w &= \lfloor \frac{W-1}{2} \rfloor, \qquad h = \lfloor \frac{H-1}{2} \rfloor \end{aligned} \tag{2.19}$$

where C denotes the number of output channels, C_{in} is the number if input channels, $\boldsymbol{b}^{(C)}$ is the bias term for the channel C, and W, H are kernel width and height respectively. The

total number of parameters in a convolutional layer is $(H \times W \times C_{in} + 1) \times C$.

To preserve the input spatial dimension, input padding schemes are employed per convolutional layer basis. For example, zero-padding adds a border of zeros to the input while reflection-padding, copies and reflects (horizontally and vertically) the input over the borders to preserve edge continuity in images. In practice, reflection-padding is a preferred padding scheme for images because of its low artifact rate near the image boundaries. Figure 2.5 illustrates the process. A $32 \times 32 \times 3$ input image is convolved with four 5×5 kernels with stride of 1 and no input padding, producing a four channel $28 \times 28 \times 4$ activation map.

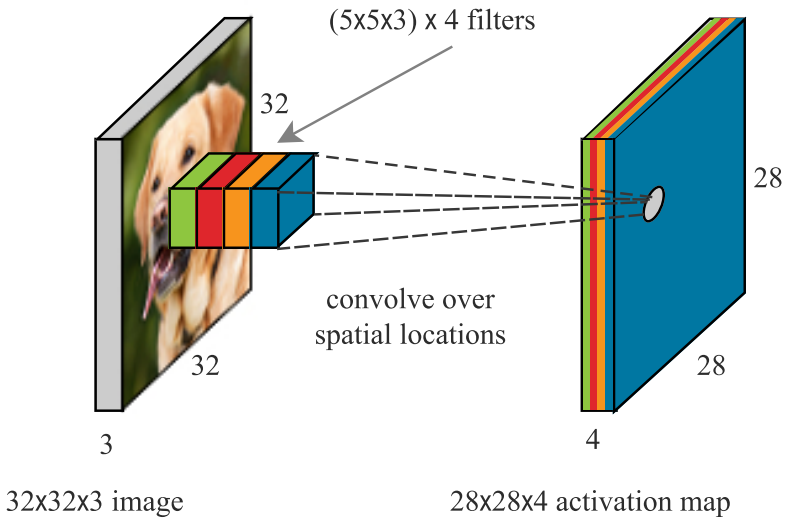


Figure 2.5: Illustration of convolving four $5 \times 5 \times 3$ filter over a $32 \times 32 \times 3$ input image with stride 1 and no input padding. Each convolution filter is expanded with the same number of channels as the input. In total four convolutional filters exists, each expanded with 3 channels producing a four channel $28 \times 28 \times 4$ activation map.

Pooling/Upsampling Layers Pooling and Upsampling layers are used in CNNs to decrease or increase the spatial dimension of the activation maps respectively. A pooling function replaces the output of a network at a certain location with a summary statistic of

the nearby outputs [44]. The pooling layer is used to control the overfitting and increasing the receptive field of the network. It also makes the inner representations invariant to small translations of the input. A pooling layer mostly uses a fixed downsampling transformation (*e.g*. max-pooling, average-pooling, *etc*.) and has no learnable parameters. To prevent information loss due to the dimensionality reduction, it is a common practice to increase the number of channels using convolution before a pooling layer. Conversely, an upsampling layer is used to undo the pooling operation. In image-to-image translation or image segmentation tasks, the output of the network needs to be the same size as the input and upsampling is used to reshape the activation maps back to their original size. Similar to a pooling layer, upsampling layers do not have learnable parameters and use some interpolation techniques (*e.g*. nearest-neighbor or bilinear interpolation) to account for missing data. Recently, an increased interest has been shown toward using strided convolution filters instead of pooling/upsampling layers [73]. These networks are being called *Fully Convolutional* or *All Convolution* and we will discuss them in Section 2.3.2.

Improved CNN Architectures for Image Generation

A convolutional neural network is built by stacking layers of convolutions and possibly pooling/upsampling layers to control and reduce/increase the complexity of the model. The architecture of a convolutional neural network is more of an art than science. However, designing a neural network for image generation problems (*e.g*. image segmentation, image-to-image translation, super-resolution, *etc*.) is different from typical classification tasks and requires some architectural considerations. Here we discuss some common heuristics to improve the performance of a CNN model for image generation tasks.

Fully Convolutional Network A typical convolutional network for classification contains several fully-connected layers at the end of the network. Neurons in these layers are

connected to all activations from the previous layer and do the high-level reasoning in the network. Most of the parameters in the network reside in these layers which make them computationally more expensive to train. Moreover, the spatial structure of the feature maps with respect to the input is being lost with fully-connected layers. As a result, these layers are not suitable to generate an output image that is highly correlated to the input. OverFeat [74] was one of the early attempts to replace fully-connected layers with convolution filters. In this work for object detection task, fully-connected layers were replaced with 1×1 convolution filters making the network invariant to the input size. Long *et al*. [75] first introduced a fully convolutional network for semantic segmentation. In their proposed model, they replaced the upsampling layers with fractionally strided convolution (also known as transposed convolution or deconvolution layer) [76] to increase the spatial dimension and reconstruct the segmentation maps. [73] showed that a learnable strided convolution kernels outperform pooling layers that use a fixed downsampling scheme. U-Net [77] is by far the most popular fully convolutional network introduced for biomedical image segmentation. This architecture is based on encoder-decoder networks [51] with a contracting path (encoder) followed by an expansive path (decoder) more or less symmetric to it. The input is being progressively downsampled in the contracting path using a series of strided convolutions and the process is reversed using a series of transposed convolutions in the expansive path. To account for the information bottleneck [78] (trade-off between accuracy and complexity) due to progressive downsampling, skip connections were added from the layers in the contractive path to their corresponding layer in the expansive path. Despite its success in segmentation tasks, recent research has shown that U-Net architecture is not well suited for problems where the output heavily relies on the spatial information of the input (*e.g*. image-to-image translation, super-resolution, *etc*.) as most of spatial information are lost due to progressive downsampling. Recently Johnson *et al*. [79] proposed a CNN architecture for style transfer and super-resolution tasks where the contractive/expansive

paths only contain two in-network downsampling/upsampling operators. The main body of the network is replaced with *residual blocks* [7] as discussed in the next section. Our proposed model in this dissertation is a modified version of the network proposed by Johnson *et al*.; we will explain the full network architecture in Section 4.3.3.

Dilated Convolution The receptive field is one of the basic concepts in convolutional neural networks. The receptive field of a layer in a deep CNN is the *field of view* of a unit in that layer [72]. Generally speaking, it is a region in the input space that a particular CNN's feature is looking at. This is especially important in visual tasks because the output needs to see a large area in the input image to capture relevant information. A receptive field of a layer R_l in a deep CNN is measured by

$$R_{l+1} = R_l + (k_l - 1) \prod_{i=1}^{l} s_i, \tag{2.20}$$

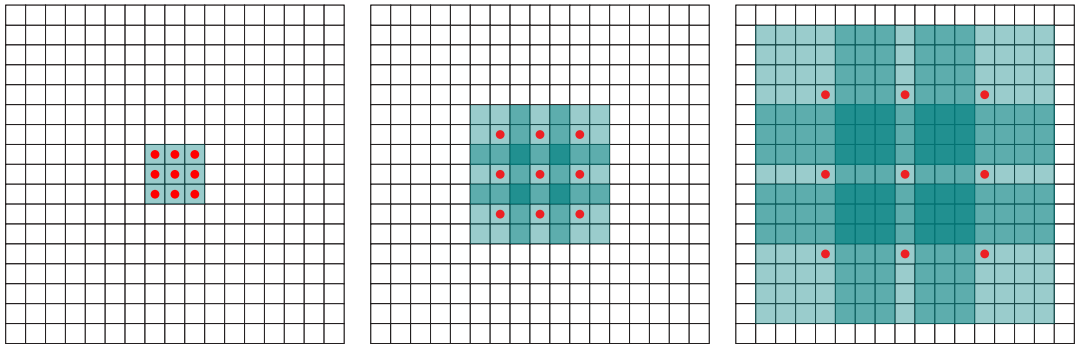
where $R_0 = 1$, k_l denotes the size of a convolutional kernel in layer l and s_l is the stride of the convolution operation in that layer. Please note that Equation 2.20 is also defined for pooling layers where k_l is then the size of the pooling layer. It can be seen that to increase the receptive field one can either increase the kernel size, stride, or the depth of the network. Increasing the kernel size and network depth comes with computational and performance costs. The easiest and most popular method to increase the receptive field has been increasing the stride by adding more pooling layers. However, as we discussed earlier, progressive downsampling in a CNN also comes with a cost of losing more and more distinctive features in the input. Yu *et al*. [6] proposed a variation of convolution kernel called *Dilated Convolution* that increases the receptive field of the kernel without any change in the number of parameters associated with it or computational cost.

Dilated convolution, also known as Atrous convolution [80] is a convolution operation with a dilated filter by introducing a dilation factor. The convolution operator is modified

to use the filter parameters at different ranges using different dilation factors. The process can be described as a modified version of Equation 2.18 for 2D convolution:

$$S(i,j) = (\mathbf{K}_\eta * \mathbf{I})(i,j) = \sum_m \sum_n \mathbf{I}(i + \eta\, m, j + \eta\, n)\mathbf{K}(m,n), \tag{2.21}$$

where $\mathbf{I}$ is an image, $\mathbf{K}$ is a two-dimensional kernel, and η is a dilation factor. This effectively increases the kernel width without increasing the number of parameters. Figure 2.6 shows the exponential expansion of the receptive field after using dilated convolutions with factors of 1, 2, and 4 respectively. All convolutional filters have identical number of parameters; the receptive field increases to 15×15 without changing the filter size or stride.



(a) $\eta = 1$ (b) $\eta = 2$ (c) $\eta = 4$

Figure 2.6: Receptive field expansion with dilated convolution, after applying dilated factors of 1, 2, and 4. **a)** Normal 3×3 1-dilated convolution filter has a receptive field of 3×3. **b)** 2-dilated convolution filter applied on (a) increases the receptive field to 7×7. **c)** 4-dilated convolution filter applied on (b) increases the receptive field to 15×15. All convolutional filters have identical number of parameters. Figure ©Yu *et al*. [6] *Multi-Scale Context Aggregation by Dilated Convolutions.*

The receptive filed after using dilated convolution is defined by modifying Equation 2.20

$$R_{l+1} = R_l + d_l.(k_l - 1) \prod_{i=1}^{l} s_i, \tag{2.22}$$

where d_l is the dilation factor associated with convolution kernel k_l in layer l. Please note that very large dilation factors (larger than 8) have a negative effect on the convergence of the network and should be used with care [80].

Residual Blocks Deep neural networks are difficult to train because of the *vanishing gradient* problem [47]. As the network depth grows, the gradient of the loss with respect to the weights becomes vanishingly small. This effectively prevents the weights from changing their values and the training becomes more difficult. He *et al*. [7] proposed a solution which effectively allows training networks with over 2000 layers. In their work, *Deep Residual Learning for Image Recognition*, they reformulated the layers as learning a residual function with respect to the layer inputs.

Formally, let $\mathcal{H}(x)$ be the desired underlying mapping of a layer, residual learning is defined by letting the layer fit another mapping of $\mathcal{F}(x) \coloneqq \mathcal{H}(x) - x$. The original mapping is then defined as $\mathcal{F}(x) + x$. This is achieved by adding shortcut connections between the input and output of a layer to perform *identity* mapping. The proposed module consists of two convolutional layers and a shortcut connection that simply adds the input to the output of the second convolutional layer followed by a ReLU [66, 67] nonlinearity. (Figure 2.7)

The advantage of using residual learning is twofold: *First*, the identity mapping preserves a strong gradient flow throughout the network effectively mitigating the vanishing gradient problem. *Second*, it simplifies the network by using fewer layers in the initial training stages. By carefully initializing the network parameters, one can completely shut down the layers inside the residual block prior to training and effectively let these layers learn the residual functions as training proceeds. This speeds up learning early in training as there are fewer parameters to update. One important property of residual learning is that we can add as many layers to the network as we computationally can afford without worrying about the vanishing gradient problem. In principle, the network should be able to

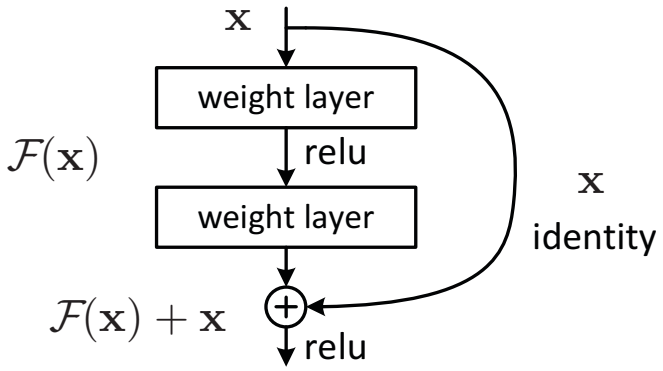


Figure 2.7: A building block of residual learning. Figure ©He *et al*. [7] *Deep Residual Learning for Image Recognition.*

choose to zero out the redundant layers by learning the identity mapping function. In practice, the ResNet model by He *et al*. [7] achieved the best performance in ILSVRC challenge [81] using 152 layers.

Normalization Schemes During training a deep neural network, the distribution of each layer's input changes by updating the parameters from the previous layer. This phenomenon is known as *internal covariate shift* and requires careful initialization of the weights and lower learning rate that slows down the training. Ioffe & Szegedy [82] proposed a normalization scheme called *Batch Normalization* that reduces the internal covariate shift of a network. This makes hyper-parameter search problem easier, allows for larger learning rates, and makes the neural network more robust. Batch normalization, normalizes the input to a layer using the mini-batch statistics.

Mathematically, normalization (also known as whitening) changes the statistics of the input distribution to have zero-mean and unit variance. For a layer with d-dimension input $\mathbf{x} = (x^{(1)}, ...x^{(d)})$ we normalize each dimension

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}, \tag{2.23}$$

where the expectation and variance are computed over the training dataset. However, since this is computationally inefficient, batch normalization computes those statistics for each mini-batch. For a mini-batch $\mathcal{B}$ of size m where $\mathcal{B} = \{x_1, ...x_m\}$ the it is defined as

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}, \tag{2.24}$$

where $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}^2$ are the mean and variance computed over the mini-batch respectively and ϵ is a small positive constant added for numerical stability. The algorithm then introduces two new learnable parameters γ and β to perform scale and shift on the whitened data. The batch-normalization module $\text{BN}_{\gamma,\beta}$ is then defined as

$$\text{BN}_{\gamma,\beta}(x_i^{(k)}) = \gamma\hat{x}_i^{(k)} + \beta. \tag{2.25}$$

Besides the learnable parameters γ and β, batch-normalization also tracks the statistics of the entire training dataset μ_t and σ_t^2 to be used at inference time. Note that the statistics in the Equation 2.24 are calculated for a mini-batch. During inference, the model may need to process data one item at a time and the mini-batch statistic simply does not exist. Hence at test time, μ_t and σ_t^2 are being used instead of $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}^2$. This is especially not favorable in deep networks that generate images. Simply changing the statistics of the layers at test time by those tracked from the training set might change luminance and the contrast of the generated image. To fix that, various normalization schemes are proposed in the literature. For example, instance-normalization [83] proposes normalization across spatial dimensions which results in a significant qualitative improvement in the generated images. Since this normalization is applied to each instance separately, the same process can be used at test time. Layer-normalization [84] and group-normalization [8] were also proposed to fix the inaccurate batch statistics estimation of batch-normalization when the batch size becomes smaller. Figure 2.8 compares different normalization schemes visually.

In practice, normalization methods significantly improve the training of the neural networks. Recent research has shown that normalizations can make optimization landscape

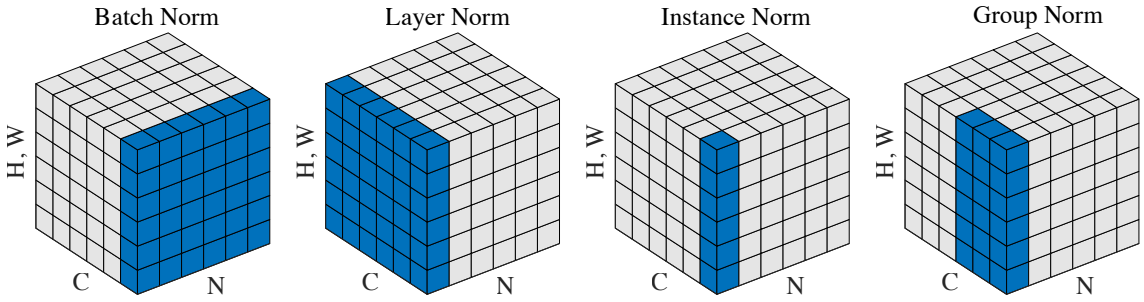


Figure 2.8: Comparison of different normalization schemes. In each case, N represents the mini-batch axis, C is the channel axis and (H, W) are the spatial axes. The pixels colored in blue are normalized by computing the statistics (mean and variance) of these pixels. Figure ©Wu *et al*. [8] *Deep Residual Learning for Image Recognition.*

significantly smoother [85] which induces a more stable behavior of the gradients and allows for faster training. As a side effect, normalizations also act as a regularizer by adding some noise to each hidden layer's activations and in some cases eliminates the need for costly regularizers such as dropout [86].

2.3.3 Generative Adversarial Networks

Generative Adversarial Networks (GANs) [9] are a new class of machine learning techniques for both semi-supervised and unsupervised learning. As the name suggests, GANs are a class of generative models. In statistical machine learning, generative and discriminative models are both methods to estimate a complex high-dimensional distribution with an approximation. Discriminative models are direct methods to find the distribution of labels Y given an observation X by estimating the conditional distribution $P(Y|X = x)$. Generative models are different from discriminative models in that they estimate the joint distribution of $P(Y, X)$ which can indirectly be used to estimate the conditional probability. This allows generative models to be used in a completely unsupervised setting where

they leverage the practically unlimited amount of unlabeled images to learn good representations of data [87]. There are two approaches to generative models. One is density estimation by taking sample points and infer a density function that describes the probability distribution that generated them. The other approach is to have a model observe many samples from a distribution and learn to create more samples from the same distribution. Many of the generative models that estimate high dimensional distributions use the second approach. Having a model to create more samples from a distribution can be used for many applications such as predicting the possible future for planning or simulated Reinforcement Learning, handling missing data or labels in the datasets, and realistic generation tasks [88]. Examples of realistic generation tasks include image-to-image translation [89], creating arts [90, 79], single image super-resolution [91, 4], and image inpainting [23, 1, 2, 24].

Most generative models use differentiable generator network and perform the principle of **maximum likelihood** to estimate the model parameters [44]. The process starts with taking samples from a distribution to create a training set. The model is then optimized to assign a high probability to those samples by maximizing a probability of observed data X, over the parameter space Θ. The parameters $\hat{\theta} \in \Theta$ that maximizes the probability is called *maximum likelihood estimate*:

$$\hat{\theta} = \arg\max_{\theta} \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \log p_{\text{model}}(x; \theta). \tag{2.26}$$

Generative models that use maximum likelihood estimation are different in the way they represent the data with an *explicit* or *implicit* density functions [88]. Models with explicit density functions are categorized based on whether the density function is tractable or not. Among the models that define a computationally tractable explicit density functions are autoregressive models such as PixelRNN [92], PixelCNN [93]. These models decompose the density function using the chain rule of probability. For example, when the input data is an image the model distribution is defined as the product of probabilities of each pixel

given all previous pixels. However, when dealing with very complicated distributions such as natural images or speech waves, it is very difficult to design a parametric function that is able to capture the distribution efficiently. To maximize the likelihood on these intractable density function, it is necessary to make either variational approximations or Monte Carlo approximations [88]. For example, Boltzmann machines [94, 95] use Markov Chain Monte Carlo approximation, whereas Variational autoencoder (VAE) [96] places a lower bound on the log likelihood $\mathcal{L}(x;\theta) \leq \log p_{\text{model}}(x;\theta)$ and maximize $\mathcal{L}(x;\theta)$. Both of these families incur some disadvantages from the approximations they use; for example, while images generated with VAEs tend to be unrealistic and suffer from blurriness [97], Markov chain approximations do not scale to problems like ImageNet generation [88]. The other family of generative models are those that represent data with implicit density functions. These methods design procedures to implicitly model and interact with the distribution, usually by drawing samples from that probability distribution. Generative adversarial networks are placed in this category of generative models. GANs provide a direct way of sampling from a distribution without explicitly defining the density functions, require no Markov chains or variational bounds approximation and relatively produce better quality samples.

Generative adversarial networks are based on a game theoretic scenario in which the generator network must compete against an adversary, discriminative network [44]. In the GAN framework, the generative network G directly produces new samples $x = G(z;\theta^{(G)})$ by sampling from a prior distribution of $z \sim p_z$. The discriminator network D attempts to distinguish between samples drawn from data distribution and samples generated by the generator network. The discriminator is trained in a supervised setting and outputs a probability value $D(y|x;\theta^{(D)})$ indicating whether or not the sample x comes from the training data distribution. The label $y = 1$ is assigned to samples from the training set, whereas $y = 0$ is assigned to samples drawn from the generator. Figure 2.9 shows an overview of the GAN pipeline.

The model is being optimized in a zero-sum game as a trade-off between the discrimi-

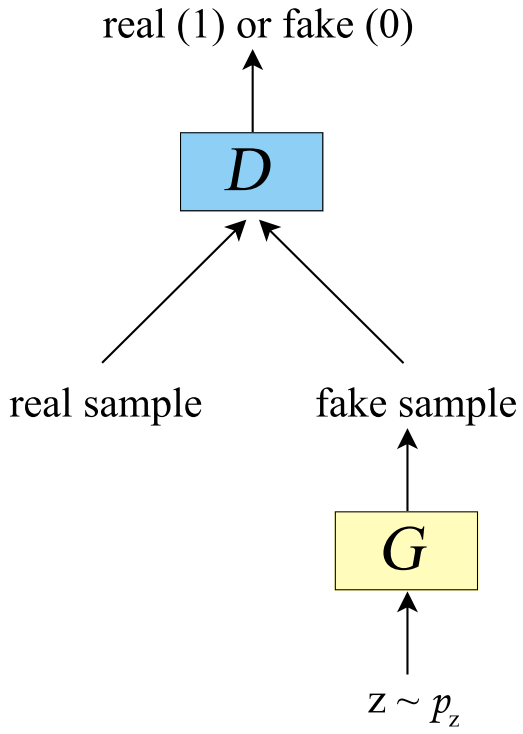


Figure 2.9: GAN framework overview. The generator network (G) directly produces samples by sampling from a prior distribution p_z. The discriminator (D) attempts to distinguish between real samples from a training set, which are labeled as 1, and samples generated by the generator which are labeled az 0. During learning, each network attempts to maximize its own performance and undo the other in a zero-sum game.

nator's payoff function $V(\theta^{(G)}, \theta^{(D)})$ and generator's payoff $-V(\theta^{(G)}, \theta^{(D)})$ in a minimax objective

$$\arg\min_{G} \max_{D} V(G, D). \tag{2.27}$$

In *game theory* and *decision theory* minimax objective is defined as minimizing the possible loss for a worst case scenario. In a two-player game, each player can maximize its performance by minimizing the other player's loss. For example, for every move that player P1 makes, there may be different countermoves by the other player and a score (loss for P1)

associated with them. Minimax objective is minimizing the largest value (loss) the player can be sure to get when they know the actions of the other player. Figure 2.10 shows the minimax objective in three steps.

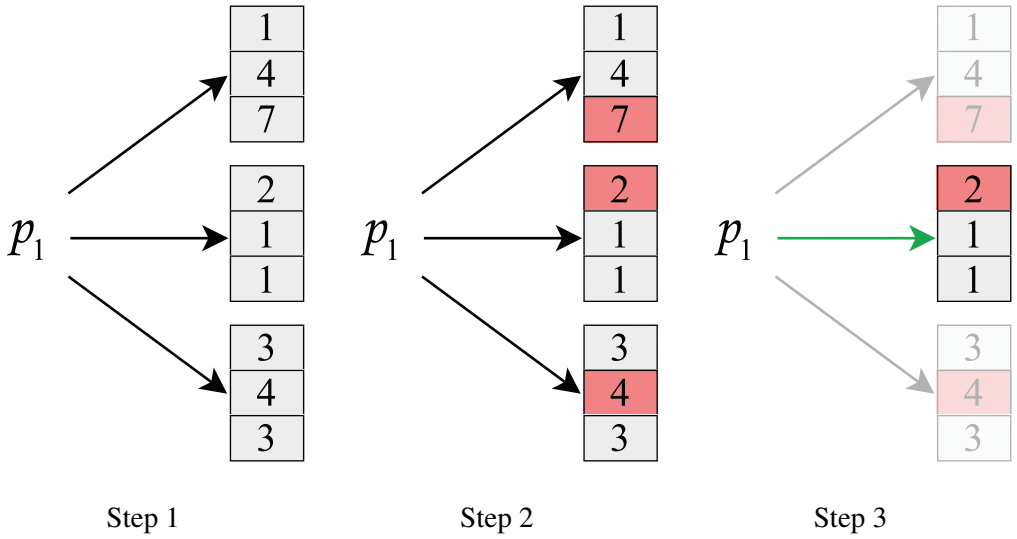


Figure 2.10: Example of the minimax objective in a two-player game. P1 tries to minimize the possible loss for a worst case scenario in three steps: Step 1) P1 can make three moves and predicts three countermoves for each of them, the score associated to each countermove is also calculated. Step 2) Maximum score (P1 loss) for each move is calculated. Step 3) P1 makes the move that has minimum value among all the values calculated in Step 2.

Both generator and discriminator are differentiable functions with respect to their input and parameters. Training GANs consists of iteratively updating each player's parameters to maximize its performance. The solution to this optimization problem is a local minimum in both networks' parameter space where no player can further improve their performance by changing their parameters. The solution point $(\hat{\theta}^{(D)}, \hat{\theta}^{(G)})$ is called a Nash equilibrium. In the next section, we mathematically formulate this optimization objective, discuss its pros and cons and review different GAN loss functions.

