DEPARTMENT OF COMPUTER SCIENCE

Generative Deep Learning for Temporally Consistent Underwater Image Enhancement

George Atkinson

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering.

Thursday 9th May, 2024

Abstract

The task of image enhancement is an important area of research in computer vision. Restoring an image could enable access to information that was otherwise lost, and could be utilized to bolster the performance of further computer vision related research. This process is made considerably more difficult for underwater images because of the occurrence of complex non-linear degradation underwater. Light absorption leads to decreased image intensities and a predominantly blue and green color distribution. Additionally, scattering causes distant objects to appear hazy, and backscattering causes noise in the image. The results of these phenomena are degraded underwater images that inhibit the performance of remotely operated vehicles (ROVs) and autonomous underwater vehicles (AUVs) which are crucial to underwater exploration, maintenance and infrastructure development. By reversing these effects, the gap between in-air and underwater computer vision tasks will be closed, allowing improved performance of processes such as object detection.

These effects are unlikely to ever be solved purely by a more capable camera, so they must be addressed using some algorithm to enhance and restore the images. This project comprehensively explores the usage of deep generative image-to-image translation models for underwater image enhancement (UIE), with an additional interest in achieving improved temporal consistency for video. To achieve this, a large data collection period was required in order to gather diverse paired and unpaired data, allowing both paired and unpaired model training. Finally, the models were extended with the goal of improving the temporal consistency achieved by the models.

The primary contributions of this project are as follows:

- Gathered a diverse unpaired dataset for UIE.
- Generated a hybrid synthetic dataset using both underwater and in-air images, by deploying state-of-the-art attenuation estimates with utilization of state-of-the-art depth estimations.
- Comprehensively studied the performance of image-to-image translation models on UIE
- Modified model architecture to permit video-to-image and video-to-video translation.
- Altered model objective functions in attempt to further correct spatial and temporal consistency.

Dedication and Acknowledgements

- I would like to express my gratitude to my supervisor, Dr Pui Anantrasirichai, for the guidance offered both related and unrelated to the project.
- Also, I would like to show appreciation to my family and friends for supporting me throughout this project, as well as occasionally providing a much needed distraction from it.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

George Atkinson, Thursday 9th May, 2024

Contents

List of Figures

List of Tables

Ethics Statement

This project did not require ethical review, as determined by my supervisor, Dr. Pui Anantrasirichai.

Supporting Technologies

This project utilized the following technologies:

- Model training was bolstered by the usage of the University of Bristol high performance computer (HPC), BluePebble, as its advanced hardware was used to improve training times.
- A range of Python versions were used throughout, as well as PyTorch's tools for creating deep learning frameworks. Also, NumPy, OpenCV, matplotlib, and scikit-learn were frequently used for handling the image data, and collecting appropriate graphs and metrics.
- Image synthesis was done by using the method outlined in SeaThru [2], with the inclusion of depth estimations from the pre-trained Depth Anything model[53]. The pre-trained model for Depth Anything can be accessed at https://github.com/LiheYoung/Depth-Anything.
- All model frameworks were based on the architecture provided by Pix2Pix [25], CycleGAN [59], and UVCGAN [44]. CycleGAN and Pix2Pix can both be accessed at https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix whilst UVCGAN's framework is available at https://github.com/LS4GAN/uvcgan.

Notation and Acronyms

- RGB: Red, Green, Blue
- RGB-D: Red, Green, Blue, Depth
- UIE: Underwater Image Enhancement
- ROV: Remotely Operated Vehicle
- AUV: Autonomously Operated Vehicle
- MLP: Multi-Layer Perceptron
- DNN: Deep Neural Network
- CNN: Convolutional Neural Network
- GAN: Generative Adversarial Network
- cGAN: Conditional Generative Adversarial Network
- HPC: High Performance Computing
- ReLu: Rectified Linear Unit
- GeLU: Gaussian Error Linear Unit
- Adam: Adaptive Moment Estimation
- ViT: Vision Transformer
- MSE: Mean Squared Error
- PSNR: Peak Signal Noise Ratio
- SSIM: Structural Similarity Index
- SAUD: Subjectively Annotated Underwater Dataset

Chapter 1

Introduction

1.1 Motivation

It is well known that technology has the potential to spark significant scientific developments and innovations. In the world of underwater exploration, this is no different, there has been a significant popularisation of remotely operated vehicles (ROVs) and autonomous underwater vehicles (AUVs). With the utilization of such machines, divers can be replaced with state of the art technology, designed to reduce risk to human lives, introduce autonomy, and address the depth-limitations of humans. [26] [22].

ROVs and AUVs are robotic devices used for exploring and performing tasks underwater. They have a wide range of applications, such as monitoring of marine species migration [42], inspection of damages [8], and underwater scene analysis. [24] Although they have their differences, for either of these to exist in a functional manner, they must both have some way of detecting their local environment in order to navigate it. Whilst other devices, such as sonar sensors [35], have been experimented with, their output is too noisy and low resolution (especially in deep water[22]) which has ultimately led to the use of optical cameras for effective navigation.

Whilst the goals of ROVs and AUVs can be diverse, they often suffer from the same technical issues. One of the biggest issues these machines face is that despite the top-grade camera systems they deploy, the images produced suffer from a loss of quality due to the inherent properties of water. Crucially, whether being used for marine biology, navigation, or 3D modelling, the degrading of the images can significantly inhibit the machine's ability to achieve its goal. Since it is likely that this problem will never be fixed by fitting a more capable camera, it is now common to look to image enhancement for improvements.

Image enhancement is a prominent area of research in computer vision which has seen lots of success in reversing image degradation. In recent years, it has further advanced thanks to innovations in machine learning, with generative models becoming ever more popular. However, the field of underwater image enhancement (UIE) has proven to be less straight-forward than most because of the unique conditions and terrain. In fact, it is ultimately an ill-posed problem, meaning that there is no ground truth. As a result, it is very difficult, if not impossible, to determine what makes a good underwater image.

The aim of this project is to experiment with using generative deep learning models, with a focus on image-to-image translation models, to help improve underwater images, and explore how these methods can be developed to better the temporal consistency exhibited when applied to videos. Additionally, the methods will have their performance analyzed using metrics for perceptual quality specific to both images and videos.

1.2 Solution

This report explores the problem described above, before proposing potential solutions using deep learning methods, as well as analysing the performance of the proposed solution in comparison to other existing methods. Generative Adversarial Networks (GANs) will be used to learn image-to-image and video-to-image translations that can proficiently map a bad underwater image/video in to a good one. The output

will also be color corrected, with restored detail, as well as displaying temporal consistency on videos.

Due to the absence of ground truth, the use of both real underwater images and synthetic underwater images will be explored using paired and unpaired translation models. Underwater image enhancement is an ill-posed problem (i.e. there is no ground truth), so the use of synthetic data (using SeaThru [2]) and unpaired training, is explored. The use of synthetic data allows the method to utilize paired methods, as paired data is emulated by adding discoloration, haze, and noise to otherwise clear images. In this model, there are a number of models being proposed. All paired models will be trained on a mix of both underwater and in-air images, whilst unpaired models will be limited to only underwater data. The use of Pix2Pix [25], CycleGAN [59], and UVCGAN [44], will be explored. One proposed method is to extend the model to learn a video-to-image mapping so that the models are no longer limited to monocular input. Another idea is to use DepthAnything [53] to provide an accurate depth estimation, which could improve the model's understanding of the scene. Additionally, the usage of a method with concern for temporal loss is explored to attempt to improve the model's performance on videos.

1.3 Project Objectives

At a high level, the project aims to develop a method for UIE, with extra emphasis on temporal consistency. This process will be broken down into more steps, as follows:

1. Collect an unpaired dataset of "good" and "bad" underwater images and videos.
2. Generate synthetic "bad" underwater images, from both in-air and "good" underwater images, producing a paired dataset.
3. Train an unpaired image-to-image and video-to-image translation model to learn mappings from the "bad" domain to the "good" domain.
4. Train a paired image-to-image and video-to-image translation model to learn mappings that can reverse the effects of the image synthesis, whilst generalizing to real "bad" underwater images.
5. Apply the models to UIE benchmark datasets, and compare performance with other state-of-the-art models.
6. Analyze the effectiveness of the models with respect to temporal consistency.

1.4 Challenges

1.4.1 Storage

Through the duration of this project, there were a number of problems encountered. Due to the requirement for high computational throughput when training models, the University of Bristol's HPC, BluePebble, was used for GPU intensive tasks. Whilst incredibly helpful, there are a large number of others that require its usage, which results in long queue times for running jobs. For this reason, this slowed down the development of models, as it prolonged the debugging process. As a result, less model's were trained, and less hyperparameters were experimented with. Additionally, it has a limited amount of storage, which was 1 Terabyte per user, which often meant the dataset often had to be made smaller than what would have otherwise been preferred. The issue of storage was also one that plagued my personal laptop, as with the constant flow of model's and data, it frequently ran out of memory.

1.4.2 Model Comparisons

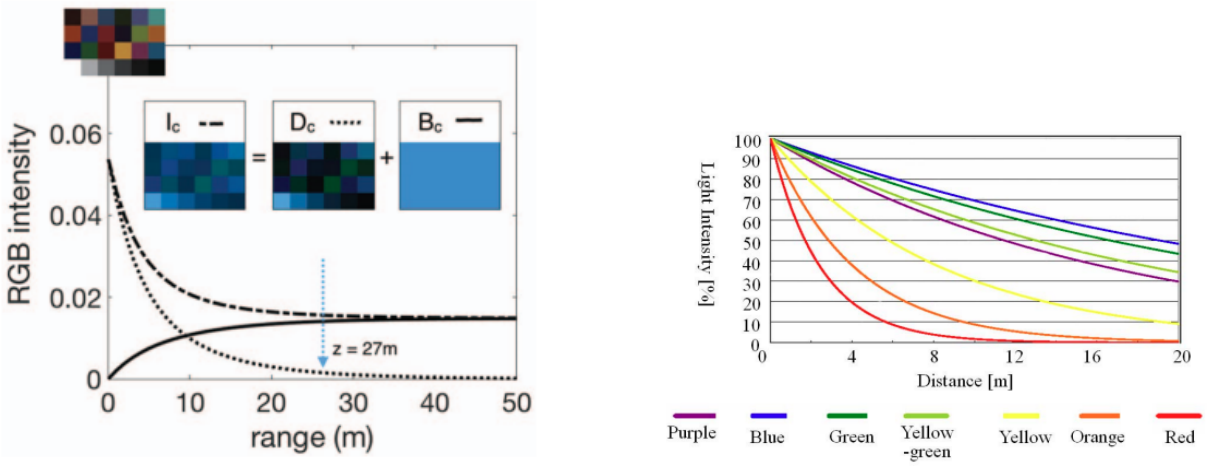
Naturally the best way to assess the performance of a model or algorithm is to compare it with other pre-existing models. All other underwater synthetic underwater datasets were published on Baidu, which is inaccessible for me as a UK citizen because it requires a Chinese or other international phone number. Similarly, this was the case for some of the pre-trained models, which meant they had to be left out of the comparisons in Chapter 4. In addition, the models that were actually accessible were often implemented in MATLAB, a language in which I am unknowledgeable. For this reason, the processing times of other models could have potentially been improved, but this is hard to conclude.

Chapter 2

Background

2.1 Underwater Light Attenuation

Underwater environments present an array of problems for optical imaging. Despite using state-of-the-art cameras, the images obtained by said cameras are prone to degradation due to the inescapable physical interactions between water and light. As light travels deeper underwater, more of it is absorbed by water particles, causing lower intensities, as shown in Figure 2.1a, below. Furthermore, the amount of attenuation differs for lights of different wavelengths, as shown below in Figure 2.1b. Light with a longer wavelength (red) is absorbed more readily than light with shorter wavelength (blue) resulting in the scene having a blue/green tinge [1]. Also, as light travels through water it is scattered such that it deviates from its initial path, resulting in a loss of information in images. Backscattering is the effect where light reflects back towards the camera, causing noise. This is especially problematic for AUVs and ROVs equipped with large underwater lights, as a fraction of this light gets returned to its camera system. Absorption, scattering, and backscattering are all forms of light attenuation [38]. As a result, there is a loss of energy in the light, which causes lower light intensities at greater depths, resulting in a hazy image, with lowered contrast, and loss of detail. [30]. In addition, the distortion of the images is considered non-linear which makes it extra difficult to model [57]. Consequently, no matter how sophisticated the camera, the images taken of underwater scenes often suffer from large amounts of degradation, and seem to require post-processing to restore the image quality. [57] [16].



(a) [2]

(b) [52]

Figure 2.1: Graphs showing the effect of water depth on the color distribution of images, and showing the effect of water depth on the intensity of observed colors, left and right, respectively.

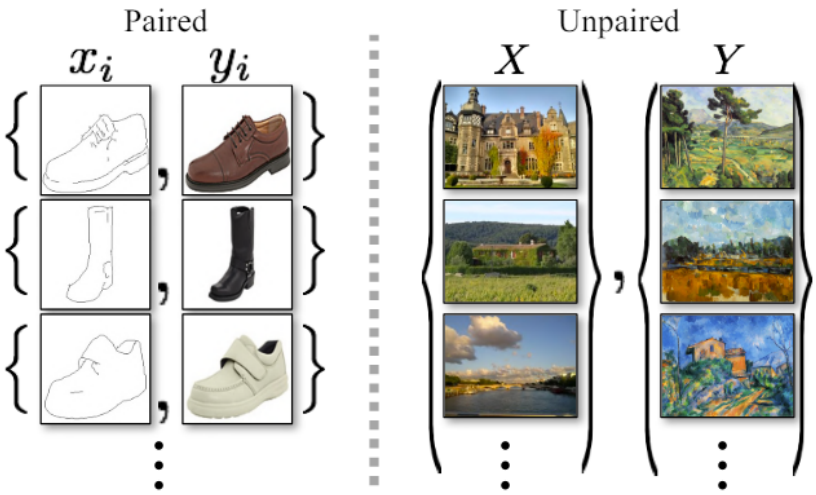


Figure 2.2: A diagram demonstrating examples of paired (left) data versus unpaired data (right). On left side, each shoe is paired with the corresponding shoe outline. On the right side, landscape photos are poised against Monet paintings in an unaligned fashion [59].

