Enhancing Visual Perception in Noisy Environments using Generative Adversarial Networks

A THESIS SUBMITTED TO THE FACULTY OF THE UNIVERSITY OF MINNESOTA BY

Cameron Fabbri

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

Junaed Sattar

August 2018
Acknowledgments

I would first like to acknowledge my advisor Professor Junaed Sattar for his continuous support of my research, as well as his patience, guidance, and advice. This thesis would not be possible without his immense help. Additionally, I would like to thank the members of the Interactive Robotics and Vision Laboratory for their inputs as well.

I would also like to thank the members of my committee, Professor Lucy Fortson and Professor Catherine Qi Zhao.

Last but certainly not least, I would like to thank my parents Bill and Laurie Fabbri who gave continuous support throughout my entire collegiate career.
Dedication

I would like to dedicate this thesis to my loving parents, who always pushed me towards higher education. Their continuous support of whichever path I chose has enabled and encouraged me to continue doing what I truly enjoy. Furthermore, setting a good example has taught me to work hard for the things I wish to achieve.
Abstract

Autonomous robots rely on a variety of sensors – acoustic, inertial, and visual – for intelligent decision making. Due to its non-intrusive, passive nature, and high information content, vision is an attractive sensing modality. However, many environments contain natural sources of visual noise such as snow, rain, dust, and other forms of distortion. This work focuses on the underwater environment, in which visual noise is a prominent component. Factors such as light refraction and absorption, suspended particles in the water, and color distortion affect the quality of visual data, resulting in noisy and distorted images. Autonomous Underwater Vehicles (AUVs) that rely on visual sensing thus face difficult challenges, and consequently exhibit poor performance on vision-driven tasks. This thesis proposes a method to improve the quality of visual underwater scenes using Generative Adversarial Networks (GANs), with the goal of improving input to vision-driven behaviors further down the autonomy pipeline. Furthermore, we show how recently proposed methods are able to generate a dataset for the purpose of such underwater image restoration. For any visually-guided underwater robots, this improvement can result in increased safety and reliability through robust visual perception. To that effect, we present quantitative and qualitative data which demonstrates that images corrected through the proposed approach generate more visually appealing images, and also provide increased accuracy for a diver tracking algorithm.
Table of contents

List of tables vi

List of figures vii

1 Introduction 1
   1.1 Image-to-Image Translation 2
   1.2 Interesting Domains 3
   1.3 Application Domain 4
   1.4 Vision-Based Underwater Algorithms 5
   1.5 Contribution 6

2 Background 8
   2.1 Vision-Guided Underwater Robotics 8
   2.2 Aqua Robot 10
   2.3 Deep Learning 12
   2.4 Generative Adversarial Networks 15
   2.5 Image-to-Image Translation 19

3 Underwater Image Enhancement 24
   3.1 Methodology 24
   3.2 Distorter GAN 34
# Table of contents

3.3 Model Inspection ............................................... 38

4 Experiments ......................................................... 40
  4.1 Datasets ......................................................... 40
  4.2 Qualitative Results ........................................... 41
  4.3 Underwater Image Quality Measure .......................... 43
  4.4 Diver Tracking using Frequency-Domain Detection ....... 49
  4.5 Comparing Generative Models ................................. 49
  4.6 Systems ......................................................... 51
  4.7 Results ......................................................... 52

5 Conclusion ......................................................... 62

References .......................................................... 65

Appendix A Additional Examples .................................. 72

Appendix B Network Architectures .................................. 93
List of tables

3.1 UIQM for the images shown in Figure 3.6. A higher UIQM corresponds to a cleaner image. ........................................ 37

4.1 Average UIQM for ImageNet test set. ................................. 46

4.2 Comparison of the average UIQM for Flickr™ test set. .......... 48

4.3 Comparison of average UIQM for the ImageNet test set over all of the UGAN models. A higher UIQM score is better, with UGANG-P with 8 layers scoring the highest. ................................. 54

4.4 Comparison of FPS for different number of layers in UGAN running on different devices. ................................. 56
List of figures

2.1 The Aqua Robot fitted with treaded legs capable of walking on the ocean floor or on ground above water. Shown with the top lid removed for internal structure viewing. ........................................... 10

2.2 A diagram of the Aqua 8 robot displaying main hardware components. . . 11

2.3 A single perceptron. ..................................................... 12

2.4 Examples of activation functions used by neural networks. .............. 14

2.5 A simple setup for a conditional GAN. Here, $x$ would be a real data point such as an image, $y$ would be a class attribute such as a label, and $z$ some random noise sampled from $P_z$. ............................................. 19


3.1 An overview of our data generation process using two domains of images. The first domain contains nondistorted underwater images, and the second contains distorted underwater images. Using CycleGAN, we translate the nondistorted images to appear distorted. Here, the function $F$ represents the generator in CycleGAN. .................................................. 25

3.2 Paired samples of ground truth and distorted images generated by CycleGAN. Top row: Ground truth. Bottom row: Generated samples. ............. 26
3.3 Overview of our method. An image $I^D$ produced by CycleGAN is used as input to the generator $G$, which is structured as a U-Net. The image is then sent to the discriminator $D$. Here, the dashed line represents an $L_1$ loss.

3.4 An overview of the UGAN generator model. Naming scheme is as follows: k4 corresponds to a kernel size of $\times 4$, n64 corresponds to the feature maps having a depth of 64, and s4 corresponds to a stride of 4.

3.5 Interpolation between $I^D$ and $I^C$ to provide additional training data.

3.6 Paired samples of ground truth and distorted images generated by Distorter GAN. Top row: Ground truth. Bottom row: Generated samples.

4.1 Qualitative results on images from our test set. UGAN$_W$ can both recover color and also enhance color if a small amount is present.

4.2 Quantitative examples of the metrics within the UIQM. (a)-(c) shows distorted images with notably lower scores, whereas (d)-(f) show nondistorted images with higher scores.

4.3 Comparisons of UIQM for original and enhanced images. (a) and (d) show two original distorted images from ImageNet [11]. (b) and (e) show the enhanced images using UGAN$_W$, whereas (c) and (f) show the enhanced images using UGAN$_W$-P.

4.4 Local image patches extracted for qualitative comparisons. Each patch was resized to $64 \times 64$, but shown enlarged for viewing ability. Sharpness is better preserved in the UGAN models, and the colors are more vibrant. Best viewed in PDF.

4.5 Performance of MDPM tracker [23] on both real (top row) and generated (second row) images; the Table compares the detection performance for both sets of images over a sequence of 500 frames.
4.6 Qualitative results on images from our test set between using the original GAN objective and the WGAN-GP objective. ........................................ 53
4.7 Comparison of UGAN\textsubscript{G} using a varying number of layers ............... 57
4.8 Comparison of UGAN\textsubscript{G}-P using a varying number of layers ............... 58
4.9 Comparison of UGAN\textsubscript{W} using a varying number of layers ............... 59
4.10 Comparison of UGAN\textsubscript{W}-P using a varying number of layers ............... 60
4.11 Comparison of all versions of UGAN with varying layers. Minor alterations between methods can be seen easiest around objects in the image such as the diver standing in the front. With UGAN-4, a grid effect can be seen whereas this is not as apparent in UGAN-16. ........................................ 61
A.1 Samples from UGAN with 16 layers using the Wasserstein metric. .............. 73
A.2 Samples from UGAN with 16 layers using the GAN metric. .................... 74
A.3 Samples from UGAN with 12 layers using the Wasserstein metric. .............. 75
A.4 Samples from UGAN with 12 layers using the GAN metric. .................... 76
A.5 Samples from UGAN with 10 layers using the Wasserstein metric. .............. 77
A.6 Samples from UGAN with 10 layers using the GAN metric. .................... 78
A.7 Samples from UGAN with 8 layers using the Wasserstein metric. .............. 79
A.8 Samples from UGAN with 8 layers using the GAN metric. .................... 80
A.9 Samples from UGAN with 4 layers using the Wasserstein metric. .............. 81
A.10 Samples from UGAN with 4 layers using the GAN metric. .................... 82
A.11 Samples from UGAN with 16 layers using the Wasserstein metric and Gradient Difference Loss. ........................................ 83
A.12 Samples from UGAN with 16 layers using the GAN metric and Gradient Difference Loss. ........................................ 84
A.13 Samples from UGAN with 12 layers using the Wasserstein metric and Gradient Difference Loss. ........................................ 85
A.14 Samples from UGAN with 12 layers using the GAN metric and Gradient Difference Loss. 86
A.15 Samples from UGAN with 10 layers using the Wasserstein metric and Gradient Difference Loss. 87
A.16 Samples from UGAN with 10 layers using the GAN metric and Gradient Difference Loss. 88
A.17 Samples from UGAN with 8 layers using the Wasserstein metric and Gradient Difference Loss. 89
A.18 Samples from UGAN with 8 layers using the GAN metric and Gradient Difference Loss. 90
A.19 Samples from UGAN with 4 layers using the Wasserstein metric and Gradient Difference Loss. 91
A.20 Samples from UGAN with 4 layers using the GAN metric and Gradient Difference Loss. 92
Chapter 1

Introduction

Many environments, whether natural or artificial, exhibit visual imagery that may inherently belong to multiple domains. An image is said to belong to a particular domain if it exhibits the properties defined in that domain. Often times, there exists an underlying relationship between two or more domains which we intend to capture. A color image may be represented as a grayscale image or edge map, each a separate domain. Translating between these domains using some mapping function is known as image-to-image translation and lies at the core of many image processing challenges such as edge detection [9], image editing [53], super resolution [34], and colorization [80]. Often times images are only available in one domain, posing an additional level of complexity for image-to-image translation. This work focuses on this class of domain transfer, addressing issues vision-based tasks encounter in noise-filled environments, and proposes an image-to-image translation solution to map these noise-filled images to a cleaner domain. The goal is to improve vision-based tasks further down the autonomy pipeline. While applicable to many domains, the direct application shown aims to enhance perception in underwater mobile robots, where vision is especially limited. This introduction serves to provide the motivation needed to set up the problem.
1.1 Image-to-Image Translation

The task of image-to-image translation is concerned with capturing certain properties and features of an image that allow it to be represented in multiple different, but similar domains. In the case of two domains, the result is a mapping from domain $\mathcal{X}$ to $\mathcal{Y}$, defined as a function $F : \mathcal{X} \rightarrow \mathcal{Y}$. Many of these tasks, such as colorization and super-resolution, are inherently ill-posed, as they require the recovery of lost or missing information. Colorization can be informally defined as computing a three-dimensional RGB vector given a single grayscale value at each pixel. Because a single grayscale value may correctly map to multiple color values, this one-to-many mapping must be able to recover color information that has been lost. Super-resolution, on the other hand, aims to increase the spatial resolution and recover fine details of an image. Just as in colorization, this involves the hallucination of information that has been lost. Image-to-image translation problems can be broken down into two main categories: supervised and unsupervised. In the supervised setting, pairs of images belonging to both domains are available (e.g., a color image can be converted to grayscale to form a pair). In the unsupervised setting, only independent sets of images in multiple domains without any paired samples are available. While this approach makes it easier to obtain data, this proves to be a more difficult problem due to the lack of any ground truth. When searching for a mapping function $F$ through some learned process, ground truth allows for a well-defined objective function, whereas the lack of ground truth creates a scenario in which the translation may not be well-defined.

The goal of unsupervised image-to-image translation in a probabilistic sense is to learn a joint probability distribution over images coming from multiple domains. Given two image domains $\mathcal{X}_1$ and $\mathcal{X}_2$, this is done by sampling images from their marginal distributions $P_{\mathcal{X}_1}(x_1)$ and $P_{\mathcal{X}_2}(x_2)$. With these marginal distributions, the Coupling Theory [39] states that there are an infinite amount of joint distributions that can be inferred, making this problem very ill-posed. On the other hand, the problem of supervised image-to-image translation
given samples \((x_1, x_2)\) from an existing joint distribution \(P(\mathcal{X}_1, \mathcal{X}_2)\) is much simpler due to the task being to only learn a transition function \(f : x_1 \rightarrow x_2\). This work takes advantage of the specified task at hand: translating an image from a noisy domain to its corresponding clean domain. Many unsupervised image-to-image translation problems involve domains in which it is useful to transition between domains (\(i.e. \), \(\mathcal{X} \rightarrow \mathcal{Y}\) and \(\mathcal{Y} \rightarrow \mathcal{X}\)), rather than only needing to solve a one-way mapping (\(i.e.\) only \(\mathcal{X} \rightarrow \mathcal{Y}\)). Learning a transition between domains is seen as a more difficult problem due to the infinite joints issue [39], as well as the need to find a more complex function \(F\) that is able to perform multiple domain translations. Given we are only interested in finding a one-way mapping, we no longer have a direct need to solve for the joint distribution, only approximate it in order to provide paired samples to learn \(f\).