GANs Objective

The GAN objective is defined by two differentiable functions. The generator G takes an input z from some prior distribution p_z and outputs an image x, the network parameters are defined as $\theta^{(G)}$. The discriminator, takes an input x, outputs a probability of the input being real, and uses $\theta^{(D)}$ as parameters. Both cost functions are defined as parameters of both networks. The discriminator wants to minimizes the loss $J^D(\theta^{(D)}, \theta^{(G)})$ by updating only $\theta^{(D)}$ parameters. The generator, minimizes the loss $J^G(\theta^{(D)}, \theta^{(G)})$ and controls only $\theta^{(G)}$. The discriminator objective is defined as

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))], \tag{2.28}$$

where the minimization only updates $\theta^{(D)}$. This loss function is a sum of two binary cross-entropy losses for real and fake inputs

$$J^D(\theta^D, \theta^{(G)}) = \mathbb{H}_b(y = 1, x) + \mathbb{H}_b(y = 0, G(\boldsymbol{z})), \tag{2.29}$$

where $\mathbb{H}_b(x, y), y \in \{0, 1\}$ is a binary cross-entropy function defined as

$$\mathbb{H}_b(x, y) = -y \log(x) - (1 - y) \log(1 - x). \tag{2.30}$$

The objective function defined in the Equation 2.28 can be seen as minimizing the log likelihood of real and fake inputs simultaneously

$$\underbrace{-\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]}_{\text{loss for real images}} \underbrace{- \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}_{\text{loss for fake images}}, \tag{2.31}$$

where the loss for real images is minimized when discriminator assigns the label $D(x) = 1$ to the input x and the loss becomes zero. Similarly, the loss for fake images is minimized when discriminator assigns the label $D(G(z)) = 0$ to the input $G(z)$. Note that the output of the discriminator is a probability value ranging between zero and one, with a unique

maximum at $D(x) = 1$, where $-\log(D)$ becomes zero. The cost for the generator can be defined using the same objective in a **zero-sum game** where sum each player's cost is zero

$$J^G = -J^D. \tag{2.32}$$

Note that, the cost for the generator is only using the second part of the objective in the Equation 2.28

$$J^G(\theta^{(D)}, \theta^{(G)}) = \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))], \tag{2.33}$$

where the loss only updates $\theta^{(G)}$ and it is at minimum when $D(G(z)) \approx 1$, which means the generator is able to fool the discriminator to misclassify its output as real. The overall cost function for both players can be summarized as a **value function** where discriminator maximizes the objective, while the generator tries to decrease the discriminator's performance by minimizing the same objective [98, 88]

$$V(\theta^{(D)}, \theta^{(G)}) = -J^D(\theta^{(D)}, \theta^{(G)}), \tag{2.34}$$

and the Nash equilibrium solution is achieved by

$$(\hat{\theta}^{(D)}, \hat{\theta}^{(G)}) = \arg\min_{\theta^{(G)}} \max_{\theta^{(D)}} V(\theta^{(D)}, \theta^{(G)}), \tag{2.35}$$

where the optimization is carried out by alternating between minimizing and maximizing the sub-objectives using one **gradient ascent** on discriminator followed by one **gradient descent** on generator. Goodfellow *et al*. [9] have shown that this objective resembles minimizing the *Jensen-Shannon divergence* between the data and the model distribution.

$$V(\theta^{(D)}, \theta^{(G)}) = -\log(4) + KL\left(p_{\text{data}} \| \frac{p_{\text{data}} + p_{\text{model}}}{2}\right) + KL\left(p_{\text{model}} \| \frac{p_{\text{data}} + p_{\text{model}}}{2}\right), \tag{2.36}$$

where KL is the Kullback–Leibler divergence and the expression can be written as

$$V(\theta^{(D)}, \theta^{(G)}) = -\log(4) + 2.JSD(p_{\text{data}} \| p_{\text{model}}). \tag{2.37}$$

One shortcoming of this objective is that loss for the generator in Equation 2.33 may not provide sufficient gradient for G to learn well [9]. The loss function is visualized in Figure 2.11 where the dashed red line shows the objective for G. This loss function provides a small gradient for the samples that are not good (classified as fake $D(G(z)) \approx 0$) and large gradient for the good samples (classified as real $D(G(z)) \approx 1$). This is especially

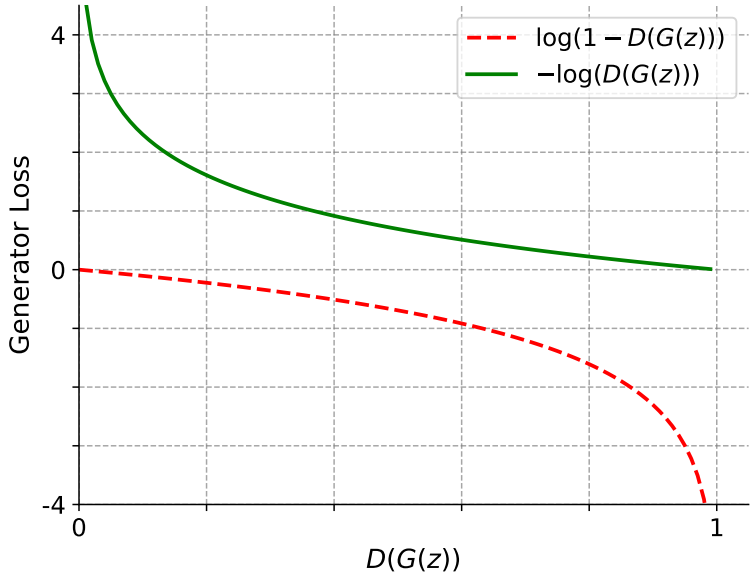


Figure 2.11: The original loss function for generator, shown in dashed red, provides small gradient for the samples that are not good ($D(G(z)) \approx 0$) and large gradient for the good samples ($D(G(z)) \approx 1$). The heuristic non-saturating loss function for generator, shown in green, is proposed by Goodfellow *et al*. [9], changes the generator loss from minimizing the log probability of the correct answer ($\log(1 - D(G(z)))$) to maximize the log probability of the wrong answer ($-\log(D(G(z)))$).

problematic early in training where the generator does not produce good samples and discriminator can reject samples with high confidence because they are clearly different than training data. To fix that, Goodfellow *et al*. [9] proposed a non-saturating loss function for

generator by changing the loss from minimizing the log probability of the correct answer $\log(1 - D(G(z)))$ to maximize the log probability of the wrong answer $-\log(D(G(z)))$, shown in Figure 2.11 in green. In practice, this loss function works better by providing large gradient early in training.

Even with the careful design of the generator loss, the objective function given in Equation 2.28 has a major flaw. The discriminator network proposed by Goodfellow *et al.* [9] outputs a probability value indicating whether or not a sample comes from the training data distribution. This achieved by adding a sigmoid non-linearity at the end of the network where the sigmoid function is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.38}$$

The output of the sigmoid function is between 0 to 1 which can be interpreted as the

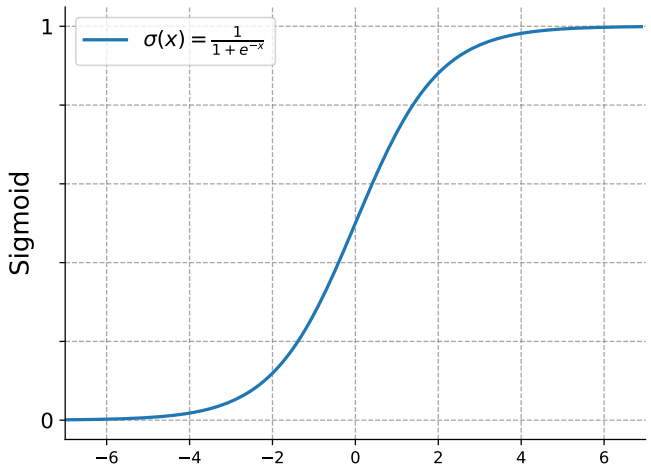


Figure 2.12: Visualization of the Sigmoid function. The function gets saturated with very large or small values where the gradient becomes very small.

probability. The problem with sigmoid function as can be seen in Figure 2.12 is that it gets saturated with very large or small inputs where the gradient becomes very small. In the

GAN objective, this corresponds to having a discriminator that can classify real and fake inputs with very high confidence while not providing enough gradient for the generator to train. The hinge version of the adversarial loss is proposed in the literature [99, 100] where the discriminator's output is no longer a probability value and instead it classifies real and fake samples with a large margin classifier. The objective function for discriminator and generator is given by

$$J^D = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\min(0, -1 + D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\min(0, -1 - D(G(z)))], \tag{2.39}$$

$$J^G = -\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[D(G(\boldsymbol{z}))]. \tag{2.40}$$

Similar to the original GAN loss function, the optimization is performed by maximizing J^D and minimizing J^G. The loss for discriminator is zero when the network's output beyond some margin $D(x) > 1$ for real samples and $D(G(z)) < -1$ for fake samples; Otherwise the loss penalizes misclassified outputs. The generator loss is a boundless minimization problem that encourages $D(G(z)) \gg 0$ where the generator manages to fool the discriminator. The optimization is carried out by alternating between maximizing and minimizing Equations 2.39 and 2.40 respectively. The hinge version of the adversarial loss does not suffer from the gradient saturation problem of Sigmoid function and is reported to be an effective loss function to train GANs [99, 100, 101, 102].

Deep Convolutional GANs

The original GAN formulation relies heavily on fully-connected neural networks. The input to the network is a vector z, normally sampled randomly from a uniform distribution and through series of fully-connected layers, the network learns the mapping from the distribution p_{z} to data distribution p_{data}. The process is very similar to **inverse transform sampling** (Smirnov transform) for random number sampling [103]. For example, pseudo-random number generators draw a scalar z from $U(0, 1)$ and apply a nonlinear transforma-

tion to a scalar x which is distributed according to $p(x)$ using the inverse of the cumulative distribution function. GANs are different in that $p(x)$ is not well defined. Instead, the network learns the mapping in an unsupervised manner [44].

Learning high dimensional distributions such as images are not very efficient with fully-connected layers and while CNNs excel at supervised learning of images, unsupervised learning with CNNs has received fewer attention [87]. Moreover, real applications of GANs oftentimes require some form of image data as input where a conditional version of generative adversarial nets is being used in an image-to-image setting [104, 89]. For example, single image super-resolution, and image inpainting problems both require models that accept a degraded image as input. Unsupervised learning for such problems, not only has the benefit of a practically unlimited amount of unlabeled images to train [105], but also has shown the advantage of being able to provide better representations for multi-modal data generation [98, 106].

Training convolutional version of GANs is more difficult than CNNs trained for supervised learning. Radford *et al*. [87] proposed an architectural topology called DCGAN (deep convolutional GAN) that makes GANs stable to train in most settings. DCGAN uses a fully-convolutional architecture in generator [75]. Fully-convolutional network replaces deterministic spatial pooling functions with strided convolutions and strided transposed convolutions [76] for spatial down-sampling and up-sampling respectively. Isola *et al*. [89] proposed a conditional GANs framework called Pix2Pix as a general-purpose solution to image-to-image translation problems. Inspired by DCGAN, they also propose a convolutional GAN architecture for both generator and discriminator that leverage convolution to handle high dimensional data such as images. In their work, they showed that in a conditional image generation setting, adversarial loss alone is not enough to generate near ground truth outputs, where they also included a reconstruction loss (in form of ℓ_1 or ℓ_2 norm) to the objective cost function of GAN. It is also shown that in a conditional convolutional

GAN setting, discriminator can also benefit from observing the input image [107, 89, 104].

In recent years, deep convolutional generative adversarial networks have had great success in generating natural images of the real world [89, 108, 36, 102]. It is also shown that a trained model learns good representations of images that can be used for generative modeling and unsupervised pre-training of supervised learning tasks [87, 56]. These methods, however, suffer from model instability problem where training of GANs sometimes becomes unstable as the generator and discriminator become stronger. In the next section, we address this problem and review some of the techniques that improve the stability of training and perceptual quality of GAN samples.

Improved GANs

One of the main challenges with training GANs is model stability. Training GANs consists in finding a Nash equilibrium to a game with two non-convex, non-cooperative neural networks. In practice, finding the equilibrium solution is a more difficult problem than optimizing an objective function mostly because the generator and discriminator do not learn with the same pace. For example, as the discriminator gets better, the updates to the generator get consistently worse. This is partly attributed to the saturation in the original GAN loss function. However, even with careful design of the loss function (see Section 2.3.3) the updates tend to get worse with stronger discriminator and optimization gets massively unstable [109]. In another scenario where the generator is trained faster sometimes the generator collapses into producing limited varieties of samples. This is a phenomenon commonly described as "mode dropping" where generator learns to ignore most of the modes in the distribution rather than including all modes of data. A complete mode collapse is not common especially for conditional GANs, however, a partial collapse can often happen. As an example, in a human faces dataset, a mode collapse happens when most of

the results share the same skin tone or have a smile on their face. The full coverage of sources of instability in GANs and the means to target them is beyond the scope of this dissertation. In this section, we review some of the techniques commonly used in practice that improve training and performance of GANs:

Feature Matching Feature matching addresses the instability of GANs by specifying a new objective for the generator that prevents it from over-training on the current discriminator [110]. The new objective requires the generator to produce images that match the statistics of the real data according to the discriminator. The feature-matching loss compares the activation maps in the intermediate layers of the discriminator between the real and generated data defined as

$$\mathcal{L}_{FM} = \mathbb{E}_{\substack{\boldsymbol{x}\sim p_{\text{data}} \\ \boldsymbol{z}\sim p_{\boldsymbol{z}}}} \left[\sum_i \frac{1}{N_i} \left\| D^{(i)}(\boldsymbol{x}) - D^{(i)}(G(\boldsymbol{z})) \right\|_1 \right], \tag{2.41}$$

where $D^{(i)}$ is the activation in the i'th layer of the discriminator and N_i is the number of elements in that layer. This stabilizes the training process by forcing the generator to produce results with representations that are most discriminative of real data versus data generated by the current model. This is similar to perceptual losses [90, 79, 111] where activation maps are compared using the pre-trained VGG network [70]. This loss is reported to be useful for high-resolution image synthesis problems in conditional GAN setting [108].

Patch-GAN Architecture One shortcoming of the discriminator network proposed in the original GAN is that the discriminator might learn to classify the fake images based on the subtle discriminative features in the generated image. For example, there might be subtle patterns or artifacts present in the generated image. Since the role of the discriminator is to classify the real and fake images, the network quickly learns to predict the fake image with high confidence solely based on subtle discrepancies and completely ignore

the perceptual "realism" in the generated images. As mentioned earlier, this discriminator becomes too powerful and the optimization gets unstable. Convolutional "PatchGAN" classifier is proposed in the literature which only penalizes structure at the scale of image patches [112, 89, 107]. In this architecture, instead of having the network output one scalar indicating the probability of an image being real, it outputs an array of scalars each representing a probability of a patch in the input image being real. Formally, let $\boldsymbol{x}$ be the input image to the discriminator D, the network maps $\boldsymbol{x}$ to a $N \times M$ array of outputs $D(\boldsymbol{x})$, where each $D^{(i,j)}(\boldsymbol{x})$ determines whether the patch $\boldsymbol{x}^{(i,j)}$ in the image is real or fake. The output $D^{(i,j)}(\boldsymbol{x})$ can be traced back to its receptive field to find the pixels it is sensitive to in the input image $\boldsymbol{x}$. The probability of an input $\boldsymbol{x}$ being real is then measured by

$$\hat{D}(\boldsymbol{x}) = \frac{1}{N \times M} \sum_{i,j} D^{(i,j)}(\boldsymbol{x}). \tag{2.42}$$

The large receptive field makes discriminator too powerful, while very small receptive field produces results with lower quality. Isola *et al*. [89] found the discriminator with the receptive fields of 70 to be most effective creating a balance between image quality and the performance of the discriminator. This means the final convolution layer produces scores predicting whether 70×70 overlapping image patches are real or fake.

Spectral Normalization Recently Miyato *et al*. [100] have proposed a new weight normalization technique called spectral normalization to stabilize the training of the discriminator. As discussed before, one of the main challenges with training GANs is to reach the right balance between the performance of both players. To solve the problem of imbalance update steps, two-timescale update rule (TTUR) [15] is suggested in the literature. It consists of providing different learning rates for optimizing the generator and discriminator. However, different learning rate steps make the GAN training slower. Spectral normalization is a way to control the performance of the discriminator by restricting the Lipschitz

constant of the entire network to one. Doing so prevents escalation of parameters magnitude in discriminator and avoids unusual gradients. This gives the generator an advantage to better match the statistics of real data when trained with the same learning rate as the discriminator. Spectral normalization is implemented by scaling down weight matrices of the discriminator by their respective largest singular values as discussed below

Let Lipschitz norm of a general differentiable function $g : \mathbb{R}^n \to \mathbb{R}^m$ be the supremum of spectral norm of its gradient over its domain

$$\|g\|_{\text{Lip}} = \sup_{\boldsymbol{h}} \sigma(\nabla g(\boldsymbol{h})), \tag{2.43}$$

where the spectral norm $\sigma(A)$ is the ℓ_2 matrix norm and the largest singular value of A:

$$\sigma(A) := \max_{\boldsymbol{h}\neq\boldsymbol{0}} \frac{\|A\boldsymbol{h}\|_2}{\|\boldsymbol{h}\|_2} = \max_{\|\boldsymbol{h}\|_2=1} \|A\boldsymbol{h}\|_2. \tag{2.44}$$

For a linear layer $g(\boldsymbol{h}) = W\boldsymbol{h}$, with W being the weight matrix, the Lipschitz norm is given by the largest singular value of the weight matrix

$$\|g\|_{\text{Lip}} = \sup_{\boldsymbol{h}} \sigma(\nabla g(\boldsymbol{h})) = \sup_{\boldsymbol{h}} \sigma(W) = \sigma(W). \tag{2.45}$$

We can write a neural network, in form of non-linear function composition of many linear layers. Let f be a neural network with the input $\boldsymbol{x}$

$$f(\boldsymbol{x}, \theta) = W^{L+1} a_L(W^L(a_{L-1}(W^{L-1}(\dots a_1(W^1\boldsymbol{x})\dots)))), \tag{2.46}$$

where $\theta := \{W^1, \dots, W^L, W^{L+1}\}$ is the learning parameters set, and a_i is an element-wise non-linear activation function of the layer i. We can use the inequality $|\langle x, y\rangle| \leq \|x\|_2\|y\|_2$ to observe the following bound on the Lipschitz norm of the network

$$\begin{aligned}\|f\|_{\text{Lip}} \leq & \|W^{L+1}\|_{\text{Lip}} \cdot \|a_L\|_{\text{Lip}} \cdot \|W^L\|_{\text{Lip}} \\ & \cdots \|a_1\|_{\text{Lip}} \cdot \|W^1\|_{\text{Lip}} = \prod_{l=1}^{L+1} \|W^l\|_{\text{Lip}} = \prod_{l=1}^{L+1} \sigma(W^l).\end{aligned} \tag{2.47}$$

Note that for a ReLU non-linearity we have $\|a_i\|_{\text{Lip}} = 1$ and the spectral norm of the network is bounded by the product of the largest singular values of all weight matrices. We can normalize the spectral norm of the weight matrix W to satisfy $\sigma(W) = 1$

$$\bar{W}_{\text{SN}}(W) := W/\sigma(W). \tag{2.48}$$

If we apply 2.48 for every weight matrix W^l in the network we see that the $\|f\|_{\text{Lip}}$ is bounded above by 1. It is worth noting that the computation can be heavy if we naively compute singular value decomposition at each iteration, instead, Yoshida & Miyato [113] proposed to use the power iteration method to estimate $\sigma(W)$. In practice, spectral normalization is computationally very efficient and the training becomes more stable compared to other regularization techniques for GANs. Although the spectral normalization was originally proposed to regularize discriminator, recent research [102, 114] have shown that generator can also benefit from it by suppressing sudden changes of parameter and gradient values in the generator.

2.4 Summary

This chapter presents an overview of deep neural networks. We briefly reviewed inductive learning of functions from examples and discussed different types of learnings based on the availability of feedbacks for an intelligent agent to learn. In particular, we discussed supervised and unsupervised learning problems and how learning involves finding a hypothesis that agrees with examples in the training set. Examples for each learning paradigms are included and learning was formulated as an optimization problem that estimates the hypothesis by reducing the disagreement between the model output and its expected value. To that end, an objective function, also called a loss function, is associated with the hypothesis and is minimized using an optimization algorithm. Stochastic gradient descent is reviewed as a common derivative-based optimization method to train neural networks and the backpropagation algorithm is discussed as an efficient way to find the derivatives in a parameterized computational graph.

The chapter continues with a formal definition of neural network and some of the most common neural networks architectures were discussed: A multilayer perceptron is a vanilla neural network architecture where neurons in each layer are connected to all neurons in the previous layer. Convolutional neural networks (CNN), on the other hand, are more suited for data with spatial structure such as images. These networks use convolution operators in place of general matrix multiplication and their parameters are the values of convolutional kernels. Various methods to improve CNNs performance such as fully convolutional architecture, residual blocks, and normalization schemes were also reviewed.

Finally, generative adversarial networks (GANs) as a type of unsupervised generative neural network framework were discussed in detail. GANs include two neural network players (generator and discriminator) that compete with each other in a zero-sum game to generate realistic looking images authentic to human observers.

3. Image Structures & Evaluations

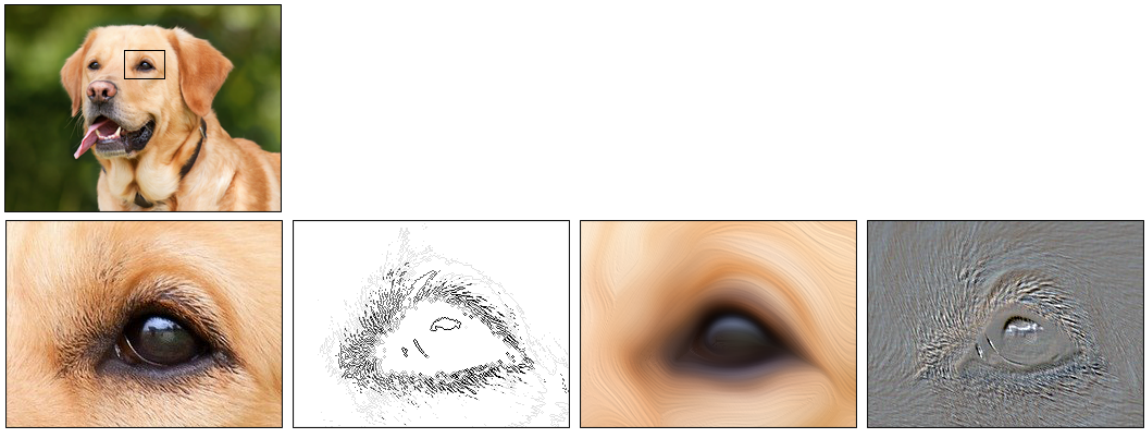
In this chapter, we discuss two main concepts used in digital image processing: structural properties of an image in the form of edge information, and image similarity metrics for evaluations and quality assessments. Related works and studies similar to those considered in this thesis are reviewed. Starting with image structure, edges as the most salient spatial information in an image are discussed. This is followed by a mathematical definition of edges and different types of edge detection schemes are reviewed.

This chapter continues with discussions about common similarity metrics and quality assessment techniques used in characterizing the performance of image restoration algorithms, more complex test analysis and optimization algorithms.

3.1 Image Structures

A digital image is a numerical representation of color information for finite number of elements in an image, each of which with a particular location and value, known as *pixels* [115]. The technical definition of a pixel is context dependent and it may refer to a physical single unit on a photosensor element in a digital camera, or a virtual unit in display devices. In digital images, color information is encoded in a *color space* and each pixel carries enough information to represent visually acceptable color in that single unit. For example in the RGB color space, each pixel carries three values, or channels, to identify the color. These values together indicate the intensity and chrominance of light. The number of pixels per distance unit (*e.g.* inch) and the number of bits used to store color information (known as color depth) determine the precision with which colors can be expressed.

Pixels together in a spatial neighborhood form structural information. Using pixel intensity values and their intrinsic correlation lets us extract more abstract visual elements from an image that each play a distinct role in semantic attributes in the scene. For example edges, regional colors, and textures each represent a visual aspect that can be extracted from the correlation of pixels in an image. Figure 3.1 shows some aspects in a digital image: *Edges* are the areas with some discontinuity or abrupt change in the brightness. These regions carry the most important semantic associations in an image and can be grouped together to construct edge-maps, contour, or lines. Edges are oftentimes referred to as regions with high spatial frequencies. *Regional Color* is simply distribution of a visible electromagnetic spectrum in a region. Most neighboring pixels exhibit a smooth transition from one color to another, making them regions with low spatial frequencies. *Texture* provides measures of properties such as smoothness, coarseness, and regularity [115]. Texture can be described by its statistical properties such as smoothness or coarseness, structural properties such as regular or stochastic pattern, or spectral properties by studying spatial frequencies.



Original Image Edge Map Color Regions Texture

Figure 3.1: Different visual components in a color image. From left to right: 1) Original color image, 2) Edge map as discontinuities in the image brightness, 3) Color as the visible spectral distribution, and 4) Texture that describes properties such as smoothness, coarseness, and regularity of the surface.

In this section, we focus on edges as the most salient structural features in an image. It is interesting that humans even young have an uncanny ability to recognize objects and scenes from line drawing or to complete the visual inconsistencies by connecting those lines [116]. However, this relatively simple task that humans can effortlessly perform, is extremely difficult for machines. For example in a single image super-resolution task, the main challenge is to reconstruct the high-frequency components (edges) lost due to the downsampling procedure. Reconstruction of color and to some extent texture from the low-resolution image is a relatively easier task compared to edges. Without proper reconstruction of edges, the high-resolution image looks over-smoothed, blurry or pixelated at best. In the following sections, we provide a formal definition of edge and discuss various edge detection methods. In section 4.3 we will present models to reconstruct high-frequency components from partially available edge-maps or low-resolution image.

3.2 Edge Detection

Edge detection is a fundamental step for many computer vision tasks such as segmentation, object detection and recognition, motion tracking, medical imaging, image-to-text analysis and many more. Since pre-historic times, humans have used lines and sketching to depict our visual world. It is shown that humans can correctly identify the object category of a sketch 73% of the time [33]. In 1962, Hubel and Wiesel's experiment led to an understanding of visual cortex of non-primate species as a population of feature detectors. Their findings showed how some neurons in the cortex, or "simple cells" as they called them, were responding to light patterns, edges, and bars of various widths and orientation. It was later shown that this perception of edges in natural images occurs over different scales [117] and while precisely localizing the edges seems to be a subjective matter, Martin *et al*. [118] showed that there is a strong consistency between humans when asked to locate the edges in an image. In their study for boundary detection, they found F-score of 0.80 among human subjects which underscores the consistency in the perception of edges among human. The F-score is a harmonic mean between the *precision* and *recall* of a binary classifier.

Edges can be seen as the boundaries between regions of different color or brightness. For example, edges can be caused by a discontinuity in surface normal, depth, surface color, and illumination. Marr *et al*. [117] in their *Theory of Edge Detection*, defined edges as spatially localized regions caused by the following factors

- Illumination changes such as shadows, visible light sources, and gradients.
- Changes in the orientation or distance from the viewer of the visible surface.
- Changes in surface reflectance.

By this definition, we differentiate edges from points, surface textures, and noise and seek methods that explain these localized changes in the spatial domain. In the past few decades, a great deal of literature was dedicated to computational edge detection. Computational

edge detection is an image processing technique designed to detect edge pixels. While early methods were mostly relying on image gradients, local image statistics, and hand designed feature extractors, recent developments are using Convolutional Neural Networks that emphasize the importance of automatic hierarchical feature learning [119]. It is important to note that detecting edge pixels that give rise to edges varies by context and application. While some applications such as segmentation require only object boundaries (contours) to be selected as edges, others like medical image processing require full edge map to find "pathological" objects in the image. Figure 3.2 shows different edge detection schemes applied on an image. (a) The contour map using Suzuki *et al*. [10] method, detects the object boundary by separating the foreground objects from background. (b) The edge map is extracted using Canny edge detection [11]. The result is a sharp binary edge mask that each pixel either belongs to the background (white) or edge map (black). (c) The image gradient is generated using the Sobel operator [12], the result looks like hand-drawn sketches with shades of gray and more emphasis on the edges.

(a) Original Image (b) Contour Map (c) Edge Map (d) Gradient Magnitude

Figure 3.2: Different edge detection algorithms. From left to right: a) Original color image, b) Contour map extracted using Suzuki *et al*. [10] method, c) Edge map retrieved using Canny edge detection [11], d) Gradient magnitude after applying Sobel operator [12].

In the following sections, we review various edge detection schemes used throughout this dissertation. We discuss mathematical definition of each edge detection and how to implement and incorporate them into deep learning models.

3.2.1 Gradient-Based Edge Detections

As we discussed earlier, edges occur at the boundaries between regions of different color or brightness, and it is a common approach to define edges as the location of pixel intensity change in the image. By quantizing digital images in a spatial domain, the maximum possible intensity change becomes finite and the minimum span of the intensity change is the distance between two adjacent pixels. Under this condition, a reasonable way to define edges is through image derivatives. For a one-dimensional function $f(x)$, an approximation for $f(x + \Delta x)$ can be achieved by expanding the Taylor series about point x

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x. \tag{3.1}$$

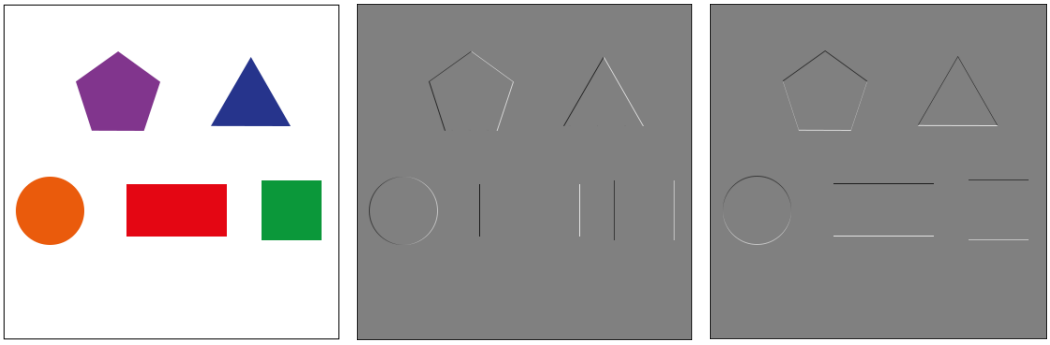
For discrete data such as images, the smallest value Δx can take on is 1, and since we are dealing with two-dimensional images we define the first order derivatives of an image as

$$\begin{aligned} \frac{\partial f(x,y)}{\partial x} &= f_x \approx f(x+1,y) - f(x,y), \\ \frac{\partial f(x,y)}{\partial y} &= f_y \approx f(x,y+1) - f(x,y). \end{aligned} \tag{3.2}$$

The partial derivative defined in the equation 3.2 is known as *forward difference*. Using the same technique, we can define *backward difference* using $f_x \approx f(x,y) - f(x-1,y)$, or *central difference* by using $f_x \approx (f(x+1,y) - f(x-1,y))\ /\ 2$. First order derivative of the image has the following properties: (1) it is zero in the areas of constant intensity; (2) it is non-zero at the onset of intensity change. (3) it is non-zero at the points along the intensity change [115]. We can easily find first derivative of an image by using convolution. For example, the following kernels can be used to approximate backward difference

$$H_x = \begin{array}{|c|c|}\hline -1 & 1 \\ \hline \end{array} \qquad H_y = \begin{array}{|c|}\hline -1 \\ \hline 1 \\ \hline \end{array}$$

Figure 3.3: Convolution kernels to approximate first order partial derivatives in an image.