2.2 Paired Data vs Unpaired Data

A problem inherently at the core of UIE, and what separates it from many similar enhancement tasks, is the lack of paired data available. Paired data means that every piece of input data has an associated correct output, also known as a ground truth. For example, in a dataset used for classifying images containing cars, this could look like a set of images, each assigned a set of bounding boxes that identify the position (if any) of any cars in the image. A model could then use this data to make predictions and directly compare its output to a correct result, and attempt to improve itself. This is called supervised learning. For image-to-image translations, this could be an image of a scene in low-light, and an image of the exact same scene in good lighting. From this, a model could learn to compensate for the bad lighting in the image. However, in the realm of UIE the problem is more challenging. Underwater environments are constantly changing as things move, water conditions are variable, and it is difficult for any camera to hold the same position. Furthermore, the logistics of accessing scenes at these depths can be costly and potentially dangerous. Despite the popularity of supervised learning for a wide-range of applications, it is not always suitable. This gives rise to the use of unpaired data and unsupervised learning. Whilst paired data has a defined output, unpaired data is more abstract. Reusing the low-light image enhancement analogy, the data could be a set of input images taken in dark scenes, as well as a set of output images containing unique bright scenes. An unsupervised learning model would then attempt to learn a relationship between the two sets of images by learning about the underlying structure and distribution of the data.[9]

2.3 Deep Learning

It is largely thanks to the recent boom in machine learning that there are now an abundance of exceptional new methods available for computer vision processing, including object detection, image segmentation, and image enhancement. Machine learning allows the computer to learn from experience, allowing it to form its own understanding of data by building an extensive network of concepts which are defined by their links to more simple concepts [15]. Deep learning is the subset of machine learning which is predominantly used in modern approaches for UIE [10]. Deep learning, from a high-level view, is just a machine learning model (in particular a neural network) but one that is allowed more layers, to allow it to make deeper inferences, with its deeper understanding enabling it to pick up more complex relationships. It is this ability to form their own understanding that allows deep learning models to excel at solving more abstract problems, the kind of problems which require a more human-like intuition, the kind of problems where a solution is hard to articulate.

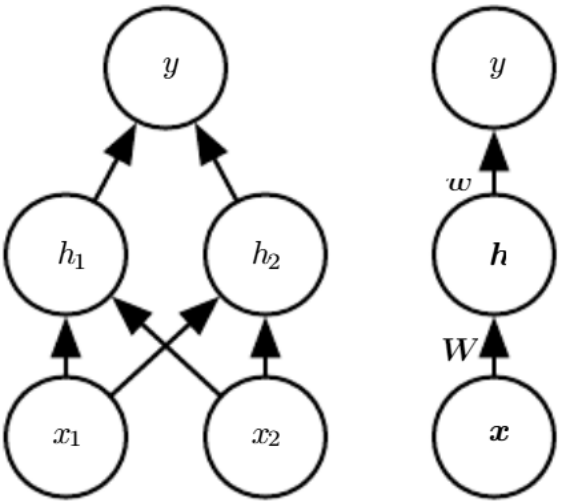


Figure 2.3: An image containing a simple MLP, in two different formats. The left shows an MLP with two inputs, two hidden layers, and one output layer, with each layer connected. The right shows a more compact version of this diagram, in which the layers are shown as one node that are a vector of the individual nodes. [15]

2.3.1 Multi Layer Perceptron (MLP)

The Multi Layer Perceptron (AKA feed-forward deep network) is the classic example used for both supervised and unsupervised deep learning; it is useful to understand the MLP before trying to understand more complex deep learning models. Put simply, it is just a function that maps a group of input values to a group of output values by a mathematical function. The input is mapped to the output by a large number of simpler mathematical functions which combine to represent more complex relationships. Each function belongs to a node, each node belongs to a layer, and the more layers there are, the deeper the inferences that can occur, but, they also become more expensive to train and run. The first layer is the input layer, the last layer is the output layer, and those in-between are the hidden layers. Each layer can be thought of as a vector of weights, biases, and activation functions. For example a node might contain something like

$$f(W^T x + b) \tag{2.1}$$

where W is a vector of weights, x is the vector of inputs, b is bias, and f is an activation function. The size and quantity of these layers, as well as the activation functions they contain, are amongst the hyper-parameters the creator must define. Hyper-parameters can greatly affect the performance of a model, and therefore should be picked carefully. Moreover, a grid search can be used to experiment with a number of different combinations of hyperparameters, from which the best performing can be used. However the weights and biases are values to be learned when training an MLP. Also, the model contains one or more loss functions which are used in training to assess the correctness of the predictions. Finally, they contain an optimizer which uses the results of the loss function to improve the model.[15].

In some ways, MLPs/neural networks behave a bit like the human brain. The nodes of a neural network are analogous to the neurons in the human brain, connected to thousands of others in a complex circuit. When one neuron is activated, it will spark another one, and another one, and another one, allowing complex relationships to be formed. Throughout the rest of this project, the terms node and neuron will be used interchangeably. Unarguably, the human brain excels in inference, intuition and perception. However, brains are also slow, prone to errors, and extremely difficult to understand [4]. Similarly MLPs, and deep models in general, become extremely difficult to understand. For example, in the same way it is difficult to kick a bad habit, it is also difficult for deep networks to unlearn a bad mapping. For this reason it is important that the data used is carefully selected, and techniques such as dropout and cross-validation are used to ensure the model continues to generalize well to new data.

2.3.2 Training an MLP

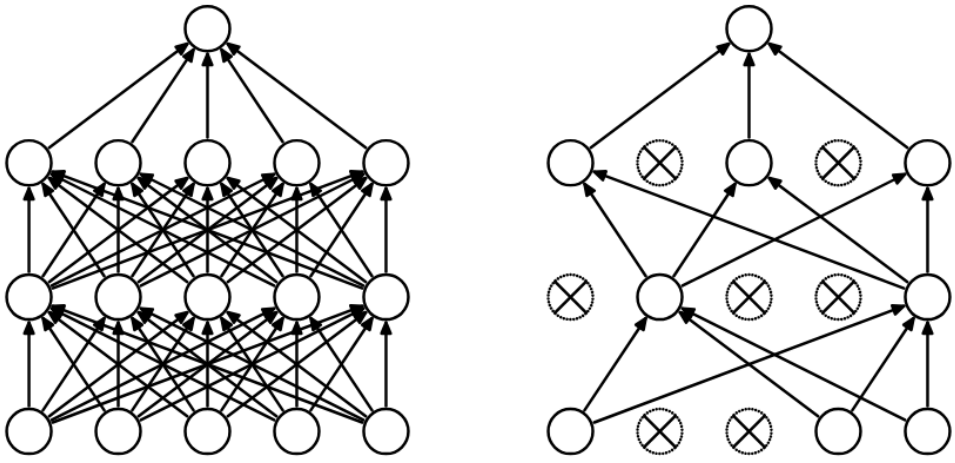
Training an MLP can be summarised into 2 steps: forwards propagation and backwards propagation. The first step, consists of an input being passed into the neural network, allowing it to be processed layer by layer until the output layer has been updated. Then, during backwards propagation, the results of the model are compared with the ground truth values to calculate the error in the model's predictions. Then the loss function's gradient is calculated with respect to every neuron's weight and bias. Finally, the optimizer uses gradient descent to update the weights and biases to minimize the model loss. This process is repeated until a certain number of epochs have been completed or the model seems to have converged.

2.4 Deep Network Architecture

As explained above, a neural network's architecture can be summarized into a sequence of hidden layers wedged between an input and output layer. The types of layers used differ depending on the use-case. Furthermore, neural networks can use a variety of techniques to ensure faster training or improved performance. This section will attempt to briefly outline the techniques and layer types that will be referenced throughout the remainder of this report.

2.4.1 Initial Parameters

For gradient descent based machine learning methods, data must be able to move through the network smoothly. If the parameter scale is poorly scaled, the data can vanish, or it can explode. As a result, the model would reach an early divergence as it gets stuck at a bad result. For this reason, it is of utmost importance to initialize the model with a good set of parameters, as a bad initialization can hinder the model's ability to learn anything. Furthermore, even if vanishing or exploding gradients aren't an issue, it is thought that the choice of initialization can impact the model's final divergence. Moreover, it allows the model to use a higher learning rate, which has been shown to allow models to reach a better divergence. In addition, model's often use weight normalization in an attempt to inhibit vanishing and exploding gradients [6].



(a) Standard Neural Net (b) After applying dropout.

Figure 2.4: A diagram showing dropout in action on an MLP [43]

2.4.2 Dropout

Over-fitting is when a model fits itself to the noise of the training data; when an over-fitted model is given new unseen data, it generalises poorly due to the bad habits it has picked up. This can cause the model to diverge early, resulting in a poor generalization. Many smaller networks trained on unique subsets of the training data can be used in an ensemble such that the inaccuracies caused by over-fitting in each individual model cancel each other out [9]. Although elegant, this method cannot efficiently be applied

to deep neural networks (DNNs) due to their sheer size. Dropout is a technique introduced by Srivastava et al. that emulates the effect of ensembling smaller networks during the training on a singular deep network. During training, dropout randomly deactivates a fraction of the neurons in a layer. By using unique subsets of the nodes, they become less likely to make complex co-adaptations because each node is allowed to have more independence. With this, the model is less likely to fit to the noise of the training data, resulting in less over-fitting. [43]

2.4.3 Activation Functions

Modelling non-linearities is a complex problem that has evaded mathematicians for years. The section above explores the suitability of deep neural networks for learning these complex relationships, but without the presence of activation functions, even the deepest neural network would behave as linearly as a single-layer perceptron (MLP with 1 hidden layer). Activation functions are mathematical functions that determine the output of a neuron or layer in the DNN. The simplest activation function is the linear activation: $f(x) = x$. With just this, the networks would still be linear, which causes the existence of an array of alternatives such as sigmoid, tanh, and three others which will be examined more closely [15].

The vanishing gradient problem is one that plagues DNNs, causing them to overfit to the training data. This problem describes the stagnation of training caused by the gradients in the DNN becoming too small, which results in slower weight updates [19]. The rectified linear unit (ReLU) attempts to counteract this problem; ReLU takes the form of $f(x) = \max(0, x)$ which means that values greater than 0 are themselves, and any negative value becomes 0. This means that for all positive values, the gradient of ReLU is 1, meaning that the gradient cannot become small enough to stagnate (unless they are all 0, but typically they are not). In addition, this makes the gradient more easily computed since ReLU is essentially just the linear activation function, except it outputs 0 for half of its domain. Leaky ReLU is an extension of ReLU, which is defined as $f(x) = \max(\alpha \cdot x, x)$, where α is a hyperparameter defined at runtime, usually as a value like 0.1. Whilst maintaining non-linearity, this means that negative values have a smaller positive gradient, which stops the model from stagnating due to small gradients [51]. In addition, the Gaussian error linear unit (GeLU) is defined as $f(x) = x\Phi(x)$, where $\Phi(x)$ is the standard Gaussian cumulative distribution function. This method offers an alternative to ReLU that models non-linearity based on the inputs value rather than its sign (i.e. positive or negative).[18].

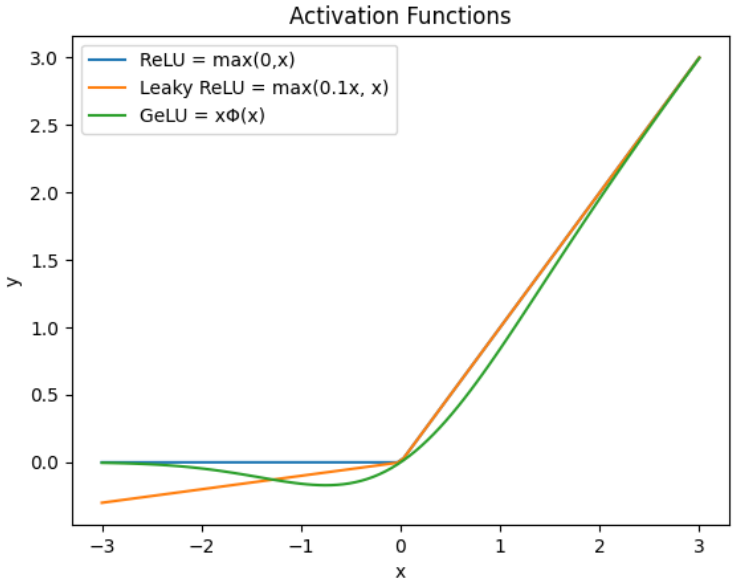


Figure 2.5: Graph showing ReLU, leaky ReLU, and GeLU plotted against each other. For ReLU, note the 0-gradient below x values of 0, and linear gradient otherwise. For leaky ReLU ($\alpha = 0.1$), notice the same as ReLU for x values greater than 0, but allowing a slightly positive gradient below that point. Finally, observe how GeLU allows a decaying positive gradient for $0 > x > -1$, which alternates to a negative gradient for larger negative values.