### 1.2 Interesting Domains

Image-to-image translation covers a wide variety of domains. Many of these can be thought of as style transfer between domains (\(e.g.\), day to night, photo to painting, etc.), while others can be thought of as information retrieval (\(e.g.\), grayscale to color, low resolution to high resolution, etc.). Inspecting these differences more closely allows one to extract useful properties each problem poses, providing disparate approaches.

Style transfer is almost always unsupervised, due to the fact that large image datasets often only contain images in a single style. This also results in a lack of ground truth, giving rise to unsupervised image-to-image translation methods. Furthermore, in many cases it is interesting to transfer between domains, rather than just a one-way mapping. For example, it is intriguing to translate a photo to a painting, but it also may be just as intriguing to translate a painting to a photo.

Information retrieval problems, which aim to generate information that has been lost due to a process such as image compression, are most often associated with ground truth pairs of
images. Many of these problems have the ability to generate associating image pairs by way of some well defined mapping function. For example, converting color images to grayscale, extracting edges from an image, or lowering the resolution of an image. In all of these cases, we are given a ground truth image and create a distorted pair by some well-defined (often linear) function, then learn the non-linear mapping back to the original image. Information retrieval problems differ from style transfer in that we are only interested in a one-way mapping.

We explore domains in which a one-way mapping is desired, but for which we have no ground truth as well as no known function available to generate ground truth. These domains include natural and man-made environments which inherently exhibit some sort of visual noise, such as rain or snow during bad weather conditions, or the use of cameras outside the visible spectrum, such as infrared or night vision. While these sorts of cameras can increase the information portrayed, it still may be desirable to view the image in the visible spectrum. The domain focus here is the natural underwater environment, which experiences a non-linear distortion measure due to many factors discussed in Section 1.4.

Given the unique situation of being unable to easily generate ground truth in order to solve a one-way mapping, we learn a nonlinear distorting function by training a neural network in order to generate image pairs. This allows us to approach the information retrieval problem without fully approaching the problem of solving for a joint distribution. While this relaxation does not fully incorporate all possible distortions due to the problem of infinite joints [39], this paired image-to-image translation problem is superior to unpaired image-to-image translation methods which we show in Chapter 4.

1.3 Application Domain

Domains experienced by outdoor mobile robots, for which vision is often a primary sensor, are considered here. Specifically, this work focuses on the underwater domain, which poses
1.4 Vision-Based Underwater Algorithms

unique challenges not seen in other outdoor environments. Light refraction, absorption, and scattering from suspended particles can greatly affect optics. Due to the absorption of red wavelengths by the water, images tend to have a blue or green hue. This effect worsens as one goes deeper and more red light is absorbed. This distortion is extremely non-linear in nature, and is affected by a large number of factors, such as the amount of light present, the amount of particles in the water, the time of day, and the camera being used. This distortion may cause difficulty in vision-based tasks such as segmentation, tracking, or classification due to their indirect or direct use of color. Furthermore, visual inspection by humans becomes more difficult as the quality of the images degrades.

As with terrestrial environments, there is high visual variation within the underwater domain, but to a greater degree. For this reason, vision-based algorithms need to be generalizable in order to work within the various depths and environments in which an underwater mobile robot may operate. When using learning-based methods, especially deep learning [33], large volumes of data across many domains are needed in order for algorithms to perform well. Due to the high cost and difficulty of acquiring data across many underwater domains, vision-based algorithms often perform poorly. A step towards mitigating this issue is to be able to enhance and restore underwater images in such a way that they appear to be above water, i.e., with noise removed and colors enhanced. With this, instead of attempting to generalize across a wide variety of noisy and potentially incomplete domains, vision-based algorithms can focus on just one clean domain by removing underwater visual artifacts.

1.4 Vision-Based Underwater Algorithms

Vision-based algorithms for mobile robots face unique challenges due to the limited hardware that is often available. Many algorithms must be able to run in real-time in order to reliably perform the task at hand. Furthermore, the variability across domains can change drastically during a single experiment due to dynamic environments, especially in the underwater
domain. It may, therefore, seem imprudent to use vision, and indeed many underwater techniques opt to use sonar instead [35, 5]. However, vision is an attractive option because of its non-intrusive, passive, and energy efficient nature. Furthermore, it is the primary sensing tool used by humans, therefore making it a very natural choice when performing any sort of inspection, tracking, or monitoring. There exists a large area of research in surveying and monitoring marine environments [77, 67, 20] which may not make use of any image processing algorithms and serve instead to capture data to be manually inspected or viewed, providing another reason for the necessity of vision. On the other hand, recent methods show autonomous algorithms are able to analyze and monitor various types of underwater domains [42]. In both cases, the distortion exhibited by the images causes difficulties in expert analysis as well as post processing done by various algorithms.

Many classical computer vision algorithms have been replaced by deep learning models which are able to learn general representations given large enough datasets. While often superior in accuracy, they require much more computational power which may not be available on some mobile robots. Still, these methods have been shown to run in real time on specialized hardware such as the NVIDIA Jetson TK1, as shown in a tracking by detection method [68]. We describe these methods in greater detail in Section 2.

The methods explored thus far in the underwater domain do not address the extreme distortion that is prevalent in virtually every underwater environment being surveyed or worked in. Lacking an active measure to remove these effects, they are simply ignored and assumed to be existent in every environment, causing visual algorithms to suffer. Rather than merely accepting this difficulty, this research leverages deep learning techniques which are able to remove distortion and enhance the visual quality of images in these domains.

1.5 Contribution

This thesis provides the following contributions:
1.5 Contribution

- A general method for approaching the problem of enhancing and improving images in a noisy domain without any ground truth

- An image-to-image translation method able to map various sub-domains across several underwater environments and depths to one clean domain

- A qualitative improvement in images for manual visual inspection done by humans as well as quantitative improvements in image processing algorithms run on Autonomous Underwater Vehicles (AUVs), as seen in Section 5

The rest of this thesis is organized as follows. Chapter 2 gives an overview of related work, as well as a background in generative models and underwater image processing. Chapter 3 gives an overview of the methodology. Chapter 4 provides experiments and an evaluation of our model. Conclusions and future work is discussed in Chapter 5. Finally, Appendix A provides additional experiments, and Appendix B provides further details for the models used throughout this work.
Chapter 2

Background

The fields of mobile robotics and machine learning have been steadily growing over the past few years. Both hardware and software advancements on both fronts have greatly accelerated their growth. For example, the Robot Operating System (ROS) [56] allows for a unified way of building applications for deployment on a wide variety of robots, reducing the time from idea to implementation. On the machine learning side, the development of the Convolutional Neural Network (CNN) [32, 28] along with advancements in hardware such as GPUs greatly improved results across many image processing tasks. Here, we take a look at the relevant advancements and state of the art methods, as well as give a background on the approach used in this work. For field experiments, the Aqua 8 amphibious robot [12] is employed for bench testing.

2.1 Vision-Guided Underwater Robotics

The use of vision in an underwater domain is not straightforward due to the many unique challenges at hand. Compared with open air environments, there exists a much greater amount of visual distortion underwater due to many factors, of which we will describe the most important here. Light refraction through the water’s surface, as governed by Snell’s
Law [29], causes natural underwater lighting to appear to come from a cone above the scene, known as Snell’s Window or the optical manhole. The red wavelengths coming from this light source are quickly absorbed by the water, giving the environment a blue/green hue. This effect quickly escalates as one goes deeper, and more red wavelengths are absorbed. An additional source of distortion comes from particles suspended in the water, which may also affect the scattering of light.

Prior to vision, methods such as sonar or other forms of specialized hardware were used [35, 18]. Many algorithms accept that the distortion is prevalent in virtually every underwater environment, and model their algorithms around it [20]. Color-based tracking algorithms have been employed, although may suffer due to the blue/green hue that distorts and obscures the true color of many objects. A study of three color-based tracking methods has been done for the underwater environment [62] in various lighting and visibility conditions.

There have been numerous approaches towards directly modeling the physical effect of underwater distortion. Physics-based models to approximate this distortion have been developed with the aim of reversing it [65]. One issue these model-based approaches face is the differences arising between various underwater environments, as the physical properties changes throughout. Fusion-based methods for enhancing underwater images and videos have also been studied [2], which enhance the contrast and degraded colors. The work of [49] gives a performance evaluation on a stereo based system for object detection. In this work, they note that a vital step in the image-processing pipeline is to improve the image quality. They use the lightness channel $L$ in the CIELAB color space [52] as input to a contrast based method, which sharpens the image. In order to correct color, they use a color histogram equalizer. An energy minimization formulation was explored by using a Markov Random Field to represent the relationship between color and color-depleted [71].

While these methods are able to enhance vision to a certain degree, due to the environment specifics they were presented in, they are unable to generalize to the variety of environments
a mobile robot may encounter in various underwater scenes. Recently, there have been several deep learning-based approaches that are concurrent with this work. WaterGAN [38] uses Generative Adversarial Networks in order to generate underwater images from in-air images, then uses a pixel-wise loss metric to learn the reverse mapping in order to correct color. The work of [36] uses a weakly supervised color transfer method in order to correct color distortion. An attractive property of their approach is the lack of a need for paired samples. Deep learning-based approaches have the advantage of being able to learn from large amounts of data, therefore not restricting them to a single environment. Due to this very attractive property, they have largely been able to outperform previous approaches qualitatively, while also generalizing to a more diverse set of environments.

2.2 Aqua Robot

The Aqua 8 “Minnebot” [12, 63] is a 6-legged amphibious robot capable of both swimming and walking motions. Using six flippers for propulsion, it is able to operate in 5 degrees of freedom underwater: surge, heave, pitch, roll, and yaw. During ground locomotion, flippers can be easily interchanged with legs for use of walking on the shore or sea bottom, as seen

![Aqua Robot](image_url)

Fig. 2.1 The Aqua Robot fitted with treaded legs capable of walking on the ocean floor or on ground above water. Shown with the top lid removed for internal structure viewing.
in Figure 2.1 (image courtesy of [30]). Aqua has been tested for diving up to a depth of 40 meters with two on-board Lithium-ion batteries providing power for up to six hours. There are three cameras on-board: stereo vision on the front, and monocular vision on the back. Information passing through these cameras is handled by one of the two on-board computers, namely the *vision stack*. The other on-board computer, the *control stack*, is responsible for handling motor commands controlling the robot either teleoperated or autonomously.

An ongoing modification to Aqua deals with the addition of the NVIDIA Jetson TX2 embedded Artificial Intelligence (AI) computing device. This chip allows for complex probabilistic graphs involving parallelized operations, such as deep neural networks, to run in real time. For direct control and data collection purposes, Aqua is equipped with a fiber optic tether connection. When this tether is not in use, Aqua is completely autonomous and is able to respond via visual cues such as hand gestures [24] or fiducial cards [64]. In addition to vision, Aqua is also fitted with an Inertial Measurement Unit (IMU) and a depth sensor. Figure 2.2 (courtesy of [30]) displays the Aqua Robot equipped with fins for underwater exploration, as well as physical specifications.
2.3 Deep Learning

Artificial Neural Networks (ANNs), also known as feedforward neural networks or multilayer perceptrons (MLP), are sequential function compositions loosely inspired by the brain. ANNs belong to the class of representation learning techniques, which allows them to be fed raw data in order to discover hidden representations as opposed to relying on hand-crafted features. A core building block of these networks is the perceptron, as seen in Figure 2.3. A perceptron (often called a node) computes the weighted sum of its inputs (plus a bias) and applies a nonlinear activation function to its value. Stacking these perceptrons results in what is commonly known as a layer. A network is composed of multiple layers, resulting in an alternating sequence of linear and nonlinear functions. The class of neural networks that employ the use of many hidden layers has become known as deep learning [33].

![Fig. 2.3 A single perceptron.](image)

Deep learning has enjoyed much success across a wide variety of problems including image classification [28], object detection [58], speech recognition [6], and reinforcement learning [47]. These networks are trained via backpropagation [60] in order to optimize a given objective function. Backpropagation is used in conjunction with an optimization method, such as gradient descent, to propagate the error back through the network from output to input in order to learn the set of weights and biases.
Convolutional Neural Networks

The most common type of neural network for visual data is the Convolutional Neural Network (CNN) [32], which is designed specifically for multidimensional data with spatially correlated features. CNNs incorporate three powerful techniques in order to provide scalability, as well as some degree of scale and shift invariance. The first is the use of shared weights, which stems from the idea that a feature detector used in one part of an image is almost certainly useful in other parts of the image. Furthermore, this allows networks to reduce the number of parameters, avoiding the curse of dimensionality which arises in high dimensional spaces. The second is the use of local receptive fields. A kernel (often called a filter or sliding window) is convolved across the entire image to produce a feature map. Each pixel in the resulting feature map is the result of the kernel convolved with a small area in the input. The number of pixels the kernel slides over as it is convolved with the image is known as the stride. The use of local receptive fields allows earlier layers in the network to learn low-level features such as edges or corners, which are then combined in successive layers throughout the network to learn high-level features. The third technique is various forms of subsampling. Convolutions with a stride greater than 1 inherently subsample, as the resulting output is of a lower dimensionality from the input. Other subsampling techniques, such as the use of pooling layers, provide a strict form of linear downsampling. The most common technique, maxpooling, computes the max of an \( n \times n \) region of a feature map. Maxpooling is very effective by acting as a form of regularization due to harshly cutting out information. This forces the network to become invariant to this missing information.