(a) Original Image (I) (b) Image Derivative ($H_x * I$) (c) Image Derivative ($H_y * I$)

Figure 3.4: Visual comparison of first order partial derivatives of an image using convolution filters. (a) original image, (b) derivative along x axis captures vertical edges, (c) derivative along y axis captures horizontal edges.

It is worth noting that image derivatives are not defined over the image boundaries. To fix that we can either assume that the derivative at the boundary is zero (Neumann boundary condition), constant (Dirichlet boundary condition), or continuous by imposing periodic or reflection boundary conditions. Throughout this dissertation, we use reflection boundary condition over the image boundaries. This technique pads the image using the reflection (about both axes) of the input image and preserves the direction of edges over the boundaries. Figure 3.4 shows the visualization of the first order derivatives of an image using convolution. Note how derivative along x and y axes capture vertical and horizontal edges respectively. Kernels showed in 3.3 are simple and easy to compute, however, symmetric kernels about the center point have shown to be more useful to capture edges while in general using more pixels to compute edges makes the procedure less sensitive to noise. The smallest symmetric kernel is of size 3×3. Prewitt operator [13], for example, consists of two symmetric 3×3 kernels that de-emphasize values near the center and are used to capture horizontal and vertical edges. The Sobel operator [12] uses the same technique with more emphasis to changes around the center pixel. It was shown that using 2 in the

center location provides image smoothing [115, 12] which makes the Sobel operator better noise-suppression edge detector. Prewitt and Sobel operators are shown in Figure 3.5.

	$\frac{\partial}{\partial x}$	$\frac{\partial}{\partial y}$
Prewitt	$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$	$P_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$
Sobel	$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Figure 3.5: Finite difference filters used to approximate derivative. Prewitt operator [13] de-emphasizes values near the center. The Sobel operator [12] gives more emphasis to changes around the center pixel. Note that these filters sum to zero.

The gradient operators discussed here are designed to capture horizontal and vertical edges and as a result their response are weaker in diagonal directions. When edges along the diagonal direction are of interest, one can use either horizontal or vertical kernels. However, it is a common practice to use image gradient magnitude and direction instead. Considering the 2D image gradient as a vector of partial derivatives at each pixel $\nabla I = \langle I_x, I_y \rangle$ that points to the direction of maximum positive change, we use ℓ_2 norm as the magnitude of the gradient vector along the direction defined as

$$\begin{aligned} \|\nabla I\|_2 &= \sqrt{I_x^2 + I_y^2}, \\ \theta &= \tan^{-1}\left(\frac{I_y}{I_x}\right). \end{aligned} \tag{3.3}$$

The gradient magnitude represents the total amount of change at each point. It is expected to be large at the edges and zero in the areas of constant intensity. While gradient orientation specifies the direction where the change occurs. Combining the gradient with thresholding can be useful to extract sharp edges and minimize the effect of noise.

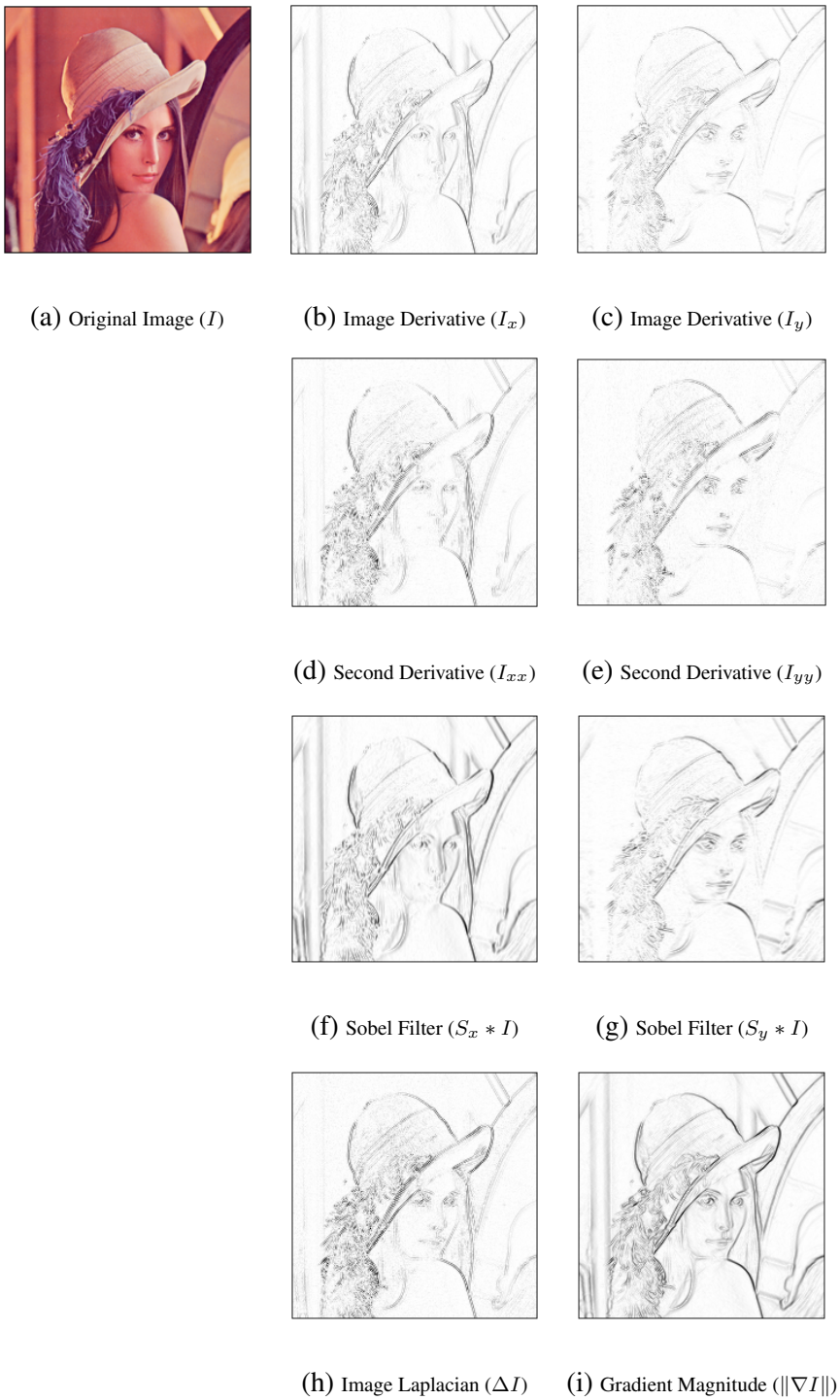
Second order derivatives can also be used to approximate edges in the image. The direction of the first derivative changes over an edge as can be seen with a dark and light color around an edge in Figure 3.4. This means that the second derivative has the following properties: (1) it is zero in the areas of constant intensity; (2) it is non-zero at the onset of intensity change (positive at the start of the ramp, negative at the end of the ramp). (3) it is **zero** at the points along the intensity ramp with constant slope [115]. The last property can be used to locate the centers of thick edges by finding the zero crossings of the second derivative. Using first derivatives in equation 3.2 we can approximate second derivatives as

$$\begin{aligned} \frac{\partial^2 f(x,y)}{\partial x^2} &= f_{xx} \approx f(x+1,y) - 2f(x,y) + f(x-1,y), \\ \frac{\partial^2 f(x,y)}{\partial y^2} &= f_{yy} \approx f(x,y+1) - 2f(x,y) + f(x,y-1). \end{aligned} \tag{3.4}$$

Using second derivatives result in stronger response and produce thinner edges. We can use the second derivatives to calculate image Laplacian:

$$\begin{aligned} \Delta I &= I_{xx} + I_{yy} \\ &\approx f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y). \end{aligned} \tag{3.5}$$

One advantage of image Laplacian is that it is invariant to rotation (isotropic), that means it captures intensity changes equally in every direction, thus avoiding using multiple kernels to calculate edges at different direction and point. Similar to first derivative operators, second derivatives can be computed using convolution. Figure 3.6 shows different gradient-based edge detectors discussed in this chapter.



(a) Original Image (I) (b) Image Derivative (I_x) (c) Image Derivative (I_y)

(d) Second Derivative (I_{xx}) (e) Second Derivative (I_{yy})

(f) Sobel Filter ($S_x * I$) (g) Sobel Filter ($S_y * I$)

(h) Image Laplacian (ΔI) (i) Gradient Magnitude ($\|\nabla I\|$)

Figure 3.6: Visual comparison of different derivative-based edge detectors. Edges are acquired by applying convolution operator on the original image using various linear difference filters.

One problem with image derivatives is that they are highly sensitive to noise. Noise is often high-frequency random perturbation in digital images and since gradient accentuates high frequencies it exacerbates the noise. This is worse with second derivatives as they will exaggerate noise even more. One solution is to smooth the image with a low-pass filter prior to computing gradient. The Gaussian operator is being proposed in the literature as a smoothing process because of its desired properties: (1) The Gaussian operator is a circularly symmetric filter which does not affect the orientation of the edges in the image. (2) Unlike average function, the Gaussian operator is smooth in both the spatial and frequency domains and less likely to introduce image artifacts [115]. (3) The amount of smoothing can be controlled by varying the standard deviation σ of the Gaussian. It is worth noting that computing the derivative on a Gaussian smoothed image can be sped up using the associative property of convolutions

$$\nabla[G_\sigma(x,y) * I(x,y)] = [\nabla G_\sigma](x,y) * I(x,y), \tag{3.6}$$

where G_σ is a Gaussian kernel with the standard deviation of σ. The derivative of a Gaussian kernel function can be seen as

$$\nabla G_\sigma(x,y) = (\frac{\partial G_\sigma}{\partial x}, \frac{\partial G_\sigma}{\partial y})(x,y) = \langle -x, -y \rangle \frac{1}{2\pi\sigma^4} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right). \tag{3.7}$$

The above equation can be used to create a symmetric $n \times n$ kernel. Computing the derivative of the Gaussian is the first step in many edge detection algorithms. For example, Marr *et al*. [117] proposed to convolve an image with Laplacian of a Gaussian kernel and then find the zero crossing of the result to determine the exact location of edges. John F. Canny [11] proposed to filter the image with x and y derivative of Gaussian prior to computing the gradient magnitude and orientation, the process continues with a non-maximum suppression and thresholding to create a binary edge map. This algorithm is known as Canny edge detector which we will discuss in the next section.

3.2.2 Canny Edge Detection

Canny edge detector is the most widely used edge detector in computer vision and digital image processing. It was proposed by John F. Canny [11] in 1986 as a computational approach to *edge linking*. In edge linking, unlinked edges are matched with their neighbors in both directions to form chains. To match neighboring lines and create a continuous chain, characteristics of edge such as orientation and phase are sometimes used. This process is followed by thresholding with hysteresis to remove low-strength edges. Canny edge detector is based on three objectives:

- *High sensitivity and specificity*. That is all relevant edges should be found and edges detected should be relevant.
- *Well localized*. Edge points should be as close as possible to the center of the edge.
- *Single edge response*. Pixels contributing to an edge must be marked only once.

A closed-form solution that satisfies these objectives may be difficult to find. Canny edge detection is a set of mathematical steps that lead to a good approximation of the optimal solution. A pseudo-code for Canny edge detection is shown in Algorithm 1. The process beings with smoothing the image with a circular 2-D Gaussian function and computing the gradient of the smoothed image. As we showed in equation 3.7, the process can be done faster using only two convolution operators to compute x and y gradients. After this, gradient magnitude and orientation are computed for every pixel in the image. The next step is to remove any unwanted pixels that do not constitute the edge. A simple gradient magnitude thresholding creates ridges and thick edges which violates the third objective. This process is usually performed by a method called *non-maximum suppression* as explained below.

Algorithm 1 Canny edge detection

```
1:  procedure CANNY(img, sigma, high, low)
2:      sm ← GAUSSIAN (img, sigma)                          ▷ Gaussian smooth image
3:      ix ← SOBEL (sm, axis=0)                             ▷ find x derivative
4:      iy ← SOBEL (sm, axis=1)                             ▷ find y derivative
5:      gm ← SQRT (ix^2 + iy^2)                             ▷ gradient magnitude
6:      go ← ARCTAN (iy, ix)                                ▷ gradient orientation
7:      nms ← NON_MAX (gm, go)                              ▷ non-maximum suppression
8:      edges[,] ← 0
9:      for (x, y) ∈ nms do
10:         if nms[x, y] > high then                        ▷ passing a high threshold
11:             edges[x, y] ← 1
12:         else if nms[x, y] > low then                    ▷ passing a low threshold
13:             if NEIGHBORS (edges[x, y]) = 1 then         ▷ connected to sure-edges
14:                 edges[x, y] ← 1
15:             else                                        ▷ not connected to sure-edges
16:                 edges[x, y] ← 0
17:             end if
18:         else                                            ▷ not passing a low threshold
19:             edges[x, y] ← 0
20:         end if
21:     end for
22:     return edges
23: end procedure
```

The non-maximum suppression procedure checks every pixel for being a local maximum in its neighborhood in the direction of the gradient vector, also known as *edge normal*. If the point is a local maximum it is considered, otherwise, it is surpassed (put to zero). To check if a pixel is a local maximum, normally a number of discrete orientations are considered in a local region (*e.g.* 3×3 window): for example vertical, horizontal, $45°$, and $-45°$. If a pixel is less than any of its two neighborhood along the gradient vector it is set to zero (suppressed); otherwise, it is considered. Figure 3.7 shows this procedure for different edge

normals, in each case pixel p_5 constitutes an edge.

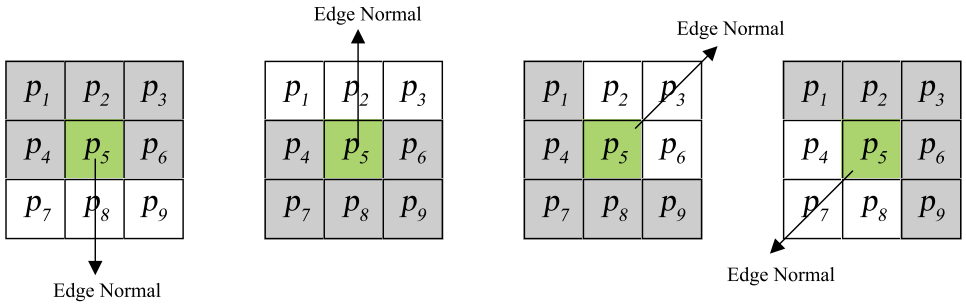


Figure 3.7: Non-maximum suppression procedure: a pixel is checked if it is a local maximum in its neighborhood along the direction of edge normal. p_5 is maximum in all cases.

The final stage in the Canny edge detection algorithm is thresholding. A simple thresholding scheme in which pixel intensities below some value are set to zero might not be very useful: if the threshold is set too low, the false-positive error increases. If the threshold is set too high, the false-negative error increases. Canny detector improves this by using *hysteresis thresholding*. In this scheme, two thresholds are defined: low and high. Each pixel on the edge is examined against both thresholds. Any pixel intensity above the high threshold is considered an edge, while the values below the low threshold are discarded. Values that lie between two thresholds are only considered to be an edge if they are connected to "sure edge" pixels, otherwise, they are discarded. At this point, the result is returned as the final edge map. In practice oftentimes edges thicker than one pixel still remain where the process is typically followed by one pass of an edge-thinning algorithm.

The Canny edge detector is a powerful and accurate edge detector with nice properties such as sharp edges and the output is a binary mask. However, it comes with some shortcomings. The width of the Gaussian filter σ, and the values of high and low thresholds must be selected with caution. Small values of σ causes lot of small edges and noise to appear in the edge-map. Increasing the σ may cause position shift in the edge-map, also edges

Original Image $\sigma = 1.0$ $\sigma = 2.0$ $\sigma = 3.0$ $\sigma = 4.0$

Figure 3.8: Edge-maps generated by Canny edge detector for different values of Gaussian width σ. Increasing σ smooths the image and reduces the amount of edge.

might merge together or split in two with large σ. Figure 3.8 shows edge-maps generated by Canny edge detector for different values of Gaussian width. Another shortcoming of the Canny edge detector is the speed. Although the algorithm is quite fast, it is difficult to parallelize the algorithm to leverage extreme performance with GPU acceleration. The hysteresis thresholding in the algorithm needs to be done sequentially which makes Canny edge detector less useful in deep-learning applications. In the next section, we will review recent developments in edge detection using Convolutional Neural Networks (CNN). Learning-based edge detections can be used in deep-learning applications as hierarchical feature extractor, or a loss function.

3.2.3 Learning-Based Edge Detections

Classic edge detection schemes such as derivative-based methods work well with localizing the edges however, they don't have high specificity (true negative rate). Canny edge detection improves derivative-based methods by using a Gaussian kernel to remove the noise and non-maximum suppression to produce single edge response. The main problem with Canny edge detection is that the standard deviation of the Gaussian kernel must be determined prior to edge detection. Also, the hysteresis thresholding that aims at creating

continuous edge-map requires the values of high and low thresholds to be handpicked for every image and as mentioned before, cannot be parallelized. Learning-based edge detections methods aim at improving the quality of the edges and generally don't require manual parameters selection. These methods are mostly dominated by deep learning techniques that leverage convolutional networks and automatic hierarchical feature learning in a supervised setting. In short, learning-based methods bring several advantages over classical methods:

- Require little or no manual parameter selection.
- Improve speed by leveraging convolution and parallelization by orders of magnitude.
- Learn deep hierarchical representations that are essential in challenging ambiguous cases of edges and object boundaries.
- Can be used in existing deep learning models as a feature extractor or an objective function in the optimization algorithm.

While humans have uncanny ability to identify objects from sketches or draw contours of natural scenes [33], designing an edge detection scheme that does not differentiate the semantic object boundaries and abrupt changes in low-level image cues is a difficult task [120]. Deep learning based methods, on the other hand, predict the probability of an edge or a local contour map for each pixel by extracting and combining hierarchy of features from correlations between many pixels. This is comparable to different neurons responding to different light patterns in the visual cortex of non-primate species. Most deep learning based edge detection methods are trained as a binary classifier on a pixel level. The binary cross-entropy loss is mostly used to train these models. By definition, the cross-entropy between two distributions p and q is given by

$$\mathbb{H}(p, q) \triangleq \mathbb{E}_p[-\log q] = -\sum_x p(x) \log q(x). \tag{3.8}$$

An information theory-based interpretation of cross-entropy is the average number of bits needed to encode data coming from a source with distribution p when we use model q as the encoder [121]. By considering Bernoulli distribution as the underlying distribution of binary masks, the binary cross-entropy is defined as

$$\mathbb{H}_b(p, q) = -p \log(q) - (1 - p) \log(1 - q), \tag{3.9}$$

where $p \in \{0, 1\}$ defines the true distribution of the binary edge-map. This loss function penalizes both false-positive and false-negative results equally and can be used to train a convolutional neural network in a supervised setting [122, 121, 123]. One shortcoming of binary-cross-entropy loss to train edge detection models is that for a typical natural image, is that almost 90% of the edge-map consists of non-edge pixels and the distribution of edge/non-edge is heavily biased. To balance the loss between positive/negative pixels, Xie *et al*. [119] introduced a class balance weight β to the loss on a per-pixel term:

$$J(Y|X) = -\beta \sum_{i \in Y_+} \log \Pr(y_i = 1|X) - (1 - \beta) \sum_{i \in Y_-} \log \Pr(y_i = 0|X), \tag{3.10}$$

where X is the input image, Y is the binary mask label, y_i is a mask prediction, $\beta = |Y_-|/|Y|$ and $|Y_-|$ is the non-edge pixels in the label. Their model, Holistically-Nested Edge Detection (HED), applied this loss on different scales and by fusing multi-scale responses they produced clean, sketch-like edge detection model. When detecting edges with different semantics and labels are required, the categorical version of the cross-entropy loss can be used. The accuracy of the edge detection is normally reported using F-score as a harmonic mean between the precision and recall of the predicted edge map. In chapter 4 we will use HED, as an alternative edge detection for our proposed inpainting model and show how it improves the inpainting quality compared to Canny edge detection.

3.3 Image Quality Assessments and Similarity Metrics

What distinguishes a good quality image from a bad one is subject to human visual perception. Humans perceive an image under some degradation process, normally a bad quality image. Degradation of visual quality in a digital image happen during image acquisition, storage, transmission, compression, and processing and may appear in various forms of noise, blur or different image artifacts. For example, different kinds of noise appear in an image by poor illumination, non-stabilized cameras, quantization, fluctuations in an electric circuit, or physical effects while out-of-focus cameras, image compressions, or down-sampling cause blurriness in the image.

Proper quality assessment techniques are vital to the field of image processing and are often used as a basis for more complex structural analysis. In this section, two class of image quality assessment techniques are reviewed and their advantages and shortcomings are discussed: 1) *Objective* evaluation techniques provide a quantitative measure to assess the perceived image quality. These metrics can be used as an optimization objective in image restoration and enhancement algorithms as well as means to monitor the visual performance of the process. Depending on the availability of a distortion-free original image, objective evaluations may be categorized into *full-reference*, *no-reference*, and *reduced-reference* quality assessments [124].

2) *Subjective* image evaluation techniques require a human to assess image quality. Although these metrics are mostly expensive, time-consuming, and may contain inaccurate predictions, they still remain the only correct method to evaluate visual image quality. In many generative models, for example, the actual results may have a very good image quality but perceptually not very relevant and require human input to evaluate the perceptual quality. In most cases, similarity tests used in experimental psychology are designed to gather and coordinate the human intelligence to perform the evaluation. [125]

In this section, the most popular quality assessment techniques and methods to evaluate the performance of image restoration models are discussed in detail.

3.3.1 Mean Absolute Error

The mean absolute error (MAE) is a dominant performance metric in the field of digital signal processing. It is a method of choice for many signal processing applications despite its weak performance and serious shortcomings when dealing with perceptually relevant signals such as images [126, 127, 128]. From a mathematical standpoint, MAE between two discrete signals represented as vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$\text{MAE}(\mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} |x_i - y_i|. \tag{3.11}$$

Where for an image signal, N is the number of pixels and x_i and y_i are the intensity values for ith pixel in $\mathbf{x}$ and $\mathbf{y}$ respectively. The $|x_i - y_i|$ in MAE is called an error term and corresponds to Manhattan distance or ℓ_1 norm $\|\mathbf{x} - \mathbf{y}\|_1$ between vectors $\mathbf{x}$ and $\mathbf{y}$. The more general distance form is the normalized ℓ_p norm for $p \geq 1$

$$d_p(\mathbf{x}, \mathbf{y}) = \frac{1}{N} \left(\sum_{i=1}^{N} |x_i - y_i|^p \right)^{1/p}. \tag{3.12}$$

It is worth noting that with $p = 2$ in equation 3.12, the distance metric is called *root mean square error* (RMSE) which is also a very popular distance metric with similar properties as MAE. In the context of deep learning, this metric can be used both as an optimization objective (ℓ_1 or ℓ_2 losses) or an evaluation metric in which the metric is defined over a mini-batch of samples. By rewriting the mean absolute error as an expectation with respect to random variables $\mathbf{X}$ and $\mathbf{Y}$ the equation becomes

$$\text{MAE} = \frac{1}{m} \sum_{i=1}^{m} \mathbb{E}\Big[|\mathbf{X}^{(i)} - \mathbf{Y}^{(i)}| \Big], \tag{3.13}$$

where m is the number of samples in the mini-batch.

The MAE metric has some nice properties which made it very popular. In particular, It is very simple, parameter-free and computationally efficient. MAE can be evaluated for each sample independently, which makes it very easy to implement on GPU to benefit from parallelism and achieve higher performance. Moreover, ℓ_1 norm is a valid distance metric in $\mathbb{R}^N$ which satisfies nice properties such as positivity, homogeneity and triangle inequality. Finally, it is a very popular optimization objective, where it is widely used in deep generative models as a *reconstruction loss*.

Despite its popularity, MAE has some fundamental shortcomings as a quality metric. Firstly, MAE evaluates at the pixel level which means it fails to capture the perceptual similarity between visual signals. As a very basic example, two exact images where one is shifted by one pixel may result in a large MAE despite the fact that images are very similar. Secondly, MAE can be approximated with Laplace distribution and since the Laplace distribution is unimodal with a sharp peak at its mean, MAE favors data points around the mean. In case of color images, for example, this means that images with the mean color across their pixel intensities, similar to "sepia effect", will have a smaller MAE. Finally, MAE fails to differentiate between different types of distortions. Wang *et al*. [126] have shown that an image altered by contrast stretch, mean luminance shift, additive Gaussian noise, blur, spatial scaling, and shift, although dramatically different in visual quality, have nearly identical MAE when compared to the original image.

Despite all its flaws, MAE will continue to be widely used as the most popular distance measure in signal processing and many research publications report this metric in their results.

3.3.2 Peak Signal to Noise Ratio (PSNR)

Signal-to-noise ratio (SNR) is a measure based on the power spectra of noise and of the undegraded image. This ratio gives a measure of the level of information bearing signal power to the level of noise power [115]. SNR is defined as

$$\text{SNR} = \frac{\frac{1}{N}\sum_{x=1}^{N} I(x)^2}{MSE}, \tag{3.14}$$

where $I(x)$ is the original image, N is number of pixels in the image and MSE is the mean square error between the original image and its estimate $\hat{I}(x)$ defined as

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left[I(x) - \hat{I}(x)\right]^2. \tag{3.15}$$

The *peak-signal-to-noise ratio* (PSNR in dB) is defined as

$$\text{PSNR} = 10\log_{10}\frac{I_{max}^2}{MSE} = 20\log_{10}(I_{max}) - 10\log 10(MSE), \tag{3.16}$$

where I_{max} is the maximum signal value, *e.g.* 255 for eight-bit images. The closer $I(x)$ and $\hat{I}(x)$ and are, the larger this ratio will be. For eight-bit image data, the PSNR values between 20 dB to 25 dB are considered to be acceptable quality while images with very good qualities yield values larger than 30 dB.

For color images, the PSNR is defined by taking MSE over all pixel values of each individual channel, divided by image size and number of channels. Alternatively we can find it on a grayscale image or a luminance channel in a color space, such as LAB [129].

PSNR is widely used in characterizing the performance of image restoration algorithms. Despite being a simple mathematical measure, none of the complex objective metrics in the literature has shown a clear advantage over PSNR under strict testing conditions [130, 128]. However, while high-quality images often yield higher PSNR, the opposite is not always true. As mentioned before, objective metrics do not necessarily correlate well to

visual quality [126, 127]. While PSNR operates on the pixel level, it suffers from the same shortcomings as MAE. A Higher level similarity measure that can exploit feature extraction, pattern matching, and perceptual understanding of an image is more desirable.