2.4.4 Optimizers

During training, neural networks attempt to process the data, and then back-propagate to update their weights depending on the output of the objective function. The optimizer is the method that is used to update the weights of each neuron. The most common and simple method for optimization is stochastic gradient descent. Like most optimizers, it uses a learning rate to determine how significantly the weights should be altered. To progress this, the learning rate decays over time to allow big changes near the start of training, and encouraging convergence later in training. In more advanced optimizers, a momentum variable is introduced to give more recognition to consistent and noisy gradients. The inclusion of momentum helps the optimizer to accelerate in the correct direction by allowing consistent or extreme gradients to maintain a significant impact. Adaptive moment estimation (Adam) is one of the most popular optimizers for DNNs. It uses an adaptive learning rate, as well as bias correction to ensure good, robust learning even from sparse and noisy data [29] [15]

2.4.5 Batch Normalization

Internal covariate shift is a phenomenon that occurs due to the changing of parameter distributions during training. This slows down model training by causing lower learning rates and more careful parameter initialization. Batch Normalization is a technique introduced by Ioffe et al. which aims to reduce the effect of internal covariate shift by normalizing each batch of inputs during training. With the insertion of a BatchNorm layer between two layers, a layer that previously received x will now receive $BN(x)$ which is x with some linear transform, scale, and/or shift applied. With this, the network can use higher learning weights, place less dependency on techniques such as dropout, and worry less about the initial network parameters. [23]

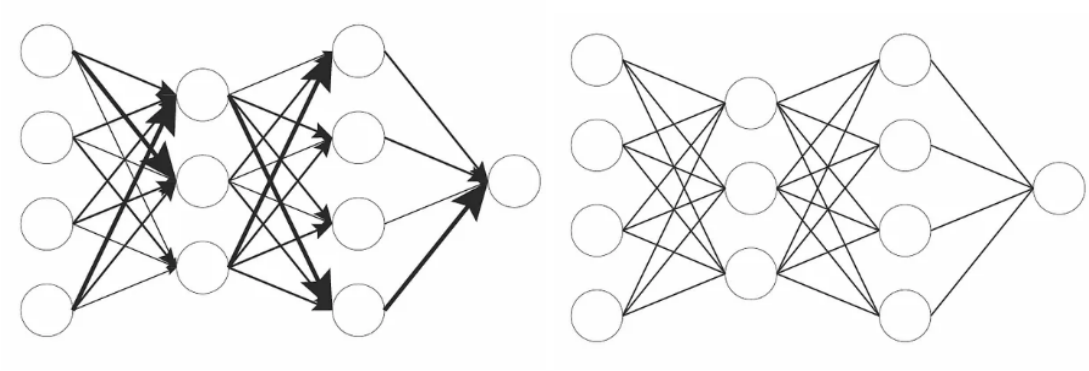


Figure 2.6: Diagrams showing an MLP with no batch normalization, left, and an MLP with batch normalization, right. The regularized arrow sizes demonstrate the normalized signal that leads to less inter-dependencies. [21]

2.5 Convolutional Neural Networks (CNNs)

Convolution is a linear mathematical operation which takes an input of two functions, f and g, and produces a third function, $h = f * g$, representing how the shape of one is modified by the other [50]. The latter of the two input functions, g, is often known as a kernel/filter; it slides over the data, applying an operation at each point in the data. In the case of object classification, this kernel could be a feature map (as in 2.7), that slides over the data in search of particular feature. Also, to make this process less computationally expensive, there exists a hyperparameter called stride which determines how many units the kernel translates between each convolution. For example, with a stride of two, the convolution would be performed on every other data value.

Convolutional Neural Networks (CNNs) are one of the newer, more exciting generations of deep learning research. CNN is the term used to describe all neural networks in which at least one of their layers uses convolution instead of matrix multiplication. Thanks to their employment of convolution, a highly useful dimensional tool, CNNs often excel at learning from data with underlying, grid-like structures.

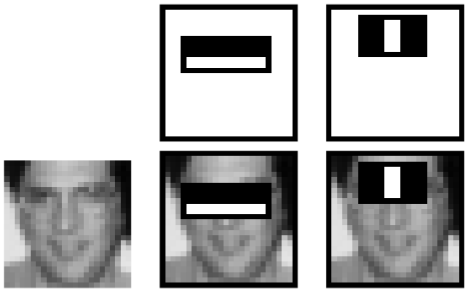


Figure 2.7: Diagram showing example convolutional feature maps that could be used for face detection [48].

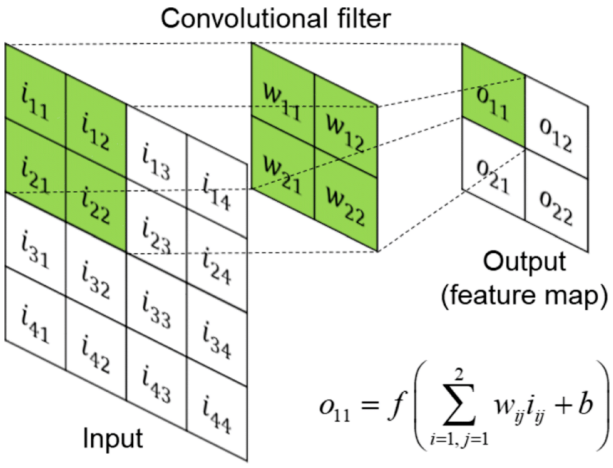


Figure 2.8: Diagram showing a 2×2 convolution kernel being applied to the top left point of an image. Also note a stride of 2, meaning the kernel shifts two pixels each iteration. [55].

Whilst using matrix multiplication means every output parameter interacts with every input parameter, convolution allows for a smaller search-space for salient features. This idea is known as sparse connectivity, which is visualized in Figure 2.11, and it means that training the model stores less parameters, and becomes less computationally expensive to train. In addition, using matrix multiplication ensures that each element of a node's weight matrix is used exactly once, whereas, when using convolution every weight value is used on every input value. With this, known as **parameter sharing**, it means that the CNN becomes less memory intensive as it can store less parameters. The last relevant property exhibited by CNNs, is **equivariance**. For a function to be equivariant, it means that if some function were applied to the input, then the same changes would be observed in the output, i.e. $f(g(x)) = g(f(x))$ [15].

As described above, a CNN is a neural network containing at least one convolutional layer. The following explains how this actually works: In general, a convolutional layer can be viewed as one complex layer, but it sometimes offers more clarity to break it down into three simpler layers:

1. **Convolution stage:** The convolution filter is slid along the data producing a number of linear activation functions representing the strength of each feature at each input point. In this stage, the size of the output data becomes smaller than the input data. Specifically, after a kernel of size x is applied to data of size y with stride s, the output node now has size

$$|c(\text{input})| = \frac{y - x}{s} + 1$$

   For example if the pooling layer takes an input of size 10, and is outputting the mean of each of itself and its direct neighbours (i.e. kernel size of 3), the output size can be reduced to size 8 because it no longer needs to store the two edge values.

2. **Detector stage:** The outputs from the convolution stage are passed through a non-linear activation function.

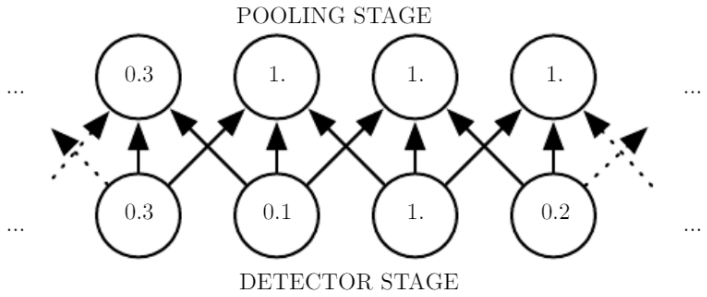


Figure 2.9: A diagram showing an example of max-pooling. [15]

3. **Pooling stage:** Finally, the outputs are processed using a pooling function. Generally, a pooling function replaces the output of the detector layer with a statistic summarising itself and other local outputs. For example, this could be a mean value of all outputs within a 3x3 radius. As a result, some of the positional information of the detections are lost, however, the detections are now more robust to small translations in the data. Because the pooling is summarising multiple nodes, it means that this layer can have fewer nodes. The output size works in the same way as in the convolution stage, with an output size of

$$|p(c(\text{input}))| = \frac{y - x}{s} + 1$$

As explained, the output dimensions of a convolutional layer are smaller than the input dimensions. For this reason, there is a possibility of the dimensions falling too low, resulting in a loss of spatial information on the features. For this reason, padding is often added to the input. Padding is basically just adding a frame of filler numbers around the data - these numbers can just be zeros. By doing this, the output size of the layer is controlled, instead of severely compromising the spatial information of the data. The process of the data's spatial information diminishing is known as down-sampling. On the contrary, up-sampling is where the spatial dimensions of the data are increased. This can be done in various ways, like with padding, or by repeating every data value to double its size (nearest neighbour up-sampling). In most image-related tasks, the networks use a type of up-sampling called bi-cubic interpolation which uses a large neighbourhood of data, and fits a polynomial to predict the expected value of a pixel. [15] [54] [41]

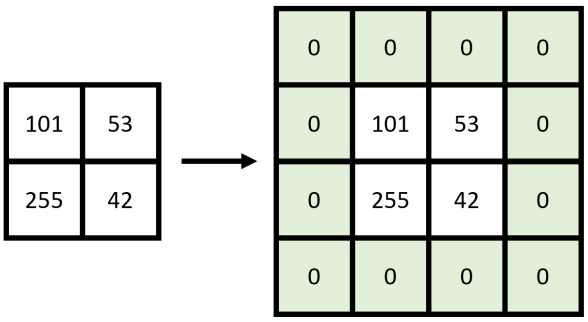


Figure 2.10: Diagram showing zero-padding being applied to a 2x2 matrix.

2.6 U-Net Architecture

In the past, training CNNs could require millions of training images. To gain access to such a large quantity of data is a difficult and expensive task. For this reason, Ronneberger et al. introduced a type of encoder-decoder architecture named U-Net. U-Net was specifically designed for biomedical image segmentation, but it is known to excel at most image-related learning. The architecture of this model

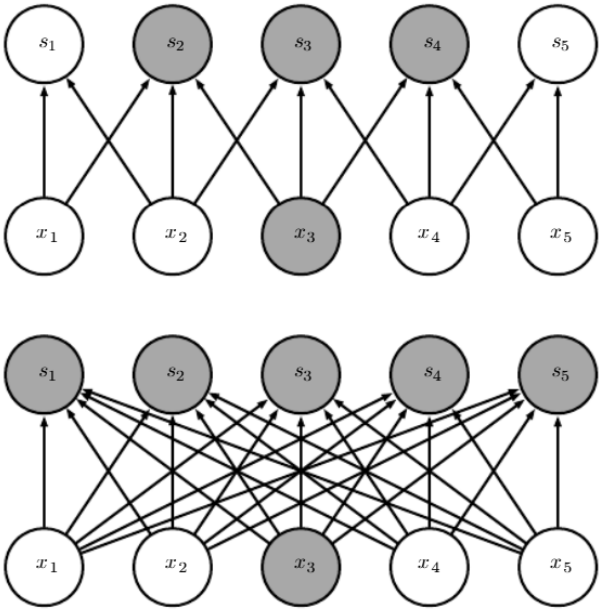


Figure 2.11: Sparse connectivity from [15]

consists of a series of down-sampling convolutional layers, and a bottleneck followed by a series of up-sampling layers. As seen in Figure 2.12, this gives the U-Net its defining U shape.[41]. Also, it contains a number of skip connections whose inclusion allows for activation functions from earlier layers to skip past a number of layers before being concatenated back at a subsequent layer. This allows the output image to retain important edge information from the input image. The benefit of using skip connections is made apparent in Figure 2.13, which shows the fitness landscape. This represents the loss at each point, as the model aims to reach the global minimum loss. Moreover, it shows that the fitness landscape becomes smoother, which makes gradient descent easier for the model.

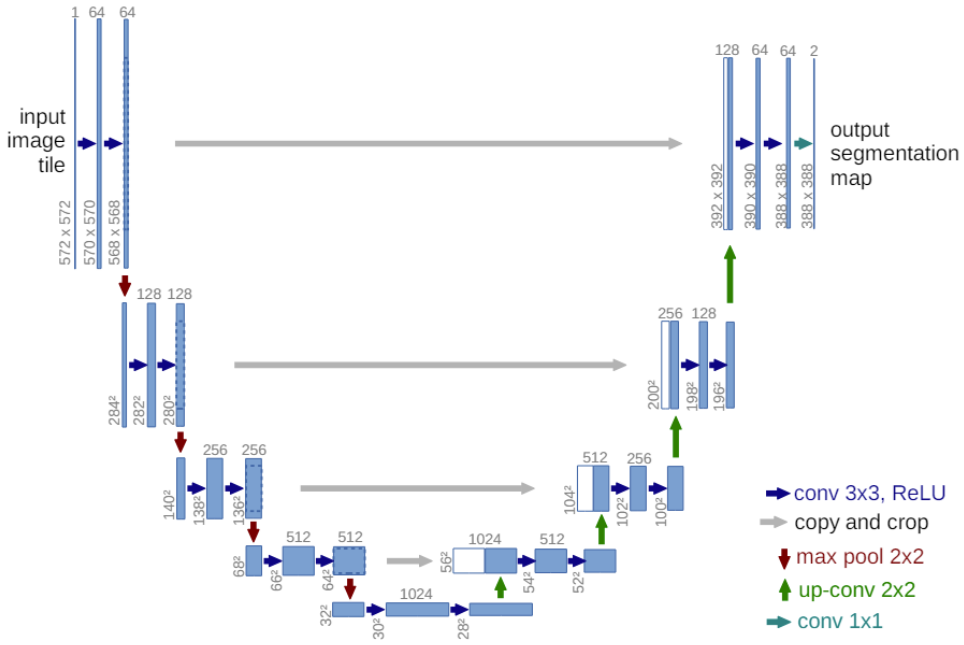


Figure 2.12: U-Net Architecture [41]

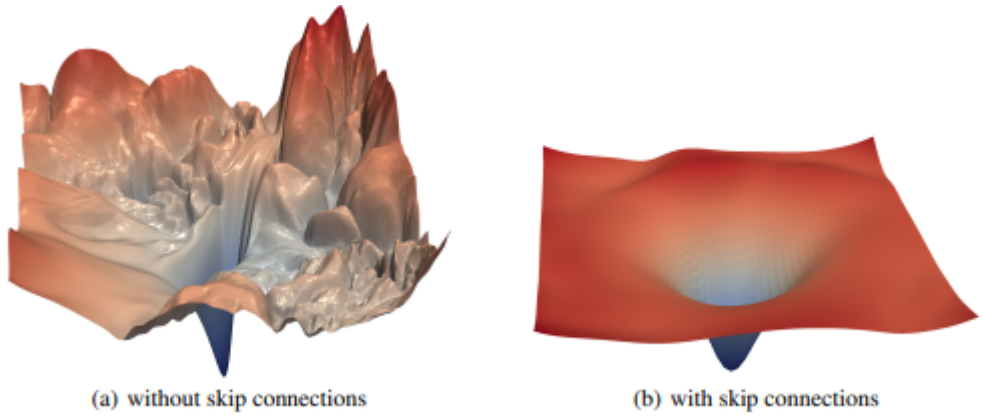


Figure 2.13: A diagram showing the fitness landscapes for a deep learning model with and without utilizing skip connections, (a) and (b), respectively [33].