Nonlinearities

In order for neural networks to learn to complex real-world functions, a form of nonlinearity is needed. This is because many real world problems have a solution that is much more complicated than simply a linear combination of its inputs. Historically, smooth nonlinear
functions such as sigmoid($x$) or tanh($x$) were employed, but it has been shown that the ReLU \[48\] and its leaky alternative are able to learn much faster in large networks. The ReLU activation function, seen in Figure 2.4b, is piecewise linear, allowing it to enjoy many of the properties that make linear models easy to optimize with gradient-based optimization methods \[15\].

![Sigmoid](image1) ![ReLU](image2) ![Leaky ReLU](image3)

Fig. 2.4 Examples of activation functions used by neural networks.

**Autoencoders** Some neural networks are designed specifically for learning efficient embeddings of data. These networks are known as *autoencoders* because they are able to automatically encode data to some lower dimensional representation in an unsupervised manner. Previous methods that do not require learning include linear techniques such as PCA \[78\]. Autoencoders have two main components: an encoder and a decoder. Given some input $x \in \mathbb{R}^d$, an encoder, which is comprised of several layers, maps $x$ to a latent embedding $z \in \mathbb{R}^k$ with $k < d$. A decoder attempts to reconstruct the original input $x$ given the embedding, $z$, such that some loss is minimized, such as the squared $L_2$ loss: $l = ||x - \text{dec}(\text{enc}(x))||_2^2$, with $\text{enc}$ and $\text{dec}$ being an encoder and decoder respectively represented by neural networks. Autoencoders are not required to reconstruct the input exactly. For example, given a color image $I^C$ and its grayscale pair $I^G$, and
2.4 Generative Adversarial Networks

Generative Adversarial Networks (GANs) [14] are a recent class of generative models able to model high dimensional probability distributions such as images. They are the current state of the art in tasks such as image generation and image-to-image translation. Therefore, they are an attractive option for our task. GANs represent a class of generative models in which a generator network $G$ competes against an adversarial network $D$. The generator is given some noise input $z$ and produces synthetic data which actively attempts to fool the discriminator network, which is trained to discriminate between the true data and data from the generator. Formally, the objective between $G$ and $D$ is defined as:

$$\min_G \max_D \mathbb{E}_{x \sim P_r}[\log D(x)] + \mathbb{E}_{z \sim P_z}[\log (1 - D(G(z)))]$$  \hspace{1cm} (2.4)$$

where $P_r$ represents the true data distribution, and $P_z$ represents a prior on the noise distribution as input to the generator, e.g., $z \sim \mathcal{N}(-1, 1)$. We could also explicitly write the second half of Eq. 2.4 in terms of the generator distribution $P_g$, where $\hat{x} \sim P_g, \hat{x} = G(z)$ and $z \sim P_z$: $\mathbb{E}_{\hat{x} \sim P_g}[\log (1 - D(\hat{x}))]$.

\[\begin{align*}
    z &= \text{enc}(I^G) \hspace{1cm} (2.1) \\
    I^C &= \text{dec}(z) \hspace{1cm} (2.2) \\
    l &= ||I^C - I^T||_2^2 \hspace{1cm} (2.3)
\end{align*}\]

displays a setup in which we use an autoencoder to learn how to perform colorization. This work uses an autoencoder framework which is further explained in Chapter 3.
Under a perfect discriminator, minimizing Eq. 2.4 amounts to minimizing the Jensen-Shannon divergence (JSD) between $P_r$ and $P_g$ [14], defined as

$$JSD(P_r||P_g) = \frac{1}{2} KL(P_r||P_a) + \frac{1}{2} KL(P_g||P_a),$$

(2.5)

where $P_a$ is the average of the two distributions, $P_a = \frac{P_r + P_g}{2}$, and $KL$ represents the Kullback-Leibler divergence. In practice, however, as the discriminator improves, gradients to the generator get worse [3]. This problem of the vanishing gradient was addressed by [14], who noted that because minimizing $\log(1 - G(z))$ saturates early on in training, we can instead train $G$ to maximize $\log(D(G(z)))$. While this helped training early on, the same issue of updates increasingly getting worse remained. Due to this, significant amounts of research have been put into efficient ways to measure the divergence between $P_r$ and $P_g$ [43, 81, 8, 3, 4, 16, 54, 46, 76]. This work highlights the Wasserstein GAN method which was found to provide the most stability in training for our specific applications.

**Wasserstein GAN**

The work of [4] studies various divergence measures and their applicability towards GANs. They show that even in simple scenarios, the JSD does not provide useful gradients. On the other hand, the Earth Mover (EM) distance contains more attractive properties and, under mild assumptions, is continuous and differentiable almost everywhere [4]. The EM distance can be informally defined as the minimum cost of transporting mass from one distribution $q$ to another distribution $p$. Formally, this is defined as:

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y) \sim \gamma} [||x - y||]$$

(2.6)

where $\Pi(P_r, P_g)$ denotes the set of joint distributions $\gamma(x,y)$ whose marginals are $P_r$ and $P_g$, respectively [4]. Because the infimum is highly intractable and cannot be computed in a
2.4 Generative Adversarial Networks

A reasonable amount of time, [4] instead proposes to approximate $W$ given a set of $K$-Lipschitz functions $f$ by solving the following:

$$\min_G \max_D \mathbb{E}_{x \sim P_r} [D(x)] - \mathbb{E}_{z \sim p(z)} [D(G(z))]$$  \hspace{1cm} (2.7)$$

where $\mathcal{D}$ is the set of all 1-Lipschitz functions. The discriminator $D$ is not surprisingly modeled as a neural network. Due to the discriminator instead giving a score to real and generated samples and no longer classifying, it is called a critic. However, in the scope of this paper, the two terms are used interchangeably. To ensure $D$ is 1-Lipschitz, [4] proposes to clamp the weights to some range $[-c, c]$ after each gradient update. Two interesting properties come from this formulation that differ from the original GAN formulation. The first is that the loss reported by the generator and discriminator in WGANs correlate well with image quality, which is very important for analyzing convergence. The second, is unlike the original GAN where the updates to the generator get worse as the discriminator gets better [3], WGAN actually relies on a perfect discriminator. For this reason, the gradients of the discriminator are updated $n$ times for every 1 update to the generator ($n = 5$ is common in practice). Although this provides stability in training, weight clamping and multiple updates to the discriminator results in very long training times.

**Improved Wasserstein GANs**

The work of [16] showed that the weight clipping method used by [4] in order to force the critic to lie in the space of 1-Lipschitz functions leads to capacity underuse as well as pathological behavior. In order to enforce the Lipschitz constraint without clipping the weights of the discriminator, [16] instead penalizes the norm of the gradient with respect to its inputs. This formulation is known as Wasserstein GAN with Gradient Penalty (WGAN-GP). A differentiable function is 1-Lipschitz if and only if it has gradients with a norm less than or
equal to 1 everywhere. With this, a new objective function is defined in order to constrain the norm of the critic with respect to its inputs:

$$\min_G \max_{D \in \mathcal{D}} \mathbb{E}_{z \sim P_z}[D(G(z))] - \mathbb{E}_{x \sim P_r}[D(x)] + \lambda \mathbb{E}_{\hat{x} \sim P_{\hat{x}}}[\left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\right)^2]$$  (2.8)

where $P_{\hat{x}}$ is defined as sampling uniformly along straight lines between pairs of sampled from the true data distribution, $P_r$, and the distribution assumed by the generator, $P_g = G(z)$. This is motivated by the intractability of enforcing the unit gradient norm constraint everywhere. Because the optimal discriminator consists of straight lines connecting the two distributions (see [16] for more details), the constraint is enforced uniformly along these lines. As with the original WGAN, the discriminator is updated $n$ times for every 1 update of the generator.

**Conditional GANs**

While normal GANs give the user no control over the output, the work of [45] showed that the generator could be conditioned on some additional information, such as a class label, in order to restrict the output. These are known as conditional GANs (cGAN). In the simplest form, this is done by concatenating an additional vector containing some extra information onto $z$ before sending the latent code to $G$ as seen in Figure 2.5. In order for the discriminator to provide useful gradients back to the generator, this information must also be presented to it as well. The discriminator now has two jobs: determining if an image is real or generated and determining if the image and corresponding attribute are realistic together.

cGANs have a wide variety of possible applications, as the conditional information is not limited to a class label. This additional information could be anything from a label, continuous value, or even an image. Conditioning GANs on images has opened up the field to image-to-image translation and is the base for our proposed method.
Fig. 2.5 A simple setup for a conditional GAN. Here, \( x \) would be a real data point such as an image, \( y \) would be a class attribute such as a label, and \( z \) some random noise sampled from \( P_z \).

2.5 Image-to-Image Translation

As mentioned in Section 1.1, image-to-image translation is the task of translating an image from one domain to another. Here, we provide further details, as well as some previous and current work for underwater colorization. There have been a large number of image-to-image translation works within the past few years, both supervised and unsupervised [25, 82, 21, 70, 40, 75, 41, 83, 72]. We put a focus on two of these which were utilized in this work and note that the use of other methods are left open as future work.

Pix2Pix

The work of [25] provided a network architecture dubbed “pix2pix” which advertised itself as a general architecture for image-to-image translation problems. This work relies on supervised training data, meaning that image pairs in both domains are needed. Demonstrated in Figure 2.6 are synthesized images from edge maps, “day to night” translations, and the colorization of black and white images.
2.5 Image-to-Image Translation

Fig. 2.6 Example results of the Pix2Pix network [25] for general image-to-image translation tasks. Image courtesy of [25].

It is known that simply minimizing the Euclidean distance between images produces blurry results due to the averaging over pixels [51]. Designing a loss function in order for image-to-image translation problems to output sharp and realistic outputs is still an open problem. However, instead of hand-crafting a loss function, [25] proposes to use the discriminator from a GAN to reject non-realistic images. Due to the fact that this method relies on training pairs being available, the final objective function comes from the combination of a Euclidean loss and a GAN loss. In order to preserve low-level structure, $L_1$ is used:

$$\mathcal{L}_{L_1}(G) = \mathbb{E}_{y \sim \mathcal{P}_r, x \sim \mathcal{P}_s} [||y - G(x)||_1]$$

(2.9)

where $y$ is the ground truth image in the target domain $r_t$, and $x$ is the input from the source domain $r_s$. While this loss captures low-level frequencies, using $L_1$ alone results in blurry images. However, blurry images will be rejected by the discriminator in a GAN due to them not looking realistic. The objective for the conditional GAN is

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{y \sim \mathcal{P}_r} [\log D(y)] + \mathbb{E}_{x \sim \mathcal{P}_s} [\log (1 - D(G(x)))]$$

(2.10)

which gives the final objective function
\[ G^* = \min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G). \]  

(2.11)

This combination of low level structure captured in the \( L_1 \) loss along with the high level structure capture in the \( cGAN \) loss allows for the generation of very realistic and high quality images that do not suffer from artifacts such as blurriness.

The architecture used in [25] has two important properties which allow it to work very well for image-to-image translation tasks. The first is the U-net design [59] which allows for structural similarity in the input and output. Encoder-decoder networks downsample (encode) the input via convolutions to a lower dimensional embedding, which is then upsampled (decode) via transpose convolutions or other form of upsampling to reconstruct an image. The advantage of using a “U-Net” comes from explicitly preserving spatial dependencies produced by the encoder, as opposed to relying on the embedding to contain all of the information. This is done by the addition of “skip connections”, which concatenate the activations produced from a convolution layer \( i \) in the encoder to the input of a transpose convolution layer \( n-i+1 \) in the decoder, where \( n \) is the total number of layers in the network.

The second addition designs the discriminator network in order to specifically model high-level frequencies, since the \( L_1 \) loss (Eq. 2.9) takes care of the low-level frequencies. The discriminator is modeled as a PatchGAN [25, 37], which discriminates at the level of image patches. As opposed to a regular discriminator, which outputs a scalar value corresponding to real or fake, a PatchGAN discriminator outputs a \( n \times n \) feature matrix providing a metric for high-level frequencies.