3.3.3 Structured Similarity (SSIM)

The *structural similarity index* (SSIM) is a full reference objective image quality assessment technique based on the degradation of structural information that takes advantage of known characteristics of the human visual system [124]. SSIM addresses the biggest shortcoming of MAE and PSNR and instead of comparing image signals directly on a pixel level, compares local patterns of images after normalizing them for luminance and contrast. SSIM separates the task of quality assessment into three components: luminance, contrast, and structure and combine them to form an overall similarity measure between two images $\mathbf{x}$ and $\mathbf{y}$ defined as

$$\text{SSIM}(\mathbf{x}, \mathbf{y}) = f(l(\mathbf{x}, \mathbf{y}), c(\mathbf{x}, \mathbf{y}), s(\mathbf{x}, \mathbf{y})), \tag{3.17}$$

where $l(\mathbf{x}, \mathbf{y})$ is a luminance comparison function with luminance of an image estimated as the mean pixel intensity values. $c(\mathbf{x}, \mathbf{y})$ is a contrast comparison function with the standard deviation as an estimate for signal contrast. $s(\mathbf{x}, \mathbf{y})$ is a structure comparison function conducted on the normalized signals (the signal after removing its mean and divided by its own standard deviation) and $f(.)$ is a combination function. The comparison functions $l(\mathbf{x}, \mathbf{y}), c(\mathbf{x}, \mathbf{y}), s(\mathbf{x}, \mathbf{y})$ and the combination function $f(.)$ are chosen such that they satisfy three conditions: *symmetry*, *bounded* below 1 and with a *unique maximum* equal to 1 if and only if $\mathbf{x} = \mathbf{y}$. From a mathematical standpoint, SSIM index is defined as

$$\text{SSIM}(\mathbf{x}, \mathbf{y}) = \left(\frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}\right)\left(\frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}\right)\left(\frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}\right), \tag{3.18}$$

where μ_x and μ_y are mean intensity values, σ_x and σ_y are signal standard deviations, and σ_{xy} is the correlation coefficient between normalized signals $\mathbf{x}$ and $\mathbf{y}$. C_1, C_2, and C_3 are small positive constants included to avoid numerical instability. Choosing $C_3 = C_2/2$ results in a specific form of SSIM index

$$\text{SSIM}(\mathbf{x}, \mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}. \tag{3.19}$$

SSIM index ranges between 0 and 1, with a higher value indicating better performance. It is a common practice to apply SSIM index locally rather than globally, where the local statistics μ_x, σ_x, and σ_{xy} are calculated using a sliding Gaussian window of size M (typically 11×11) and the mean SSIM (MSSIM) is reported as the overall image similarity index

$$\text{MSSIM}(\mathbf{x}, \mathbf{y}) = \frac{1}{M}\sum_{i=1}^{M} \text{SSIM}(\mathbf{x}_i, \mathbf{y}_i), \tag{3.20}$$

where $\mathbf{x}_i$ and $\mathbf{y}_i$ are the ith square window in $\mathbf{x}$ and $\mathbf{y}$ respectively. For color images, SSIM is taken over each channel independently and the average is reported. To leverage image details at different resolutions and viewing conditions a multi-scale structural similarity approach is also proposed [131] that outperforms the single-scale SSIM models.

Overall, SSIM and its variants such as multi-scale SSIM are among the most commonly used similarity metrics to evaluate image compressions, image restorations, and pattern recognition and their advantages over MAE, MSE, and PSNR have been shown using various visual examples [124]. One drawback of SSIM is that it does not properly address geometrical distortions such as translation, scaling, rotation caused by movement of the image acquisition devices [132]. Similarity measures based on complex wavelet transform domain, or deep features extracted from a pre-trained neural network capture these nonstructural distortions and overcomes this problem.

3.3.4 Deep Features as Perceptual Metric

Image similarity metrics that are closer to human visual perception are extremely challenging [130]. While human beings can effortlessly perceive and assess visual similarities, a great deal of efforts has been made by computer vision and image processing community to develop a similarity measure that can correlate well with human perceptual judgments. These metrics are difficult to develop especially because human visual perception depends on higher-order structural information and to some extent is context-dependent [14]. For example, one might ask if a red circle is more similar to a blue circle or a red square? The answer to this question and many similar philosophical questions depend on the context in which it is asked and our understanding of a representation of the world! It is because of these fundamental ambiguities that pair-wise comparisons and methods relying on pixel similarity, such as MAE, MSE, PSNR are subject to failure. These methods, as we discussed earlier, assume pixel independence and fail to capture the underlying perceptual structures in an image.

Over the past few years, deep-learning techniques have demonstrated impressive progress in many areas of computer vision as they now underpin many complex tasks. For example, recent studies have found that features extracted from a VGG network [70] trained on ImageNet classification task [81] can be used as loss function for image synthesis tasks such as image style transfer [133, 111], super-resolution [79, 134, 4], and image-inpainting [135, 3, 1]. Zang *et al*. [14] have shown that compared to pixel-wise objective metrics, features extracted from activation maps of deep network correlates surprisingly well with human judgment (Figure 3.9). Their finding shows that these features are not restricted to pre-trained VGG network on ImageNet, but extends across different deep network architectures and tasks (supervised, unsupervised, and self-supervised learning).

Comparison between two images using deep features is performed by feeding the im-

Figure 3.9: Choosing the patch which is more "similar" to the reference in the middle. In each case, classical similarity metrics (L1/L2, PSNR, SSIM) disagree with human perceptual judgment. Features extracted from deep networks trained for different tasks and level of supervision (supervised, unsupervised, and self-supervised) agree with human visual perception. Figure ©Zang *et al*. [14] *The unreasonable effectiveness of deep features as a perceptual metric.*

ages to a pre-trained neural network. The similarity is then measured by computing a distance metric (such as ℓ_1) across different channels of some selected activation maps. These metrics are known as *Learned Perceptual Image Patch Similarity* (LPIPS) [14]. In the following section, we cover two well-known LPIPS metrics, together known as *Perceptual Losses* used in training and evaluation of deep networks for image synthesis tasks. We also review the recently proposed *Fréchet Inception Distance* as a non-referenced based similarity measure to evaluate unsupervised networks and generative models.

Perceptual Losses

The perceptual loss is a term primarily used in image transformation problems and is referred to as two image similarity metrics based on high-level features extracted from pre-trained networks. The term first appeared in image style transfer [90] where an optimization algorithm was introduced that could separate the content and style of natural images.

The algorithm allowed synthesizing new images that combine content from an arbitrary image with the style of an artwork creating a new artistic image. Separation of content and style was achieved by introducing *Content Reconstructions* and *Style Reconstructions* losses on the basis of the VGG network [70] meaning these loss functions are themselves deep convolutional neural networks. Here we review these reconstruction losses and show how they can be used as an objective function to train neural networks as well as a similarity metric that agrees with human visual perception. It is worth noting that since these metrics are computed on pre-trained deep neural networks, they normally require GPUs to run efficiently, running these metrics on CPUs is not recommended as they are is not computationally economic.

Content Reconstructions

Rather than comparing a target image $\mathbf{x}$ with the prediction $\hat{\mathbf{x}}$ on the pixel level, this loss measures the similarity based on their corresponding feature representations extracted from a deep network. Different layers of a convolutional neural network provide nonlinear filter banks that encode the input image into different feature maps. The complexity of feature maps increase depending on the position of the layer in the network. Early layers of a convolutional neural network capture low level features such as edges and curves. As the depth of the network increases more abstract concepts are represented through series of convolution filters. Let ϕ_i be the activation map of the i'th layer of a pre-trained network of the size $C_i \times H_i \times W_i$ with C_i being the number of convolutional channels in the layer and W_i and H_i as the width and height of the activation map, the perceptual loss is defined as the normalized Manhattan distance between feature representations

$$\ell^{\phi,i}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{C_i \times H_i \times W_i} \left\| \phi_i(\mathbf{x}) - \phi_i(\hat{\mathbf{x}}) \right\|_1 . \tag{3.21}$$

To capture different feature representations with different complexity and scale, it is common practice to combine feature maps from different layers. The loss is defined as

$$\mathcal{L}_{perc}(\mathbf{x}, \hat{\mathbf{x}}) = \sum_i \frac{1}{C_i \times H_i \times W_i} \left\| \phi_i(\mathbf{x}) - \phi_i(\hat{\mathbf{x}}) \right\|_1 , \tag{3.22}$$

where the summation is taken over layers at different positions in the network. For example, Gatys *et al*. [90] suggested using `conv1_2`, `conv2_2`, `conv3_2`, `conv4_2`, `conv5_2` of the 19-layer VGG network. As mentioned before, early activation layers in the network best measure the similarity of the low level features in the image while deep layers measure it in more abstract contexts.

Style Reconstructions

To measure style similarities such as color, texture, and visual patterns between two images, Gatys *et al*. [111] proposed a style representation loss that captures texture information. To formulate this similarity metric, the correlation between activation maps of a layer in a deep network are calculated. Let ϕ_i be the activation map of the i'th layer of the size $C_i \times H_i \times W_i$, the Gram matrix G_i^{ϕ} of size $C_i \times C_i$ is defined as

$$G_i^{\phi}(\mathbf{x}) = \frac{1}{C_i \times H_i \times W_i} \psi_i \psi_i^T , \tag{3.23}$$

where ψ_i is a matrix formed by reshaping $\phi_i(x)$ into $C_i \times H_i W_i$ shape. Elements of this Gram matrix are correlation between each two feature maps at layer i, thus they represent features that tend to activate together. For example, in a pre-trained network, if one set of feature maps activate in the presence of a car, and another set of features activate with red objects in an image, then one element of the Gram matrix is dedicated to red cars and its value is larger when a red car appears somewhere in the image. The style loss is defined as a ℓ_1 norm between two Gram matrices. Similar to the perceptual loss, a multi-scale

representation is obtained by combining information from different layers

$$\mathcal{L}_{style}(\mathbf{x}, \hat{\mathbf{x}}) = \sum_i \|G_i^\phi(\mathbf{x}) - G_i^\phi(\hat{\mathbf{x}})\|_1, \tag{3.24}$$

where the summation is taken over different layers with various choices of layers proposed the literature [111, 90, 79, 134]. It is worth noting that since the size of the Gram matrix in the equation 3.23 is always $C_i \times C_i$, the style reconstruction metric is defined even for images with different shapes.

Fréchet Inception Distance

The Fréchet Inception Distance (FID) is a distance measure introduced to evaluate the quality of images generated by GANs [15]. Generative models such as GANs, generate images without specific labels, instead, they learn to model the distribution of the data and generate samples close to that distribution. For example, in an image inpainting problem, the missing region in the image should be filled with the most "plausible" structure and texture such that the inpainted image agrees with human perceptual judgment. The plausible structure and texture is entirely subjective and the lack of a "true" reference makes most objective similarity metrics incapable of quantifying the performance of the models. A non-referenced based objective similarity metric that can quantify the "realism" of images is highly desired.

FID measures the Wasserstein-2 distance between the feature space representations of real and synthesized images using a pre-trained Inception-V3 model [136, 137]. FID starts with the assumption that data distribution (*e.g.* natural images) follow a multidimensional Gaussian distribution. In order to obtain vision-relevant features, the data is encoded with a function mapping $f(\mathbf{x})$ from input distribution to feature distribution using a pre-trained inception model. For practical reasons, only the first two moments of the Gaussian are considered: mean and covariance. The FID distance between two Gaussians with mean

and covariance $(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$ and $(\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y)$ is given by

$$\text{FID}((\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x), (\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y)) = \|\boldsymbol{\mu}_x - \boldsymbol{\mu}_y\|_2^2 + \text{Tr}\big(\boldsymbol{\Sigma}_x + \boldsymbol{\Sigma}_y - 2\big(\boldsymbol{\Sigma}_x\boldsymbol{\Sigma}_y\big)^{1/2}\big), \tag{3.25}$$

where Tr is the matrix trace operator and the mean and covariance are computed for large samples of data (commonly larger than $10,000$ samples) to capture the true statistics of the distributions. Lower FID means closer distance between model distribution (synthetic images) and data distribution (real images). Recent studies have shown that FID strongly agrees with human perception of realism [15, 102]. Figure 3.10 shows the value of FID for different levels of image degradations by adding Gaussian noise, Gaussian blur, and implanted black rectangle on a CelebA dataset [35]. In each case FID captures the distortions very well.

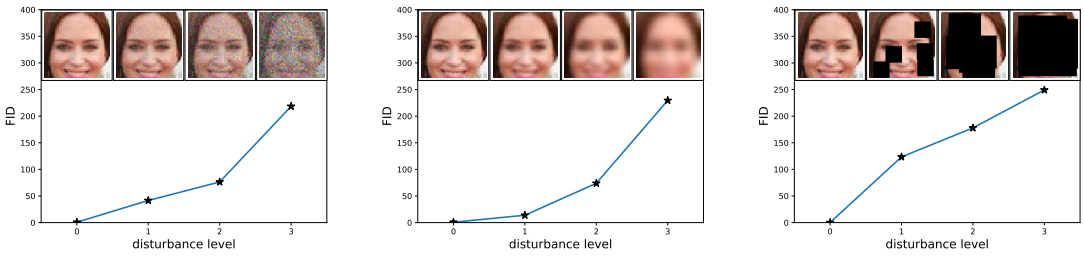


Figure 3.10: Left to right: FID evaluated for Gaussian noise, Gaussian blur, and implanted black rectangles on images from CelebA dataset. The degradations level starts from zero and increased to highest value. In each case, FID shows a monotonically increasing behavior and captures the distortions very well. Figure ©Heusel *et al*. [15] *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium.*

At the time of writing, FID remains the best quality measure available to evaluate the performance of unsupervised generative models. One caveat with FID is that it oftentimes requires large sampling to capture the statistics of the underlying distribution, this requires a large memory footprint and significant computations which make this metric less effective

for real-time performance evaluations.

3.3.5 Human Study & Psychophysical Similarity Measurements

For many image processing and computer vision applications, where images are being processed or generated for a human end user, the only "correct" method of quantifying the quality of the images is by subjective evaluations. These techniques require a human to assess image quality and oftentimes are more expensive, time-consuming (preparation and running), and may contain inaccurate predictions. Subjective tests may also be used to benchmark the performance of objective quality assessment metrics. Most subjective image quality tests are in fact psychophysical methods designed to measure human sensation given set of stimuli. For example, viewers may be given a set of images and their job is to rate them according to a rating scale from the lowest perceived quality to highest perceived quality. Mean opinion score (MOS) is the arithmetic mean for all viewers opinion scores of the perceived quality and is oftentimes reported as the overall quality of the system.

From a statistical standpoint, subjective tests are related to finite-sample distribution where random observations are used to approximate statistics of the underlying distribution of a population. For practical reasons calculating the statistics of an entire population is not feasible, instead, sampling is employed to characterize the true distribution. The idea is that, if a large number of samples are taken to compute a statistical property such as sample mean or variance, then the sampling distribution of that property is a probability distribution that can describe statistics of all possible samples taken from the same population. A very useful sampling distribution is "sampling distribution of the sample mean" where according to the *central limit theorem* follows a normal distribution with the same mean as the original distribution and its variance inversely proportional to the sample size: $\mathcal{N}(\mu, \sigma^2/n)$. Here n is the sample size and for the distribution to be normal $n > 30$ samples are required. The

mean opinion score of a psychophysical test mentioned before can be used to approximate the true perceived quality mean of a process with a larger sample size leading to a better approximation. It is a common practice to report the MOS with 95% confidence interval. Here we briefly review two similarity tests based on experimental psychology mostly used in subjective image quality assessments.

Two-Alternative Forced Choice (2AFC)

Two-Alternative Forced Choice (2AFC) is a subjective test for measuring perceptual responses from subjects through their choices and response time [138]. As the name suggests, participants are presented with only two choices and they must respond in the expected time. For examples 3.11 shows a simple 2AFC test for comparing two deblurring algorithms. Users were asked to choose an image (image1 or image2) that look more similar to a reference image (in the middle). Participants are exposed to multiple test subjects and are expected to an answer each test in a pre-determined time. The mean opinion score is calculated for all answers as a metric to favor the better deblurring algorithm.

One potential drawback of 2AFC method is that perception is oftentimes biased by secondary factors such as noise, spatial context, or artifacts in an image [139]. Participants may instead of the perceptual similarity focus on their choice of similarity to complete the task which can lead to incorrect estimates of perceptual bias. As a result, it is a common practice to conduct the 2AFC test in a *Yes-No task* (Y-N) where a single image is randomly sampled and participants are asked whether the sampled image is real or not.

Just Noticeable Differences (JND)

Just Noticeable Differences (JND) [140] is a less biased subjective test in experimental psychology introduced by physiologist Ernst Heinrich Weber (1795–1878) that quantifies

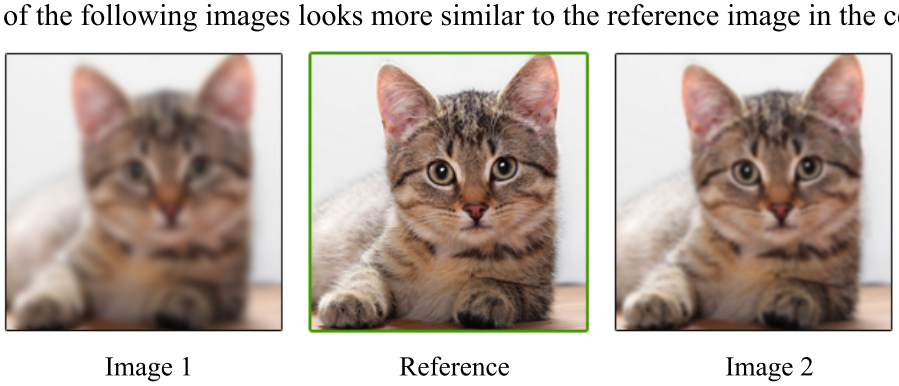


Figure 3.11: Two-alternative forced choice example for comparing two deblurring algorithms. Image1 (left) and Image2 (right) are the results of two different deblurring methods and the reference image (center) is the ground truth. Participants are asked to choose the image that looks more similar to the reference. In each test, the position of Image1 and Image2 are selected randomly to prevent bias.

the smallest perceivable difference between two stimuli in order to produce a noticeable variation in perception. The test is used a lot in computer vision subjective evaluations. For example, 3.12 shows a JND test for evaluating the performance of a deblurring algorithm where two images are shown to the participants and they are asked to select the image that looks more real. For each test case, the position of images is randomly chosen to prevent bias in decision-making. The average true answers are calculated for all participants as an evaluation metric of the deblurring algorithm.

JND test are oftentimes conducted in a limited time setting and the statistics are reported for different amounts of time the tests were presented to participants; this essentially measures how long it takes for an observer to spot a real image from the synthesized or degraded image. It is worth noting that the best performance achieved by JND test is 50% where two images are almost indistinguishable in terms of their perceptual quality.

Which of the following images looks more real?

Image 1 Image 2

Figure 3.12: Just noticeable differences example to evaluate a deblurring algorithm. For each test, the result of a deblurring algorithm (left) and a reference image (right) are randomly positioned and the participants are asked to choose the image that looks more real.

3.4 Summary

This chapter discussed image structures in the form of edges. Edge information carries the most important semantic associations with the human visual perception and are a fundamental step for many computer vision tasks such as image restoration, semantic segmentation, optical flow, object detection, motion tracking, and medical image processing. Two main categories of edge detection systems were presented: *Classical edge detections* take advantage of the correlation between neighboring pixels. These methods use a variety of techniques for computing image gradients. This is followed by non-maximal suppression in the Canny edge detection to accurately localize on the center of the edge. *Learning-based edge detections* follow data-driven, supervised approach to learn edge-data distribution. Instead of complex hand-designed feature extractors, these methods rely on neural networks for hierarchical feature learning. These methods are valuable in modern image processing that attempts to mimic human ability to resolve ambiguity in natural image edge detection and improved processing performance. The choice of one edge detection technique

over another is dictated mostly by the application and the context of the problem being considered.

This chapter also surveyed popular evaluation metrics for image quality assessment and a number of objective and subjective quality measures have been presented. Objective evaluations take advantage of known characteristics of the human visual system (HVS) which make them effective in measuring the level of information in a signal. These methods are widely used for performance evaluations despite their limitations in capturing the structure and perceptual similarity between images. A new class of objective metrics for no-reference and reduced-reference image quality assessment based on deep features were reviewed. These metrics are especially helpful to capture nonstructural distortions and are closer to human visual perception. Subjective metrics, on the other hand, are more useful for applications where the quality is ultimately subject to human understanding and perception. For example in generative models, there are more than one "correct" answer and evaluation often requires human judgment. Some of the subjective testing methods mentioned in the literature were also discussed in this chapter.

Although these methods each measure the magnitude of degradation and/or the quality of an image, there's no single metric that can measure all impairments. A combination of different numerical and subjective measures may prove to be more useful quality assessment method.

4. Image Inpainting

This chapter specifically addresses the problem of image inpainting. We propose a two-stage deep-convolutional adversarial pipeline for image inpainting that disentangles edge generation and image completion. The edge generator takes an image with a missing region as input and generates its full structure. The image completion stage employs the structures to guide the inpainting. This model is trained using a joint optimization of image contents (texture and color) and structures (edges). Quantitative and qualitative comparisons and user study show our model outperforms current state-of-the-art techniques.

4.1 Introduction

Image inpainting, or image completion, involves filling in missing regions of an image. We propose a model of "lines first, color next" that combines two different approaches to inpainting problem *Structural Inpainting* [28, 29, 30] and *Textural Inpainting* [31, 32] and we simultaneously try to perform texture and structure filling in regions of missing image information. We divide image inpainting into a two-stage process (Figure 4.1): edge generation and image completion. Edge generation is solely focused on hallucinating edges in the missing regions. The image completion network uses the hallucinated edges and

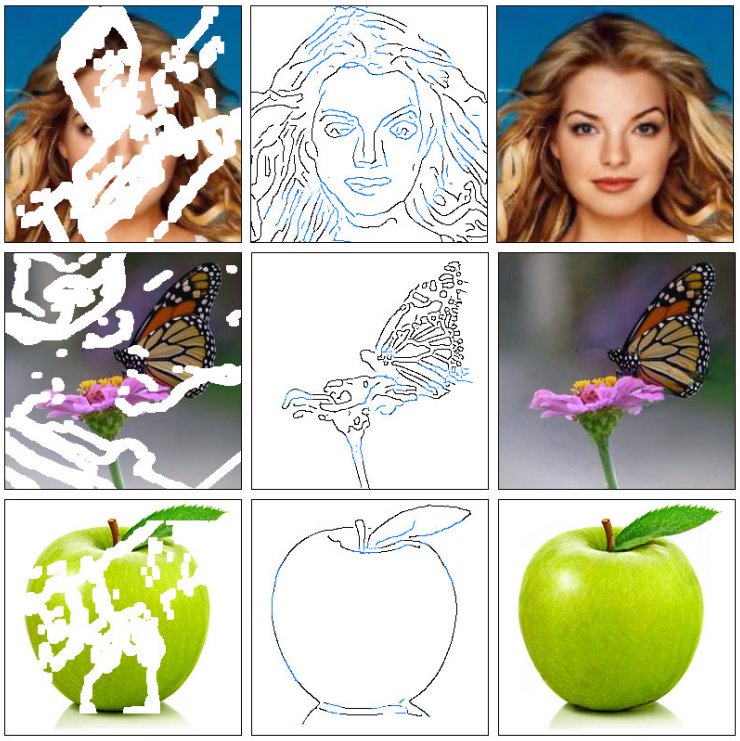


Figure 4.1: (Left) Input images with missing regions. The missing regions are depicted in white. (Center) Computed edge masks. Edges drawn in black are computed (for the available regions) using Canny edge detector; whereas edges shown in blue are hallucinated (for the missing regions) by the edge generator network. (Right) Image inpainting results of the proposed approach.

estimates RGB pixel intensities of the missing regions. Both stages follow an adversarial framework [9] to ensure that the hallucinated edges and the RGB pixel intensities are visually consistent. Both networks incorporate losses based on deep features to enforce perceptually realistic results.

Like most computer vision problems, image inpainting predates the wide-spread use of deep learning techniques. Broadly speaking, traditional approaches for image inpainting can be divided into three groups: **diffusion-based**, **patch-based**, and **learning-based** methods. Diffusion-based methods propagate background data into the missing region by following a diffusive process typically modeled using differential operators [141, 142, 143, 144]. Patch-based methods, on the other hand, fill in missing regions with patches from a collection of source images that maximize patch similarity [145, 146]. These methods, however, do a poor job of reconstructing complex details of the missing region, are normally slow and do not consider the semantics of the scene. Learning-based methods fill the missing pixels using learned data distribution and are superior to classical methods in every aspect. Table 4.1 shows comparison between different methods.

Algorithm	Fast	Semantics	Non-Local	High-Quality
Diffusion-based	-	-	-	-
Patch-based	-	-	✓	✓
Learning-based	✓	✓	✓	✓[1]

Table 4.1: Comparison of different approaches for image inpainting. Diffusion-based methods propagate background data into the missing region by following a diffusive process. Patch-based methods fill in missing regions with patches from a collection of source images that maximize patch similarity and provide better inpainting quality. Learning-based methods fill the missing pixels using learned data distribution and are superior to classical methods in every aspect.

[1] Inpainting results of the learning-based methods are oftentimes over-smoothed and/or blurry.

4.2 Related Work

4.2.1 Diffusion-Based Inpainting

Diffusion-based methods propagate neighboring information into the missing regions [141, 144]. [142] adapted the Mumford-Shah segmentation model for image inpainting by introducing Euler's Elastica. Structure-guided diffusion-based methods have also been proposed such as [28, 29, 30]. Reconstruction of the missing part is restricted to locally available information for these diffusion-based methods, and these methods fail to recover meaningful structures in the missing regions especially for cases with large missing regions. Moreover since in these methods optimization is performed at runtime, they are normally slow and not suitable in practical settings (see Table 4.1).

4.2.2 Patch-Based Inpainting

Patch-based methods fill in missing regions (*i.e.*, targets) by copying information from similar regions (*i.e.*, sources) of the same image (or a collection of images). Source regions are often blended into the target regions to minimize discontinuities [145, 146]. These methods are computationally expensive since similarity scores must be computed for every target-source pair. PatchMatch [147] addressed this issue by using a fast nearest neighbor field algorithm. These methods, however, assume that the texture of the inpainted region can be found elsewhere in the image. This assumption does not always hold. Consequently, these methods excel at recovering highly patterned regions such as background completion but struggle at reconstructing patterns that are locally unique. Finally, these methods exhibit subtle color inconsistencies between the inpainted area and the surrounding regions. To fix that, inpainting is normally followed by a post-processing blending algorithm such as Poisson image blending [148].

4.2.3 Learning-Based Inpainting

One of the first *deep learning* methods designed for image inpainting is context encoder [23], which uses an encoder-decoder architecture [51]. The encoder maps an image with missing regions to a low-dimensional feature space, which the decoder uses to construct the output image. However, the recovered regions of the output image often contain visual artifacts and exhibit blurriness due to the information bottleneck in the channel-wise fully connected layer. This was addressed by Iizuka *et al*. [2] by reducing the number of downsampling layers, and replacing the channel-wise fully connected layer with a series of dilated convolution layers [6]. The reduction of downsampling layers is compensated by using varying dilation factors (see Chapter 2.3.2). However, training time was increased significantly[2] due to extremely sparse filters created using large dilation factors. Yang *et al*. [135] uses a pre-trained VGG network [70] to improve the output of the context-encoder, by minimizing the feature difference of image background. This approach requires solving a multi-scale optimization problem iteratively, which noticeably increases computational cost during inference time. Liu *et al*. [3] introduced "partial convolution" for image inpainting, where convolution weights are normalized by the mask area of the window that the convolution filter currently resides over. This effectively prevents the convolution filters from capturing too many zeros when they traverse over the incomplete region.

Recently, several methods were introduced by providing additional information prior to inpainting. Yeh *et al*. [25] trains a GAN for image inpainting with uncorrupted data. During inference, back-propagation is employed for $1,500$ iterations to find the representation of the corrupted image on a uniform noise distribution. However, the model is slow during inference since back-propagation must be performed for every image it attempts to recover. Dolhansky and Ferrer [24] demonstrate the importance of exemplar information for in-

[2]Model by [2] required two months of training over four GPUs.

painting. Their method is able to achieve both sharp and realistic inpainting results. Their method, however, is geared towards filling in missing eye regions in frontal human face images. It is highly specialized and does not generalize well. Contextual Attention [1] takes a two-step approach to the problem of image inpainting. First, it produces a coarse estimate of the missing region. Next, a refinement network sharpens the result using an attention mechanism by searching for a collection of background patches with the highest similarity to the coarse estimate. [149] takes a similar approach and introduces a "patch-swap" layer which replaces each patch inside the missing region with the most similar patch on the boundary. These schemes suffer from two limitations: 1) the refinement network assumes that the coarse estimate is reasonably accurate, and 2) these methods cannot handle missing regions with arbitrary shapes. SPG-Net [150] also follows a two-stage model which uses semantic segmentation labels to guide the inpainting process. Free-form inpainting method proposed in [26] is perhaps closest in spirit to our scheme. It uses hand-drawn sketches to guide the inpainting process. Our method does away with hand-drawn sketches and instead learns to hallucinate edges in the missing regions.