2.7 Generative Adversarial Networks (GANs)

In a way, Generative Adversarial Networks (GANs) are like an extension of CNNs. In a simple scenario, a GAN can be supposed to be a pair of CNNs: a generator network, and a discriminator network. The generator, G, attempts to represent a data distribution by building a mapping between some prior noise distribution, $p_z(z)$, to the data space. With this, the output from the generator attempts to be a randomly generated bit of data that appears to be indistinguishable from the rest of the data. Alternatively, the discriminator, D, outputs a single scalar value which is a probability of whether its input is from the generator or the training data. With this, their respective loss functions can be defined: the discriminator D tries to minimise $\log(D(x))$ and the generator G tries to minimise $\log(1 - D(G(x)))$. With this the GAN loss is defined as follows:

$$\mathcal{L}_{GAN}(D, G) = \mathbb{E}_{x \sim p_{\text{data}}}\Big[\log(D(x))\Big] + \mathbb{E}_{z \sim p_z(z)}\Big[\log(1 - D(G(z))\Big] \tag{2.2}$$

Where $\mathbb{E}$ is the expectation value of the term. These models get caught up in a game-like min-max game with roots embedded in co-evolution. It can be likened to the vicious cycle of co-evolutionary competition between predator and prey. The predator kills the prey, so the prey learns to outsmart the predator, causing the predator to improve at hunting, and so on. Back to the context of GANs, this cycle is described by the following min-max equation on the GAN loss [37]:

$$\Omega_{cGAN} = \min_G \max_D \mathcal{L}_{GAN}(D, G) \tag{2.3}$$

2.8 Conditional GANs (cGANs)

Whilst GANs are suitable for randomly generating imposter images, this is not that useful for image-to-image translations. Conditional GANS (cGANs) are just like GANs, but in addition to the input of random noise, it also receives some form of auxiliary information, y, such as a class label. In the generator, the input is now a combination of random noise and the auxiliary information, and in the discriminator, the input is now the auxiliary information and the real image. With this, the new GAN loss and objective functions are as follows:

$$\mathcal{L}_{cGAN} = \mathbb{E}_{x \sim p_{\text{data}}}\Big[\log(D(x|y))\Big] + \mathbb{E}_{z \sim p_z(z)}\Big[\log(1 - D(x, G(z|y))\Big] \tag{2.4}$$

$$\Omega_{cGAN} = \min_G \max_D \mathcal{L}_{cGAN}(G, D) \tag{2.5}$$

In practice, this could be an image generation model that takes an auxiliary string variable which labels the brand of car, e.g. "Ford", "Ferrari", etc. Then, with the random noise and the label, it would generate an image that it believes to be a car of that brand. [37]

2.9 Mode Collapse Problem

Despite the success of traditional GANs, they face a critical challenge, which is known as "mode collapse." This is a phenomenon that inhibits the generator from producing diverse samples, which leads to stagnation in the competition between the generator and the discriminator. This usually occurs due to a large strength imbalance between the generator and discriminator, which causes the generator to overpower the discriminator by exploiting its weaknesses [31]. There are a number of remedies to this problem, for example, Wasserstein GAN (WGAN) was introduced by Arjovsky et al. which proposes using a Wasserstein distance that produces a continuous confidence score rather than a discrete classification [5]. Additionally, GANs can deploy a gradient penalty, which is a method that aims to keep the outputs gradient around a constant value, such as 1, in order to stop the model stagnating.

2.10 Pix2Pix

Building on the idea of cGANs is an architecture named Pix2Pix produced by Isola et al. [25] which is one of the most famous examples of an image-to-image translation model. In this architecture the generator produces an image with the auxiliary data of yet another image. The model now is trained on two sets of images, one from domain X, and the other from domain Y. Now the goal of the generator is to produce a mapping X to Y such that the result of inputting an image from domain X will result in an image that is stylistically indistinguishable from images in domain Y. This mapping is defined as function $F : X \to Y$, and say Real X and Real Y to refer to images that actually belong to the set. Finally, the impersonator of an image in set Y can be referred to as Fake Y, i.e. $G(x)$ where $x \in X$.

2.10.1 Objective Functions

Whilst the GAN loss provides a sound method for emulating the style of a dataset, only having a GAN loss would mean that the images produced by the model would be allowed to go further astray from the ground-truth - leading to less accuracy and loss of structure. For this reason, Pix2Pix employs an L1 loss to supplement the GAN loss for the objective function. L1 loss can be defined as

$$\mathcal{L}_{L1} = \sum |y_{\text{true}} - y_{\text{predicted}}| \tag{2.6}$$

Or in GAN terms:

$$\mathcal{L}_{L1} = \sum \mathbb{E}_{x,y,z}\Big[||y - G(x,z)||\Big] \tag{2.7}$$

Resulting in a new objective function of:

$$\Omega_{\text{Pix2Pix}} = \min_G \max_D \mathcal{L}_{cGAN}(G,D) + \lambda \mathcal{L}_{L1}(G) \tag{2.8}$$

Where λ is a weight constant to be defined; the greater value of λ the closer the model's structure will stay to the ground truth.

2.10.2 Architecture

Both the Generator and Discriminator use the same UNet architecture with Convolution-BatchNorm-ReLU. Since the model hopes to keep the same underlying structure from input image to output image, the architecture was designed with this in consideration. Prior models commonly used an encoder-decoder architecture, in which the image is down-sampled until a bottleneck layer, at which point it would be up-sampled to reverse the process. Due to the unrecoverable loss of spatial information in this process, the output images inevitably lose accuracy information such as the location of edges and other salient features. Such information should be shared between the input and output images, yet they are lost due to their low-level nature. For this reason, they utilize skip connections. The inclusion of skip connections (following the U-Net architecture) allows the model to retain important edge information.

2.11 CycleGAN

Most neural networks employ a method that relies upon, and is inherently limited by the use of, paired data. However, in most practical cases, there is an inherent absence of paired training data due to the

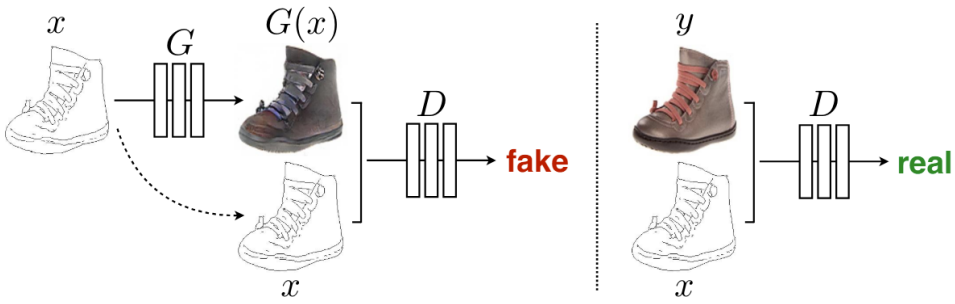


Figure 2.14: Diagram demonstrating Pix2Pix architecture with a generator that attempts to make an image from an outline, and a discriminator which attempts to spot fake images [25].

economic and time factors of attaining such data. CycleGAN [59] is a image-to-image translation model which doesn't need paired data, the first of its kind. Much like Pix2Pix, it is able to capture special characteristics of one set of images and find a way of mapping them onto another set of images whilst preserving their content. To achieve this, it assumes that two sets, say X and Y, have some kind of relationship, and then looks for a mapping from X to Y that would produce an image that is stylistically indistinguishable by the discriminator from the rest of the set. With this, functions G and F are defined as $G : X \to Y$ and $F : Y \to X$, respectively.

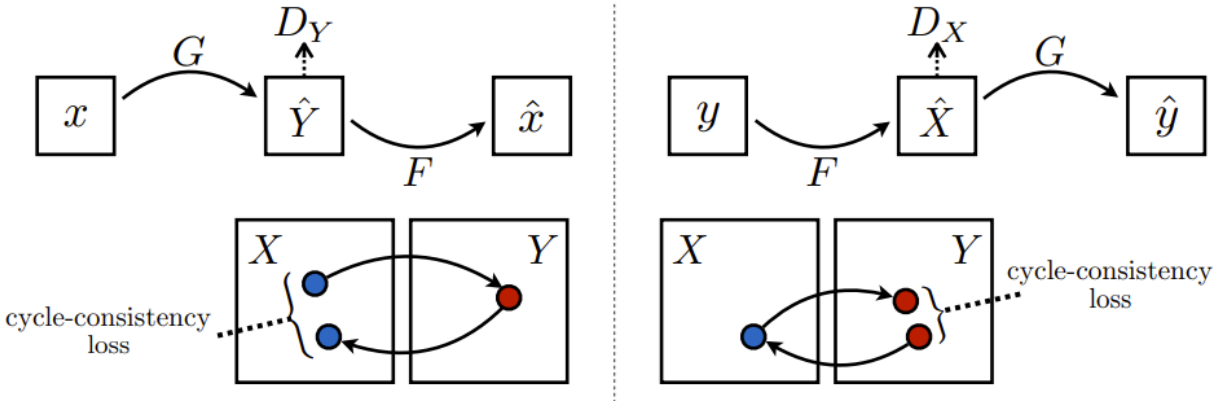
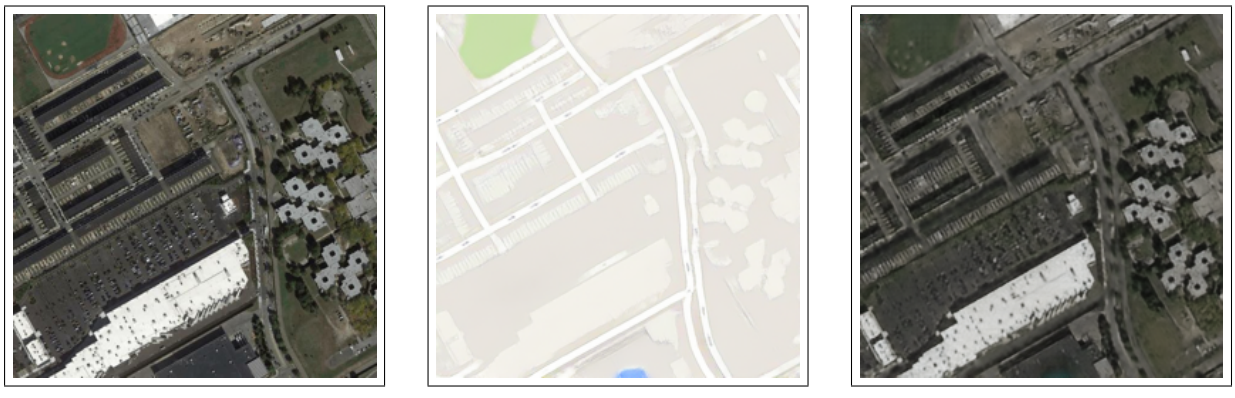


Figure 2.15: Diagrams showing the forwards and backwards consistency, as well as the forwards and backwards loss of each function of CycleGAN (respectively) - i.e. that $F(G(X) \approx X$ and vice-versa.[59]



(a) Sample X (b) Fake Y (c) Recreated X

Figure 2.16: 3 images showing an example of mapping $X \to Y \to X$ with (a) $= X_{real}$, (b) $= Y_{fake}$, and (c) $= X_{rec}$.

The method used to train CycleGAN is essentially an extension of that to train Pix2Pix. It builds on the

assumption that were a Pix2Pix-like model trained to map G, as well as one to map F, then you would expect them to be the inverse of each other - i.e. $F(G(x)) \approx x$ and $G(F(y)) \approx y$. Similarly, if you were to translate a sentence from English to French, and then from French back into English, you would expect the output to be the same as the original sentence. Using this concept, Zhu et al. define two GANs (as opposed to cGANs). Within these GANs exist generator networks G_G and G_F, with each modelling the opposite transformations $F : X \to Y$ and $G : Y \to X$ respectively. As well as this, there are two discriminator networks, D_G and D_F, training adversarially to G_X and G_Y respectively. However, training two models without formally interlinking them will just result in two models that have complex mappings to generate infinitely many statistically equivalent (to the target domain) but incorrect images. This leads to the novel idea here of adding a new loss to the objective function from Pix2Pix: cycle-consistency loss. Returning to the translation analogy, some phrases may not have direct translations, so you must attempt to translate whilst minimising the loss of semantics in the phrase. Similarly in CycleGAN, there is a loss measure on the cycle-consistency so that the image-to-image translations retain the content of the images. This can be defined as

$$\mathcal{L}_{cyc}(G_F, G_G) = \mathbb{E}_x\Big[G_F(G_G(x)) - x\Big] + \mathbb{E}_y\Big[G_G(G_F(y)) - y\Big] \tag{2.9}$$

Which allows Zhu et al. to extend $\mathcal{L}_{Pix2Pix}$ to construct a new loss function:

$$\mathcal{L}_{\text{CycleGAN}}(G_F, G_G, D_F, D_G) = \mathcal{L}_{GAN}(G_F, D_F, X, Y) + \mathcal{L}_{GAN}(G_G, D_G, Y, X) + \lambda\mathcal{L}_{cyc} \tag{2.10}$$

Resulting in a full objective function of:

$$\Omega_{\text{CycleGAN}} = \min_{G_F, G_G} \max_{D_F, D_G} \mathcal{L}_{CycleGAN}(G_F, G_G, D_F, D_G) \tag{2.11}$$

2.12 CycleGAN Succesors

Despite the novel success of CycleGAN for unpaired one-to-one image translation mappings, it can often be weak at comprehending finer details and preserving salient attributes of the original image. As a result, some images don't keep the scene information after translation to the target domain, resulting in less realistic outputs. Least squares GAN (LSGAN), is an extension of CycleGAN that acknowledges its vanishing gradient problem by employing a mean squared error adversarial loss. In 2020, three years after the success of CycleGAN, Kim et al. proposed U-GAT-IT [28] which uses a new attention mechanism as well as adaptive instance normalization to better detect the important features of images. Meanwhile, Nizan et al. hypothesized a GAN ensemble, as they introduced Council GAN [40], replacing traditional cycle-consistency with a "council loss." This method leveraged multiple generator discriminator pairings that work together as a "council" to make agreements on mappings which tend to be more diverse. One year later, Zhao et al introduced ACL-GAN [58] which relaxed the cycle-consistency loss to allow for larger shape changes by being less strict on pixel-level mappings. It breaks the assumption that the target images contain the same information as the input images, as they introduce an "adversarial-consistency loss", allowing for larger pixel-level changes whilst maintaining distribution-level features of the input image. Whilst improving upon predecessors, both ACL-GAN and Council GAN come at the cost of introducing randomness to the output, which renders it unsuitable for more scientific applications that require one-to-one mappings. Furthermore, each of these models require far more parameters to achieve their results, leading to a more computationally expensive model that takes longer to train [44].