**CycleGAN**

Forgoing the need for paired training data, CycleGAN [82] instead aims to perform image-to-image translation by enforcing a cycle consistency loss. Given two domains, \( D_1 \) and \( D_2 \), CycleGAN trains two generators \( G_1 \) and \( G_2 \) such that an image \( X \in D_1, Y \in D_2, G_1(X) \in D_2 \)
and $G_2(Y) \in D_1$. To ensure that the translated image still captures the same properties of the original image, the cycle consistency loss enforces that $G_2(G_1(X)) \approx X$, and similarly for $Y$.

The network architecture used in CycleGAN comes from the work of [26], which showed impressive results on super-resolution and style-transfer. Like Pix2Pix, they also employ a PatchGAN discriminator. We take advantage of the fact that CycleGAN does not need training pairs in order to use it as a *distorter* network to create ourselves a dataset. While we acknowledge that CycleGAN inherently learns to map distorted images to non-distorted images, we argue in favor of our method due to paired image-to-image translation problems being easier than unpaired problems. Further experiments and comparisons to CycleGAN can be found in Chapter 4.

**Underwater Image Enhancement**

Specifically towards underwater image enhancement, a few methods have run parallel to this work. WaterGAN [38] has a very similar approach in that they generate a paired dataset using a GAN. Their generator network is comprised of three components: 1) Attenuation, which models the range-dependent attenuation of light particles. 2) Scattering, which models the hazing effect common in underwater environments due to the back scattering of photons. 3) Vignetting, which causes shading near the edges of images, and is also related to Snell’s Window. To capture range-dependency, a depth map computed using a Microsoft Kinect is used. After WaterGAN has been trained, a Euclidean loss is used to perform color correction. This loss has been shown to produce blurry results [80], which we overcome by using an adversarial loss. Additionally, our method does not require depth information during training time, only images of objects in two separate domains throughout the entire process.

The work of [10] also creates a paired dataset in order to train a GAN. They propose a filtering-based restoration scheme (FRS) in order to create ground truth pairs from distorted underwater images. This can be broken down into two steps: 1) Filtering Method, which
2.5 Image-to-Image Translation

captures the distortion of underwater images through absorption, forward scattering, and back scattering. A degeneration model is used to remove haziness caused from scattering, which is then processed by a Wiener filter in order to remove noise. 2) Filtering-Based Color Correction: The output of the Wiener filter provides a form of color correction by removing low-frequency information. The Least Squares GAN [43] approach is used in place of the vanilla GAN loss. Besides fooling the discriminator, their objective function also includes a Euclidean loss between the darkest channel for each pixel, inspired by [19]. Additionally, a novel underwater index loss is employed which aims to pull the distribution of colors in the LAB color-space towards that of in-air images.

Finally, the work of [36] showed that they could improve the quality of underwater images by using a weakly supervised color transfer method that shifts the color of the image as if it was taken in-air. This method has the advantage of being unpaired. The goal of color transfer methods is to learn a mapping between two domains such that the characteristics representing color information can be translated. Similar to that of [82], a cycle consistency loss is employed to retain content information.

GANs are notoriously difficult to train in practice. Often either the discriminator or generator will outperform the other, leading to the models diverging. One distinction our method makes from those mentioned is the use of the Wasserstein metric, which greatly stabilizes training and provides a view of the models’ convergence. Although this comes at the cost of a longer training time, the results argue in favor of that sacrifice. The next chapter goes into detail of our method, which is followed by experiments.
Chapter 3

Underwater Image Enhancement

Image transformations can be applied at various granularities, spanning from per pixel editing to applying filters over an image as a whole. Many of these transformations are linear in nature, applying a simple function over the entire image (e.g., color \(\rightarrow\) grayscale). The distortion exhibited in various underwater domains can vary greatly, depending on weather conditions, time of day, depth, and the environment itself. Due to these factors, the distortion present in underwater domains is extremely nonlinear in nature. Therefore, simple methods such as adding hue to an image do not capture all of the dependencies, and a more expressive nonlinear method is desired.

3.1 Methodology

Underwater images distorted by lighting or other forms of distortion inherently lack ground truth, which is required for previous reference-based colorization or noise removal approaches. Additionally, the distortion exhibited is extremely nonlinear in nature as shown by complicated physics-based approaches attempting to model it. Therefore, simply adding a blue or green hue to a clean image in order to create a training pair does not capture all of the dependencies we would like it to as it would not account for other sources of distortion. To-
Dataset Generation

The first domain contains nondistorted underwater images, and the second contains distorted underwater images. Using CycleGAN, we translate the nondistorted images to appear distorted. Here, the function \( F \) represents the generator in CycleGAN.

Towards this end, we use an unpaired image-to-image translation model as a distortion method in order to generate paired images for training. While in theory any unpaired image-to-image translation model can be used, we choose to use CycleGAN [82] as discussed in Chapter 2.  

**Dataset Generation**  There are many factors that can affect the amount of distortion an underwater image may be subjected to. In certain conditions, an image may have very little distortion or none at all. Denote \( I^C \) as a “clean” underwater image with no distortion, and \( I^D \) to be the *same* image with distortion. The goal is to learn a mapping function \( f : I^D \rightarrow I^C \).

Because of the difficulty of collecting underwater data, more often than not only \( I^D \) or \( I^C \) exist but not both. Gathering training pairs is not as simple as converting a color image to black and white in the case of colorization. Figure 3.1 displays our data generation process. Note the lack of image pairs in Figure 3.1 compared to those in Figure 3.2.

This issue of lacking image pairs in both domains has brought a surge of research in unpaired image-to-image translation methods (also known as domain transfer) [82, 21, 70, 40, 75, 41, 83, 72]. In many of these methods, translating from \( A \) to \( B \) is just as important.
3.1 Methodology

![Paired samples of ground truth and distorted images generated by CycleGAN. Top row: Ground truth. Bottom row: Generated samples.](image)

as translating from $B$ to $A$. Our method differs in this aspect due to the end goal being only the need to translate from $I^D$ to $I^C$. Aside from generating training data, there is no need to distort an image. Because of this, a paired image-to-image translation is desirable. **In order to circumvent the issue of image pairs not being readily available, CycleGAN is used to generate $I^D$ from $I^C$, which allows the generation of a paired dataset of images.**

Given two datasets of unpaired images $X$ and $Y$, where $I^C \in X$ and $I^D \in Y$, CycleGAN learns a mapping $F : X \rightarrow Y$. Figure 3.2 shows paired samples from our generated dataset. From this paired dataset we train a generator $G$ to learn the mapping $f : I^D \rightarrow I^C$. We note that during training, CycleGAN also learns the mapping $G : Y \rightarrow X$ which is similar to $f$. We show a comparison in Chapter 4 with our method, which proves to be superior due to the task of paired image-to-image translation being easier than that of unpaired image-to-image translation.

**Underwater GAN** The main contribution of this thesis is Underwater GAN (UGAN), which is able to remove distortion and enhance colors for underwater images. As discussed in Chapter 2, GANs represent a class of generative models that are based on game theory. Given two differentiable functions modeled as neural networks, a generator $G$ attempts to generate
data points given input noise \((z)\) that are able to fool a discriminator \(D\). The discriminator is given either real data points or data points produced by \(G\) and aims to classify them as real or fake. In our conditional GAN, the generator is given as input a distorted underwater image \(I^D\) and attempts to generate its nondistorted pair \(I^C\) (note \(z\) is omitted). The goal for the discriminator is to be able to distinguish between true non-distorted instances coming from the dataset and false instances produced by the generator. Considering the original GAN [14], the objective function becomes:

\[
\mathcal{L}_{GAN}(G, D) = \min_G \max_D \mathbb{E}_{I^C \sim P_r}[\log D(I^C)] + \mathbb{E}_{I^\gamma \sim P_g}[\log (1 - D(I^\gamma))].
\] (3.1)

where \(I^C\) represents a nondistorted image sampled from the true data distribution \(P_r\), and \(I^\gamma = G(I^D)\) represents the output from the generator given a distorted image \(I^D\). Note for simplicity in notation, we will further omit \(I^C \sim P_r\) and \(I^\gamma \sim P_g\), and simply refer to them as \(I^C\) and \(I^\gamma\) respectively.

In this formulation, the discriminator is modeled as a classifier, with a sigmoid cross-entropy loss function [14], which in practice may lead to issues such as the vanishing gradient and mode collapse. As shown by [3], as the discriminator improves, the gradient of the generator vanishes, making it difficult or impossible to train. Mode collapse occurs when the generator “collapses” onto a single point, fooling the discriminator with only a single instance of data [61]. To illustrate the effect of mode collapse, imagine a GAN being trained to generate digits from the MNIST [31] dataset, but it is only able to generate a single digit repeatedly (e.g., the number 7). In reality, the desired outcome would be to generate a diverse collection of all the digits. To this end, there have been a number of recent methods which hypothesize a different loss function for the discriminator [43, 4, 16, 81].

We focus on the Wasserstein GAN (WGAN) [4] formulation as discussed in Section 2.4, which proposes to use the Earth-Mover or Wasserstein-1 distance \(W\) by constructing a value function using the Kantorovich-Rubinstein duality [74]. In this formulation, \(W\) is
approximated given a set of \(k\)-Lipschitz functions \(f\) modeled as neural networks. To ensure \(f\) is \(k\)-Lipschitz, the weights of the discriminator are clipped to some range \([-c, c]\). In our work, we adopt the Wasserstein GAN with gradient penalty (WGAN-GP) [16], which instead of clipping network weights like in [4], ensures the Lipschitz constraint by enforcing a soft constraint on the gradient norm of the discriminator’s output with respect to its input. Following [16], our new objective then becomes:

\[
\mathcal{L}_{WGAN-GP}(G, D) = \mathbb{E}[D(I^C)] - \mathbb{E}[D(I^\gamma)] + \lambda_{GP}\mathbb{E}_{\hat{x} \sim P_{\hat{x}}}[\left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\right)^2],
\]

where \(I^\gamma = G(I^D)\) and \(P_{\hat{x}}\) is defined as samples along straight lines between pairs of points coming from the true data distribution and the generator distribution, and \(\lambda_{GP}\) is a weighing factor. While we focus on WGAN-GP, a comparison and discussion of the tradeoffs between using Eq. 3.1 and Eq. 3.2 are shown in Chapter 4, giving both functions an analysis on their accuracy and training behaviors. In order to give \(G\) some sense of ground truth by regressing towards our ground truth image, as well as capture low level frequencies in the image, we also consider the \(L_1\) loss:

\[
\mathcal{L}_{L_1} = \mathbb{E}[\|I^C - I^\gamma\|_1].
\]

Combining these, we get our final objective function for our network, which we call Underwater GAN (UGAN),

\[
\mathcal{L}_{UGAN}^* = \min_G \max_D \mathcal{L}_{WGAN-GP}(G, D) + \lambda_{1} \mathcal{L}_{L_1}(G).
\]

The original GAN loss is also considered as described by Eq. 3.1. When using this, the objective for UGAN becomes:

\[
\mathcal{L}_{UGAN}^* = \min_G \max_D \mathcal{L}_{GAN}(G, D) + \lambda_{1} \mathcal{L}_{L_1}(G).
\]
3.1 Methodology

It should be noted that in practice \( \log(1 - D(I^\gamma)) \) will saturate early in training, causing little to no gradient to be sent to \( G \) [14]. This is due to the generated data clearly looking very different than the true data, causing \( D \) to reject samples from \( G \) with high confidence. Therefore, instead of minimizing \( \log(1 - D(I^\gamma)) \) we maximize \( D(I^\gamma) \). We use the naming scheme of UGAN\(_W\) when referring to Eq. 3.4 and UGAN\(_G\) when referring to Eq. 3.5. When just “UGAN” is used, it is implied the context refers to both.

**Image Gradient Difference Loss**  Often times generative models produce blurry images due to an averaging effect over pixels or noise induced. We explore a strategy to sharpen these predictions by directly penalizing the differences of image gradient predictions in the generator, as proposed by [44]. Given a ground truth image \( I^C \), generated image \( I^\gamma = G(I^D) \), and \( \alpha \) which is an integer greater than or equal to 1, the Gradient Difference Loss (GDL) is given by

\[
L_{GDL}(I^C, I^\gamma) = \sum_{i,j} ||I^C_{i,j} - I^C_{i-1,j}|| + ||I^\gamma_{i,j} - I^\gamma_{i-1,j}||. \tag{3.6}
\]

where \( i, j \) are pixel locations. In our experiments, we denote our network as UGAN-P when considering the GDL, which can be expressed as

\[
L^*_U(GANw-P) = \min_G \max_D L_{GANw-GP}(G, D) + \lambda_1 L_{L1}(G) + \lambda_2 L_{GDL}. \tag{3.7}
\]

Similarly, when using the original GAN objective as in Eq. 3.5 we use a naming scheme of UGAN\(_G\)-P.