4.2.4 Image-to-Edges vs. Edges-to-Image

The image restoration technique proposed in this work subsumes two disparate computer vision problems: Image-to-Edges and Edges-to-Image. There is a large body of literature that addresses "Image-to-Edges" problems [122, 151, 152, 123] (see Section 3.2 for a detailed study on edge detection). Canny edge detector, an early scheme for constructing edge maps, for example, is roughly 30 years old [11]. Dollár and Zitnikc [153] use *structured learning* [154] on random decision forests to predict local edge masks. Holistically-nested Edge Detection (HED) [119] is a fully convolutional network that learns edge information based on its importance as a feature of the overall image. In our work, we train on edge

maps computed using Canny detector. We explain this in detail in Sections 4.3.4 and 4.4.4.

Traditional "Edges-to-Image" methods typically follow a bag-of-words approach, where image content is constructed through a pre-defined set of keywords. These methods, however, are unable to accurately construct fine-grained details, especially near object boundaries. Scribbler [155] is a learning-based model where images are generated using line sketches as the input. The results of their work possess an art-like quality, where the color distribution of the generated result is guided by the use of color in the input sketch. Isola *et al*. [89] proposed a conditional GAN framework [104], called pix2pix, for image-to-image translation problems. This scheme can use available edge information as *a priori*. CycleGAN [107] extends this framework and finds a reverse mapping back to the original data distribution. This approach yields superior results since the aim is to learn the inverse of the forward mapping.

4.3 Model

We propose an image inpainting network that consists of two stages: 1) edge generator, and 2) image completion network (Figure 4.2). Both stages follow an adversarial model [9], *i.e*. each stage consists of a generator/discriminator pair. Let G_1 and D_1 be the generator and discriminator for the edge generator, and G_2 and D_2 be the generator and discriminator for the image completion network, respectively. To simplify notation, we will use these symbols also to represent the function mappings of their respective networks.

Our generators follow an architecture similar to the method proposed by Johnson *et al*. [79], which has achieved impressive results for style transfer, super-resolution [4, 134], and image-to-image translation [107]. Specifically, the generators consist of encoders that down-sample twice, followed by eight residual blocks [7] and decoders that up-sample

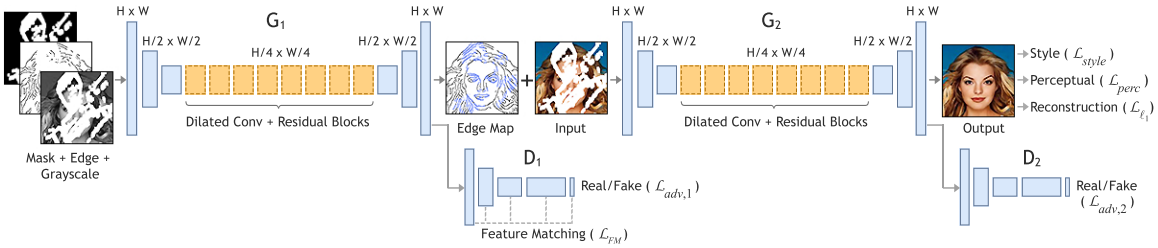


Figure 4.2: Summary of our proposed method. Incomplete grayscale image and edge map, and mask are the inputs of G_1 to predict the full edge map. Predicted edge map and incomplete color image are passed to G_2 to perform the inpainting task.

images back to the original size. Dilated convolutions with a dilation factor of two are used instead of regular convolutions in the residual layers, resulting in a receptive field of 205 at the final residual block. For discriminators, we use a 70×70 PatchGAN [89, 107] architecture, which determines whether or not overlapping image patches of size 70×70 are real. We use instance normalization [83] across all layers of the network. See Sections 2.3.2 and 2.3.3 for a detailed explanation on these architectural designs.

4.3.1 Edge Generation

Let $\mathbf{I}_{gt}$ be ground truth images. Their edge map and grayscale counterpart will be denoted by $\mathbf{C}_{gt}$ and $\mathbf{I}_{gray}$, respectively. In the edge generator, we use the masked grayscale image $\tilde{\mathbf{I}}_{gray} = \mathbf{I}_{gray} \odot (\mathbf{1} - \mathbf{M})$ as the input, its edge map $\tilde{\mathbf{C}}_{gt} = \mathbf{C}_{gt} \odot (\mathbf{1} - \mathbf{M})$, and image mask $\mathbf{M}$ as a pre-condition (1 for the missing region, 0 for background). Here, $\odot$ denotes the Hadamard product. The generator predicts the edge map for the masked region

$$\mathbf{C}_{pred} = G_1 \left(\tilde{\mathbf{I}}_{gray}, \tilde{\mathbf{C}}_{gt}, \mathbf{M} \right). \tag{4.1}$$

We use $\mathbf{C}_{gt}$ and $\mathbf{C}_{pred}$ conditioned on $\mathbf{I}_{gray}$ as inputs of the discriminator that predicts whether or not an edge map is real. The network is trained with an objective comprised of

the hinge variant of GAN loss [100] and feature-matching loss [108]

$$\mathcal{J}_{G_1} = \lambda_{G_1}\mathcal{L}_{G_1} + \lambda_{FM}\mathcal{L}_{FM}, \tag{4.2}$$

where λ_{G_1} and λ_{FM} are regularization parameters. The hinge losses over the generator and discriminator are defined as

$$\mathcal{L}_{G_1} = -\mathbb{E}_{\mathbf{I}_{gray}}\left[D_1(\mathbf{C}_{pred}, \mathbf{I}_{gray})\right], \tag{4.3}$$

$$\mathcal{L}_{D_1} = \mathbb{E}_{(\mathbf{C}_{gt}, \mathbf{I}_{gray})}\left[\max(0, 1 - D_1(\mathbf{C}_{gt}, \mathbf{I}_{gray}))\right] + \mathbb{E}_{\mathbf{I}_{gray}}\left[\max(0, 1 + D_1(\mathbf{C}_{pred}, \mathbf{I}_{gray}))\right]. \tag{4.4}$$

See Section 2.3.3 for a detail explanation on adversarial losses. The feature-matching loss $\mathcal{L}_{FM}$ compares the activation maps in the intermediate layers of the discriminator. This stabilizes the training process by forcing the generator to produce results with representations that are similar to real images. This is similar to perceptual loss [79, 90, 111], where activation maps are compared with those from the pre-trained VGG network. However, since the VGG network is not trained to produce edge information, it fails to capture the result that we seek in the initial stage. The feature matching loss $\mathcal{L}_{FM}$ is defined as

$$\mathcal{L}_{FM} = \mathbb{E}_{(\mathbf{C}_{gt}, \mathbf{C}_{pred}, \mathbf{I}_{gray})}\left[\sum_i \frac{1}{N_i}\left\|D_1^{(i)}(\mathbf{C}_{gt}, \mathbf{I}_{gray}) - D_1^{(i)}(\mathbf{C}_{pred}, \mathbf{I}_{gray})\right\|_1\right], \tag{4.5}$$

where, N_i is the number of elements in the i'th activation layer, and $D_1^{(i)}$ is the activation in the i'th layer of the discriminator. Spectral normalization (SN) [100] further stabilizes training by scaling down weight matrices by their respective largest singular values, effectively restricting the Lipschitz constant of the network to one. Although this was originally proposed to be used only on the discriminator, recent works [102, 114] suggest that generator can also benefit from SN by suppressing sudden changes of parameter and gradient values. We apply SN to both generator and discriminator (see Section 2.3.3). Spectral normalization was chosen over Wasserstein GAN (WGAN) [109], as we found that WGAN was several times slower in our tests. For our model, we choose $\lambda_{G_1} = 1$ and $\lambda_{FM} = 10$.

4.3.2 Image Completion

The image completion network uses the incomplete color image $\tilde{\mathbf{I}}_{gt} = \mathbf{I}_{gt} \odot (\mathbf{1} - \mathbf{M})$ as input, conditioned using a composite edge map $\mathbf{C}_{comp}$. The composite edge map is constructed by combining the background region of ground truth edges with generated edges in the corrupted region from the previous stage, *i.e.* $\mathbf{C}_{comp} = \mathbf{C}_{gt} \odot (\mathbf{1} - \mathbf{M}) + \mathbf{C}_{pred} \odot \mathbf{M}$. The network returns a color image $\mathbf{I}_{pred}$, with missing regions filled in, that has the same resolution as the input image:

$$\mathbf{I}_{pred} = G_2\left(\tilde{\mathbf{I}}_{gt}, \mathbf{C}_{comp}\right). \tag{4.6}$$

This is trained over a joint loss that consists of an ℓ_1 loss, hinge loss, perceptual loss, and style loss. To ensure proper scaling, the ℓ_1 loss is normalized by the mask size. The hinge loss is similar to Equations 4.3 and 4.4:

$$\mathcal{L}_{G_2} = -\mathbb{E}_{\mathbf{C}_{comp}}\left[D_2(\mathbf{I}_{pred}, \mathbf{C}_{comp})\right], \tag{4.7}$$

$$\mathcal{L}_{D_2} = \mathbb{E}_{(\mathbf{I}_{gt}, \mathbf{C}_{comp})}\left[\max(0, 1 - D_2(\mathbf{I}_{gt}, \mathbf{C}_{comp}))\right] + \mathbb{E}_{\mathbf{C}_{comp}}\left[\max(0, 1 + D_2(\mathbf{I}_{pred}, \mathbf{C}_{comp}))\right]. \tag{4.8}$$

We include the two losses proposed in [90, 79] commonly known as perceptual loss $\mathcal{L}_{perc}$ and style loss $\mathcal{L}_{style}$ (see Section 3.3.4). As the name suggests, $\mathcal{L}_{perc}$ penalizes results that are not perceptually similar to labels by defining a distance measure between activation maps of a pre-trained network. Perceptual loss is defined as

$$\mathcal{L}_{perc} = \mathbb{E}_{(\mathbf{I}_{gt}, \mathbf{I}_{pred})}\left[\sum_i \frac{1}{C_i \times H_i \times W_i} \left\|\phi_i(\mathbf{I}_{gt}) - \phi_i(\mathbf{I}_{pred})\right\|_1\right], \tag{4.9}$$

where ϕ_i is the activation map of the i'th layer of a pre-trained network, C_i is the number of convolutional channels in that layer and W_i and H_i as the width and height of the activation map. For our work, ϕ_i corresponds to activation maps from layers `relu1_1`,

`relu2_1`, `relu3_1`, `relu4_1` and `relu5_1` of the VGG-19 network pre-trained on the ImageNet dataset [81]. These activation maps are also used to compute style loss which measures the differences between correlations of the activation maps. Given feature maps of sizes $C_j \times H_j \times W_j$, style loss is computed by

$$\mathcal{L}_{style} = \mathbb{E}_{(\tilde{\mathbf{I}}_{gt}, \tilde{\mathbf{I}}_{pred})} \left[\sum_j \| G_j^{\phi}(\tilde{\mathbf{I}}_{gt}) - G_j^{\phi}(\tilde{\mathbf{I}}_{pred}) \|_1 \right], \tag{4.10}$$

where G_j^{ϕ} is a $C_j \times C_j$ Gram matrix constructed from activation maps ϕ_j [90] (see Equation 3.23). The style loss was shown by Sajjadi *et al*. [4] to be an effective tool to combat "checkerboard" artifacts caused by transpose convolution layers [156]. We select layers `relu2_2`, `relu3_4`, `relu4_4`, and `relu5_2` from VGG-19 network. Our overall loss is

$$\mathcal{J}_{G_2} = \lambda_{\ell_1} \mathcal{L}_{\ell_1} + \lambda_{G_2} \mathcal{L}_{G_2} + \lambda_p \mathcal{L}_{perc} + \lambda_s \mathcal{L}_{style}. \tag{4.11}$$

For our experiments, we choose $\lambda_{\ell_1} = 1$, $\lambda_{G_2} = \lambda_p = 0.1$, and $\lambda_s = 250$. We noticed that the training time increases significantly if spectral normalization is included. We believe this is due to the network becoming too restrictive with the increased number of terms in the loss function. Therefore we choose to exclude it from the image completion network.

4.3.3 Network Architecture

Generators We follow a similar naming convention as those presented in [107]. Let `c7s1-k` denote a 7×7 Convolution-SpectralNorm-InstanceNorm-ReLU layer with k filters and stride 1 with reflection padding. Let `dk` denote a 4×4 Convolution-SpectralNorm-InstanceNorm-ReLU layer with k filters and stride 2 for down-sampling. Let `uk` be defined in the same manner as `dk` with transpose convolution for up-sampling. Let `Rk` denote a residual block of channel size k across both layers. We use dilated convolution in the first layer of `Rk` with dilation factor of 2, followed by spectral normalization and instance normalization. The architecture of our generators is adopted from the model proposed by Johnson *et al.* [79]:

`c7s1-64, d128, d256, R256, R256, R256, R256, R256, R256, R256, R256, u128, u64, c7s1-⋆`.

The final layer `c7s1-⋆` varies depending on the generator. In the edge generator G_1, `c7s1-⋆` has channel size of 1 with sigmoid activation for edge prediction. In the image completion network G_2, `c7s1-⋆` has channel size of 3 with $\tanh$ (scaled) activation for the prediction of RGB pixel intensities. In addition, we remove spectral normalization from all layers of G_2.

Discriminators The discriminators D_1 and D_2 follow the same architecture based on the 70×70 PatchGAN [89, 107]. Let `Ck-s` denote a 4×4 Convolution-SpectralNorm-LeakyReLU layer with k filters of stride s. The discriminators have the architecture `C64-2, C128-2, C256-2, C512-1, C1-1`. The network maps input image $\mathbf{I}$ to a matrix of outputs $\mathbf{X}$, where each $\mathbf{X}_{(i,j)}$ determines whether the 70×70 patch $\mathbf{I}_{(i,j)}$ in the image is real or fake. See Section 2.3.2 for a detail explanation on Patch-GAN architecture. LeakyReLU [66] with slope 0.2 is employed in all layers of discriminator except for the last layer.

4.3.4 Training

Edge Information and Image Masks

To train G_1 for an inpainting task, we generate training labels (*i.e*. edge maps) using Canny edge detector. The sensitivity of Canny edge detector is controlled by the standard deviation of the Gaussian smoothing filter σ. For our inpainting model, we empirically found that $\sigma \approx 2$ yields the best results (Figure 4.8). In Section 4.4.4, we investigate the effect of the quality of edge maps on the overall image completion.

For the inpainting task, we use two types of image masks: regular and irregular. Regular masks are square masks of fixed size (25% of total image pixels) centered at a random location within the image. We obtain irregular masks from the work of Liu *et al*. [3]. Irregular masks are classified based on their sizes relative to the entire image in increments of 10% (*e.g*. 0-10%, 10-20%, *etc*.). All bins are divided into two batches of 1,750 and 250 masks for training and testing purposes respectively. Once separated, masks are augmented by introducing four rotations ($0°, 90°, 180°, 270°$) and a horizontal reflection for each mask.

Training Setup and Strategy

Our proposed models are implemented in PyTorch. The networks were trained with 256×256 images with batch size of eight to obtain results for quantitative comparisons with existing methods. The models were optimized using Adam optimizer [63] with $\beta_1 = 0$ and $\beta_2 = 0.9$. Generators G_1, G_2 are trained separately using Canny edges with learning rate 10^{-4} until the losses plateau. We lower the learning rate to 10^{-5} and continue to train G_1 and G_2 until convergence. Finally, we freeze training on G_1 while continue to train G_2. For visual comparisons presented in this thesis, our models were trained with 512×512 images using pre-trained weights from the 256×256 model with the same hyper-parameters.

4.4 Experiments

Our proposed models are evaluated on publicly available datasets. Results are compared against the current state-of-the-art methods both qualitatively and quantitatively.

4.4.1 Datasets

We evaluate our proposed models on the following publicly available standard datasets.

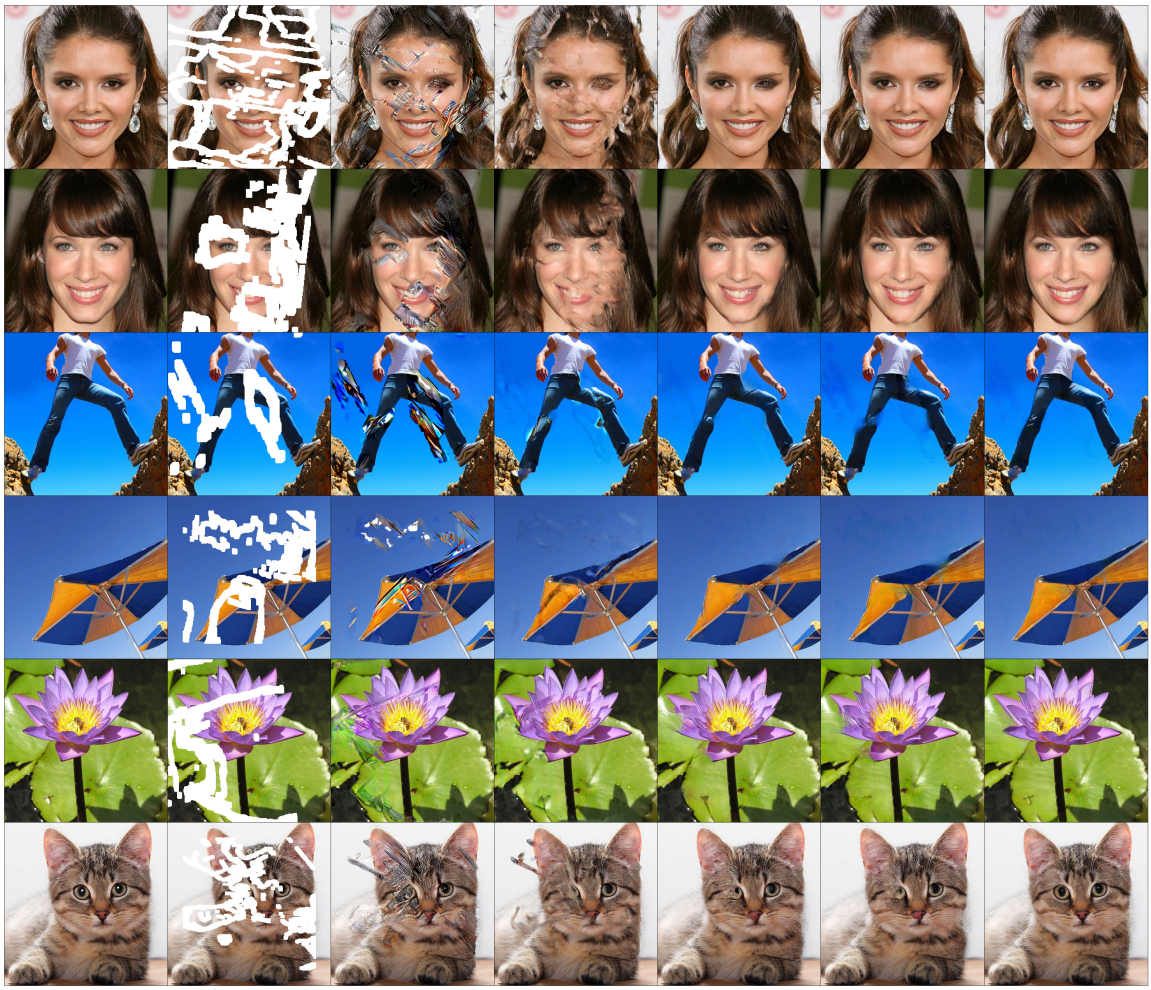
- CelebA [35]. A large-scale face attributes dataset with 200K celebrity images.
  `http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html`
- Celeb-HQ [42]. High-quality version of the CelebA dataset with 30K images.
  `https://github.com/tkarras/progressive_growing_of_gans`
- Places2 [37]. More than 10 million images comprising 400+ unique scene categories.
  `http://places2.csail.mit.edu/`
- Paris StreetView [38] Geotagged imagery of Paris from Google Street View.
  `https://github.com/pathak22/context-encoder`

  Datasets can be downloaded from their official websites.

For CelebA, we crop the center of the image and resize it to the appropriate resolution. For Paris StreetView, since the images in the dataset are elongated (936×537), we separate each image into three: 1) Left 537×537, 2) middle 537×537, 3) right 537×537, of the image for a total of $44,700$ images. All images are rescaled to 256×256 for quantitative results, and 512×512 for qualitative results.

4.4.2 Qualitative Evaluation

Figure 4.3 compares inpainting results generated by our method with those generated by other state-of-the-art techniques for 512×512 images.



Ground Truth | Input | Iizuka *et al*. [2] | Yu *et al*. [1] | Liu *et al*. [3] | Baseline | Ours

Figure 4.3: Comparison of qualitative results of 512×512 image inpainting with existing models. From left to right: Ground Truth, Masked Image, Iizuka *et al*. [2] (Globally and Locally Image Completion), Yu *et al*. [1] (Contextual Attention), Liu *et al*. (Partial Convolution) [3], Baseline (no edge data, G_2 only), Ours (Full Model).

Figure 4.4: Qualitative results of 512×512 image inpainting. (Left to Right) Original image, input image, generated edges, inpainted results without any post-processing.

The images generated by our proposed model are closer to the ground truth than images from other methods. We conjecture that when edge information is present, the network only needs to learn the color distribution, without having to worry about preserving image structure. This is especially visible with the Baseline in Figure 4.3 with no-edge data, where the output lacks sharp edges and/or structure around the missing regions.

Figure 4.4 shows a sample of images generated by our model with their predicted edge-map. For visualization purposes, we reverse the colors of $\mathbf{C}_{comp}$ and delineate predicted edge-maps in blue. Our model is able to generate photo-realistic results with a large fraction of image structures remaining intact. Furthermore, by including style loss, the inpainted images lack any "checkerboard" artifacts [156] in the generated results. As importantly, the inpainted images exhibit minimal blurriness.

For more qualitative inpainting results see Appendix A.

4.4.3 Quantitative Evaluation

Inpainting Numerical Metrics

Since existing models were evaluated using 256×256, we evaluated our model trained on images of the same resolution to ensure fair comparisons. The performance of our model was measured using the following metrics: 1) relative ℓ_1; 2) structural similarity index (SSIM) [124], with a window size of 11; 3) peak signal-to-noise ratio (PSNR); and 4) Fréchet Inception Distance (FID) [15]. Since relative ℓ_1, SSIM, and PSNR assume pixel-wise independence, these metrics may assign favorable scores to perceptually inaccurate results. Recent works [14, 102, 24] have shown that FID serves as the preferred metric for human perception (see Section 3.3.4). Note that since FID is a dissimilarity measure between high-level features, it may not reflect low-level color consistencies that attribute to visual quality. While FID may not be the ideal metric to measure inpainting quality, we believe the combination of the listed metrics provided a better picture of inpainting performance. Tables 4.2, 4.3, and 4.4 show the performance of our model compared to existing methods over the datasets Places2, CelebA, and Paris StreetView respectively. Figures 4.5, 4.6, and 4.7 display these results graphically. Our method produces noticeably better results. Note that these statistics are based on the synthesized image which mostly comprises of the ground truth image. Therefore our reported FID values are lower than other generative models reported in [157]. Statistics for competing techniques are obtained using their respective pre-trained weights, where available[3] [4]. The full model of Partial Convolution (PConv) is not available at the time of writing. We implemented PConv based on the guidelines in [3] using the PConv layer that is publicly available [5]. Our statistics are

[3] https://github.com/JiahuiYu/generative_inpainting
[4] https://github.com/satoshiiizuka/siggraph2017_inpainting
[5] https://github.com/NVIDIA/partialconv

calculated over $10,000$ random images in the test set.

	Mask	CA	GLCIC	PConv	Ours
ℓ_1 (%)†	0-10%	0.97	1.02	0.60	**0.51**
	10-20%	2.41	2.66	1.55	**1.50**
	20-30%	4.23	4.70	2.71	**2.59**
	30-40%	6.15	6.78	3.94	**3.77**
	40-50%	8.03	8.85	5.35	**5.14**
	50-60%	10.32	10.64	7.63	**7.41**
	Fixed	4.37	4.12	3.95	**3.86**
SSIM*	0-10%	0.959	0.945	**0.968**	**0.968**
	10-20%	0.893	0.862	0.916	**0.920**
	20-30%	0.815	0.771	0.854	**0.861**
	30-40%	0.739	0.686	0.789	**0.799**
	40-50%	0.662	0.603	0.720	**0.731**
	50-60%	0.582	0.539	0.628	**0.641**
	Fixed	0.818	0.814	0.818	**0.823**
PSNR*	0-10%	30.52	28.98	**33.40**	33.39
	10-20%	24.36	23.49	27.54	**27.95**
	20-30%	21.19	20.45	24.47	**24.92**
	30-40%	19.13	18.50	22.42	**22.84**
	40-50%	17.75	17.17	20.77	**21.16**
	50-60%	16.38	16.42	18.71	**18.99**
	Fixed	20.65	21.34	21.54	**21.75**
FID†	0-10%	1.76	3.68	**0.76**	081
	10-20%	6.16	11.84	**2.26**	2.32
	20-30%	14.17	25.11	**4.88**	4.91
	30-40%	24.16	39.88	**8.84**	8.91
	40-50%	35.78	54.30	15.18	**14.98**
	50-60%	42.26	53.30	28.11	**25.75**
	Fixed	8.31	8.42	10.53	**8.16**

Table 4.2: Comparison of quantitative results (256×256) over **Places2** with CA [1], GLCIC [2], PConv [3], Ours (end-to-end). The best result of each row is boldfaced. †Lower is better. *Higher is better.

	Mask	CA	GLCIC	PConv	Ours
ℓ_1 (%)†	0-10%	1.33	0.91	**0.29**	**0.29**
	10-20%	2.48	2.53	0.78	**0.76**
	20-30%	3.98	4.67	1.42	**1.38**
	30-40%	5.64	6.95	2.19	**2.13**
	40-50%	7.35	9.18	3.08	**3.03**
	50-60%	9.21	11.21	4.96	**4.89**
	Fixed	2.80	3.83	**2.35**	**2.39**
SSIM⋆	0-10%	0.947	0.947	**0.985**	**0.985**
	10-20%	0.888	0.865	0.956	**0.961**
	20-30%	0.819	0.773	0.924	**0.928**
	30-40%	0.750	0.689	0.884	**0.890**
	40-50%	0.678	0.609	0.840	**0.846**
	50-60%	0.614	0.560	0.768	**0.771**
	Fixed	0.882	0.847	**0.891**	**0.891**
PSNR⋆	0-10%	31.16	30.24	**39.65**	39.60
	10-20%	25.32	24.09	33.19	**33.51**
	20-30%	22.09	20.71	29.68	**30.02**
	30-40%	19.94	18.50	27.15	**27.39**
	40-50%	18.41	17.09	25.15	**25.28**
	50-60%	17.18	16.24	22.00	**22.11**
	Fixed	25.34	22.13	**25.63**	25.49
FID†	0-10%	3.24	16.84	**0.20**	**0.20**
	10-20%	13.12	58.74	**0.53**	**0.53**
	20-30%	29.47	102.97	**1.08**	**1.08**
	30-40%	47.55	136.47	1.81	**1.80**
	40-50%	68.40	163.95	**2.81**	2.82
	50-60%	76.70	167.07	5.46	**5.30**
	Fixed	1.90	25.21	1.92	**1.90**

Table 4.3: Comparison of quantitative results (256×256) over **CelebA** with CA [1], GLCIC [2], PConv [3], Ours (end-to-end). The best result of each row is boldfaced. †Lower is better. ⋆Higher is better.