2.13 UVCGAN

Following the relative success of CycleGAN and it's successors, as well as the recent boom in the use of transformer-based architectures, Torbunov et al. introduced UVCGAN. Unlike previous methods, they adapted the UNet architecture to include a transformer that allows for capturing more precise detail and spatial information.

2.13.1 Transformers

Introduced in 2017 by Vaswani et al. [47], transformers have revolutionized the field of deep learning. Incredibly, they often manage to achieve better results, with less training time, relative to the standard encoder-decoder attention mechanisms. They achieve this by processing sequences in parallels rather

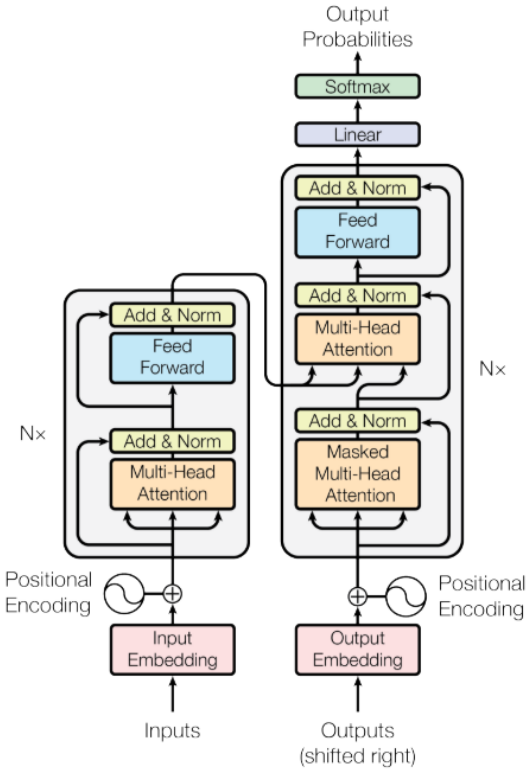


Figure 2.17: Transformer model architecture - encoder and decoder are left-side and right-side respectively. [47]

than sequentially. At first, they were introduced for application in natural language processing, but have since adapted for utilization in computer vision tasks.
The transformer architecture (see 2.17) is made up of an encoder and a decoder. The encoder is composed of 6 identical layers, each with 2 sub-layers. The first sub-layer is a multi-head self-attention mechanism, whereas the second one is a position-wise feed-forward network. The feed-forward network is just an MLP used to apply linear transformations to each token separately and identically with the following equation

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2.12}$$

The MLP is the same for each token but has different parameters in different layers. Residual connections are used in-between each sub-layer, such that its input is added to the output, and then normalized. In doing so, more information is preserved throughout the layers of the network. Similarly, the decoder is built of 6 identical layers and uses residual connections, but each layer also contains an additional sub-layer which performs multi-head attention over the output of the encoder. However, this sub-layer is masked to prevent data from the decoder path influencing later layers. [47].

2.13.2 Attention

An attention function is a function that maps a query vector and key-value pair vector to an output vector. Put simply, the output is the weighted sum of the values, keys, and the query. In their model, scaled dot-product attention is used to take a set of keys and values, with dimensions d_k and d_v, process dot products of the query with all of the keys, and divide each by $\sqrt{d_k}$. Then, a softmax function is used to calculate the weights on the values. Overall, this looks like:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.13}$$

[47] where Q, K, and V are the query, the keys, and the values respectively. Sometimes, additive attention functions are used as an alternative. However, dot-product attention functions are faster, and more space efficient. Continuing, Vaswani et al. go on to explain the novel multi-head attention. They found

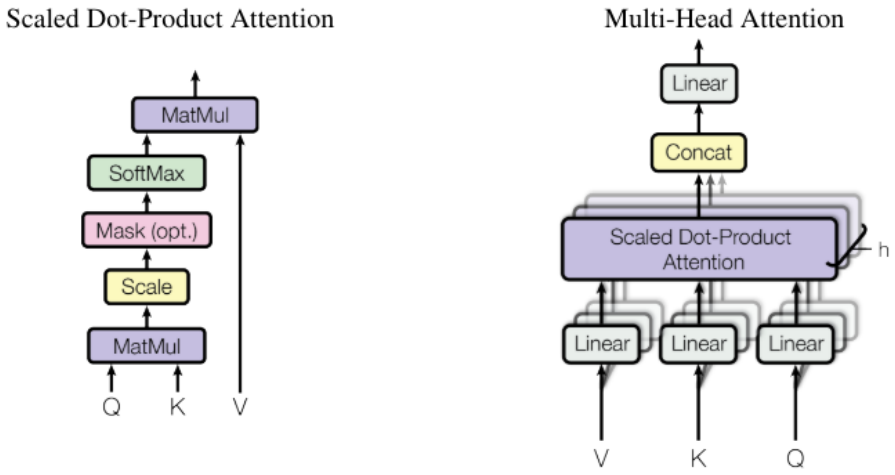


Figure 2.18: Scaled dot-product attention, and multi-head attention architectures, left and right respectively. [47]

that instead of performing one attention function on d_m-dimensional inputs (like in Equation 2.13), it is preferable to learn unique linear transformations to project the inputs (Q, K, V) to have dimensions d_k, d_k, and d_v respectively. Subsequently, they apply the attention function to each projection, in parallel. This results in a d_v-dimensional output, which is then concatenated and projected back to achieve the final values. This is known as multi-head attention, which can be comparatively visualized in Figure 2.18, and can be summed up as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{2.14}$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.15}$$

where $W_i^Q \in \mathbb{R}^{d_m \times d_k}$, $W_i^K \in \mathbb{R}^{d_m \times d_k}$, and $W_i^V \in \mathbb{R}^{d_m \times d_v}$ are the initial projection matrices, and $W^O \in \mathbb{R}^{hd_v \times d_m}$ is the output projection matrix. By utilizing multi-head attention, the transformer is able to capture more diverse relationships by giving significance to multiple parts of the data at a time.[47]

2.13.3 Vision Transformer (ViT)

Initially, transformers were built for natural language processing, and have since shown astonishing results on such tasks. State-of-the-art performance was seen on language translation tasks, and researchers could only assume that similar results would be seen on other data formats, whether it be audio, video, or images [47]. Later, assumptions would prove correct, as Dosovitskiy et al. introduced the Vision Transformer (ViT), which showed state-of-the-art success in computer vision without the support of CNNs. The model, as seen in 2.19, splits an image into a number of fixed-size patches, which are then flattened and concatenated with positional embeddings, which are then given as input to a transformer encoder, as seen in 2.17. Finally, this output is decoded for some output such as class labels or another image. Using ViTs enables image models to understand global relationships in the image, whilst CNNs are burdened with an inductive bias due to their two-dimensional neighbourhood structure and translational equivariance. However, Dosovitskiy et al suggested the potential for a hybrid architecture consisting of a CNN partnered with a transformer that uses feature maps from the CNN to learn patterns. Importantly, the CNNs neighbourhood structure is used frugally, such that positional embeddings don't actually contain information about the position of the patch, instead leaving the spatial relations to be learnt by the model. In doing so, the ViT and hybrid models have far more inductive power than a traditional CNN. [12].

2.13.4 Architecture

Furthermore, UVCGAN builds upon the UNet architecture (discussed in 2.6) utilized by most CycleGAN-like models, they additionally use a ViT transformer block at the UNet bottleneck. As per usual, in the encoding path of UNet, the model extracts low-level features using convolutional layers and down-sampling whilst high-level features are passed to the decoding path using skip connections. At the

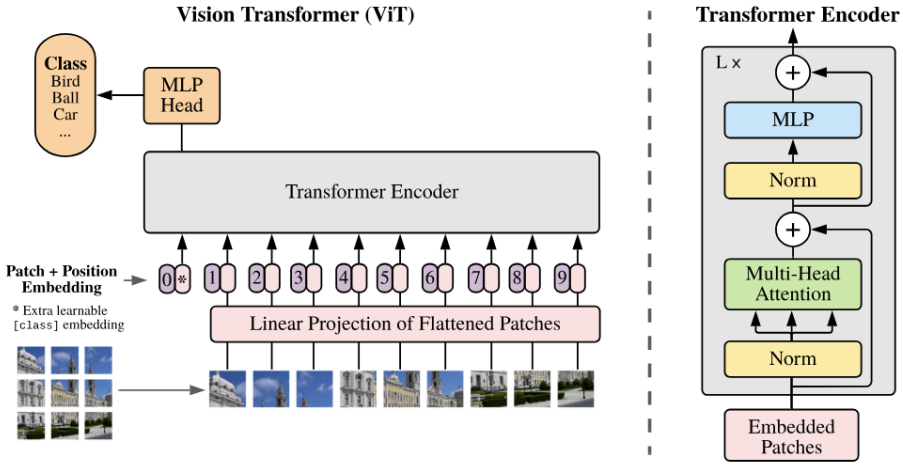


Figure 2.19: ViT architecture

bottleneck, low-level features are given to the ViT transformer block to learn pairwise relationships of low-level features.[44]

After encoding a image of shape (w,h,f), the output of shape $(w/16,\ h/16,\ 8 \cdot f)$ is passed into the bottleneck. Upon receipt of the low-level feature maps from the encoder path, the ViT flattens them into a sequence of tokens, with length $w \cdot h$. Subsequently, the tokens are concatenated with their respective Fourier positional embedding of dimension f_p, and then given as input to the 12 consecutive transformer encoder blocks, which uses a feed-forward network to expand the tokens into a higher dimension, f_h. This facilitates more complex interactions, allowing more "expressive power", and returns an f_v-dimensional output. After processing, the tokens are linearly projected back to their original sizes, and sent to the UNet decoding path for upwards sampling. In their paper, f, f_p, $f_v = 384$, and $f_h = 4 \cdot f_v$.

In addition, the UVCGAN's discriminator uses LSGAN loss, which means using L_2 error as opposed to L_1 as in 2.10. Furthermore, the discriminator is provided with a gradient penalty term to ensure the mapping undergoes changes smoothly, without gradients exploding. This is done by leveraging the 1-Lipschitz function [5]:

$$|f(x) - f(y)| \geq L\|x - y\| \tag{2.16}$$

which is simply to say the difference between the output of function f must be less than or equal to the Euclidean distance between the inputs. With this, the new discriminator loss is defined:

$$\mathcal{L}^{GP}_{disc,A} = \mathcal{L}_{disc,A} + \lambda_{GP}\,\mathbb{E}\Big[\frac{(\|\nabla_x \mathcal{D}_A(x)\|_2 - \gamma)^2}{\gamma^2}\Big] \tag{2.17}$$

where $\mathcal{L}_{disc,A}$ is the original discriminator loss, λ_{GP} is the coefficient for the gradient penalty, and the remainder is the gradient penalty term, that rewards gradients close to γ (usually $\gamma = 1$).

2.13.5 Self-Supervised Pre-training

Until recently, it was common-practice to randomly initialize the model weights. However, it has been found that performance can be enhanced by employing self-supervised pre-training. In this method, the generator model attempts to predict some unobserved training data from parts it has observed, which results in an initialization that has a better representation of the data. For example, in the domain of natural language processing, the model may attempt to predict the next word of a sentence based on priorly observed words. On the other hand, in the case of image-to-image translation, the model utilizes inpainting. This is where the model masks the input images by setting pixel values to 0, and attempts to predict the unmasked pixel values. In the case of UVCGAN, the images are given a grid made up of 32×32 squares, 40% of which have their values set to 0. Then, the generator trains to restore the lost information of these images.[44]