**Network Architecture**  Our generator network is a fully convolutional autoencoder, similar to the work of [25], which is designed as a “U-Net” [59] due to the structural similarity between input and output as seen in Figure 3.3. The advantage of using a “U-Net” comes
3.1 Methodology

Fig. 3.3 Overview of our method. An image $I^D$ produced by CycleGAN is used as input to the generator $G$, which is structured as a U-Net. The image is then sent to the discriminator $D$. Here, the dashed line represents an $L_1$ loss.

from explicitly preserving spatial dependencies produced by the encoder, as opposed to relying on the embedding to contain all of the information. This is done by the addition of “skip connections”, which concatenate the activations produced from a convolution layer $i$ in the encoder to the input of a transpose convolution layer $n - i + 1$ in the decoder, where $n$ is the total number of layers in the network. Each convolutional layer in our generator uses kernel size $4 \times 4$ with stride 2 which downsamples the input by half. Convolutions in the encoder portion of the network are followed by batch normalization [22] and a leaky ReLU activation with slope 0.2, while transpose convolutions in the decoder are followed by a ReLU activation [48] (no batch norm in the decoder). Exempt from this is the last layer of the decoder, which uses a TanH nonlinearity which is similar to the input distribution of $[-1, 1]$ (images are normalized to be in a range of $[-1, 1]$). Together, the encoder and decoder contain 16 layers (8 each).

Our fully convolutional discriminator is modeled after that of [57], except no batch normalization is used. This is due to the fact that WGAN-GP penalizes the norm of the
discriminator’s gradient with respect to each input individually, which batch normalization would invalidate. Our discriminator is modeled as a PatchGAN [25, 37], which discriminates at the level of image patches. As opposed to a regular discriminator, which outputs a scalar value corresponding to real or fake, our PatchGAN discriminator outputs a $32 \times 32 \times 1$ feature matrix, which provides a metric for high-level frequencies. The final output for $D$ is given by taking the mean from this patch. Our generator model is displayed in Figure 3.4. Below we show detailed tables of our generator and discriminator. Note that in the generator decoder, filter sizes are before concatenation from the encoder occurs. In all tables, “Convolution” is represented by “Conv”, “Transpose Convolution” is represented by “T Conv”, and Leaky ReLU is represented by L-ReLU.
### Generator Encoder

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Stride</th>
<th>Filters</th>
<th>Batch Norm</th>
<th>Activation</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>256 × 256</td>
</tr>
<tr>
<td>Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>64</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>128 × 128</td>
</tr>
<tr>
<td>Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>128</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>64 × 64</td>
</tr>
<tr>
<td>Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>256</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>32 × 32</td>
</tr>
<tr>
<td>Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>512</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>16 × 16</td>
</tr>
<tr>
<td>Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>512</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>8 × 8</td>
</tr>
<tr>
<td>Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>512</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>4 × 4</td>
</tr>
<tr>
<td>Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>512</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>2 × 2</td>
</tr>
<tr>
<td>Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>512</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>1 × 1</td>
</tr>
</tbody>
</table>

### Generator Decoder

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Stride</th>
<th>Filters</th>
<th>Batch Norm</th>
<th>Activation</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>T Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>512</td>
<td>No</td>
<td>ReLU</td>
<td>2 × 2</td>
</tr>
<tr>
<td>T Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>512</td>
<td>No</td>
<td>ReLU</td>
<td>4 × 4</td>
</tr>
<tr>
<td>T Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>512</td>
<td>No</td>
<td>ReLU</td>
<td>8 × 8</td>
</tr>
<tr>
<td>T Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>512</td>
<td>No</td>
<td>ReLU</td>
<td>16 × 16</td>
</tr>
<tr>
<td>T Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>256</td>
<td>No</td>
<td>ReLU</td>
<td>32 × 32</td>
</tr>
<tr>
<td>T Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>128</td>
<td>No</td>
<td>ReLU</td>
<td>64 × 64</td>
</tr>
<tr>
<td>T Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>64</td>
<td>No</td>
<td>ReLU</td>
<td>128 × 128</td>
</tr>
<tr>
<td>T Conv</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>3</td>
<td>No</td>
<td>Tanh</td>
<td>256 × 256</td>
</tr>
</tbody>
</table>
3.1 Methodology

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Stride</th>
<th>Filters</th>
<th>Batch Norm</th>
<th>Activation</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>64</td>
<td>No</td>
<td>L-ReLU</td>
<td>128 × 128</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>128</td>
<td>No</td>
<td>L-ReLU</td>
<td>64 × 64</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>256</td>
<td>No</td>
<td>L-ReLU</td>
<td>32 × 32</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>512</td>
<td>No</td>
<td>L-ReLU</td>
<td>32 × 32</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>1</td>
<td>1</td>
<td>No</td>
<td>Linear</td>
<td>32 × 32</td>
</tr>
</tbody>
</table>

Additional Data  Because distortion can behave in many different granularities, in order to provide additional data during training we linearly interpolate between the clean image $I^C$ and the distorted version $I^D$ generated by CycleGAN. This allows the network to better generalize in terms of the different types of environments that may be encountered. For example, in a setting in which a robot is performing a survey by exploring both shallow and deeper waters, the shallow portions may not exhibit any distortion at all. By forcing our network to realize that a nondistorted image should receive no alteration (clean image as input, clean image as output), we can avoid situations where the network disrupts the visual quality of an already-clean image. Figure 3.5 shows example images that are generated by interpolating linearly from $I^D$ to $I^C$ in the pixel space. Any one of these images (including $I^C$) is sent to the generator as input where $I^C$ is the target output.

Fig. 3.4 An overview of the UGAN generator model. Naming scheme is as follows: k4 corresponds to a kernel size of ×4, n64 corresponds to the feature maps having a depth of 64, and s4 corresponds to a stride of 4.
Further data augmentation can be performed to increase the size of the training set while also providing additional means of generalization. Simple alterations such as flipping images horizontally and vertically are employed. Underwater images also contain distortions besides color imbalance, such as blur. By convolving the input image $I^D$ with a Gaussian to induce artificial blur, we can effectively train our network to not only enhance color, but sharpen images as well. These transformations can be employed to occur randomly with a probability of 50% during training.

### 3.2 Distorter GAN

CycleGAN is an effective method with which we can create a dataset. However, there exists wasted computation because of the cycle consistency loss it enforces. Aside from comparing results, we are only interested in the mapping from nondistorted to distorted images in order to create a paired training dataset (not the mapping from distorted to nondistorted). Therefore, we are effectively doubling the computation needed. Furthermore, the requirement that the distorted image must be able to be transformed back into its original domain limits the amount of distortion that it may exhibit.

In order to circumvent these issues, we introduce a new GAN-based method to provide a nonlinear amount of distortion to clean underwater images. First, we observe that much of the distortion an image exhibits is color-based. For this reason we choose to use the CIELAB color space [52], which is perceptually uniform with respect to human color vision.
Color expressed in the CIELAB color space (often just abbreviated LAB) is expressed as a lightness channel $L$, a green-red channel $a$, and a blue-yellow channel $b$. Another advantage the LAB color space offers is computational efficiency when compared to using standard RGB. Given a grayscale image as input to a deep neural network, the output using RGB would be $y' \in \mathbb{R}^{n \times m \times 3}$, where $n$ and $m$ are the height and width of the output, respectively. Then, something such as the $L_2$ loss could be used in order to minimize the predicted $y'$ with the true color image $y$. However, the use of $L_2$ here combined with an autoencoder approach would cause the output to be blurry, losing its spatial structure. When using the LAB color space, the input is a grayscale image into a deep neural network. However, the network only needs to predict an output $y' \in \mathbb{R}^{n \times m \times 2}$, where $n$ and $m$ are again the height and width of the output, respectively. Computation is saved by only needing to predict two dimensions, $a$ and $b$, instead of three, $r$, $g$, and $b$. After prediction, $a$ and $b$ can be concatenated onto the original lightness channel $L$ which was used as input. Because this explicitly preserves spatial structure, there is no blurring induced.

**Method**  As discussed in Chapter 2, the distortion exhibited by underwater vision is highly nonlinear and not easily modeled. Therefore, we use a convolutional neural network in order to induce a nonlinear amount of noise in nondistorted underwater images. We take an approach very similar to that of UGAN. Given a set of nondistorted underwater images $X$ and a set of distorted underwater images $Y$, we want to perform image-to-image translation on images $I^C \in X$ such that they appear to have come from $Y$. During training, we only consider images from the distorted set $Y$. Given an image $I^D$, we separate it into its LAB color components, $L$, $a$, and $b$. A generator network with the same structure as UGAN is used. Given the lightness channel $L$ as input, $G$ predicts the two color channels $a$ and $b$. The predicted color channels are combined with the original channel $L$ in order to construct the final image. A discriminator network then discriminates between images coming from the true distorted dataset which consist of the lightness channel $L$ and the true $a$ and $b$ channels,
3.2 Distorter GAN

and images constructed by the generator, which consist of the same lightness channel $L$, but use the predicted $a$ and $b$ channels.

As with UGAN$_W$, we use the improved WGAN method (WGAN-GP) [16]. We also consider the $L_1$ loss, which provides the network with a sense of ground truth. Formally given a distorted image $I^D \in Y$, we deconstruct it into its LAB components $L, a, \text{and } b$. A generator network predicts the $a$ and $b$ color channels given a lightness channel $L$. The objective in an adversarial sense becomes:

$$
L_{WGAN-GP}(G, D) = \mathbb{E}[D(L, ab)] - \mathbb{E}[D(G(L))] + \lambda_{GP}\mathbb{E}_{\hat{x} \sim P_{\hat{x}}}[(||\nabla_{\hat{x}}D(\hat{x})||_2 - 1)^2],
$$

(3.8)

where $P_{\hat{x}}$ is defined as samples along straight lines between pairs of points coming from the true data distribution and the generator distribution, and $\lambda_{GP}$ is a weighing factor. We also consider the $L_1$ loss between the true $ab$ color channels and the $ab$ channels predicted by the generator.

$$
L_{L_1} = \mathbb{E}[||ab - G(L)||_1].
$$

(3.9)

Combining these, we get our final objective function for our network, which we call Distorter GAN (DGAN),

$$
L^*_{DGAN} = \min_G \max_D L_{WGAN-GP}(G, D) + \lambda_1 L_{L_1}(G).
$$

(3.10)

Figure 3.6 displays samples generated by DGAN. We can use a metric that measures the quality of the images called the Underwater Image Quality Measure (UIQM) [50] (described in further detail in Chapter 4) to estimate the level of distortion the image exhibits. The UIQM is comprised of three underwater quality measures: colorfulness, sharpness, and contrast.
3.2 Distorter GAN

One may question the choice to use DistorterGAN over CycleGAN when given similar performance. Besides the wasted computation mentioned earlier, we can turn towards the difficulty in convergence in generative models. Generative models, especially image-to-image translation methods, are very difficult to quantify. Whereas in other machine learning domains, such as classification, we can stop training when the accuracy on a validation set meets some criteria or is no longer getting smaller, image-to-image translation methods do not have such a luxury. Many times we must use visual quality metrics, meaning we only eyeball the results and stop training when the results look “good enough”. One major reason to use the Wasserstein metric in a GANs architecture besides the stability of training is the correlation between the loss function and visual quality. In the original GAN formulation, viewing the losses for the generator and discriminator show them to be stagnant throughout training, which is seen during the training of CycleGAN. However, when using a Wasserstein metric, we can see that the loss for both $G$ and $D$ are converging. Thus, we can stop training after viewing that the loss for $G$ has converged.

In our preliminary experiments in using DistorterGAN rather than CycleGAN as a distortion method, we found that the results produced by UGAN were very poor. While these results were able to remove the blue/green hue often found in distorted images, it was unable to produce any vibrant colors in the output. We believe that this was due to the colorspace swap. In training DistorterGAN, the LAB colorspace is used. However, UGAN is trained on the more classical RGB colorspace. Future work will focus on implementing a LAB colorspace option for UGAN to take this into account, with ideally more colorful and vibrant results.

Table 3.1 UIQM for the images shown in Figure 3.6. A higher UIQM corresponds to a cleaner image.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>8.97</td>
<td>4.73</td>
<td>6.53</td>
<td>3.12</td>
<td>4.49</td>
</tr>
<tr>
<td>Distorted</td>
<td>4.59</td>
<td>4.02</td>
<td>3.36</td>
<td>2.57</td>
<td>3.26</td>
</tr>
</tbody>
</table>
3.3 Model Inspection

Determining the intrinsics of deep learning models can be very tricky due to high number of parameters and training methods used. It has been observed that earlier layers learn low level information such as edges and corners, which are then combined to learn higher level abstract concepts in later layers. This is also the reason we found skip connections to be very effective in preserving spatial structure. We now turn towards examining the structure of our generator, which is in the form of an autoencoder. Each subsequent layer downsamples its input, which at the final layer of the encoder leaves us with an embedding of a much smaller size than the original input. This embedding is upsampled through a series of transpose convolutions to create an image. Now, intuition tells us that because the skip connections in the network are preserving spatial dependencies, then the embedding must contain the additional information required for the task at hand: color. Therefore, if we can explore this latent space, then perhaps we can control the final color of our output.