	Mask	CA	GLCIC	PConv	Ours
ℓ_1 (%)†	0-10%	0.75	0.86	**0.43**	**0.43**
	10-20%	2.10	2.20	1.14	**1.09**
	20-30%	3.80	3.86	2.04	**1.91**
	30-40%	5.53	5.58	3.02	**2.82**
	40-50%	7.23	7.34	4.17	**3.94**
	50-60%	9.06	9.02	6.12	**5.87**
	Fixed	3.22	3.23	2.92	**2.77**
SSIM⋆	0-10%	0.964	0.949	**0.975**	**0.975**
	10-20%	0.905	0.878	0.933	**0.938**
	20-30%	0.835	0.800	0.881	**0.892**
	30-40%	0.766	0.724	0.826	**0.842**
	40-50%	0.695	0.648	0.765	**0.784**
	50-60%	0.625	0.588	0.678	**0.700**
	Fixed	0.847	0.840	0.847	**0.860**
PSNR⋆	0-10%	32.45	30.46	**36.39**	36.31
	10-20%	26.09	25.72	30.71	**31.23**
	20-30%	22.80	22.90	27.57	**28.26**
	30-40%	20.74	21.02	25.43	**26.05**
	40-50%	19.35	19.66	23.66	**24.20**
	50-60%	18.17	18.71	21.34	**21.73**
	Fixed	23.68	24.07	24.78	**25.23**
FID†	0-10%	2.26	6.50	**0.43**	0.44
	10-20%	9.10	18.77	1.32	**1.20**
	20-30%	20.62	35.66	2.97	**2.49**
	30-40%	34.31	53.53	5.65	**4.35**
	40-50%	49.80	70.36	10.00	**7.20**
	50-60%	55.78	69.95	21.10	**13.98**
	Fixed	7.26	7.18	6.44	**4.57**

Table 4.4: Comparison of quantitative results (256×256) over **Paris StreetView** with CA [1], GLCIC [2], PConv [3], Ours (end-to-end). The best result of each row is boldfaced. †Lower is better. ⋆Higher is better.

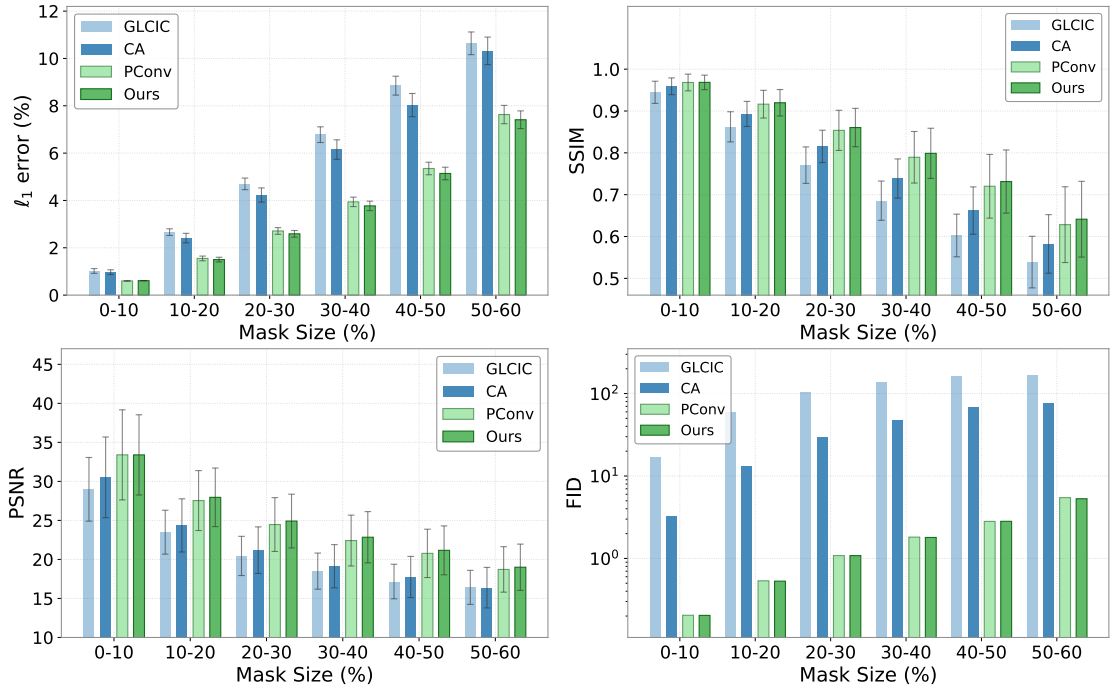


Figure 4.5: Effect of mask sizes on ℓ_1, SSIM, PSNR, and FID for Places2 dataset.

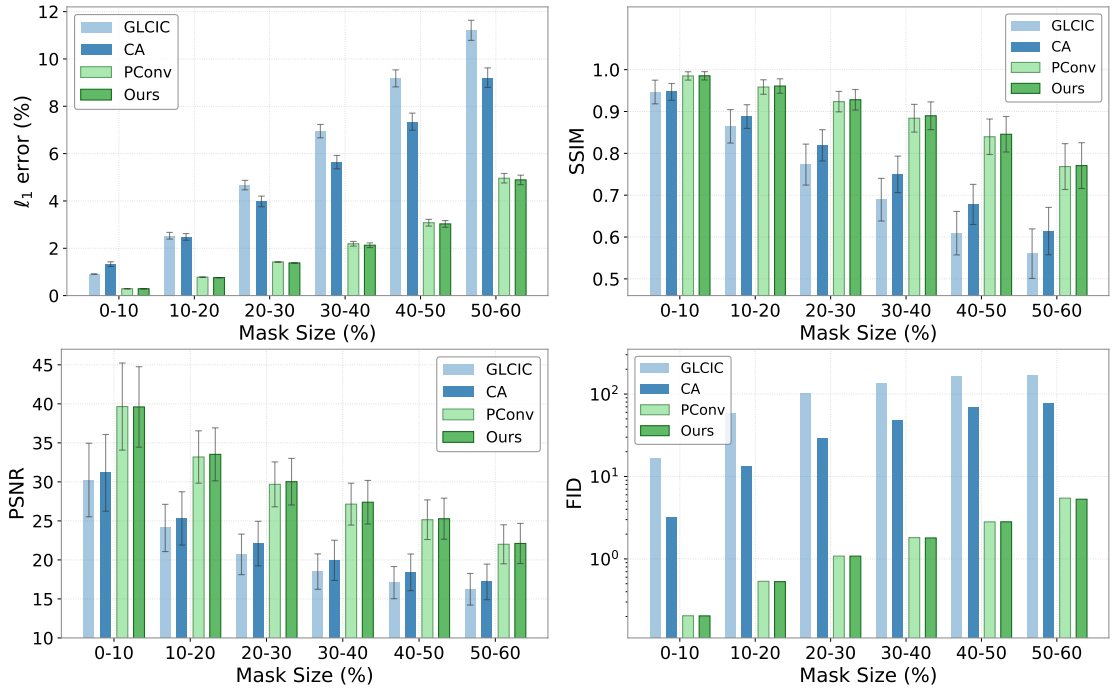


Figure 4.6: Effect of relative mask sizes on ℓ_1, SSIM, PSNR, and FID for CelebA dataset.

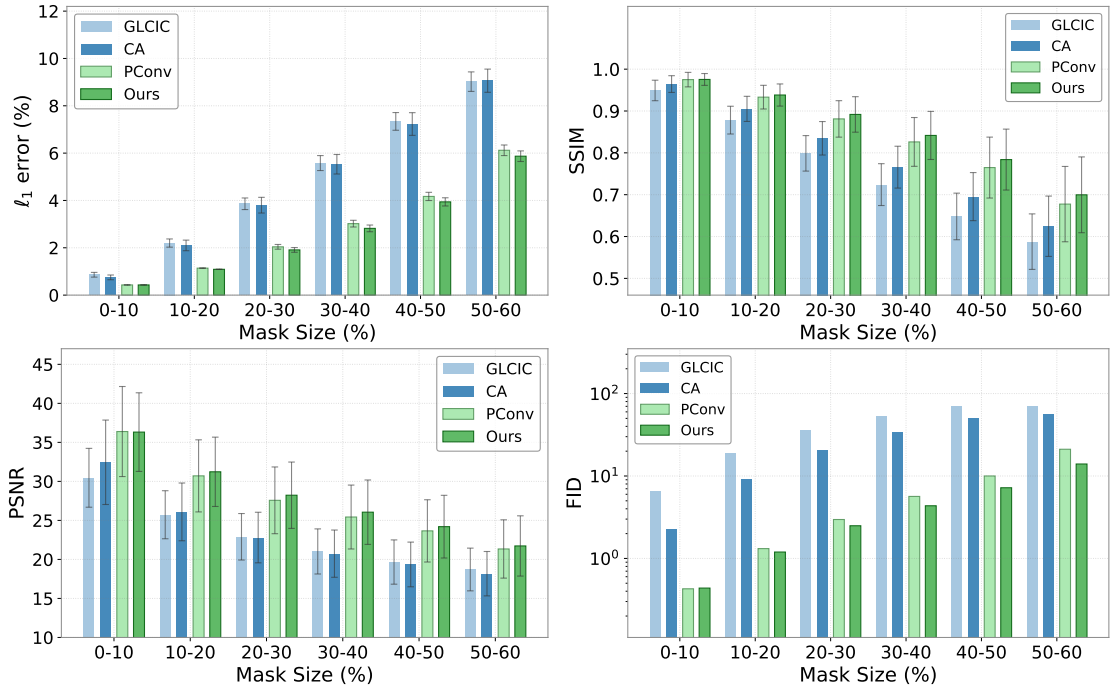


Figure 4.7: Effect of relative mask sizes on ℓ_1, SSIM, PSNR, and FID for Paris StreetView.

Visual Turing Tests

For objective evaluation of the results of the inpainting model we perform *yes-no tasks* (Y-N) and *just noticeable differences* (JND) (see Sections 3.3.5 and 3.3.5 for detail). For Y-N, a single image was randomly sampled from either ground truth images, or images generated by our model. Participants were asked whether the sampled image was real or not. For JND, we asked participants to select a more realistic image from pairs of real and generated images. For both tests, two seconds were given for each image set(s). The tests were performed over 300 images for each model and mask size. Each image was shown 10 times in total. The results are summarized in Table 4.5. The margin of error is reported at 95% confidence interval.

	Mask	CA [1]	GLCIC [2]	PConv [3]	Ours
JND (%)	10-20%	$20.98 \pm 1.2\%$	$16.91 \pm 1.1\%$	$36.04 \pm 2.4\%$	$\mathbf{39.69} \pm 1.5\%$
	20-30%	$15.45 \pm 1.1\%$	$14.27 \pm 1\%$	$30.09 \pm 2.4\%$	$\mathbf{36.99} \pm \mathbf{1.5}\%$
	30-40%	$12.86 \pm 1\%$	$12.29 \pm 1\%$	$20.60 \pm 2.1\%$	$\mathbf{27.53} \pm \mathbf{1.3}\%$
	40-50%	$12.74 \pm 1\%$	$10.91 \pm 0.9\%$	$18.31 \pm 2\%$	$\mathbf{25.44} \pm \mathbf{1.3}\%$
Y-N (%)	10-20%	$38.71 \pm 1.8\%$	$22.46 \pm 1.5\%$	$79.72 \pm 2.4\%$	$\mathbf{88.66} \pm \mathbf{1.2}\%$
	20-30%	$23.44 \pm 1.5\%$	$12.09 \pm 1.2\%$	$64.11 \pm 2.9\%$	$\mathbf{77.59} \pm \mathbf{1.5}\%$
	30-40%	$13.49 \pm 1.3\%$	$4.32 \pm 0.7\%$	$52.50 \pm 2.9\%$	$\mathbf{66.44} \pm \mathbf{1.8}\%$
	40-50%	$9.89 \pm 1\%$	$2.77 \pm 0.6\%$	$37.73 \pm 2.7\%$	$\mathbf{58.02} \pm \mathbf{1.8}\%$

Table 4.5: Comparison of Y-N and JND scores for various mask sizes on **Places2** with CA [1], GLCIC [2], PConv [3], and Ours. Y-N score for ground truth images is 94.6%.

Accuracy of Edge Generator

Table 4.6 shows the accuracy of our edge generator G_1 across all three datasets for the inpainting task. We measure precision and recall for various mask sizes. For the precision we measure how many selected pixels as edges are relevant. For recall we measure how many relevant edges are selected. We emphasize that the goal of this experiment is not to achieve the best precision and recall results, but instead to showcase how close the generated edges are to the ground truth edges.

	Mask	Precision	Recall
CelebA	0-10%	51.38	48.64
	10-20%	46.05	42.28
	20-30%	40.98	36.97
	30-40%	35.96	30.57
	40-50%	32.34	25.48
	50-60%	30.17	20.26
Places2	0-10%	48.68	46.70
	10-20%	43.55	41.22
	20-30%	38.71	36.20
	30-40%	34.51	31.36
	40-50%	31.85	27.04
	50-60%	30.53	22.42
PSV	0-10%	56.57	53.95
	10-20%	52.03	48.71
	20-30%	47.56	43.35
	30-40%	43.63	38.07
	40-50%	41.19	32.93
	50-60%	39.44	27.48

Table 4.6: Quantitative performance of edge generator for inpainting trained on Canny edges with $\sigma = 2$ for 256×256 images. Statistics are calculated over the standard test sets of each dataset

4.4.4 Ablative Study

Quantity of Edges versus Inpainting Quality

We now turn our attention to the key assumption of this work: edge information helps with image inpainting. Table 4.7 shows inpainting results with and without edge information. Our model achieved better scores for every metric when edge information was incorporated into the inpainting model, even when a significant portion of the image is missing.

	CelebA		Places2	
Edges	No	Yes	No	Yes
ℓ_1 (%)	4.11	3.03	6.69	5.14
SSIM	0.802	0.846	0.682	0.731
PSNR	23.33	25.28	19.59	21.16
FID	6.16	2.82	32.18	14.98

Table 4.7: Comparison of inpainting results with edge information (our full model) and without edge information (G_2 only, trained without edges). Statistics are based on $10,000$ random masks with size 40-50% of the entire image.

Next, we turn to a more interesting question: How much edge information is needed to see improvements in the generated images? We again use Canny edge detector to construct edge information. We use the parameter σ to control the amount of edge information available to the image completion network. Specifically, we train our image completion network using edge maps generated for $\sigma = 0, 0.5, \ldots, 5.5$, and we found that the best image inpainting results are obtained with edges corresponding to $\sigma \in [1.5, 2.5]$, across all datasets shown in Figure 4.8. For large values of σ, too few edges are available to make a difference in the quality of generated images. On the other hand, when σ is too small, too

many edges are produced, which adversely affect the quality of the generated images. We used this study to set $\sigma = 2$ when creating ground truth edge maps for the training of the edge generator network.

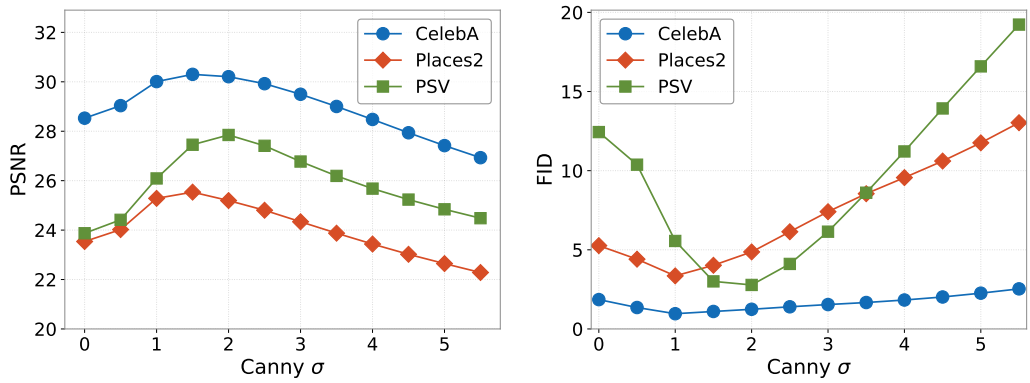


Figure 4.8: Effect of σ in Canny detector on PSNR and FID.

Figure 4.9 shows how different values of σ affects the inpainting task. Note that in a region where edge data is sparse, the quality of the inpainted region degrades. For instance, in the generated image for $\sigma = 5$, the left eye was reconstructed much sharper than the right eye.

Alternative Edge Detection Systems

We use Canny edge detector to produce training labels for the edge generator network due to its speed, robustness, and ease of use. Canny edges are one-pixel wide, and are represented as binary masks (1 for edge, 0 for background). In drawing, an edge is a boundary that separates two areas. A thick line brings the shape forward thin line indicates a plane receding into the background. In other words, edges create a sense of distance and are not just about lines. Here we use HED [119] as an alternative edge detection system. Edges produced with HED, are of varying thickness, and pixels can have intensities ranging between 0 and 1 (see Section 3.2.3). We noticed that it is possible to create edge maps that look

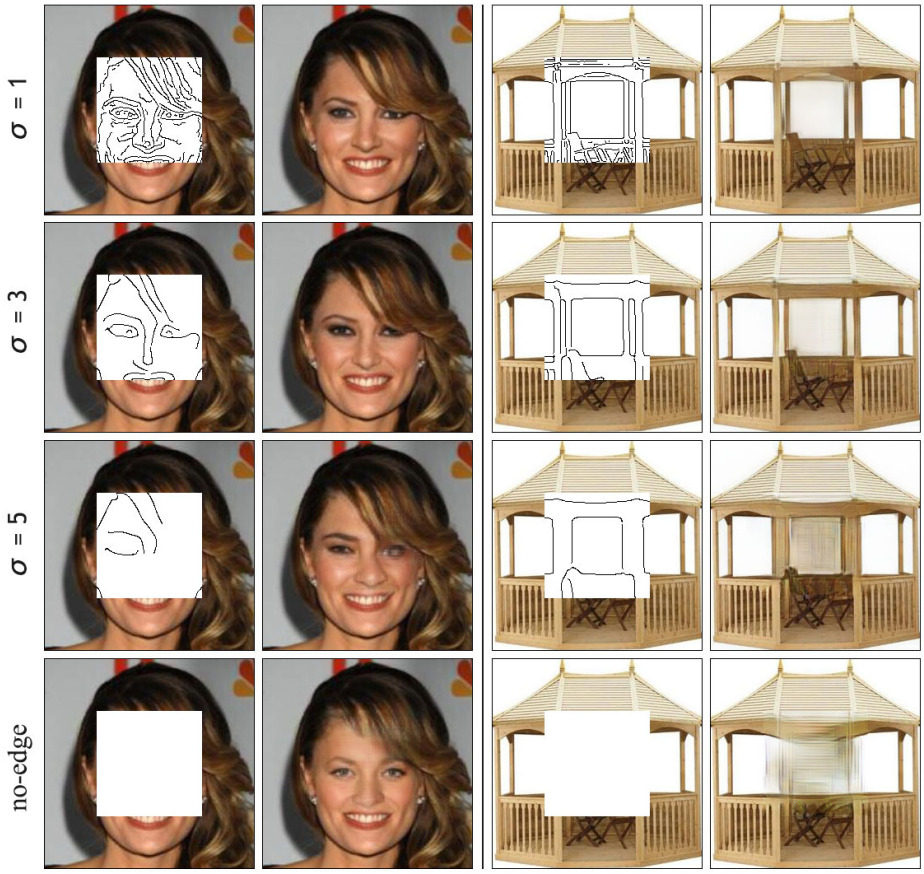
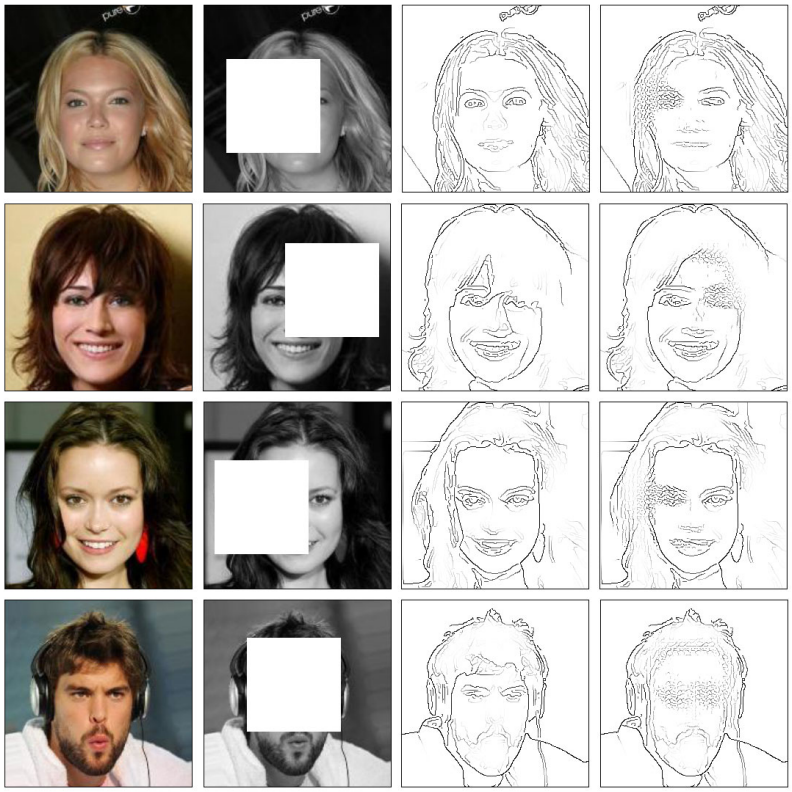


Figure 4.9: Effect of σ in Canny edge detector on inpainting results. Top to bottom: $\sigma = 1, 3, 5$, no edge data.

eerily similar to human sketches by performing element-wise multiplication on Canny and HED edge maps. We compare the quantitative results between Canny and a combination of HED and Canny edges (*i.e.* HED$\odot$Canny). Generated images based on the combined edges gave the best performance. However, our generator G_1 is unable to generate these type of edges accurately during training. Table 4.8 shows G_1 trained on HED$\odot$Canny had the poorest performance out of all methods despite its ground truth counterpart achieving the best performance. These results suggest that better edge detectors result in better inpainting, however, effectively drawing those edges remains an open question in our research. Figure 4.11 shows the results using hybrid edges.

(a) (b) (c) (d)

Figure 4.10: (a) Image. (b) Canny. (c) HED. (d) Canny⊙HED.



(a) (b) (c) (d)

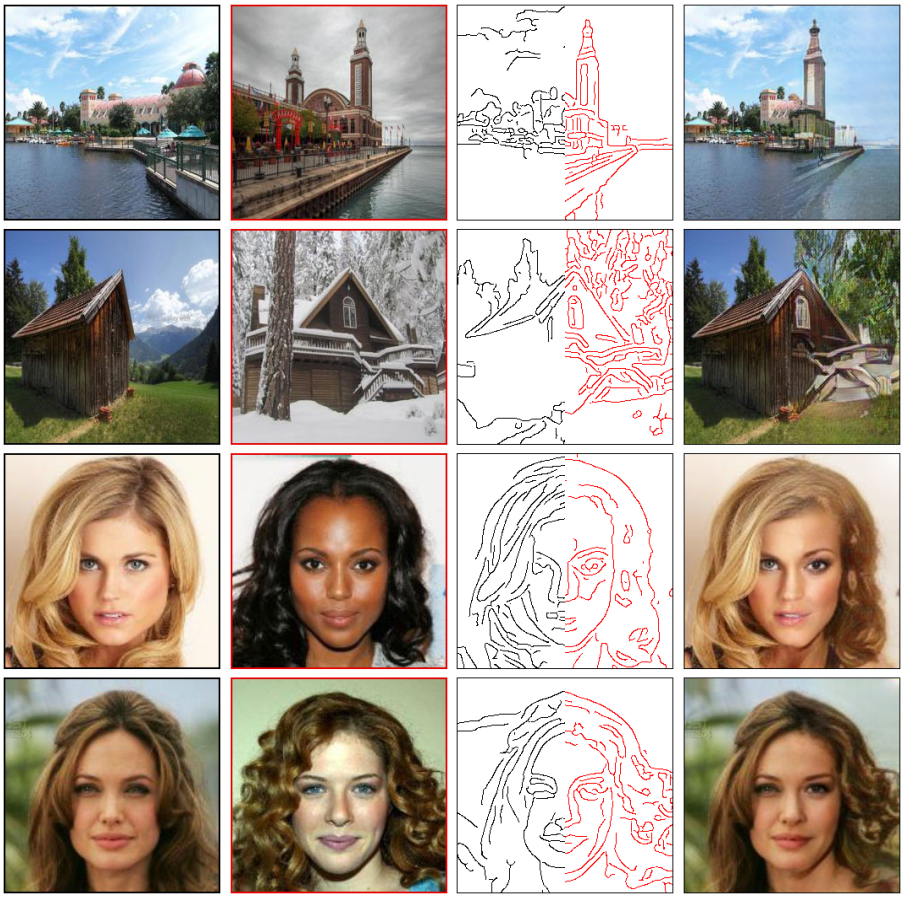
Figure 4.11: Generated edges by G_1 trained using hybrid (HED⊙Canny) edges. Images are best viewed in color. (a) Original Image. (b) Image with Masked Region. (c) Ground Truth Edges. (d) Generated Edges.

	Mask	Hybrid		Canny	
		G_1	GT	G_1	GT
ℓ_1 (%)†	0-10%	0.31	0.23	0.29	0.25
	10-20%	0.79	0.55	0.76	0.59
	20-30%	1.42	0.93	1.38	1.00
	30-40%	2.19	1.35	2.13	1.45
	40-50%	3.10	1.82	3.03	1.97
	50-60%	4.95	2.61	4.89	2.88
SSIM⋆	0-10%	0.985	0.990	0.985	0.988
	10-20%	0.959	0.978	0.961	0.972
	20-30%	0.926	0.959	0.928	0.951
	30-40%	0.886	0.940	0.890	0.930
	40-50%	0.841	0.920	0.846	0.906
	50-60%	0.767	0.891	0.771	0.872
PSNR⋆	0-10%	39.24	42.43	39.60	41.77
	10-20%	33.26	37.48	33.51	36.81
	20-30%	29.80	34.65	30.02	34.00
	30-40%	27.21	32.59	27.39	31.92
	40-50%	25.12	30.87	25.28	30.21
	50-60%	22.03	28.49	22.11	27.68
FID†	0-10%	0.22	0.11	0.20	0.13
	10-20%	0.56	0.24	0.53	0.31
	20-30%	1.13	0.41	1.08	0.57
	30-40%	1.90	0.61	1.80	0.88
	40-50%	2.99	0.83	2.82	1.25
	50-60%	5.67	1.14	5.30	1.79

Table 4.8: Comparison of quantitative results between Hybrid (HED$\odot$Canny) and Canny edges over CelebA. Statistics are shown for generated edges (G_1) and ground truth edges (GT). †Lower is better. ⋆Higher is better.

4.4.5 Applications

Our trained model can be used as an interactive image editing tool. We can, for example, manipulate objects in the edge domain and transform the edge maps back to generate a new image. This is demonstrated in Figure 4.12. Here we have removed the right-half of a given image to be used as input. The edge maps, however, are provided by a different image. The generated image seems to share characteristics of the two images. Figure 4.13 shows examples where we attempt to remove unwanted objects from existing images.



(a) (b) (c) (d)

Figure 4.12: Edge-map (c) generated using the left-half of (a) (black edges) and right-half of (b) (red edge). Input is (a) with the right-half removed, producing the output (d).