2.14 Related Work

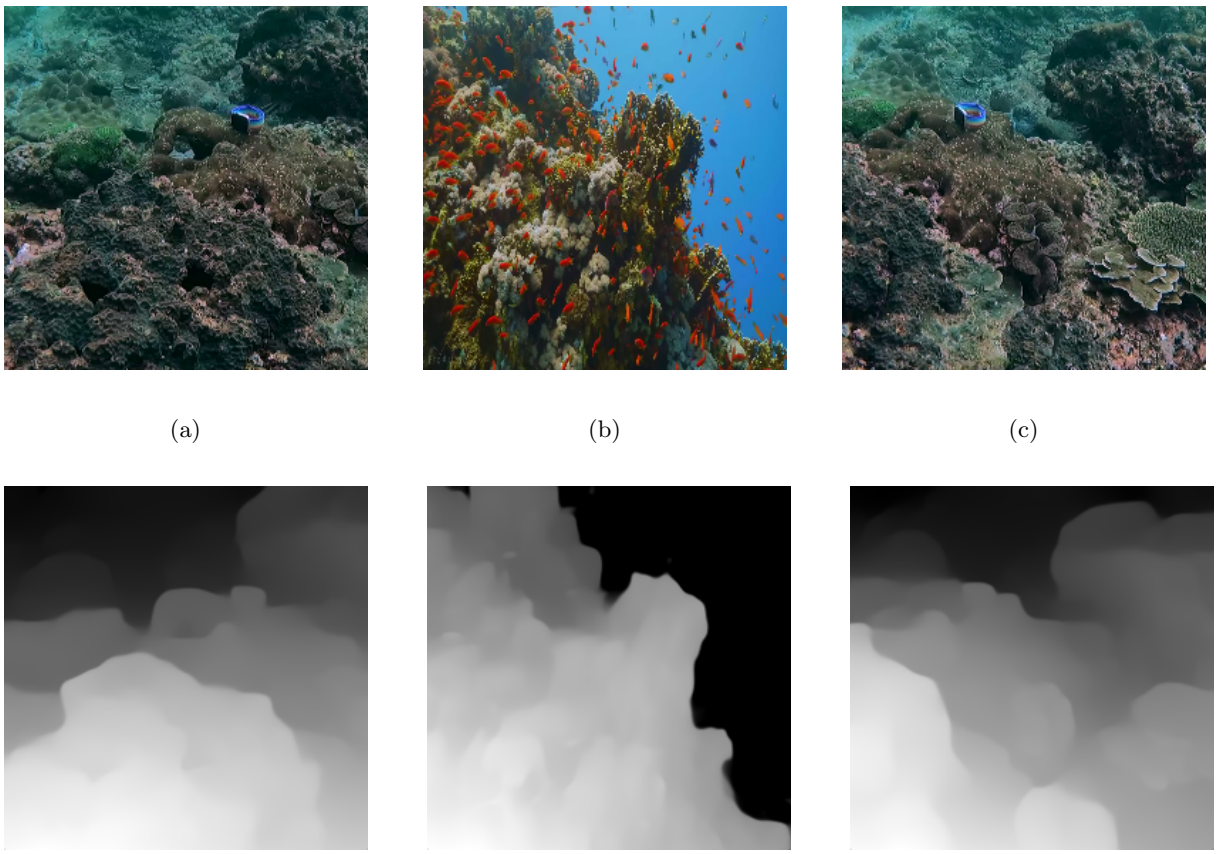
With underwater image degradation being such a common problem, it is perhaps unsurprising that there exists a wide range of solutions for UIE. In this section, a number of unique models will have their methods briefly explained, with an acknowledgement of their respective shortcomings, if deemed relevant.

2.14.1 SeaThru for Synthetic Data

Akkanayak et al. proposed a state-of-the-art UIE model that has the capability of "removing water from underwater images." The method realizes the non-linearity of the attenuation coefficient and its independence of the backscatter coefficient. Previous models had assumed underwater attenuation to follow an atmospheric mode, but this new model shows state of the art results for UIE. This model requires knowledge of the scene depth. With this, it uses the dark pixels in the image combined with their known range information to estimate the range-dependent attenuation coefficient. [2]

2.14.2 Depth Anything

Computer vision tasks are often made easier with access to more information about the scene. In UIE, depth information is a crucial aspect of modelling underwater light attenuation, yet enhancements are mostly required to be made from a monocular image. Depth Anything, by Yang et al., is a practical, robust solution for monocular depth estimation. It can produce highly accurate depth map estimations from just an image. Although it has not been extensively tested on underwater images, it could help bolster model performance by providing deeper knowledge of the scene [53].



(a) (b) (c)

(d) (e) (f)

Figure 2.20: Depth-Anything-Large being applied to (a), (b), and (c) producing output (d), (e), and (f) respectively.

2.14.3 Histogram Distribution Prior

In 2016, Li et al. separated the UIE process into two separate algorithms, one for de-hazing, and the other for contrast enhancement. They build the de-hazing algorithm to minimize information loss, in an attempt to restore visibility in the image. Contrast enhancement is achieved using a histogram distribution prior to alter the brightness and contrast of the underwater image to resemble that of an in-air image. Despite its simplicity, the proposed method is very effective for UIE, even in challenging scenes [32]. For shorthand, this model will often be referred to as the HP-based model.

2.14.4 UDCP

In 2013, Drews Jr. et al. introduced UDCP, a method that adapts a statistical method known as the dark channel prior (DCP) [17] to estimate the transmission of light underwater. [14]. DCP is a method that had previously been used as a statistical prior for image properties in outdoor scenes. It assumes that in almost all parts of the image, at least one color channel is almost zero. With this, the minimum color channel value is taken, and from these the minimum for a particular patch is calculated. Then the patches with highest values are assumed to be altered by additional atmospheric light. Then, a transmission map is created and used to remove haze from the image. Drews Jr. et al adapted this by disregarding the red channel and applying the same method.

2.14.5 UWCNN

In 2018, Li et al. introduced UWCNN, a CNN trained on a synthetic underwater dataset to restore images. As one of the first approaches using deep learning, it was a powerful display of possibility for models that do not require estimating the underwater image model parameters. Using 10 different marine databases, they produced separate models for different water environments. Synthetic images were created using an atmospheric model with a range of variables to emulate haze, discoloration, and turbidity.

2.14.6 Water-Net

In 2020, Li et al. proposed a new set of UIE benchmark experiments. They use these UIE benchmarks to train a CNN, known as Water-Net, that aims to maximize the model's scores on the benchmark. The model utilizes three inputs, which each consist of the image but after the following algorithms: white balance, histogram equalization, and gamma correction. Water-Net utilizes a gated fusion network architecture that learns confidence maps to determine the most significant features of the three inputs, which are then combined to create an enhanced image.

2.14.7 Temporal Consistency

A video can be thought of as a 3-dimensional image, such that it is a stack of images, with a third axis corresponding to the temporal information. Assuming the frames to be related, i.e. of the same scene from the same camera, neighboring frames on the temporal axis should be related. By observing along the temporal axis, frames can be compared, and should contain the same spatial features. Temporal consistency is the concept that a pixel in one frame should have the same intensity as the corresponding pixel in the next frame. Intuitively, even if an object has moved in a video frame, it should still be the same color in the next video frame, even if it has moved position. The simplest form of temporal smoothing is for a static scene, where each pixel value can be averaged along the temporal axis using its k-nearest neighbours, resulting in a smooth color gradient. However, in practice the corresponding pixels will not remain in the same place, and the spatial locations have to be tracked. The related pixels can be used to estimate the motion in an image, which is a concept known as optical flow [7]. Furthermore, the relationship of these pixels can be used to unlock more information about the scene, and can be used for temporal smoothing. More modern methods use ground truth temporally smooth videos as a prior for modelling, such that there is a mapping from raw video to temporally smooth video [11].

Chapter 3

Methodology

Previously, there have been a number of varied approaches for UIE. Whilst they been successful on image data, they can often lack temporal consistency when extended onto video datasets. Both paired and unpaired training was considered for improving upon this. This chapter will outline the data collection process, propose a number of methods, and explain how they were trained.

3.1 Dataset(s)

In the domain of UIE there is a distinct lack of publicly available datasets. For this project, data came from a range of sources, and had to be categorized into a "good" category and a "bad" category. Data was deemed to belong to the "bad" category if it contained significant hazing, color degradation, or loss of visibility. With UIE being an ill-posed problem, it is hard to define what exactly a "good" underwater image should look like; it is difficult to define what amount of discoloration is acceptable, or how much loss of visibility should be tolerated before an image is considered bad. It is an acceptable assumption to make that a "good" underwater image should exhibit likeness to an in-air image, and so color corrections aim to restore the red-channel information. Whilst it is difficult to articulate, dividing these images into separate domains was fairly intuitive, albeit arduous. This section will outline information about the dataset, from how it was sourced, to any key features.

3.1.1 Data Collection

Edited Paired Dataset

Initially, Dr Anantrasirichai provided a dataset containing 40 underwater videos taken by divers: 20 raw, and 20 post-processed with a color correction. These videos are all about 30 seconds in length, featuring footage of divers panning around an underwater terrain. These videos are fairly high definition, containing minimal hazing, and the enhanced videos exhibiting less of a blue-green tinge. Whilst this is a paired dataset, the enhanced data did not show significant enough change from the raw data for the model to learn anything other than a simple color correction. As a result, only the enhanced videos were added to my dataset, and the raw data was discarded.

DRUVA, SMD, & YouTube

Assuming videos in Table 3.1 to be good enough in quality, the dataset now needed some "bad" quality videos to include. For bad data, the DRUVA dataset [46] was used. This consists of 20 videos, each about a minute long, taken in quite shallow water featuring a strong blue tinge as well as significant haze. In addition to being quite degraded, these videos also contain a nearly 360 degree view of some artifact (e.g. rock, stick, etc.) which would allow completion of model evaluation on temporal consistency and, if time allows, scene reconstruction.[46] Also, a few videos were provided by sub-sea engineering company SMD. Thanks to their specialist equipment for deep underwater exploration, the dataset is made more diverse by including clips exhibiting stronger degradation. Finally, the dataset was made larger using videos from YouTube. Generally, bad videos tended to come from small scuba-diving channels whereas good videos would usually come from long-form documentary-style content. As a result, the dataset is now at length **11:19:40**. Then, the video frames are shuffled, and the dataset is truncated so that the first 8000 are for training, and the subsequent 2000 are split between test and validation sets. All frames

Dr Anantrasirichai's Dataset			
Video Names		Properties	
Original	HD	Duration	Good?
Original11404	HD11404	00:00:20	✓
Original11409	HD11409	00:00:59	✓
Original11410	HD11410	00:00:17	✓
Original11411	HD11411	00:00:16	✓
Original11412	HD11412	00:00:16	✓
Original11413	HD11413	00:00:14	✓
Original11414	HD11414	00:00:15	✓
Original11416	HD11416	00:00:43	✓
Original11417	HD11417	00:00:20	✓
Original11419	HD11419	00:01:33	✓
Original11432	HD11432	00:02:43	✓
Original11435	HD11435	00:00:19	✓
Original11457	HD11457	00:00:41	✓
Original11459	HD11459	00:00:49	✓
Original11460	HD11460	00:01:12	✓
Original11461good	HD11461good	00:01:05	✓
Original11462	HD11462	00:01:06	✓
Original11463	HD11463	00:00:26	✓
Original11465	HD11465	00:00:29	✓
Original11468	HD11468	00:00:18	✓
Original11469	HD11469	00:00:34	✓
OriginalGH011605	HDGH011605	00:00:28	✓
OriginalPC210120	HDPC210120	00:00:26	✓
Total Duration		**00:15:49**	

DRUVA Dataset		
Video Names	Properties	
	Duration	Good?
A1	00:01:08	✗
A2	00:00:37	✗
A3	00:01:09	✗
A4	00:01:02	✗
A5	00:00:53	✗
A6	00:01:13	✗
A7	00:00:50	✗
A8	00:01:17	✗
A9	00:00:54	✗
A10	00:01:14	✗
A11	00:00:53	✗
A12	00:00:50	✗
A13	00:00:50	✗
A14	00:01:17	✗
A15	00:00:28	✗
A16	00:00:45	✗
A17	00:00:57	✗
A18	00:01:13	✗
A19	00:01:02	✗
A20	00:01:10	✗
Total Duration	**00:19:42**	

Table 3.1: Table of videos from Dr Anantrasirichai's dataset and DRUVA dataset [46], left and right respectively.

are cropped to square aspect ratio, 800×800 or smaller if necessary, and then resized to be 256×256 so that models train in reasonable amounts of time, and maintain even divisions as they are down-sampled and up-sampled.

3.2 Unpaired Training

3.2.1 CycleGAN for Image-to-Image Translation

The training procedure for CycleGAN involved a similar method to that described in the paper, with the usage of the same hyperparameters and loss functions [59]. Both generators are initialized with ResNet 9-block architecture During training, the model iterates through 8000 image pairs, Real A and Real B, processing them batches of size 4, resulting in the generator producing new images Fake A and Fake B. These fake inputs are then input to their respective discriminators, which are initialized with 70×70 PatchGAN architecture. The fake inputs are then put through the opposite generator to make Reconstruct A and Reconstruct B. Finally, an Identity A and Identity B are formed by putting Real A and Real B through the opposite generator, which should make no change to the image. Training CycleGAN involves using all of the mentioned images to run a min-max game between the generator and the discriminator, which is defined by the following objective function:

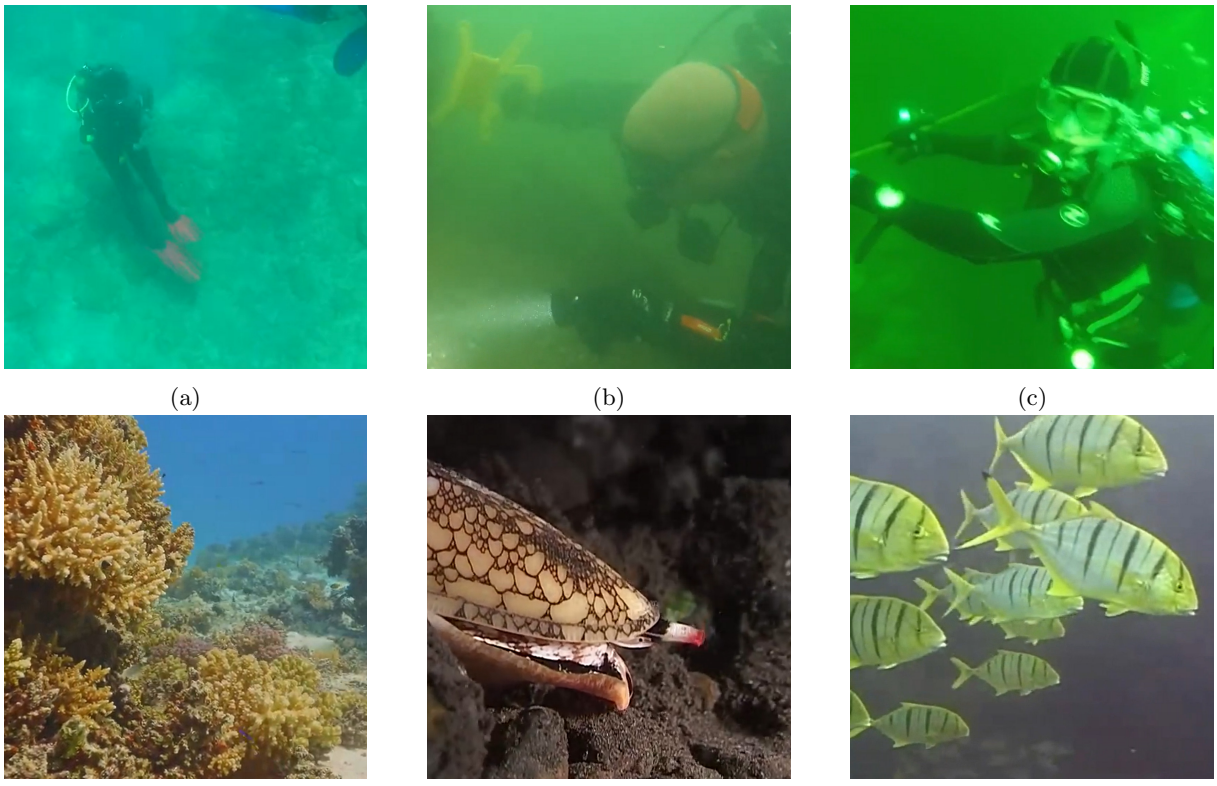
$$\Omega_{\text{CycleGAN}} = \min_{G_F, G_G} \max_{D_F, D_G} \mathcal{L}_{\text{CycleGAN}}(G_F, G_G, D_F, D_G) \tag{3.1}$$