First, small modifications were made in order to introduce a small perturbation into the embedding. When no difference in the output was observed, larger perturbations were made. When still no change in the output was observed, Gaussian noise was used to
completely replace the embedding. Again, no visual change was observed in the output. In fact, the output was the exact same, pixel for pixel. This tells us that the majority of the information is being contained in previous layers, and the layers near the embedding are wasted computation. While slightly frustrating, this fact is exciting. Running UGAN in real-time is a very important goal, and due to the computational limits of many mobile robots, this becomes difficult. However, because we now know that part of our model is wasted computation, we can effectively reduce the number of parameters, and therefore the runtime, while keeping the accuracy of our model. We explore this further in the experiments section in Chapter 4.
Chapter 4

Experiments

In this section, we describe experiments we have conducted to evaluate the proposed UGAN model for enhancing underwater imagery. Qualitative results are shown, displaying an improvement in visual quality. Without a reference ground truth image, underwater images are difficult to evaluate quantitatively. We provide quantitative results in the form of a nonreference-based measure, which is able to measure local statistics in images that correspond to high visual quality. Furthermore, we show improvements to a vision-based algorithm for tracking scuba divers. We found that the dataset and method described in Section 3.2 did not improve results significantly, so the following sections show results using CycleGAN as a distrotter method.

4.1 Datasets

We use several datasets and methods throughout the development of UGAN. Final experiments used CycleGAN as a distortion method as seen in [13]. During the training of CycleGAN, several subsets from the ImageNet [11] dataset (e.g., scuba divers, fish, brain corrals, etc.) were chosen. These were manually classified into two classes, distorted and nondistorted, based on visual inspection. Our recent approach has attempted to improve
4.2 Qualitative Results

Upon this by using a larger dataset and DistorterGAN as a distortion method. DistorterGAN uses a greater amount of images from ImageNet, as well as images scraped from Google using the Google Images Download tool [73]. These images are again classified into two classes. Additionally, we extract frames from diving videos found on Youtube™ which are known to be distorted

We let $X$ be the set of underwater images with no distortion, and $Y$ be the set of underwater images with distortion. We then train DistorterGAN to learn the mapping $F : X \rightarrow Y$ such that we have paired datasets as seen in Figure 3.6. However, this did not show any improvement over using CycleGAN. Evaluation is done on distorted images from the datasets previously mentioned. In order to compare with CycleGAN, we use a test set of images acquired from Flickr™. We also evaluate a frequency and spatial-domain diver-tracking algorithm on a video of scuba divers taken from YouTube™.

4.2 Qualitative Results

Here we take a look at some of the qualitative results produced by UGAN. Figure 4.1 shows samples from our test set. The input columns display a variety distortion levels, ranging from very green (column 1 row 1), to very blue (column 1 row 3). While many of the distorted images contain a blue or green hue over the entire image space, that is not always the case. In certain environments, it is possible that objects close to the camera are undistorted with correct colors, while the background of the image contains distortion due to it being farther away from the camera. In these cases, we would like the network to only enhance parts of the image that appear distorted. The last row in Figure 4.1 shows a sample of such an image. The orange of the clownfish is left unchanged while the distorted sea anemone in the background has its color enhanced.

1 https://www.youtube.com/watch?v=QmRFmhILd5o
2 https://www.youtube.com/watch?v=ALN6y2PLCq0
3 https://www.youtube.com/watch?v=QmRFmhILd5o
4.2 Qualitative Results

The human visual system is adept at qualitatively evaluating the images shown in Figure 4.1. It is obvious that the output images have the blue/green haze removed and their colors enhanced. While autonomous computer vision algorithms have shown incredible performance on many tasks in the past few years, there are still circumstances in which human inspection may be preferred or even necessary. For example, monitoring tasks may use an automated method which is then checked by a human counterpart, or video footage of an area may be viewed by environment experts. In these cases, visual quality is important for the viewer. Figure 4.1 shows visual improvement in a variety of environments a human may encounter. Despite this, a quantitative evaluation method is needed. While subjective evaluations can be done by human volunteers, this is a tedious process which cannot be automated. We thus show quantitative results after evaluating UGAN outputs using a metric which incorporates measures known to exist in the human visual system.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Input</th>
<th>Output</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Input Image 1" /></td>
<td><img src="output1" alt="Output Image 1" /></td>
<td><img src="image2" alt="Input Image 2" /></td>
<td><img src="output2" alt="Output Image 2" /></td>
<td><img src="image3" alt="Input Image 3" /></td>
<td><img src="output3" alt="Output Image 3" /></td>
</tr>
<tr>
<td><img src="image4" alt="Input Image 4" /></td>
<td><img src="output4" alt="Output Image 4" /></td>
<td><img src="image5" alt="Input Image 5" /></td>
<td><img src="output5" alt="Output Image 5" /></td>
<td><img src="image6" alt="Input Image 6" /></td>
<td><img src="output6" alt="Output Image 6" /></td>
</tr>
<tr>
<td><img src="image7" alt="Input Image 7" /></td>
<td><img src="output7" alt="Output Image 7" /></td>
<td><img src="image8" alt="Input Image 8" /></td>
<td><img src="output8" alt="Output Image 8" /></td>
<td><img src="image9" alt="Input Image 9" /></td>
<td><img src="output9" alt="Output Image 9" /></td>
</tr>
</tbody>
</table>

Fig. 4.1 Qualitative results on images from our test set. UGAN_W can both recover color and also enhance color if a small amount is present.
4.3 Underwater Image Quality Measure

Underwater images that have been enhanced are in a unique (and frustrating) situation of not containing a ground truth reference image to compare to. Subjective evaluations performed by humans may be employed but may contain bias and cannot be fully automated. It is therefore desirable to obtain a non-reference measure in order to give a quantitative metric for enhanced images. We turn to the work of Panetta et al. [50] who developed a metric called the Underwater Image Quality Measure (UIQM) able to assess the quality of an underwater image without a reference image. This metric is comprised of three underwater image attributes: color, sharpness, and contrast. These metrics were specifically chosen in accordance with the human visual system (HVS), attempting to capture the subjectivity humans exhibit. We describe the three components used by the UIQM and give qualitative results for each component. Each individual score shows that images which are qualitatively clearer receive a higher score.

**Underwater Image Colorfulness Measure (UICM)** Underwater images often contain a blue or green hue which worsens as the depth increases. This is due to the red wavelength being the longest of the visual spectrum, causing it to be absorbed by the water first. The first component of the UIQM is the Underwater Image Colorfullness Measure (UICM), which measures the colorfulness of an image. A higher UICM corresponds to a more colorful image and a higher quality image. The HVS is known to capture colors in the opponent color plane [50], which gives reason to use the two opponent color components related with chrominance: $RG$ and $YB$.

\[
RG = R - G \\
YB = \frac{R + G}{2} - B \tag{4.1}
\]
4.3 Underwater Image Quality Measure

Fig. 4.2 Quantitative examples of the metrics within the UIQM. (a)-(c) shows distorted images with notably lower scores, where as (d)-(f) show nondistorted images with higher scores.

Due to the heavy distortion and noise exhibited by underwater images, asymmetric alpha-trimmed statistical values [7] are used in place of normal statistical values. Given an image $x$ of size $K = m \times n \times c$, the image is flattened to a vector of pixels which is sorted into a sequence $s$ such that $x_1 \leq x_2 \leq \ldots \leq x_K$. We now define $T_{\alpha_l} = \lceil \alpha_l K \rceil$ to be the number of pixels to be trimmed from the beginning of sequence $s$, and $T_{\alpha_R} = \lfloor \alpha_l K \rfloor$ to be the number of pixels to be trimmed from the end of sequence $s$. This trims the pixels that are either on the very high end of the color space (such as bubbles) that are excluded, or pixels that are very dark and may not be related to the overall scene. Following [7], the asymmetric alpha-trimmed mean is defined as

$$
\mu_{\alpha, RG} = \frac{1}{K - T_{\alpha_l} - T_{\alpha_R}} \sum_{i=T_{\alpha_l}+1}^{K-T_{\alpha_R}} \text{Intensity}_{RG,i}, \quad (4.2)
$$

where $\text{Intensity}$ is the pixel value at location $i$ in sequence $s$. The asymmetric alpha-trimmed mean is meant to exclude pixels which may not be representative of the scene as a whole,
4.3 Underwater Image Quality Measure

such as light rays or bubbles. The results shown in this work use $\alpha_L = \alpha_R = 0.1$. Note that $\mu_{\alpha, YB}$ is defined similarly.

Following this trend, the metric to measure the variance assesses the pixel activity within each color component. A higher variance would imply a higher dynamic range within an image, causing the differences in colors to become more apparent, meaning the image overall has a higher quality of color. Formally, this is defined by

$$\sigma^2_{\alpha, RG} = \frac{1}{N} \sum_{p=1}^{N} (Intensity_{RG, p} - \mu_{\alpha, RG})^2$$ (4.3)

Combining these two terms gives us an overall metric for measuring color:

$$UICM = \lambda_1 \sqrt{\mu^2_{\alpha, RG} + \mu^2_{\alpha, YB}} + \lambda_2 \sqrt{\sigma^2_{\alpha, RG} + \sigma^2_{\alpha, YB}}$$ (4.4)

Following [50], we use the values of $\lambda_1 = -0.0268$ and $\lambda_2 = 0.1586$, which were obtained by linear regression. Figure 4.2 displays the UICM for different images. It is seen that Fig. 4.2(a) contains diluted colors and distortion, which is reflected by a lower UICM. On the other hand, Fig. 4.2(d) contains less distortion and qualitatively displays better colors, which is reflected by the higher UICM.

**Underwater Image Sharpness Measure (UISM)** Forward scattering underwater is known to cause a blurring effect on images [66]. This effect reduces the sharpness and fine details that are desired in enhanced images. The Underwater Image Sharpness Measure (UISM) is an effective measure to capture the level of detail preserved in images such as the sharpness of edges. A Sobel edge detector [69] is applied to each RGB color channel. The Hadamard product is then taken between the edge map and the original image to retrieve the final edge map. As shown by [17], the Measure of Enhancement by Entropy (EME) can be used to measure the sharpness of edges. The EME is formally defined as

$$EME = \frac{2}{k_1k_2} \sum_{l=1}^{k_1} \sum_{k=1}^{k_2} \log \frac{I_{\max, k, l}}{I_{\min, k, l}}$$ (4.5)
where the image is divided into \( k_1k_2 \) blocks, and \( I_{\text{max}} \) and \( I_{\text{min}} \) are the maximum and minimum for that block, respectively. The EME is then used in conjunction with the edge maps obtained by the Sobel edge detector to obtain the UISM:

\[
UISM = \sum_{c=1}^{3} \lambda_c EME(\text{grayscale edge}_c)
\]  (4.6)

In calculating the EME, the image is divided into \( k_1k_2 \) blocks. This work uses a block size of \( 10 \times 10 \) for each image. The EME is computed for each edge map computed from the image’s color channels and linearly combined with coefficient \( \lambda_c \). Following [55], we used \( \lambda_R = 0.299, \lambda_G = 0.587, \) and \( \lambda_B = 0.144 \) due to their relative visual responses. Figure 4.2(b) appears blurry due to the scattering of particles, whereas Figure 4.2(e) is much sharper, as confirmed by a higher UISM.