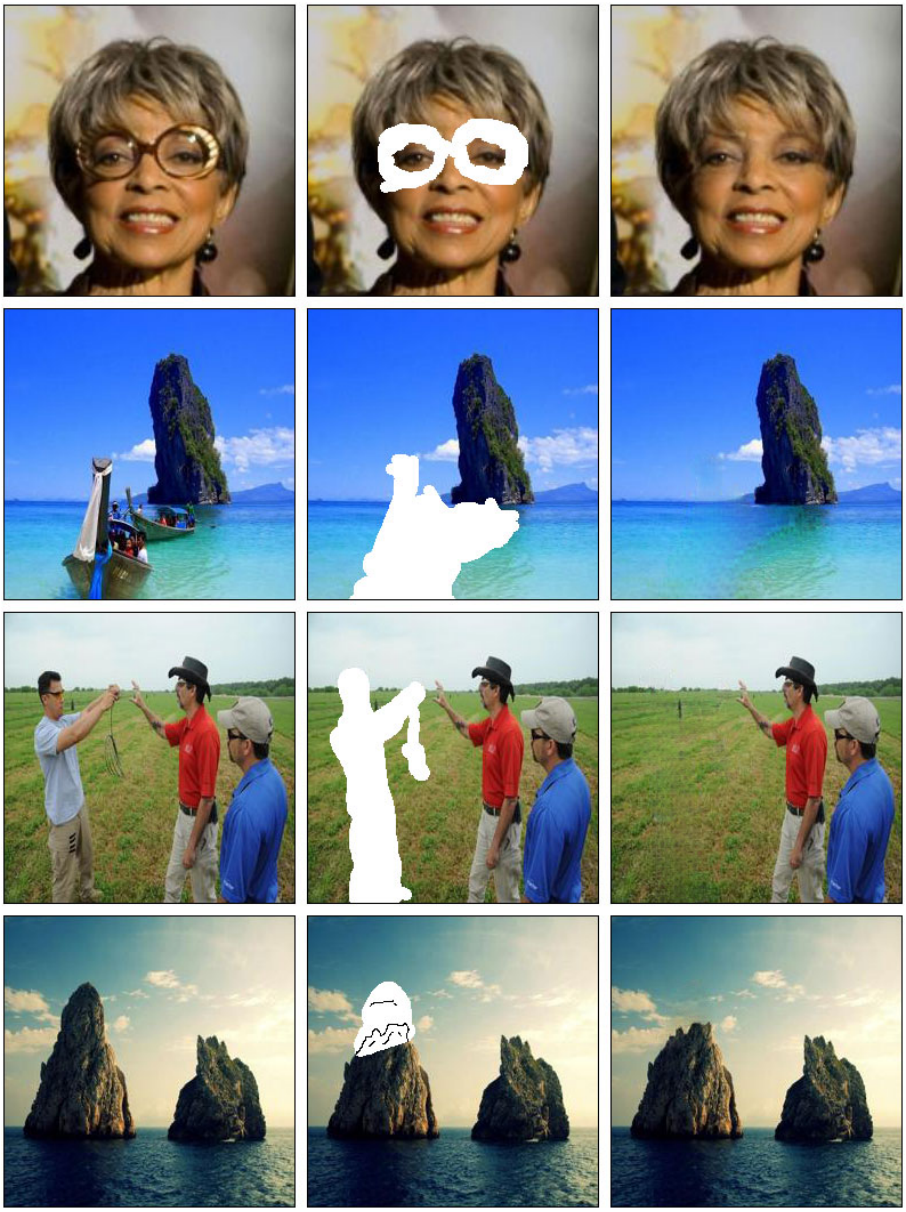


Figure 4.13: Examples of object removal and image editing using our EdgeConnect model. (Left) Original image. (Center) Unwanted object removed with optional edge information to guide inpainting. (Right) Generated image.

Figure 4.14: Inpainting results where the edge generator fails to produce relevant edges.

4.5 Summary

This chapter discusses a new structure-driven deep learning model for image inpainting tasks. The proposed model consisted of a two-stage pipeline that disentangles edge generation and image completion. In particular, we present an edge generator network followed by an image completion network, both following an adversarial model. The edge generator network hallucinates the edges for the missing region using Canny edge detector as reference. The image completion network uses the edge information to better reconstruct the missing region by adding color and texture on top of the structure. Style, perceptual, reconstruction, and adversarial losses are used to train this network. Quantitative and qualitative comparisons and visual Turing test show the effectiveness of the proposed model.

We show that the quality of the edge information plays a vital role in image inpainting task. Edges aren't just about lines, effectively depicting edges will recreate the sense. One limitation of our proposed method is that it sometimes fails to accurately depict the edges in highly textured areas, or when a large portion of the image is missing (Figure 4.14). Any improvement in the edge generation process will greatly enhance the quality of inpainting.

5. Single Image Super-Resolution

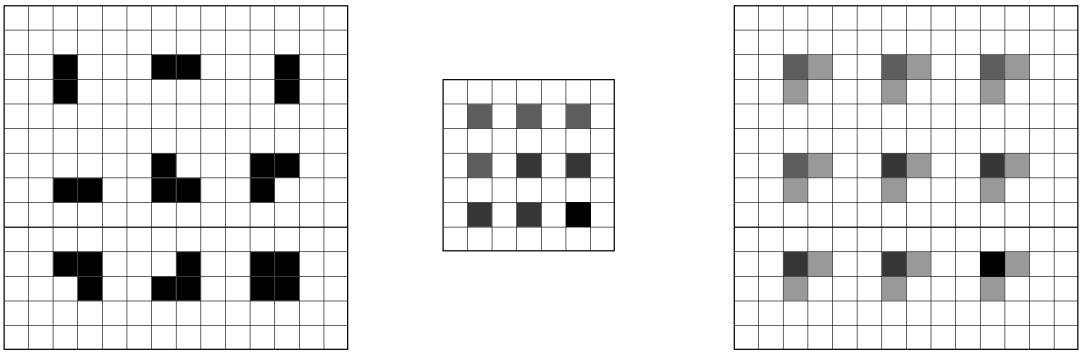
This chapter a new approach to single image super-resolution (SISR) is presented by re-formulating the problem as an in-between pixels inpainting task. We propose our existing two-stage inpainting model as a baseline for super-resolution and show its effectiveness for different scale factors ($\times 2$, $\times 4$, $\times 8$) compared to basic interpolation schemes. This model is trained using a joint optimization of image contents (texture and color) and structures (edges). Quantitative and qualitative comparisons are included and the proposed model is compared with current state-of-the-art techniques.

5.1 Introduction

Super-Resolution or SR is a task of inferring a high-resolution (HR) image from one or more of its low-resolution (LR) versions. It has direct applications in medical images, face recognition, satellite imaging, and surveillance. Some SR methods require multiple instances of LR images with different perspectives to reconstruct the HR image. These are called *Multi-Image Super-Resolution* (MISR) methods. In most cases, however, only a single image is available and the goal is to recover missing HR information from a single LR image. This category of SR is called *Single-Image Super-Resolution* (SISR). SISR is a challenging ill-posed problem and normally requires prior information and reconstruction constraints to restrict the solution space of the problem[158]: A low-resolution image is created by cropping high-frequency information in the HR and is limited by Nyquist sampling theorem which makes the HR space that we intend to map the LR image to, oftentimes intractable [159].

Figure 5.1 shows a toy example of SISR problem: when downsampling, many different HR images may end up with the same LR image. This becomes a challenging one-to-many problem when we try to estimate the HR image from the LR, rendering a blurry image at best with most distinctive features in the original image being lost. This becomes more challenging for higher SISR magnification rations as the one-to-many mapping becomes worse with dimensionality providing multiple solutions to the problem, of which determining the correct solution is non-trivial [158].

Following the deep learning success in reconstruction accuracy and computational efficiency for single image super-resolution, we propose a novel approach to SISR problem based on our deep learning model for image inpainting. Figure 5.2 illustrates the process where increasing the resolution of an image corresponds to interpolating between every two adjacent pixels. We can treat this as a missing region in an image that needs to be inpainted



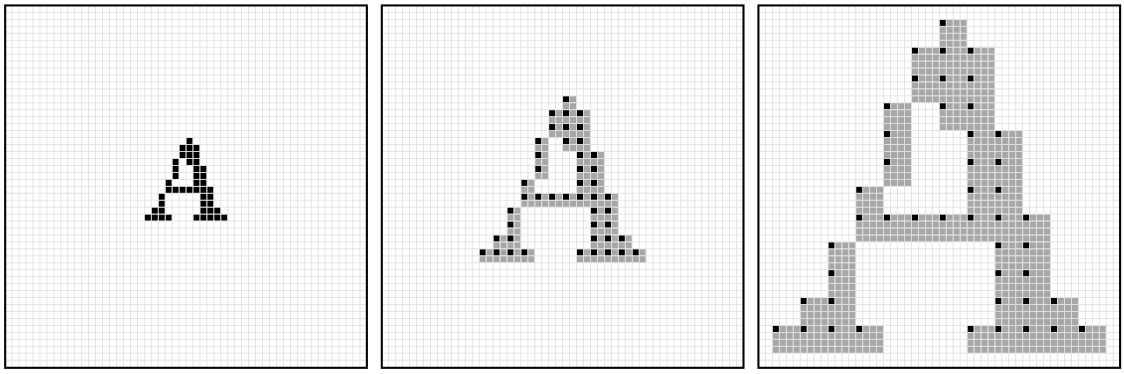
(a) Ground Truth (b) LR Image (c) HR Estimate

Figure 5.1: Schematic illustration of super-resolution problem. (a) The ground truth image, (b) The image downsampled by a factor of two. Each four-pixel information on the left turn into one pixel in the middle, as a result, the structure and orientation of edges are not distinguished anymore showing the problem is ill-posed. (c) The reconstruction of a high-resolution image from one-pixel information using bilinear interpolation. Most distinctive features in the original image are lost and the result is blurry around the edges.

effectively reformulating SISR as an in-between pixels inpainting task. Our approach to SISR follows the same procedure as the proposed inpainting model. We first create a mask for every extra rows and columns that we need to fill for the HR image. The process then follows the same two-stage pipeline for inpainting by first hallucinating the edges for the empty region and use them as a priori for the next stage where we estimate the RGB pixel intensities of the missing region.

5.2 Related Work

Many approaches to SISR problem have been proposed in the literature. In a comprehensive study, Yang *et al*. [160] categorized SISR algorithms into several types according to the image priors: **Prediction models** generate HR image through predefined mathematical



(a) LR image (b) Upsample by $2\times$ (c) Upsample by $4\times$

Figure 5.2: An illustration of the proposed inpainting-based method for SISR problem. (a) The original LR image. (b) Upsampling by a factor of two corresponds to interpolating one pixel between every two adjacent pixels. We add an extra empty row and column for every rows and columns in the ground truth image (shown in gray) which we fill by an inpainting process. (c) Upsampling by a factor of four corresponds to interpolating three pixels between every two adjacent pixels where we can add three extra empty rows and columns for every rows and columns in the ground truth image to be inpainted.

functions. Examples include bilinear and bicubic interpolation [161], and Lanczos [162] resampling. **Edge-based methods** learn priors from features such as width of an edge [163], or parameter of a gradient profile [164] to reconstruct the high-resolution image. **Statistical methods** exploit different image properties such as gradient distribution [165] to predict HR images. **Patch-based methods** use exemplar patches from external datasets [166, 167] or the image itself [168, 169] to learn mapping functions from LR to HR.

Deep learning-based methods have achieved great performance on SISR using deep convolutional networks with a per-pixel Euclidean loss [158, 170, 171, 172]. Euclidean loss, however, is less effective to reconstruct high-frequency structures such as edges and textures. Recently, Johnson *et al*. [79] proposed feed-forward CNN using a perceptual loss

(see Section 3.3.4). In particular, they used a pre-trained VGG network [70] to extract high-level features from an image effectively separating content and style. Their model was trained with a joint optimization of *Feature reconstruction loss* and *Style reconstruction loss* and achieved state-of-the-art results on SISR for challenging $\times 8$ magnification ratio. To encourage spatial smoothness and mitigate the checkerboard artifact [156] of using feature reconstruction loss, they introduced *total variation regularizer* [173] to their model objective. Sajjadi *et al*. [4] proposed to use style loss in a patch-wise fashion to reduce the checkerboard artifact [156] and enforce locally similar textures between the HR and the ground truth image. They also used the adversarial loss to produce sharp results and further improve the SISR. Adversarial loss is shown to be very effective in realistically synthesized high-frequency textures in SISR problem[91, 174, 175], however, the results of these GAN-based approaches tend to include less meaningful high-frequency noise around the edges that is irrelevant to the input image [175]. Our work herein is inspired by the model proposed by Liu *et al*. [3] which extended their image inpainting framework to image super-resolution tasks by offsetting pixels and inserting holes. We present a SISR model that simultaneously improves structure, texture, and color to generate a photo-realistic high-resolution image.

5.3 Model

To extend our proposed model to SISR problem we follow the same formulation discussed in previous chapter. However, instead of the edge generation step, we slightly modify the network to enhance the edges from a low-resolution image to a high-resolution edge map.

Let $\mathbf{I}^{(LR)}$ and $\mathbf{I}^{(HR)}$ be the low-resolution and high-resolution images. Their corresponding edge maps will be denoted as $\mathbf{C}^{(LR)}$ and $\mathbf{C}^{(HR)}$ respectively and $\mathbf{I}^{(LR)}_{gray}$ is a grayscale counterpart of the low-resolution image. The edge enhancement network G_1

predicts the high-resolution edge map

$$\mathbf{C}_{pred}^{(HR)} = G_1(\mathbf{I}_{gray}^{(LR)}, \mathbf{C}^{(LR)}), \tag{5.1}$$

where the $\mathbf{I}_{gray}^{(LR)}$ and $\mathbf{C}^{(LR)}$ are the inputs to the networks and we use the same objective as the one in Equation 4.2 to train the model. The generator network architecture is slightly changed to handle a low-resolution image as input. Particularly, we add a nearest-neighbor interpolation module at the beginning of the network in 4.2 to resize the low-resolution image and its Canny edge-map to the same size as the HR image. The modified architecture of $G1$ can be seen Figure 5.3 for SISR with $\times 4$ scale factor.

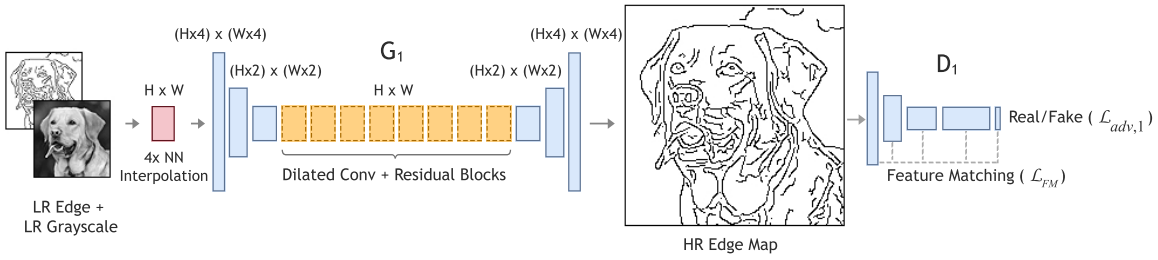


Figure 5.3: Summary of our proposed edge enhancement network for $\times 4$ SISR. Low-resolution grayscale image and edge map are the inputs of G_1 to predict the high-resolution edge map. Predicted edge map will be used in an inpainting network to perform SISR.

We use the same image completion network as discussed in Section 4.3.2 to perform SISR. To offset the pixels of the LR image and create an incomplete HR image, we use a fixed fractionally strided convolution kernels at the beginning of the network. For example, to offset the pixels and increase the size of an image by a factor of s we use a $s \times s$ convolution kernel with stride of $1/s$. The kernel's for $\times 2$ and $\times 4$ SISR factors are shown in Figure 5.4.

Let K denote a fixed strided convolution kernel, $\hat{\mathbf{I}}^{(HR)} = \mathbf{I}^{(LR)} * K$ is the high-resolution image constructed by offsetting the pixels (horizontally and vertically) from LR image.

$$K_2 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \qquad K_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 5.4: Fixed fractionally strided convolution kernels to offset the pixels of the LR image and create an incomplete HR image for $\times 2$ and $\times 4$ SISR factors.

The HR image is generated using G_2 network:

$$I_{pred}^{(HR)} = G_2\left(\hat{\mathbf{I}}^{(HR)}, \mathbf{C}_{pred}^{(HR)}\right). \tag{5.2}$$

We train G_2 using the same objective and regularization parameters from Equation 4.11, where we effectively minimize the reconstruction, style, perceptual, and adversarial loss to generate a photo-realistic high-resolution image.

5.4 Experiments

Our proposed models are evaluated on the following publicly available datasets.

- Celeb-HQ [42]. High-quality version of the CelebA dataset with 30K images.
  `https://github.com/tkarras/progressive_growing_of_gans`
- Places2 [37]. More than 10 million images comprising 400+ unique scene categories.
  `http://places2.csail.mit.edu/`
- Set5, Set14, BSDS100, Urban100 [43]. Standard SISR evaluation datasets.
  `http://vllab.ucmerced.edu/wlai24/LapSRN/`

Results are compared against the current state-of-the-art methods both qualitatively and quantitatively.

5.4.1 Qualitative Evaluation

Figure 5.5 and 5.6 show results of the proposed SIRS method for scale factors of $\times 4$ and $\times 8$ respectively. For visualization purposes, the LR image is resized using nearest-neighbor interpolation. All HR images are cropped at 512×512, which means the LR images are 128×128 and 64×64 for scale factors of $\times 4$ and $\times 8$ respectively. We obtain the LR images by blurring the HR with a Gaussian kernel of width $\sigma = 1$ and downsampling with bilinear interpolation. The results are compared against Bicubic interpolation and our proposed model without the edge generation network as a baseline. Despite having almost high PSNR/SSIM, the baseline model produces blurry results around the edges while our full model (with edge-maps) remains faithful to the high-frequency edge data and produces sharp photorealistic images.

5.4.2 Quantitative Evaluation

We evaluate our model using PSNR and SSIM for $\times 2$, $\times 4$ and $\times 8$ SISR scale factors. We obtain the LR images by blurring the HR with a Gaussian kernel of width $\sigma = 1$ and downsampling with bilinear interpolation. Table 5.1 shows the performance of our model against Bicubic interpolation and current state of the art SISR models over datasets Set5, Set14, BSD100, and Celeb-HQ. Statistics for competing models for $\times 2$ and $\times 4$ SR were obtained from their respective papers where available. Results for a challenging case of $\times 8$ are only compared against Bicubic interpolation. Note that the PSNR in our results is lower than competing models. In particular, EDSR by Lim *et al*. [5] has achieved the best PSNR for every dataset. However, their model is only trained with per-pixel ℓ_2 loss and fails to reconstruct sharp edges despite having higher PSNR. Similar results in recent research [79, 4, 14] show that PSNR favors smooth/blurry results.

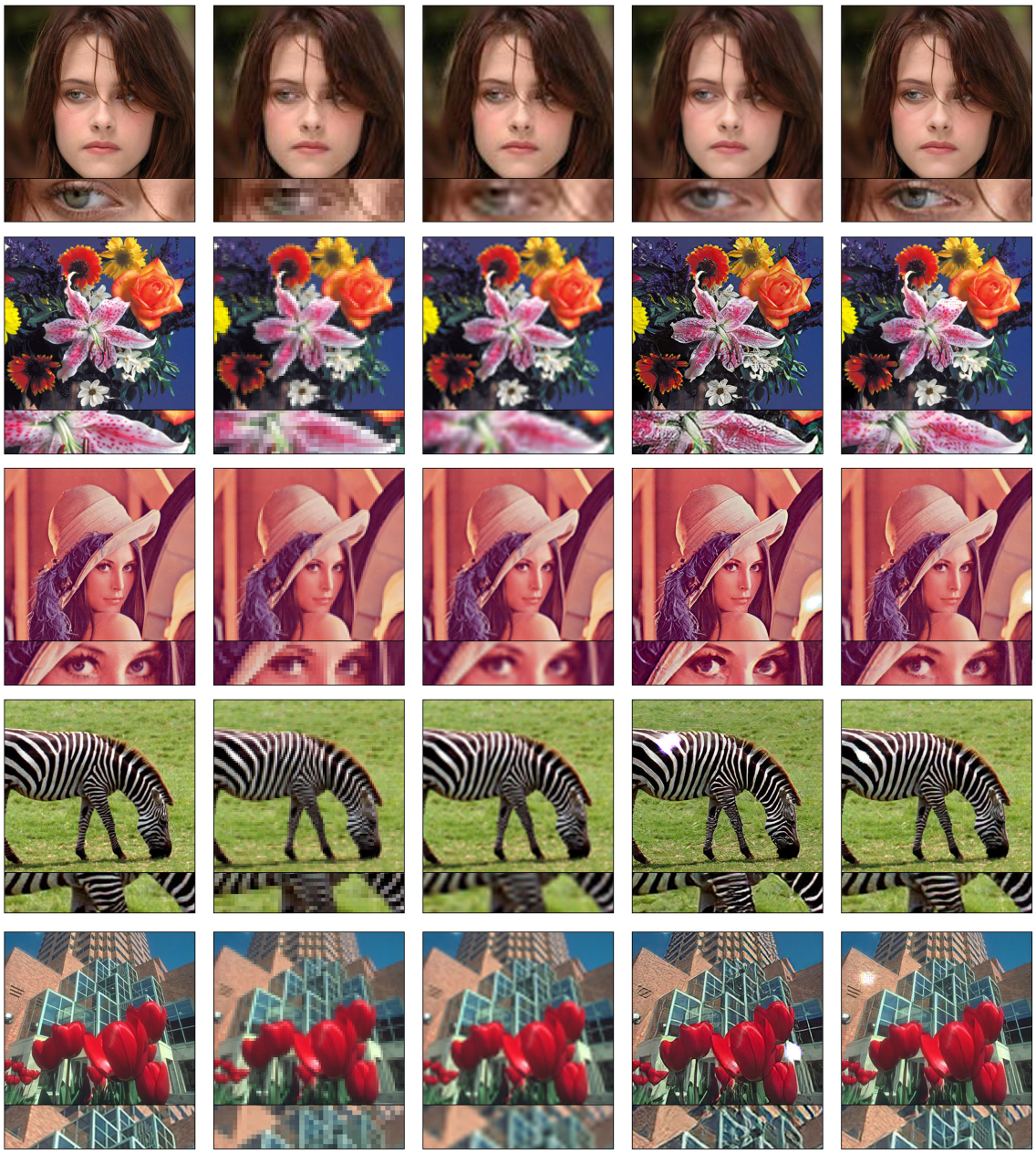


Figure 5.5: Comparison of qualitative results of images for $\times 4$ scale factor SISR cropped at 512×512. Left to right: Ground Truth HR, LR image upscaled using nearest-neighbor interpolation, SISR using Bicubic interpolation, Baseline (no edge data), Ours (Full Model)

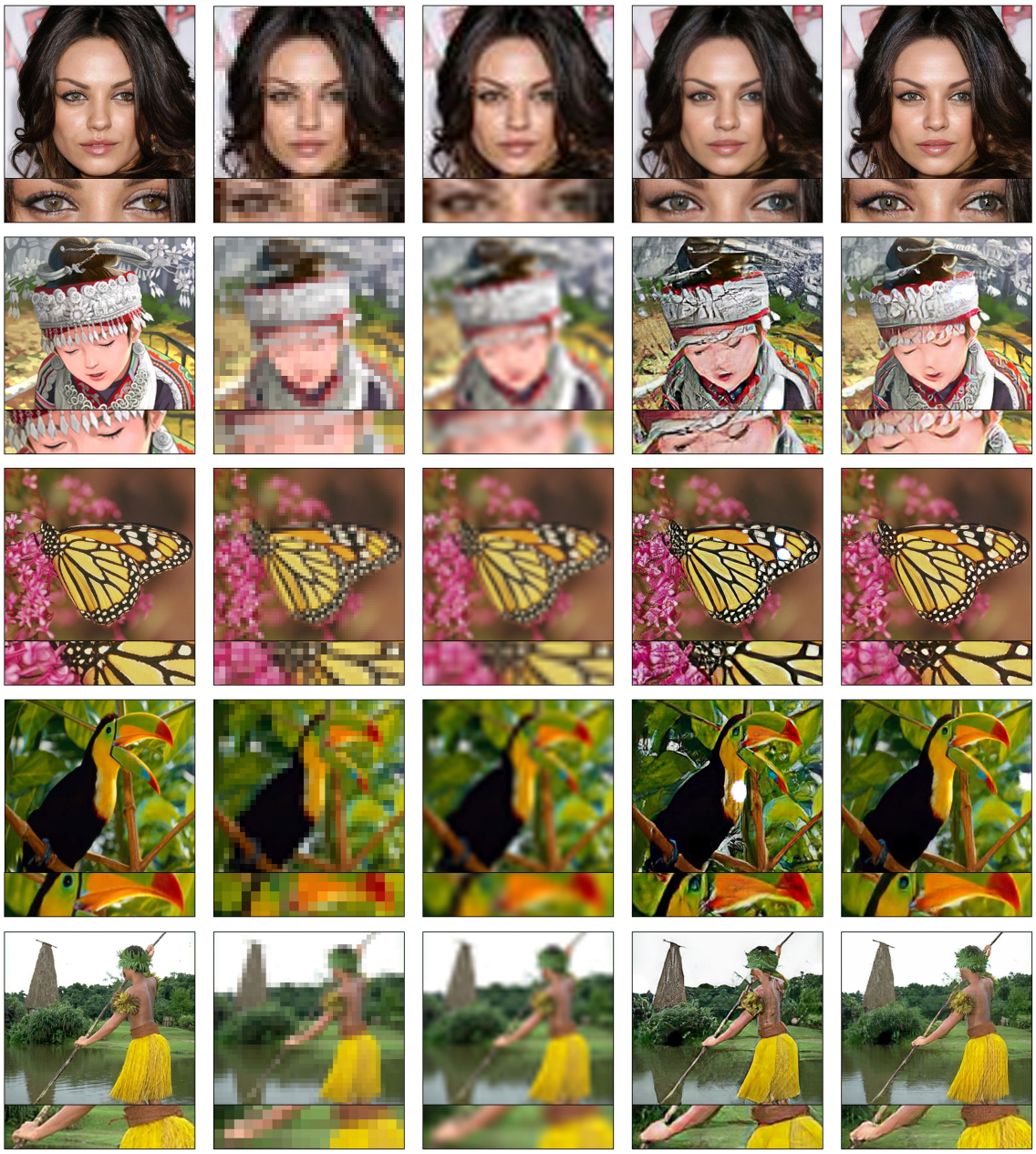


Figure 5.6: Comparison of qualitative results of images for $\times 8$ scale factor SISR cropped at 512×512. Left to right: Ground Truth HR, LR image upscaled using nearest-neighbor interpolation, SISR using Bicubic interpolation, Baseline (no edge data), Ours (Full Model)

		Dataset	Bicubic	ENet	EDSR	Baseline	Ours
PSNR	$\times 2$	Set5	33.66	33.89	**38.20**	27.32	33.60
		Set14	30.24	30.45	**34.02**	24.86	29.24
		BSD100	29.56	28.30	**32.37**	23.97	28.12
		Celeb-HQ	**33.25**	-	-	31.33	32.12
	$\times 4$	Set5	28.42	28.56	**32.62**	24.22	28.59
		Set14	25.99	25.77	**28.94**	21.56	25.19
		BSD100	25.96	24.93	**27.79**	20.78	24.25
		Celeb-HQ	**29.59**	-	-	27.94	28.23
	$\times 8$	Set5	**23.80**	-	-	19.32	23.73
		Set14	**22.37**	-	-	18.47	21.44
		BSD100	**22.11**	-	-	18.65	21.63
		Celeb-HQ	**26.66**	-	-	25.46	25.56
SSIM	$\times 2$	Set5	0.930	0.928	0.961	0.974	**0.985**
		Set14	0.869	0.862	0.920	0.930	**0.954**
		BSD100	0.843	0.873	0.902	0.909	**0.932**
		Celeb-HQ	0.967	-	-	0.957	**0.968**
	$\times 4$	Set5	0.810	0.809	0.898	0.929	**0.965**
		Set14	0.703	0.678	0.790	0.832	**0.894**
		BSD100	0.668	0.627	0.744	77.29	**0.851**
		Celeb-HQ	0.834	-	-	0.910	**0.912**
	$\times 8$	Set5	0.646	-	-	0.801	**0.904**
		Set14	0.552	-	-	0.708	**0.793**
		BSD100	0.532	-	-	0.663	**0752**
		Celeb-HQ	0.782	-	-	0.841	**0.857**

Table 5.1: Comparison of PSNR and SSIM for $\times 2$, $\times 4$, and $\times 8$ factor SISR over **Set5**, **Set14**, **BSD100**, and **Celeb-HQ** datasets with Bicubic interpolation, ENet [4], EDSR [5], and baseline (without edge-data). The best result of each row is boldfaced.

Accuracy of Edge Generator

Table 5.2 shows the accuracy of our edge enhancer G_1 for Celeb-HQ and Places2 datasets for the Single Image Super-Resolution task. We measure precision and recall for various scale factors SISR. In all experiments the width of the Gaussian smoothing filter $\sigma = 2$ for Canny edge detection.

	Scale	Precision	Recall
Celeb-HQ	$\times\mathbf{2}$	74.27	73.21
	$\times\mathbf{4}$	45.14	43.04
	$\times\mathbf{8}$	23.23	19.09
Places2	$\times\mathbf{2}$	79.18	80.24
	$\times\mathbf{4}$	60.80	58.19
	$\times\mathbf{8}$	31.06	23.93

Table 5.2: Quantitative performance of edge enhancer for Single Image Super-Resolution trained on Canny edges with $\sigma = 2$ for 512×512 images. Statistics are calculated over the standard test sets of each dataset.