Essentially, the generators try to minimize the adversarial loss, whilst the discriminators try to maximize it, by correctly deciphering between real and fake images. The loss function is further broken down as

SMD Dataset		
Video	Properties	
	Duration	Good?
canyon1	00:02:06	✗
rockgarden	00:00:28	✗
zhong1	00:01:04	✗
zhong2	00:00:22	✗
Total Duration	**00:04:00**	

YouTube Dataset		
Video Names	Properties	
	Duration	Good?
Scuba Diving emergency...	00:03:19	✗
Scuba Diving In NEW...	00:11:36	✗
Groupers and Big Fish...	00:05:51	✓
Panic diver training...	00:01:26	✗
Diving in Bali (720p)	00:54:41	✓
Under Red Sea 4K...	08:09:25	✓
Diving MAURITIUS...	00:02:26	✓
SHipwreck dive in Coron	01:14:34	✗
bottom of the sea for...	00:01:00	✗
Snorkel 2 en la playa...	00:00:51	✗
Total Duration	**10:40:09**	

Table 3.2: Table of videos from SMD and YouTube, left and right respectively.



(a) (b) (c)

(d) (e) (f)

Figure 3.1: Examples, from YouTube, of some good underwater images, (c), (d), and (e), as well as some bad underwater images, (a), (b), and (c).

follows:

$$\mathcal{L}_{\text{CycleGAN}}(G_F, G_G, D_F, D_G) = \mathcal{L}_{GAN}(G_F, D_F, X, Y) + \mathcal{L}_{GAN}(G_G, D_G, Y, X) + \lambda\mathcal{L}_{cyc} + \lambda_{idt}\mathcal{L}_{idt} \quad (3.2)$$

which defines the overall loss as a summation of the forward adversarial loss, the backwards adversarial loss, the cycle consistency loss, and the identity loss. The latter two losses are weighted by constants $\lambda = (\lambda_A, \lambda_B)$, and λ_{idt}, respectively. The weightings are used, and modified to give more importance to particular loss functions. In this project, the weights in Table 3.3 were used to experiment with the performance of CycleGAN. These weights were chosen to see if giving the loss an imbalance could be beneficial to the image enhancements.

λ_A	λ_B	$\lambda_{\text{identity}}$
10	10	0.5
10	10	10
5	10	0.5
10	5	0.5
50	50	0.5

Table 3.3: Weight combinations used for training CycleGAN image-to-image.

The adversarial loss loss is calculated using the loss defined in LSGAN [36], which utilizes a mean squared error as follows:

$$\mathcal{L}_{GAN}(G_F, D_F, X, Y) = \mathbb{E}_{x\in X, y\in Y}\left[\left|D_F(y) - 1\right|^2 + \left|D_F(G_F(x))\right|^2\right] \quad (3.3)$$

$$\mathcal{L}_{GAN}(G_G, D_G, Y, X) = \mathbb{E}_{x\in X, y\in Y}\left[\left|D_G(x) - 1\right|^2 + \left|D_G(G_G(y))\right|^2\right] \quad (3.4)$$

Subsequently, the cycle-consistency loss is calculated using an L1 loss as follows:

$$\mathcal{L}_{cyc}(G_F, G_G, X, Y) = \mathbb{E}_{x\in X, y\in Y}\left[\lambda_A\left|G_G(G_F(x)) - x\right| + \lambda_B\left|G_F(G_G(y)) - y\right|\right] \quad (3.5)$$

Finally, the identity loss also uses L1 loss, with the following function:

$$\mathcal{L}_{idt} = \mathbb{E}_{x\in X, y\in Y}\left[\lambda_A\left|G_F(y) - yG\right| + \lambda_B\left|G_G(x) - x\right|\right] \quad (3.6)$$

Hyperparameter	Value	Description
Generator Model	Res-Net 9-Block	Architecture used for the generators
Discriminator Model	N-Layer Discriminator	Architecture used for the discriminators
Optimizer	Adam	The type of optimizer used for each generator and discriminator
Optimizer First Moment, Second Moment decay	0.5, 0.99	Weights defining the momentum decay of gradients, allowing the model to remember more previous gradients.
(λ_A, λ_B)	(10, 10)	The weight for the forward and backwards mapping, respectively
λ_{idt}	0.5	The weight for the identity loss
Pool Size	50	The number of previously processed images that are stored
Learning Rate Policy	Linear	The method used for updating learning rates
Learning Rate Decay Iterations	100	Multiply the learning rate by γ after this many iterations
Learning Rate	0.0002	The initial learning rate for the model
Total Epochs	200	The number of epochs the model trains for.

Table 3.4: Table showing training hyperparameters for CycleGAN

The overall structure for the training of CycleGAN, including its predefined hyperparameters for training are summarized in Table 3.4. Over the training iterations the model decays its learning rate using a linear decay using the hyperparameters in Table 3.4. The equation for linear decay is as follows:

$$\text{learning rate decay} = \max(0, \frac{e_{\text{current}} + e_{\text{start}} - e_{\text{constant}}}{e_{\text{decay}} + 1}) \quad (3.7)$$

where e_{current} is the current epoch, e_{start} is the start epoch, e_{constant} is the number of epochs learning rate should stay constant, and e_{decay} is the number of epochs it will take to decay to zero. It repeats this process for 200 epochs and achieved bittersweet results (see Chapter 4).

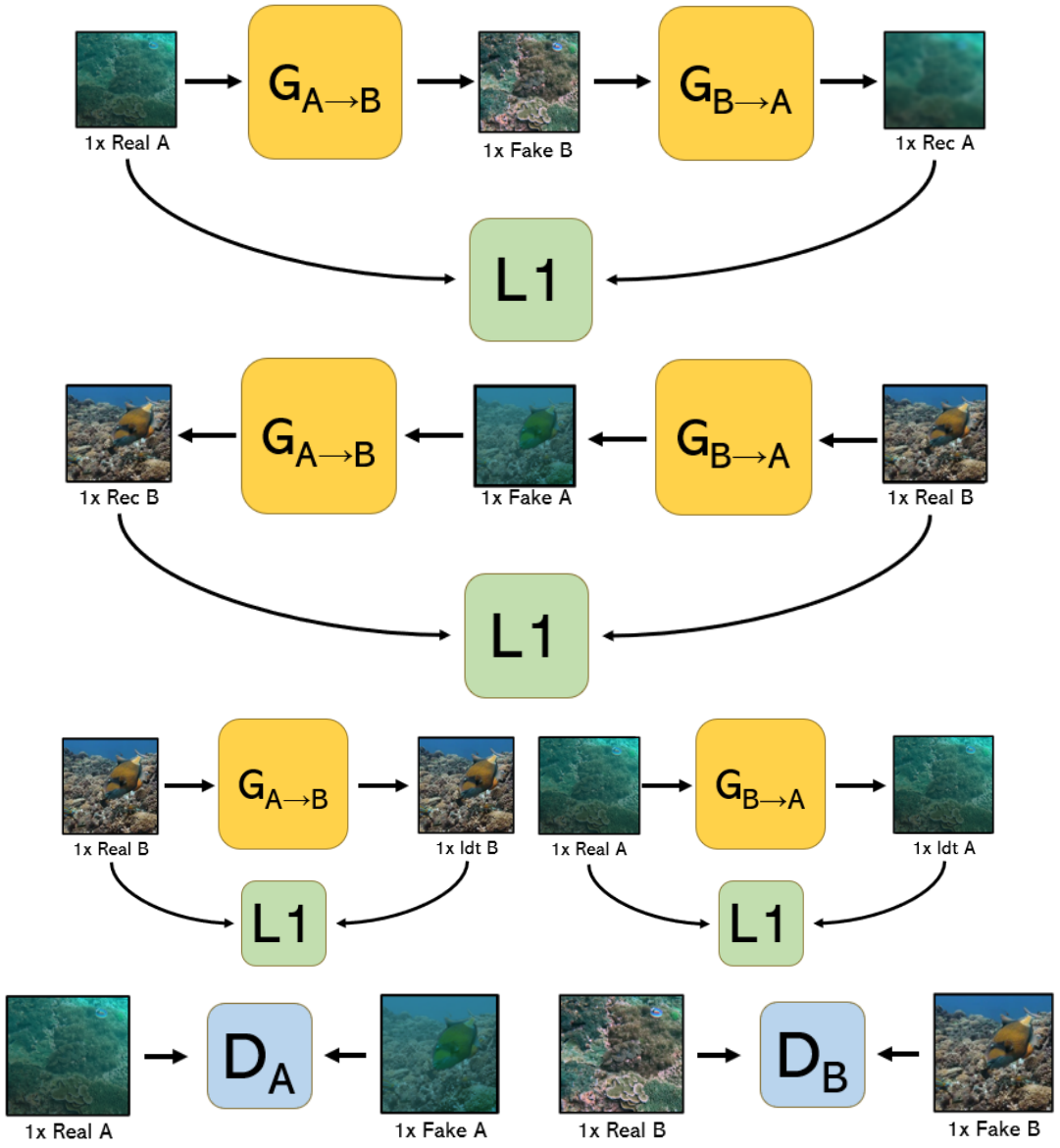


Figure 3.2: Diagram showing CycleGAN image-to-image method for calculating loss of outputs.

3.2.2 CycleGAN for Video-to-Video Translation

UIE models often struggle with temporal consistency with their focus on monocular image enhancement rather than their applications to videos. To achieve enhancements with improved temporal consistency, the model was extended to take an input of 5 consecutive frames, and map them to an output of 5 frames. The hypothesis was that providing the model with multiple consecutive frames could allow it to pick up temporal patterns, and produce a smoother output. With this in mind, the training dataset was altered to contain 8000 TIFF images each containing 5 consecutive frames. The models, method, and hyperparameters used were similar to in Section 3.2.1, except both the input and output had 15 channels (5×RGB) rather than the usual 3 (1× RGB). As a result, it would be calculating the L1 loss across 5 consecutive frames, and the discriminator would be trying to scrutinize across 15 channels instead of 3. This meant the loss functions were updated to the following:

$$\mathcal{L}_{VGAN}(G_F, D_F, X, Y) = \mathbb{E}_{x \in X, y \in Y}\left[\left|D_F(y) - 1\right|^2 + \left|D_F(G_F(x_i))\right|^2\right] \tag{3.8}$$

$$\mathcal{L}_{VGAN}(G_G, D_G, Y, X) = \mathbb{E}_{x\in X, y\in Y}\Big[\big|D_G(x)-1\big|^2 \;+\; \big|D_G(G_G(y))\big|^2\Big] \tag{3.9}$$

$$\mathcal{L}_{Vcyc}(G_F, G_G, X, Y) = \mathbb{E}_{x\in X, y\in Y}\Big[\lambda_A\big|G_G(G_F(x))-x\big| \;+\; \lambda_B\big|G_F(G_G(y))-y\big|\Big] \tag{3.10}$$

$$\mathcal{L}_{Vidt} = \mathbb{E}_{x\in X, y\in Y}\Big[\lambda_A\big|G_F(y)-y\big| \;+\; \lambda_B\big|G_G(x)-x\big|\Big] \tag{3.11}$$

where $\{x_1, x_2, x_3, x_4, x_5\} = x \in X$ and $\{y_1, y_2, y_3, y_4, y_5\} = y \in Y$

Essentially, the loss on each iteration becomes a sum of the loss for 5 consecutive frames, with the intention that the generator must have consistent mappings across slightly different frames. With this, the whole objective function for training CycleGAN for video-to-video translation is as follows:

$$\Omega_{VGAN} = \min_{G_F, G_G}\max_{D_F, D_G}\mathcal{L}_{VGAN}(G_F, G_G, D_F, D_G) \tag{3.12}$$

where

$$\mathcal{L}_{VGAN}(G_F, G_G, D_F, D_G) = \mathcal{L}_{VGAN}(G_F, D_F, X, Y) + \mathcal{L}_{VGAN}(G_G, D_G, Y, X) + \lambda\mathcal{L}_{Vcyc} + \lambda_{idt}\mathcal{L}_{Vidt} \tag{3.13}$$

3.2.3 CycleGAN for Video-to-Image Translation

The model was again adapted to take an input of 15 channels and an output of 3 channels. This model has an architecture which can be summarized as a hybrid of the image-to-image and video-to-video models. It required a new dataset which is also a hybrid, with the A domain consisting of 5 frame TIFF images of bad underwater images, and the B domain containing PNGs of good underwater images. As a result, generator A now produces images with generator B producing videos. The only new loss function required are for the forward and backward identity loss, which are now asymmetrical.

- For processing identity loss on Real A, the model now had to duplicate the input image 5 times to form an impromptu video for inputting. Then it is input into $G_{A\to B}$, and uses L1 loss to measure the difference between the real and identity image.
- To find the identity loss on Real B, the model is now expecting an input of 1 frame, to produce 5. To accomplish this, the middle (3rd) frame was chosen as the frame to compare with.