**Underwater Image Contrast Measure (UIConM)**  The final metric used in computing the UIQM is the Underwater Image Contrast Measure (UIConM). Contrast levels in underwater images have been shown to degrade due to backward scattering [66]. We measure this using an entropy-based approach as seen in [17]. The Logarithmic AME by Entropy (AMEE) is defined as:

\[
AMEE = -\frac{1}{k_1k_2} \sum_{l=1}^{k_1} \sum_{k=1}^{k_2} \left[ \alpha \left( \frac{I_{\text{max},k,l} - I_{\text{min},k,l}}{I_{\text{max},k,l} + I_{\text{min},k,l}} \right)^\alpha \times \log \left( \frac{I_{\text{max},k,l} - I_{\text{min},k,l}}{I_{\text{max},k,l} + I_{\text{min},k,l}} \right) \right]
\]  (4.7)

For computational simplicity, we set \( \alpha = 1 \). Similar to the UISM, the image is divided into \( k_1k_2 \) blocks. Here we use the same block size of \( 10 \times 10 \). Figure 4.2(c) shows a distorted
4.3 Underwater Image Quality Measure

The Underwater Image Quality Measure (UIQM) is given by a linear combination of the UICM, UISM, and UIConM. This is defined by

\[
UIQM = c_1 \times UICM + c_2 \times UISM + c_3 \times UIConM
\]  

(4.8)

The parameters \(c_1\), \(c_2\), and \(c_3\) can be application dependent and give weight to the importance of the specific measures. We use the coefficients obtained from [79], namely \(c_1 = 0.4680\), \(c_2 = 0.2745\), \(c_3 = 0.2576\). The UIQM proves to be an accurate metric for measuring the quality of enhanced underwater images. Table 4.1 displays the average UIQM for our test set containing approximately 1,800 images. The “Original” column in Table 4.1 corresponds
4.3 Underwater Image Quality Measure

Fig. 4.4 Local image patches extracted for qualitative comparisons. Each patch was resized to 64 × 64, but shown enlarged for viewing ability. Sharpness is better preserved in the UGAN models, and the colors are more vibrant. Best viewed in PDF.

to the set of distorted images we used as a test set. The images that have been enhanced by UGAN_W and UGAN_W-P receive a higher UIQM score, with UGAN_W scoring the highest. Figure 4.3 displays examples of the original distorted images (a,d), their enhancement from UGAN_W (b,e), and their enhancement from UGAN_W-P (c,f) along with their respective UIQM scores.

We use a smaller test set of images acquired from Flickr™ to compare UGAN_W with CycleGAN, as CycleGAN inherently learns the mapping \( F : X \rightarrow Y \), where \( X \) is the set of distorted images and \( Y \) is the set of nondistorted images. Figure 4.4 shows a qualitative comparison by zooming in on local image patches. Edges are not preserved as cleanly in CycleGAN as they are in UGAN_W. Furthermore, the color in UGAN_W and UGAN_W-P is brighter and more vibrant. Quantitative results are shown in Table 4.2.

Table 4.2 Comparison of the average UIQM for Flickr™ test set.

<table>
<thead>
<tr>
<th>Original</th>
<th>CycleGAN</th>
<th>UGAN_W</th>
<th>UGAN_W-P</th>
</tr>
</thead>
<tbody>
<tr>
<td>UIQM</td>
<td>3.83</td>
<td>4.38</td>
<td>4.73</td>
</tr>
</tbody>
</table>
4.4 Diver Tracking using Frequency-Domain Detection

We investigate the frequency-domain characteristics of the restored images through a case-study of periodic motion tracking in sequence of images. Particularly, we compared the performance of Mixed Domain Periodic Motion (MDPM)-tracker [23] on a sequence of images of a diver swimming in arbitrary directions. MDPM tracker is designed for underwater robots to follow scuba divers by tracking distinct frequency-domain signatures (high-amplitude spectra at 1-2Hz) pertaining to human swimming. Amplitude spectra in frequency-domain correspond to the periodic intensity variations in image-space over time, which are often eroded in noisy underwater images [68].

Fig. 4.5 illustrates the improved performance of MDPM tracker on generated images compared to the real ones. Underwater images often fail to capture the true contrast in intensity values between foreground and background due to low visibility. The generated images seem to restore these eroded intensity variations to some extent, causing much improved positive detection (a 350% increase in correct detections) for the MDPM tracker. This shows the capability of running UGAN on the Aqua robot described in Section 2.2.

4.5 Comparing Generative Models

Here we investigate reasons for and against using the Improved Wasserstein GAN [16] for UGANW versus the original GAN method [14] for use in UGANG. GANs are notoriously difficult to train in practice [3], and mostly rely on a set of tricks in order to stabilize. Using Eq. 3.5 as an objective function may not provide sufficient gradient for G to learn well [14, 16]. This is due to G performing very poorly early in training. In a typical GAN setting (no conditioning) the weights for G are randomly initialized and the input noise vector z is drawn randomly from some distribution. Therefore, at the start of training the output from G is essentially noise, causing D to easily distinguish between real and generated data points.
4.5 Comparing Generative Models

In order to provide stronger gradients, [14] proposes to maximize \( \log D(G(z)) \) rather than minimize \( \log(1 - D(G(z))) \). While this provides stronger gradients for \( G \) early on, the work of [3] showed that as \( D \) improves, the gradients updates to the generator get worse, meaning the same problem still persists.

The use of a conditional GAN by introducing some additional information (such as a label) has also shown to stabilize training in practice and produce sharp results. UGAN is inherently a conditional GAN, as we condition the generator on a distorted image. We train \( \text{UGAN}_G \) using the original GAN objective function from [14] and compare to the Wasserstein metric from [16]. We found that \( \text{UGAN}_G \) was stable during training when combining the original GAN loss with the \( L_1 \) loss (Eq. 3.3). Due to the only difference being the objective function, inference speed of the model stays the same. However, training time is massively reduced because the discriminator and generator are updated once each as opposed to the discriminator being updated \( n = 5 \) times for every generator update in WGAN-GP.

Section 4.7 gives a qualitative and quantitative comparison between \( \text{UGAN}_G \) and \( \text{UGAN}_W \). We found the results to be very similar, which can be attributed to the addition of the \( L_1 \)
4.6 Systems

This section describes the system specifications that were used in this work, as well as experiments done in order to run UGAN on a mobile robot. An in-house deep learning machine was built allowing training to be done in reasonable time. The machine uses an ASUS X99-E Motherboard which allows for four GPUs to run at maximum power. This allows the machine to use four GTX 1080 GPUs which require custom watercooling in order to keep temperatures low and prevent overheating. Additionally, an i7 Intel CPU is used along with 64 GB of RAM. As the Aqua robot (discussed in Section 2.2) is not equipped with a GPU, running UGAN in real time would be nearly impossible. In order for UGAN to be feasible to run in the field in real time on Aqua, an NVIDIA Jetson TX2™ is in the process of being installed. The Jetson is an embedded AI computing platform with GPU-accelerated parallel processing able to run machine learning models. Experiments were conducted on the Jetson on the bench giving a reasonable metric for real time applicability.

In all of our experiments, the input images have dimensions $256 \times 256 \times 3$ (height $\times$ width $\times$ color channels), which are acceptable measures for underwater tasks. A video of our method is also available. There are many hyperparameters able to be tuned in the objective function in Eq. 3.4 and Eq. 3.5. Following the work of [25] we use $\lambda_1 = 100$, and following [16] we use $\lambda_{GP} = 10$. A batch size of 32 was used to take advantage of GPU hardware. The Adam Optimizer [27] was seen to work very well with a learning rate of $\eta = 1e^{-4}$. Following WGAN-GP [16], the discriminator is updated $n = 5$ times for every update of the

---

4Video of our method can be found at http://bit.ly/2CnwStI
4.7 Results

For UGAN-P which incorporates the GDL (Eq. 3.6 and 3.7), we set $\lambda_2 = 1.0$ and $\alpha = 1.0$.

Implementation is done using the Tensorflow library [1]. All networks were trained on an NVIDIA GTX 1080 GPU for 100 epochs. Initial experiments showed inference for UGAN on the GPU takes on average 0.0138s, which is about 72 Frames Per Second (FPS). On a CPU (Intel Core i7-5930K), inference takes on average 0.1244s, which is about 8 FPS. Note that inference for UGAN$_W$ and UGAN$_G$ takes the same amount of time, as the only difference between the two is the objective function during training. The Aqua robot is currently in the stages of incorporating an NVIDIA Jetson™ which will allow UGAN to be run faster than it would on the CPU. Initial testing on the NVIDIA Jetson showed that UGAN ran far too slowly to be used in real time. However, as discussed in Section 3.3, it may be possible to alter the architecture of UGAN in order to reduce the computation without the possibility of losing performance. We experimented by progressively removing internal layers which were found to contain little to no information regarding the task. The original UGAN design based off of the pix2pix network [25] contains 16 layers for the generator (the discriminator is discarded after training). We modified this network architecture to create a multitude of networks with similar structures but a reduced number of layers. All networks contain skip connections and an internal embedding of dimension 512 as in the original architecture. Specifically, we experimented with 4, 8, 10, 12, and 16 layers. The naming scheme associated with the modified networks is UGAN-$n$, with $n$ being the number of layers in that network. Network architecture details for UGAN-$n$ can be found in Appendix B.

4.7 Results

Here we compare all of the aforementioned variations of UGAN. We qualitatively and quantitatively compare the number of layers used, the GAN metric used (WGAN or GAN),

---

$^5$/Code is available at https://github.com/cameronfabbri/UGAN-V2
4.7 Results

and the use of the GDL from Eq. 3.6 (UGAN-P). The choice of one model over another comes directly from the application at hand. If processing happens offline, then a higher UIQM may be desired without much of an issue with low FPS. On the other hand, if real-time results are desired, then one may accept a lower quality result in order for quicker processing times. If training time is required to be low, then UGAN\(_G\) would be preferred over UGAN\(_W\). This is due to the discriminator in UGAN\(_W\) being updated \(n = 5\) times for every update for the generator in order to satisfy the Lipschitz constraint, whereas UGAN\(_G\) updates the generator and discriminator once each in an alternating fashion. The use of the GDL as part of the objective function showed no noticable difference in training time as the computation is relatively fast.

Figure 4.6 displays a qualitative comparison between using the original GAN loss [14] with UGAN\(_G\) and the WGAN-GP loss [16] with UGAN\(_W\). Visually they are very similar and
Table 4.3 Comparison of average UIQM for the ImageNet test set over all of the UGAN models. A higher UIQM score is better, with UGAN$_{G}$-P with 8 layers scoring the highest.

<table>
<thead>
<tr>
<th>Layers</th>
<th>Original</th>
<th>UGAN$_{G}$</th>
<th>UGAN$_{G}$-P</th>
<th>UGAN$_{W}$</th>
<th>UGAN$_{W}$-P</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4.53</td>
<td>4.70</td>
<td>4.73</td>
<td>4.64</td>
<td>4.42</td>
</tr>
<tr>
<td>8</td>
<td>4.53</td>
<td>5.03</td>
<td><strong>5.20</strong></td>
<td>4.99</td>
<td>4.94</td>
</tr>
<tr>
<td>10</td>
<td>4.53</td>
<td>4.97</td>
<td>5.03</td>
<td>5.03</td>
<td>4.89</td>
</tr>
<tr>
<td>12</td>
<td>4.53</td>
<td>4.85</td>
<td>4.81</td>
<td>4.83</td>
<td>4.91</td>
</tr>
<tr>
<td>16</td>
<td>4.53</td>
<td>4.94</td>
<td>4.84</td>
<td>4.93</td>
<td>4.81</td>
</tr>
</tbody>
</table>

both realistically restore color and remove distortions. Each column of the original images contain very different forms of distortions, ranging from a light blue hue in the first column to a green hue with a haze effect in the fourth column. Even when the distortion exhibited is the lack of light as seen in the second column, UGAN is able to uncover the details of the fish that is barely noticable in the original image. For a quantitative comparison we again turn to the UIQM. Table 4.3 shows the average UIQM obtained over all variations of UGAN for our test set. If results are indistinguishable for a certain task, then UGAN$_{G}$ is the preferred network due to the short training time compared to UGAN$_{W}$. The short training time comes from the need of UGAN$_{W}$ to update the discriminator $n = 5$ times for every generator update, whereas UGAN$_{G}$ updates the discriminator once each alternately.

We now look at the effect the number of layers has on UGAN in terms of speed and accuracy. Figures 4.7-4.10 display a qualitative comparison between UGAN$_{G}$, UGAN$_{G}$-P, UGAN$_{W}$, and UGAN$_{W}$-P using a varying number of layers. Amazingly with only 4 layers, UGAN-4 gives acceptable qualitative results. This suggests that the reduction in quality may be sufficiently low in order for UGAN-4 to run in real time on Aqua. To measure this quantitatively, we turn to the UIQM. Table 4.3 displays the average UIQM for all variations of UGAN with varying layers. The highest average UIQM obtained is by UGAN$_{G}$-P with a UIQM of 5.20. Noticably, the UIQM does not drastically increase with the addition of more layers, which confirms our suspicions of the model’s internal waste of computation.
as discussed in Section 3.3. The decrease in computational complexity and relatively small change in performance gives motivation for running UGAN in real time on the NVIDIA Jetson TX2 for use in mobile settings.