5.5 Summary

This chapter discusses a new structure-driven deep learning model for single image super-resolution (SISR) by recasting the problem as an in-between pixels inpainting task. One benefit of this approach over most deep-learning based SISR models is that we only have one model that is used for different SISR scales. Most deep-learning based SISR models take the LR image as input and generate the HR by in-network upsampling layers. This requires different network architectures and training different models for every HR resolution. Whereas our model takes the LR image and adds empty space between pixels before using it as input to the network. Our proposed model learns to fill in the missing pixels by relying on the available edge information to create the high-resolution image and effectively applies parameter sharing for different scales of SISR. Quantitative results show the effectiveness of the structure-guided inpainting model for SISR problem where it achieves state-of-the-art results on standard benchmarks.

One shortcoming of the proposed inpainting-based SISR is that it requires minimizing two disjoint optimizing algorithms. A better approach is to incorporate the edge generation stage into the inpainting model's objective. This model could be trained using a joint optimization of image contents and structures and potentially outperform disjoint two-stage optimization algorithm computationally while preserving sharp details and high image quality.

Another limitation of this model is that it is not easily scalable to non-integer scale factors. One solution would be to use an interpolation method on a low-resolution image as well as the inpainting mask as inputs to the network. In this scheme, the inpainting mask is no longer a binary mask but a heat-map that guides the completion network through the inpainting process.

We leave these limitations as an interesting direction for future works.

6. Conclusions

Image restoration is an unsolved computer vision problem. In this dissertation, we made a small step ahead by focusing on the most salient aspect of an image — image structures. We propose an image restoration model that effectively disentangles structure inference and image completion. We show that this disentanglement substantially improves the performance of image restoration and achieves superior qualitative results for image inpainting and single image super-resolution problems.

A great deal of this research has focused on edge information as image structures. Chapter 3, offered two fundamental approaches to edge detection: *Computational edge detection* methods are local image processing techniques that rely on local image statistics such as image gradients, phase, or histogram to compute edges. *Learning-based edge detection* methods are mostly dominated by deep learning techniques and leverage automatic hierarchical feature learning presented by deep convolutional networks. While computational methods heavily rely on hand designed feature extractors and normally require manual parameters selection, learning-based methods require no human supervision and oftentimes produce superior edges for challenging ambiguous cases of edges and object boundaries. Moreover, when properly parallelized, learning-based methods can be computed orders of magnitude faster than computational methods which allow these techniques to be used as objective functions in optimization algorithms or in real-time applications such as objects and boundary detection for self-driving cars.

Inspired by artists' work in line drawing and the success of learning-based edge detec-

tion methods, we propose a novel "line first, color next" approach for image restoration. Chapter 4 presents a new deep learning model for the image inpainting problem. The proposed framework is a two-stage pipeline that comprises an edge generator and an image completion model. The edge generator creates an entire structure of the image by hallucinating the edges of the missing region using learned data distribution. The image completion uses the structure as a guideline to the inpainting process by adding texture and color to the missing region. The proposed method combines two different approaches to inpainting problem: *Structural Inpainting* and *Textural Inpainting*. Both stages use deep convolutional networks that are trained in a conditional unsupervised adversarial setting using the recently proposed adversarial loss in their optimization objectives. To that end, we introduce two discriminator networks that validate the results of each stage by measuring how realistic they look. Our method achieves state-of-the-art results on standard benchmarks and is able to deal with images with multiple, irregularly shaped missing regions.

In Chapter 5 expanded the image inpainting model to single image super-resolution (SISR). Increasing the resolution of an image corresponds to interpolating between every two adjacent pixels. We address this as a missing region in an image that can be filled effectively reformulating SISR as an in-between pixels inpainting task. The model proposed for SISR is similar to our two-stage pipeline for the inpainting problem. The edge generator for image inpainting can be seen as an edge enhancer for SISR that maps a low-resolution image to a high-resolution edge-map; albeit in principle, both models perform exactly the same function. Qualitative and quantitative results in this chapter, underline the effectiveness of reconstructing high-frequency image structures for SIRS problem.

While the model presented in this thesis outperforms current state-of-the-art techniques for image restoration, extending it to higher resolution images is extremely challenging. One possible future direction would be to address this with a multi-scale approach, by first predicting a low-resolution variant of edge data, and recursively up-sampled and refine the

edges until the desired resolution is reached. This allows image structure to be scaled up with minor degradation using common interpolation techniques. Since image completion is able to produce photo-realistic results provided that the edge data is accurate, the model can be extended to very high-resolution restoration applications by following a pyramid model for edge prediction.

There is no convincing quantitative measure that can truly evaluate the image restoration process. The intuitive story for Chapter 3 is to present basic image quality assessment techniques using computational models and/or perceptual human assessment tests. *Objective* evaluation techniques provide quantitative measures to assess the perceived image quality. Although these methods each, to some extent, measure the magnitude of degradation or similarity between a reference and the restored degraded image, there's no single metric that can measure all degradation. A combination of different numerical measures may prove to be more useful objective quality assessment method. *Subjective* evaluation techniques, however costly and time-consuming, still remain the only correct method to evaluate visual image quality. Combining the best of both methods as a unified image quality assessment method that can measure most impairments, and agrees with human visual perception, should be a fruitful direction for future work.

The fundamental question of this research is "How to accurately generate structures from a partially observed image?". We demonstrate in this thesis that edge information plays an important role in image restoration. While effectively delineating these edges is more useful than hundreds of detailed lines, our model sometimes struggles to accurately depict the edges. The question that motivated this thesis remains tantalizingly unanswered. In actual line drawing, lines play a vital role in the composition: A thick line may bring a shape forward, sharp lines indicate a focal point, and thin lines may recede objects to background. An image restoration model that can learn to capture these subtle nuances in drawing will provide vast array of open and interesting avenues for future work.

A. Inpainting Results

Figure A.1: Sample of results with CelebA dataset (512×512). Images are best viewed in color. From left to right: Original Image. Input Image, Generated Result.

Figure A.2: Sample of results with Places2 dataset (512×512). Images are best viewed in color. From left to right: Original Image. Input Image, Generated Result.

Figure A.3: Sample of results with Places2 dataset (512×512). Images are best viewed in color. From left to right: Original Image. Input Image, Generated Result.

Figure A.4: Sample of results with Places2 dataset (512×512). Images are best viewed in color. From left to right: Original Image. Input Image, Generated Result.

Bibliography

[1] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, "Generative image inpainting with contextual attention," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. x, xvi, 3, 35, 74, 90, 99, 103, 104, 105, 108

[2] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Globally and locally consistent image completion," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 107, 2017. x, xvi, 3, 35, 89, 99, 103, 104, 105, 108

[3] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro, "Image inpainting for irregular holes using partial convolutions," in *European Conference on Computer Vision (ECCV)*, September 2018. x, xvi, 3, 74, 89, 97, 99, 102, 103, 104, 105, 108, 122

[4] M. S. M. Sajjadi, B. Scholkopf, and M. Hirsch, "Enhancenet: Single image super-resolution through automated texture synthesis," in *The IEEE International Conference on Computer Vision (ICCV)*, IEEE, 2017. xi, 35, 74, 91, 95, 122, 125, 128

[5] B. Lim, S. Son, H. Kim, S. Nah, and K. Mu Lee, "Enhanced deep residual networks for single image super-resolution," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017. xi, 125, 128

[6] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *International Conference on Learning Representations (ICLR)*, 2016. xiii, 29, 30, 89

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. xiii, 22, 29, 31, 32, 91

[8] Y. Wu and K. He, "Group normalization," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018. xiii, 33, 34

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014. xiv, 13, 34, 40, 41, 42, 87, 91

[10] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985. xiv, 55

[11] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, pp. 679–698, 1986. xiv, 55, 61, 62, 90

[12] J. Kittler, "On the accuracy of the sobel edge detector," *Image and Vision Computing*, vol. 1, no. 1, pp. 37–42, 1983. xiv, xv, 55, 57, 58

[13] J. M. Prewitt, "Object enhancement and extraction," *Picture processing and Psychopictorics*, vol. 10, no. 1, pp. 15–19, 1970. xv, 57, 58

[14] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 586–595, 2018. xv, 74, 75, 102, 125

[15] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in Neural Information Processing Systems*, 2017. xv, 47, 78, 79, 102

[16] "History of the digital camera and digital imaging." Digital Camera Museum, available as `https://www.digitalkameramuseum.de/en/`. 1

[17] E. Lee, "Our best photos deserve to be printed," 2008. available as `http://blog.infotrends.com/our-best-photos-deserve-to-be-printed`. 1

[18] B. K Gunturk and X. Li, *Image Restoration: Fundamentals and Advances (Digital Imaging and Computer Vision)*. CRC Press, 2012. 1, 2

[19] K. Armanious, Y. Mecky, S. Gatidis, and B. Yang, "Adversarial inpainting of medical image modalities," *arXiv preprint arXiv:1810.06621*, 2018. 3

[20] L. M. White and K. A. Buckwalter, "Technical considerations: Ct and mr imaging in the postoperative orthopedic patient," in *Seminars in musculoskeletal radiology*, vol. 6, pp. 005–018, Copyright© 2002 by Thieme Medical Publishers, Inc., 333 Seventh Avenue, New . . . , 2002. 3

[21] M.-J. Lee, S. Kim, S.-A. Lee, H.-T. Song, Y.-M. Huh, D.-H. Kim, S. H. Han, and J.-S. Suh, "Overcoming artifacts from metallic orthopedic implants at high-field-strength mr imaging and multi-detector ct," *Radiographics*, vol. 27, 2007. 3

[22] Z. Feng, S. Chi, J. Yin, D. Zhao, and X. Liu, "A variational approach to medical image inpainting based on mumford-shah model," in *2007 International Conference on Service Systems and Service Management*, pp. 1–5, IEEE, 2007. 3

[23] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2536–2544, 2016. 3, 35, 89

[24] B. Dolhansky and C. C. Ferrer, "Eye in-painting with exemplar generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7902–7911, 2018. 3, 35, 89, 102

[25] R. A. Yeh, C. Chen, T.-Y. Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do, "Semantic image inpainting with deep generative models.," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 3, 89

[26] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, "Free-form image inpainting with gated convolution," *arXiv preprint arXiv:1806.03589*, 2018. 3, 90

[27] Y. Li, S. Liu, J. Yang, and M.-H. Yang, "Generative face completion," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 3

[28] M. Bertalmio, L. Vese, G. Sapiro, and S. Osher, "Simultaneous structure and texture image inpainting," *IEEE transactions on image processing*, vol. 12, no. 8, pp. 882–889, 2003. 4, 86, 88

[29] J. Sun, L. Yuan, J. Jia, and H.-Y. Shum, "Image completion with structure propagation," in *ACM Transactions on Graphics (TOG)*, vol. 24, pp. 861–868, ACM, 2005. 4, 86, 88

[30] H. Huang, K. Yin, M. Gong, D. Lischinski, D. Cohen-Or, U. M. Ascher, and B. Chen, "" mind the gap": tele-registration for structure-driven image completion.," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, pp. 174–1, 2013. 4, 86, 88

[31] S. Hesabi, M. Jamzad, and N. Mahdavi-Amiri, "Structure and texture image inpainting," in *2010 International Conference on Signal and Image Processing*, pp. 119–124, IEEE, 2010. 4, 86

[32] M. Ashikhmin, "Synthesizing natural textures," in *Proceedings of the 2001 symposium on Interactive 3D graphics*, pp. 217–226, Citeseer, 2001. 4, 86

[33] M. Eitz, J. Hays, and M. Alexa, "How do humans sketch objects?," *ACM Transactions on graphics (TOG)*, vol. 31, no. 4, pp. 44–1, 2012. 4, 54, 66

[34] B. Edwards, *Drawing on the Right Side of the Brain: The Definitive, 4th Edition*. Penguin Publishing Group, 2012. 4

[35] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision*, 2015. 4, 8, 79, 98

[36] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *CoRR*, vol. abs/1710.10196, 2017. 4, 45

[37] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 4, 8, 98, 124

[38] C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. Efros, "What makes paris look like paris?," *ACM Transactions on graphics (TOG)*, vol. 31, no. 4, 2012. 4, 8, 98

[39] K. Nazeri, E. Ng, T. Joseph, F. Qureshi, and M. Ebrahimi, "Edgeconnect: Generative image inpainting with adversarial edge learning," *arXiv preprint arXiv:1901.00212*, 2019. 6

[40] "The python programming language." TIOBE Index, available as `https://www.tiobe.com/tiobe-index/python/`. 7

[41] "Automatic differentiation package - autograd." PyTorch, available as `https://pytorch.org/docs/autograd`. 7

[42] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation," in *International Conference on Learning Representations*, 2018. 8, 98, 124

[43] J.-B. Huang, A. Singh, and N. Ahuja, "Single image super-resolution from transformed self-exemplars," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5197–5206, 2015. 8, 124

[44] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016. 9, 10, 13, 17, 22, 25, 27, 35, 36, 44

[45] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012. 9, 22

[46] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001. 9

[47] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010. 9, 31

[48] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal, and V. K. Asari, "The history began from

alexnet: a comprehensive survey on deep learning approaches," *arXiv preprint arXiv:1803.01164*, 2018. 9

[49] C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*. Springer, 2018. 10, 20

[50] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (3rd Edition)*. Pearson, 2009. 10, 12, 13, 15

[51] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006. 13, 28, 89

[52] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of machine learning research*, vol. 11, no. Dec, pp. 3371–3408, 2010. 13

[53] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, ACM, 2008. 13

[54] R. Salakhutdinov and G. Hinton, "Semantic hashing," *International Journal of Approximate Reasoning*, vol. 50, no. 7, pp. 969–978, 2009. 13

[55] A. Krizhevsky and G. E. Hinton, "Using very deep autoencoders for content-based image retrieval.," in *ESANN*, 2011. 13

[56] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?," *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 625–660, 2010. 14, 45

[57] S. Edelkamp and S. Schroedl, *Heuristic search: theory and applications*. Elsevier, 2011. 15

[58] J. Kiefer, J. Wolfowitz, *et al.*, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952. 16

[59] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016. 16

[60] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018. 16

[61] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964. 17

[62] T. Tieleman and G. Hinton, "Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning," *Technical Report.*, 2017. 17

[63] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015. 17, 97

[64] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016. 17

[65] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988. 18

[66] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, p. 3, 2013. 21, 31, 96

[67] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010. 21, 31

[68] A. Karpathy, *Connecting images and natural language*. PhD thesis, Ph. D. thesis, Stanford University, 2016. 22

[69] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, 1998. 22

[70] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. 22, 46, 74, 76, 89, 122

[71] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017. 22

[72] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 4898–4906, 2016. 24, 29

[73] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *ICLR (workshop track)*, 2015. 27, 28

[74] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2014. 28

[75] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015. 28, 44

[76] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks.," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 10, pp. 2528–2535, 2010. 28, 44

[77] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015. 28

[78] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," *arXiv preprint physics/0004057*, 2000. 28

[79] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *European Conference on Computer Vision (ECCV)*, pp. 694–711, Springer, 2016. 28, 35, 46, 74, 78, 91, 93, 94, 96, 121, 125

[80] 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," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2018. 29, 31

[81] 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," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015. 32, 74, 95

[82] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, pp. 448–456, 2015. 32

[83] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 33, 92

[84] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016. 33

[85] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?," in *Advances in Neural Information Processing Systems 31*, pp. 2483–2493, 2018. 34

[86] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014. 34

[87] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *International Conference on Learning Representations (ICLR)*, 2016. 35, 44, 45

[88] I. Goodfellow, "Nips 2016 tutorial: Generative adversarial networks," *arXiv preprint arXiv:1701.00160*, 2016. 35, 36, 40

[89] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 35, 44, 45, 47, 91, 92, 96

[90] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2414–2423, 2016. 35, 46, 75, 77, 78, 93, 94, 95

[91] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4681–4690, 2017. 35, 122

[92] A. V. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1747–1756, PMLR, 2016. 35

[93] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, *et al.*, "Conditional image generation with pixelcnn decoders," in *Advances in neural information processing systems*, pp. 4790–4798, 2016. 35

[94] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley, *Boltzmann machines: Constraint satisfaction networks that learn*. Carnegie-Mellon University, Department of Computer Science Pittsburgh, 1984. 36

[95] G. E. Hinton, T. J. Sejnowski, *et al.*, "Learning and relearning in boltzmann machines," *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1, no. 282-317, p. 2, 1986. 36

[96] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *International Conference on Learning Representations (ICLR)*, 2014. 36

[97] A. Dosovitskiy and T. Brox, "Generating images with perceptual similarity metrics based on deep networks," in *Advances in neural information processing systems*, pp. 658–666, 2016. 36

[98] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018. 40, 44

[99] J. H. Lim and J. C. Ye, "Geometric gan," *arXiv preprint:1705.02894*, 2017. 43

[100] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *International Conference on Learning Representations*, 2018. 43, 47, 93

[101] D. Tran, R. Ranganath, and D. M. Blei, "Deep and hierarchical implicit models," *arXiv preprint arXiv:1702.08896*, vol. 7, 2017. 43

[102] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," *arXiv preprint arXiv:1805.08318*, 2018. 43, 45, 49, 79, 93, 102

[103] C. R. Vogel, *Computational methods for inverse problems*, vol. 23. Siam, 2002. 43

[104] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014. 44, 45, 91

[105] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, p. 5, 2017. 44

[106] J.-Y. Zhu, R. Zhang, D. Pathak, T. Darrell, A. A. Efros, O. Wang, and E. Shechtman, "Toward multimodal image-to-image translation," in *Advances in Neural Information Processing Systems*, pp. 465–476, 2017. 44

[107] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *The IEEE International Conference on Computer Vision (ICCV)*, 2017. 45, 47, 91, 92, 96

[108] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, "High-resolution image synthesis and semantic manipulation with conditional gans," in

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), p. 5, 2018. 45, 46, 93

[109] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017. 45, 93

[110] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, pp. 2234–2242, 2016. 46

[111] L. Gatys, A. S. Ecker, and M. Bethge, "Texture synthesis using convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 262–270, 2015. 46, 74, 77, 78, 93

[112] C. Li and M. Wand, "Precomputed real-time texture synthesis with markovian generative adversarial networks," in *European Conference on Computer Vision*, pp. 702–716, Springer, 2016. 47

[113] Y. Yoshida and T. Miyato, "Spectral norm regularization for improving the generalizability of deep learning," *arXiv preprint arXiv:1705.10941*, 2017. 49

[114] A. Odena, J. Buckman, C. Olsson, T. B. Brown, C. Olah, C. Raffel, and I. Goodfellow, "Is generator conditioning causally related to gan performance?," in *Proceedings of the 35th International Conference on Machine Learning*, 2018. 49, 93

[115] R. C. Gonzalez, *Digital Image Processing (4th Edition)*. Pearson, 2017. 52, 56, 58, 59, 61, 71

[116] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010. 53

[117] D. Marr and E. Hildreth, "Theory of edge detection," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 207, 1980. 54, 61

[118] D. R. Martin, C. C. Fowlkes, and J. Malik, "Learning to detect natural image boundaries using local brightness, color, and texture cues," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 5, pp. 530–549, 2004. 54

[119] S. Xie and Z. Tu, "Holistically-nested edge detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1395–1403, 2015. 55, 67, 90, 111

[120] W. Shen, X. Wang, Y. Wang, X. Bai, and Z. Zhang, "Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3982–3991, 2015. 66

[121] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012. 67

[122] G. Bertasius, J. Shi, and L. Torresani, "Deepedge: A multi-scale bifurcated deep network for top-down contour detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4380–4389, 2015. 67, 90

[123] Y. Liu, M.-M. Cheng, X. Hu, K. Wang, and X. Bai, "Richer convolutional features for edge detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5872–5881, IEEE, 2017. 67, 90

[124] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004. 68, 72, 73, 102

[125] M. J. Crump, J. V. McDonnell, and T. M. Gureckis, "Evaluating amazon's mechanical turk as a tool for experimental behavioral research," *PloS one*, vol. 8, no. 3, p. e57410, 2013. 68

[126] Z. Wang and A. C. Bovik, "Mean squared error: Love it or leave it? a new look at signal fidelity measures," *IEEE signal processing magazine*, vol. 26, no. 1, pp. 98–117, 2009. 69, 70, 72

[127] T. N. Pappas, R. J. Safranek, and J. Chen, "Perceptual criteria for image quality evaluation," *Handbook of image and video processing*, pp. 669–684, 2000. 69, 72

[128] A. M. Eskicioglu, P. S. Fisher, and S.-Y. Chen, "Image quality measures and their performance," 1994. 69, 71

[129] R. S. Hunter, "Photoelectric color difference meter," *Josa*, vol. 48, no. 12, pp. 985–995, 1958. 71

[130] Z. Wang, A. C. Bovik, and L. Lu, "Why is image quality assessment so difficult?," in *Proceedings of International Conference on Acoustics, Speech and Signal Processing (CASSP*, pp. IV–3313, IEEE, 2002. 71, 74

[131] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, vol. 2, pp. 1398–1402, Ieee, 2003. 73

[132] Z. Wang and A. C. Bovik, "Modern image quality assessment," *Synthesis Lectures on Image, Video, and Multimedia Processing*, vol. 2, no. 1, pp. 1–156, 2006. 73

[133] L. A. Gatys, A. S. Ecker, M. Bethge, A. Hertzmann, and E. Shechtman, "Controlling perceptual factors in neural style transfer," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 74

[134] M. W. Gondal, B. Schölkopf, and M. Hirsch, "The unreasonable effectiveness of texture transfer for single image super-resolution," in *Workshop and Challenge on Perceptual Image Restoration and Manipulation (PIRM) at the 15th European Conference on Computer Vision (ECCV)*, 2018. 74, 78, 91

[135] C. Yang, X. Lu, Z. Lin, E. Shechtman, O. Wang, and H. Li, "High-resolution image inpainting using multi-scale neural patch synthesis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 74, 89

[136] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016. 78

[137] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 78

[138] G. T. Fechner, *Elemente der psychophysik*. Leipzig: Breitkopf, 2012. 81

[139] M. Jogan and A. A. Stocker, "A new two-alternative forced choice method for the unbiased characterization of perceptual bias and discriminability," *Journal of Vision*, vol. 14, no. 3, pp. 20–20, 2014. 81

[140] L. L. Thurstone, "A law of comparative judgment.," *Psychological review*, vol. 34, no. 4, p. 273, 1927. 81

[141] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, "Image inpainting," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 417–424, 2000. 87, 88

[142] S. Esedoglu and J. Shen, "Digital inpainting based on the mumford–shah–euler image model," *European Journal of Applied Mathematics*, vol. 13, no. 4, pp. 353–370, 2002. 87, 88

[143] D. Liu, X. Sun, F. Wu, S. Li, and Y.-Q. Zhang, "Image compression with edge-based inpainting," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 10, pp. 1273–1287, 2007. 87

[144] C. Ballester, M. Bertalmio, V. Caselles, G. Sapiro, and J. Verdera, "Filling-in by joint interpolation of vector fields and gray levels," *IEEE transactions on image processing*, vol. 10, no. 8, pp. 1200–1211, 2001. 87, 88

[145] S. Darabi, E. Shechtman, C. Barnes, D. B. Goldman, and P. Sen, "Image melding: Combining inconsistent images using patch-based synthesis.," *ACM Transactions on graphics (TOG)*, vol. 31, no. 4, pp. 82–1, 2012. 87, 88

[146] J.-B. Huang, S. B. Kang, N. Ahuja, and J. Kopf, "Image completion using planar structure guidance," *ACM Transactions on graphics (TOG)*, vol. 33, no. 4, p. 129, 2014. 87, 88

[147] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "Patchmatch: A randomized correspondence algorithm for structural image editing," *ACM Transactions on graphics (TOG)*, vol. 28, no. 3, p. 24, 2009. 88

[148] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," *ACM Transactions on graphics (TOG)*, vol. 22, no. 3, pp. 313–318, 2003. 88

[149] Y. Song, C. Yang, Z. Lin, X. Liu, Q. Huang, H. Li, and C. Jay, "Contextual-based image inpainting: Infer, match, and translate," in *European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018. 90

[150] Y. Song, C. Yang, Y. Shen, P. Wang, Q. Huang, and C. J. Kuo, "Spg-net: Segmentation prediction and guidance network for image inpainting," in *British Machine Vision Conference 2018, BMVC 2018, Northumbria University, Newcastle, UK, September 3-6, 2018*, p. 97, 2018. 90

[151] P. Dollar, Z. Tu, and S. Belongie, "Supervised learning of edges and object boundaries," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 1964–1971, IEEE, 2006. 90

[152] Y. Li, M. Paluri, J. M. Rehg, and P. Dollár, "Unsupervised learning of edges," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1619–1627, 2016. 90

[153] P. Dollár and C. L. Zitnick, "Fast edge detection using structured forests," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 8, pp. 1558–1570, 2015. 90

[154] S. Nowozin, C. H. Lampert, *et al.*, "Structured learning and prediction in computer vision," *Foundations and Trends® in Computer Graphics and Vision*, vol. 6, no. 3–4, pp. 185–365, 2011. 90

[155] P. Sangkloy, J. Lu, C. Fang, F. Yu, and J. Hays, "Scribbler: Controlling deep image synthesis with sketch and color," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, 2017. 91

[156] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts," *Distill*, vol. 1, no. 10, p. e3, 2016. 95, 101, 122

[157] M. Lučić, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, "Are gans created equal? a large-scale study," in *Advances in Neural Information Processing Systems*. 102

[158] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient subpixel convolutional neural network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 119, 121

[159] W. Yang, X. Zhang, Y. Tian, W. Wang, and J.-H. Xue, "Deep learning for single image super-resolution: A brief review," *arXiv preprint arXiv:1808.03344*, 2018. 119

[160] C.-Y. Yang, C. Ma, and M.-H. Yang, "Single-image super-resolution: A benchmark," in *European Conference on Computer Vision*, pp. 372–386, Springer, 2014. 120

[161] C. De Boor, C. De Boor, E.-U. Mathématicien, C. De Boor, and C. De Boor, *A practical guide to splines*, vol. 27. springer-verlag New York, 1978. 121

[162] C. E. Duchon, "Lanczos filtering in one and two dimensions," *Journal of applied meteorology*, vol. 18, no. 8, pp. 1016–1022, 1979. 121

[163] R. Fattal, "Image upsampling via imposed edge statistics," *ACM transactions on graphics (TOG)*, vol. 26, no. 3, p. 95, 2007. 121

[164] J. Sun, Z. Xu, and H.-Y. Shum, "Image super-resolution using gradient profile prior," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, 2008. 121

[165] Q. Shan, Z. Li, J. Jia, and C.-K. Tang, "Fast image/video upsampling," in *ACM Transactions on Graphics (TOG)*, vol. 27, p. 153, ACM, 2008. 121

[166] H. Chang, D.-Y. Yeung, and Y. Xiong, "Super-resolution through neighbor embedding," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1, pp. I–I, IEEE, 2004. 121

[167] W. T. Freeman, T. R. Jones, and E. C. Pasztor, "Example-based super-resolution," *IEEE Computer graphics and Applications*, no. 2, pp. 56–65, 2002. 121

[168] D. G. S. B. M. Irani, "Super-resolution from a single image," in *Proceedings of the IEEE International Conference on Computer Vision, Kyoto, Japan*, 2009. 121

[169] G. Freedman and R. Fattal, "Image and video upscaling from local self-examples," *ACM Transactions on Graphics (TOG)*, vol. 30, no. 2, p. 12, 2011. 121

[170] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *European conference on computer vision*, pp. 184–199, Springer, 2014. 121

[171] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1646–1654, 2016. 121

[172] J. Kim, J. Kwon Lee, and K. Mu Lee, "Deeply-recursive convolutional network for image super-resolution," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1637–1645, 2016. 121

[173] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: nonlinear phenomena*, vol. 60, pp. 259–268, 1992. 122

[174] M. Haris, G. Shakhnarovich, and N. Ukita, "Deep back-projection networks for super-resolution," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1664–1673, 2018. 122

[175] S.-J. Park, H. Son, S. Cho, K.-S. Hong, and S. Lee, "Srfeat: Single image super-resolution with feature discrimination," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 439–455, 2018. 122