To test the possibility of running UGAN on Aqua in real time, we turn towards frames per second (FPS) as a metric. The Aqua robot has three cameras equipped, each with a frame-rate of 30 FPS. Exceeding this frame-rate with UGAN is not needed, and the minimum frame-rate needed is application dependent. For tasks such as monitoring or surveying, a low FPS may not be an issue as data is processed offline. For tasks such as tracking a slow moving object, a low FPS may be acceptable (such as 5 FPS) as the object is tracked multiple times per second. However tracking a fast moving object which may require frame-by-frame tracking would require higher FPS. Table 4.4 displays the FPS obtained by the variations of UGAN. Not surprisingly, UGAN obtains a higher FPS when using a smaller number of layers due to the computational complexity decrease. Using 4 layers shows that 8 FPS are attainable with good performance (as measured by the UIQM). Further improvements to the model may be able to increase the FPS count while maintaining a reasonable sense of accuracy, but we leave that for future work.

It is important to notice that UGAN does not just simply remove or reduce the blue/green color from the image. Seen in column 2 of Figures 4.7 - 4.10 is a blue fish which retains its blue color after enhancement. Had a linear operation been used to reduce the blue in the image in an attempt to enhance it, the fish would look washed out. This also shows the ability for UGAN to recognize objects in a scene and avoid reducing their natural color despite it containing similar properties to the distortion that we are aiming to remove. Figure 4.11 displays a qualitative comparison of all models with varying layers in order to provide an easier comparison visually. Here it is easier to see the differences between models, especially between layers. When using 4 layers, small perturbations are visible in the image, such as a grid effect due to the use of transpose convolutions and a generally more washed out image.
### Table 4.4 Comparison of FPS for different number of layers in UGAN running on different devices.

<table>
<thead>
<tr>
<th>Device</th>
<th>4 Layers</th>
<th>8 Layers</th>
<th>10 Layers</th>
<th>12 Layers</th>
<th>16 Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>378</td>
<td>104</td>
<td>116</td>
<td>117</td>
<td>73</td>
</tr>
<tr>
<td>CPU</td>
<td>31</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Jetson</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

in terms of color. Qualitatively, using 16 layers appears to produce the smoothest image with the most vibrant colors.
Fig. 4.7 Comparison of UGAN<sub>G</sub> using a varying number of layers
Fig. 4.8 Comparison of UGAN\textsubscript{G}-P using a varying number of layers
4.7 Results

Fig. 4.9 Comparison of UGAN_w using a varying number of layers
Fig. 4.10 Comparison of UGAN_w-P using a varying number of layers
Fig. 4.11 Comparison of all versions of UGAN with varying layers. Minor alterations between methods can be seen easiest around objects in the image such as the diver standing in the front. With UGAN-4, a grid effect can be seen whereas this is not as apparent in UGAN-16.
Chapter 5

Conclusion

Many natural domains explored by mobile robots contain visual noise to some degree. Vision is an attractive option due to it being non-intrusive, passive, energy efficient, as well as the primary sensing tool used by humans. This makes it a natural choice when performing tasks such as inspection, monitoring, or tracking (in any environment). When vision is used as a primary sensor in an environment which experiences visual noise, a robot’s ability to reason about that environment can degrade significantly. The underwater domain contains a vast variety of environments which experience many types of distortions based on many factors such as the time of day, weather, depth, and water quality.

Unlike many other image-to-image translation problems such as translating an edge map to a natural image or converting a grayscale image to color, underwater environments do not contain image pairs that enable supervised image-to-image translation methods applicable. Our first contribution is the application of CycleGAN and DistorterGAN in order to circumvent this issue and create a paired dataset such that supervised image-to-image translation is possible.

Following, this work presents an approach for enhancing underwater images through the use of generative adversarial networks. UGAN is able to model the complex physical properties exhibited by several sources of underwater distortion including particle scattering,
light refraction, and absorption. Rather than modeling these processes explicitly, learning a model allows for generalization to multiple environments, and therefore multiple types of distortion. Chapter 4 displayed qualitative and quantitative measurements which show UGAN’s ability to generalize to many of these environments and types of distortion.

We demonstrate the use of CycleGAN and DistorterGAN to generate a dataset of paired images to provide a training set for the proposed restoration model. Quantitative and qualitative results demonstrate the effectiveness of this method, and using a diver tracking algorithm on enhanced images of scuba divers show higher accuracy compared to the distorted image sequence. This shows the applicability of UGAN as a precursor towards underwater autonomous algorithms which use vision as a primary sensor. Furthermore, enhanced images allow human monitors to view images or videos without distortion.

Underwater domains have the unique problem of lacking a reference image during evaluation. Due to no ground truth existing for natural distorted underwater images, a non-reference based approach must be taken. We use the Underwater Image Quality Measure as our non-reference based metric, and find that the scores enhanced images receive form well with human perception. The UIQM allows for a quantitative measure between the different UGAN models.

Running this in real time on limited hardware is a primary goal. By inspecting the intrinsics of our model, we were able to decrease the computational complexity without a great sacrifice in performance. This allows for mobile hardware on Aqua 8 to process incoming frames in near real time, which is essential to many applications such as tracking or monitoring. Future work will focus on creating a larger and more diverse dataset from underwater objects, thus making the network more generalizable. Augmenting the data generated by CycleGAN with noise such as particle and lighting effects would improve the diversity of the dataset. While the datasets used in this work are constrained to ocean data,
we acknowledge that underwater scenes vary greatly across different bodies of water, and plan to incorporate those into our training sets.
References


Appendix A

Additional Examples

This Appendix provides more examples from each network for a qualitative evaluation to the reader. Samples were randomly taken from our ImageNet test set.
Fig. A.1 Samples from UGAN with 16 layers using the Wasserstein metric.
Fig. A.2 Samples from UGAN with 16 layers using the GAN metric.
Fig. A.3 Samples from UGAN with 12 layers using the Wasserstein metric.
Fig. A.4 Samples from UGAN with 12 layers using the GAN metric.
Fig. A.5 Samples from UGAN with 10 layers using the Wasserstein metric.
Fig. A.6 Samples from UGAN with 10 layers using the GAN metric.
Fig. A.7 Samples from UGAN with 8 layers using the Wasserstein metric.
Fig. A.8 Samples from UGAN with 8 layers using the GAN metric.
Fig. A.9 Samples from UGAN with 4 layers using the Wasserstein metric.
Fig. A.10 Samples from UGAN with 4 layers using the GAN metric.
Fig. A.11 Samples from UGAN with 16 layers using the Wasserstein metric and Gradient Difference Loss.
Fig. A.12 Samples from UGAN with 16 layers using the GAN metric and Gradient Difference Loss.
Fig. A.13 Samples from UGAN with 12 layers using the Wasserstein metric and Gradient Difference Loss.
Fig. A.14 Samples from UGAN with 12 layers using the GAN metric and Gradient Difference Loss.
Fig. A.15 Samples from UGAN with 10 layers using the Wasserstein metric and Gradient Difference Loss.
Fig. A.16 Samples from UGAN with 10 layers using the GAN metric and Gradient Difference Loss.
Fig. A.17 Samples from UGAN with 8 layers using the Wasserstein metric and Gradient Difference Loss.
Fig. A.18 Samples from UGAN with 8 layers using the GAN metric and Gradient Difference Loss.
Fig. A.19 Samples from UGAN with 4 layers using the Wasserstein metric and Gradient Difference Loss.
Fig. A.20 Samples from UGAN with 4 layers using the GAN metric and Gradient Difference Loss.
Appendix B

Network Architectures

Here we show detailed architectures of the UGAN variations. “Convolution” is represented as “Conv”, “Transpose Convolution” is represented as “T Conv”, “Leaky ReLU” is represented as “L-ReLU”, and “Fully Connected” is represented as “FC”.

UGAN-4 Generator Encoder

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Stride</th>
<th>Filters</th>
<th>Batch Norm</th>
<th>Activation</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>256 × 256</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>64</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>128 × 128</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>128</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>64 × 64</td>
</tr>
</tbody>
</table>

UGAN-4 Generator Decoder

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Stride</th>
<th>Filters</th>
<th>Batch Norm</th>
<th>Activation</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>64</td>
<td>No</td>
<td>ReLU</td>
<td>128 × 128</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>3</td>
<td>No</td>
<td>Tanh</td>
<td>256 × 256</td>
</tr>
</tbody>
</table>
## UGAN-8 Generator Encoder

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Stride</th>
<th>Filters</th>
<th>Batch Norm</th>
<th>Activation</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>256 × 256</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>64</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>128 × 128</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>128</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>64 × 64</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>256</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>32 × 32</td>
</tr>
<tr>
<td>FC</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>512</td>
</tr>
</tbody>
</table>

## UGAN-8 Generator Decoder

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Stride</th>
<th>Filters</th>
<th>Batch Norm</th>
<th>Activation</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>No</td>
<td>ReLU</td>
<td>262144</td>
</tr>
<tr>
<td>Reshape</td>
<td>-</td>
<td>-</td>
<td>256</td>
<td>No</td>
<td>None</td>
<td>32 × 32</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>128</td>
<td>No</td>
<td>ReLU</td>
<td>64 × 64</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>64</td>
<td>No</td>
<td>ReLU</td>
<td>128 × 128</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>3</td>
<td>No</td>
<td>Tanh</td>
<td>256 × 256</td>
</tr>
</tbody>
</table>
### UGAN-10 Generator Encoder

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Stride</th>
<th>Filters</th>
<th>Batch Norm</th>
<th>Activation</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>256 × 256</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>64</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>128 × 128</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>128</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>64 × 64</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>256</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>32 × 32</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>512</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>16 × 16</td>
</tr>
<tr>
<td>FC</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>512</td>
</tr>
</tbody>
</table>

### UGAN-10 Generator Decoder

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Stride</th>
<th>Filters</th>
<th>Batch Norm</th>
<th>Activation</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>No</td>
<td>ReLU</td>
<td>131072</td>
</tr>
<tr>
<td>Reshape</td>
<td>-</td>
<td>-</td>
<td>512</td>
<td>No</td>
<td>None</td>
<td>16 × 16</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>256</td>
<td>No</td>
<td>ReLU</td>
<td>32 × 32</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>128</td>
<td>No</td>
<td>ReLU</td>
<td>64 × 64</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>64</td>
<td>No</td>
<td>ReLU</td>
<td>128 × 128</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>3</td>
<td>No</td>
<td>Tanh</td>
<td>256 × 256</td>
</tr>
</tbody>
</table>
### UGAN-12 Generator Encoder

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Stride</th>
<th>Filters</th>
<th>Batch Norm</th>
<th>Activation</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>256 × 256</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>64</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>128 × 128</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>128</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>64 × 64</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>256</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>32 × 32</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>512</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>16 × 16</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>512</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>8 × 8</td>
</tr>
<tr>
<td>FC</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>512</td>
</tr>
</tbody>
</table>

### UGAN-12 Generator Decoder

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Stride</th>
<th>Filters</th>
<th>Batch Norm</th>
<th>Activation</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>No</td>
<td>ReLU</td>
<td>32768</td>
</tr>
<tr>
<td>Reshape</td>
<td>-</td>
<td>-</td>
<td>512</td>
<td>No</td>
<td>None</td>
<td>8 × 8</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>512</td>
<td>No</td>
<td>ReLU</td>
<td>16 × 64</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>256</td>
<td>No</td>
<td>ReLU</td>
<td>32 × 32</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>128</td>
<td>No</td>
<td>ReLU</td>
<td>64 × 64</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>64</td>
<td>No</td>
<td>ReLU</td>
<td>128 × 128</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>3</td>
<td>No</td>
<td>Tanh</td>
<td>256 × 256</td>
</tr>
</tbody>
</table>
### UGAN-16 Generator Encoder

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Stride</th>
<th>Filters</th>
<th>Batch Norm</th>
<th>Activation</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>256 × 256</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>64</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>128 × 128</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>128</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>64 × 64</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>256</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>32 × 32</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>512</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>16 × 16</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>512</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>8 × 8</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>512</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>4 × 4</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>512</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>2 × 2</td>
</tr>
<tr>
<td>Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>512</td>
<td>Yes</td>
<td>L-ReLU</td>
<td>1 × 1</td>
</tr>
</tbody>
</table>

### Generator Decoder

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kernel</th>
<th>Stride</th>
<th>Filters</th>
<th>Batch Norm</th>
<th>Activation</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>512</td>
<td>No</td>
<td>ReLU</td>
<td>2 × 2</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>512</td>
<td>No</td>
<td>ReLU</td>
<td>4 × 4</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>512</td>
<td>No</td>
<td>ReLU</td>
<td>8 × 8</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>512</td>
<td>No</td>
<td>ReLU</td>
<td>16 × 16</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>256</td>
<td>No</td>
<td>ReLU</td>
<td>32 × 32</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>128</td>
<td>No</td>
<td>ReLU</td>
<td>64 × 64</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>64</td>
<td>No</td>
<td>ReLU</td>
<td>128 × 128</td>
</tr>
<tr>
<td>T Conv</td>
<td>4 × 4</td>
<td>2</td>
<td>3</td>
<td>No</td>
<td>Tanh</td>
<td>256 × 256</td>
</tr>
</tbody>
</table>