This is defined as follows:

$$\mathcal{L}_{V2I-idt} = \mathbb{E}_{x\in X, y\in Y}\Big[\lambda_A\big|G_F(y_{\times 5})-y\big| \;+\; \lambda_B\big|G_G(x_2)-x\big|\Big] \tag{3.14}$$

where $\{x_1, x_2, x_3, x_4, x_5\} = x \in X$ and $y_{\times 5} = \{y, y, y, y, y\}$ where $y \in Y$

3.2.4 UVCGAN for Image-to-Image translation

UVCGAN offers a more modern architecture that can hopefully further the performance of image enhancements on the dataset. The training process of UVCGAN contains two stages, the first of which is to pre-train the model. This is done through in-painting, a process where the input image is given a 32×32 grid and 40% of the tiles have their pixels set to 0. In doing so, the model should have a better weight initialization that should ultimately lead to better results. For pre-training, the model is initialized with a ViT-UNet-12 generator, and uses the hyperparameters in Table 3.5.
Notably, the pre-training phase utilizes a cosine annealing learning decay with the Adam optimizer. This is instead of the linear policy used in CycleGAN, and should help the model's gradients to converge. This is defined by the following

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(\cos(\frac{T_{cur}}{T_i}\pi) + 1) \tag{3.15}$$

where η_{max} is the initial learning rate, η_{min} is the minimum learning rate, T_{cur} is the number of epochs since the last restart, and T_i is the number of epochs between two restarts. When $T_{cur} = T_i$, η_t is set

Hyperparameter	Value	Description
Generator Model	ViT-UNet-12	Generator architecture used
Optimizer	Adam	The type of optimizer used for the generator, initialized with first moment decay of 0.90, and second moment decay of 0.99.
Gradient Penalty	None	The type of gradient penalty used, to improve training stability
Batch Size	4	The number of images processed on each iteration
Learning Rate Policy	Cosine Annealing with Warm Restarts	The method used for updating learning rates.
Total Epochs	100	The total number of epochs in the model's pre-training phase

Table 3.5: Table showing pre-training hyperparameters for UVCGAN

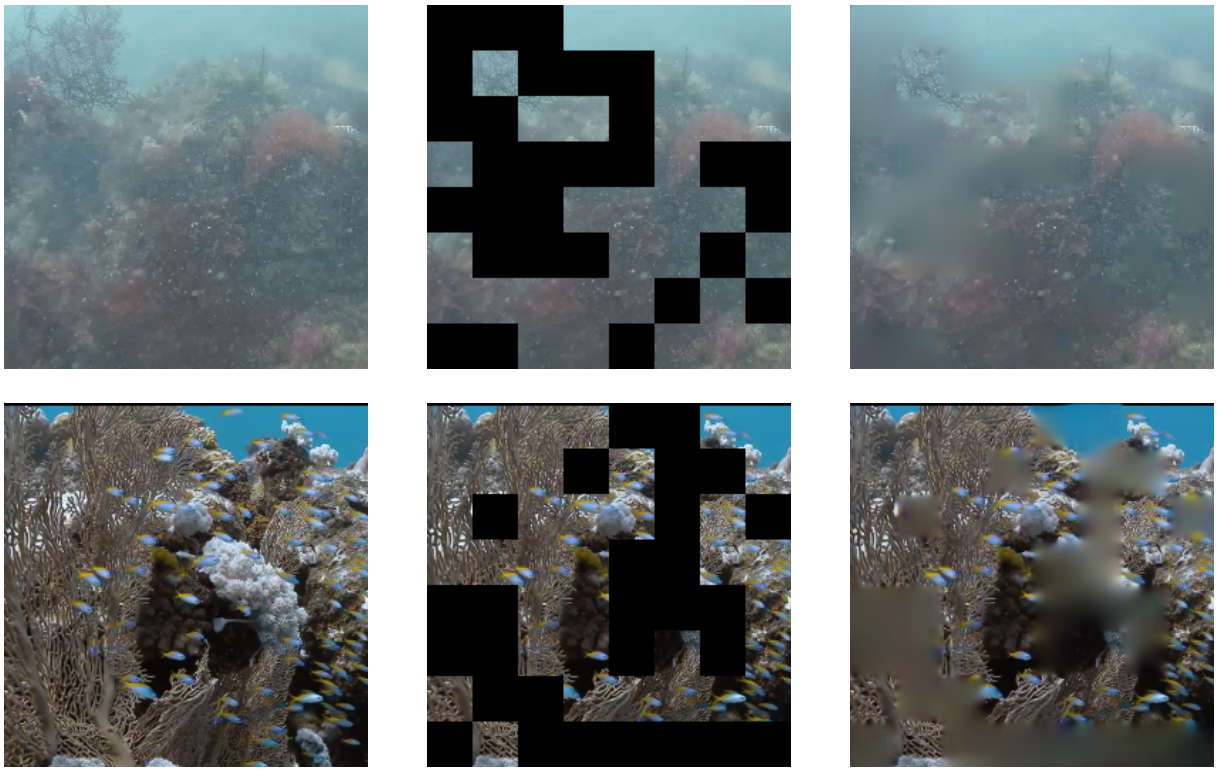


Figure 3.3: Examples of UVCGAN pretraining, as it masks 40% of its 32×32 grid, and learns to unmask itself.

equal to η_{min}, and after the restart, when $T_{cur} = 0$, set η_t equal to η_{max}. [34]. For the second training phase, UVCGAN is initialized with a ViT-UNet-12 generator, as well as an N-layer discriminator. The generator is initialized with the weights from the pre-training model, and training commences for 500 epochs. The model is trained on the unpaired dataset from Section 3.2.1, using the hyperparameters in Table 3.6, and almost the same objective function as in 3.2.1. The only difference in the objective function is that the adversarial loss includes a gradient penalty term, which is defined as follows:

$$\mathcal{L}^{GP}_{GAN}(G, D, X, Y) = \mathcal{L}_{GAN} + \lambda_{GP}\mathbb{E}_{x\in X}\Big[\frac{||\Delta_x D(x)||_2 - \gamma}{\gamma^2}\Big] \tag{3.16}$$

As a result, the whole objective function is now the following:

$$\Omega_{\text{UVCGAN}} = \min_{G_F, G_G} \max_{D_F, D_G} \mathcal{L}_{\text{UVCGAN}}(G_F, G_G, D_F, D_G) \tag{3.17}$$

where

Hyperparameter	Value	Description
Generator Model	ViT-UNet-12	Architecture used for each generator, loaded from pre-training
Discriminator Model	N-Layer Discriminator	Architecture used for each discriminator, featuring 3 layers.
Optimizer	Adam	The type of optimizer used for each generator and discriminator. Initialized with a first moment and second moment of 0.50 and 0.99, respectively.
Batch Size	4	The number of images processed on each iteration.
Learning Rate Policy	Linear	The method used for updating learning rates. This is given 250 epochs to warm up, and 250 to anneal.
Gradient Penalty, λ_{gp}	1/1,000,000	The constant for gradient penalty. This is kept constant for 100 epochs.
Total Epochs	500	The total number of epochs in the model's training phase.
Pool Size	50	The number of previously processed images that are stored.

Table 3.6: Table showing pre-training hyperparameters for UVCGAN

$$\mathcal{L}_{\text{UVCGAN}}(G_F, G_G, D_F, D_G) = \mathcal{L}^{GP}_{GAN}(G_F, D_F, X, Y) + \mathcal{L}^{GP}_{GAN}(G_G, D_G, Y, X) + \lambda\mathcal{L}_{cyc} + \lambda_{idt}\mathcal{L}_{idt} \quad (3.18)$$

3.3 Paired Training

3.3.1 SeaThru for Synthetic Data

In 2019, Akkaynak et al. proposed SeaThru [2]: a novel physical model for removing the water from images. In this section, the model is used to create synthetic underwater images that I can then use for paired training.The underwater image model is as follows:

$$I_c = D_c + B_c \quad (3.19)$$

where c represents R, G, and B color channels. Here, I_c represents the distorted image captured by the camera, D_c is the direct signal containing information about the scene, and B_c is the backscatter. Both D_c and B_c are influenced by their respective coefficients: β_c^D and β_c^B. Beforehand, it was assumed that the coefficients for the signal and backscatter were the same, but Akkaynak et al. novelly show otherwise. Equation 3.19 can be expanded to the following:

$$I_c = J_c e^{-\beta_c^D(\mathbf{v}_d)\cdot z} + B_c^{\infty}(1 - e^{-\beta_c^B(\mathbf{v}_B)\cdot z}) \quad (3.20)$$

where z is the distance from the camera to objects along the principal axis, B_c^{∞} is scattered light, and J_c is the scene that would have been captured if there had been no attenuation. The vectors $\mathbf{v}_D = \{z, \rho, E, S_c, \beta\}$ and $\mathbf{v}_B = \{E, S_c, b, \beta\}$ reflect the dependencies of β_c^D and β_c^B on the following, which are all functions of wavelength:

- z, the range between camera and objects along the line of sight of the camera
- ρ, the reflectance
- E, the spectrum of ambient light
- S_c, the spectral response from the camera
- b, the physical scattering attenuation coefficient of the water
- β, the beam attenuation coefficient of the water

Equation 3.20 is made for horizontal views, but for practical applications it is a assumed still to work sufficiently.

3.3.2 Applying SeaThru

Crucially, SeaThru requires knowledge of distances from the monocular viewpoint to objects in the image. DepthAnything is a model introduced by Yang et al. that can produce accurate depth map estimations for monocular images [53]. This method utilizes DepthAnything-Large, their largest model which has 335.3 million parameters. Instead of using equation 3.20 to solve for J_s, the formulation was used to create synthetic images, I_c. This requires knowledge of coefficients β_c^D and β_c^B. These are obtained through replicating the SeaThru method on the set of real underwater images. Having obtained a set of coefficients from the bad videos, the back-scattering is estimated through the following equation:

$$\hat{B}_c = B_c^{\infty}(1 - e^{\beta_c^B z}) + J_c' e^{-\beta_c^{D'} z} \tag{3.21}$$

Then, the attenuation coefficient is estimated by

$$\beta_c^D(z) = a \cdot e^{b \cdot z} + c \cdot e^{d \cdot z} \tag{3.22}$$

where z is depth, and a, b, c, and d are coefficients describing rate of loss of either red, blue or green. With the capability of synthesizing underwater images, incorporating in-air images is now an option. For this reason, the NYUD[39] was added to the dataset. This dataset contains 1449 images taken of indoor scenes using a Kinect sensor to capture an RGB-D image. Incorporating accurately measured depth maps and scenes taken out of water in unison with underwater data should improve the colors and the visibility exhibited in the resulting images. By applying these equations, substituting in the coefficients from [49], synthetic underwater images are produced, as in Figure 3.4 [2].

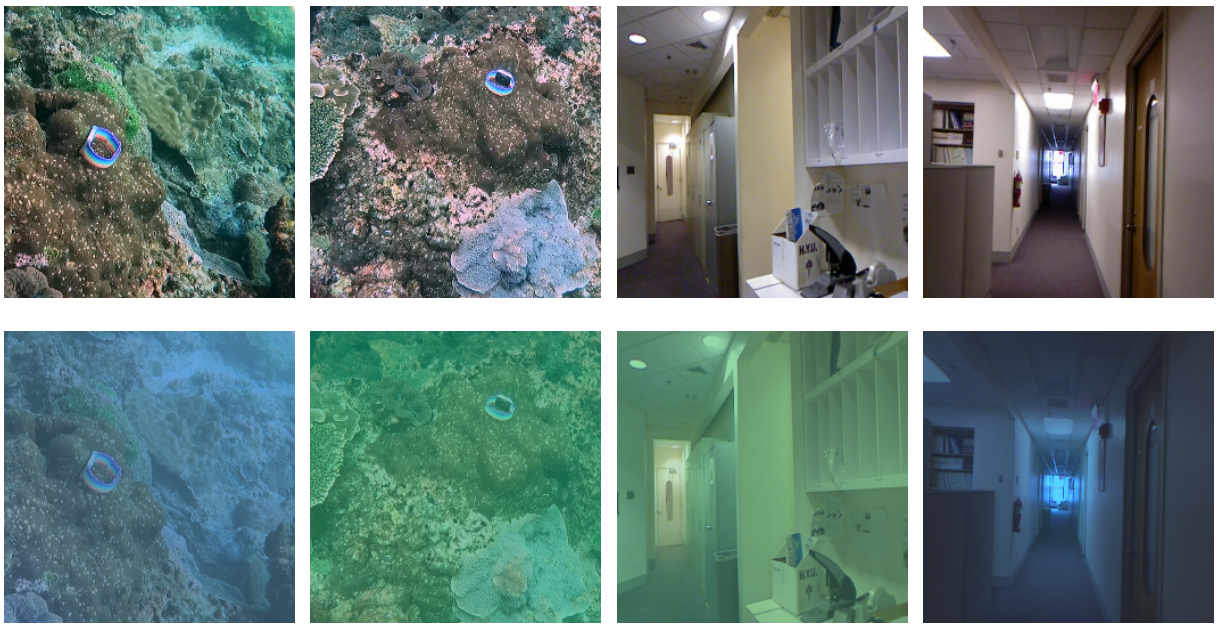


Figure 3.4: Synthetic underwater data being produced by SeaThru method. The top row contains real underwater and in-air images, and the bottom row is the result of synthesizing images. The synthetic images exhibit more green/blue colors, and feature a light haze. [2]

3.3.3 Pix2Pix for Image-to-Image Translation

The attained paired dataset was used to train Pix2Pix to try and learn a mapping to reverse the effect of the synthesis. On each training epoch, the model iterates through a training dataset consisting of 9449 images. Of these images, 1449 are in-air, and the other 8000 are from underwater videos. The model is initialized with the architecture in Table 3.7, including a ResNet 9-block generator and a 70×70 PatchGAN discriminator. On each iteration an image from Real A is passed through the generator to produce a Fake B. The objective function for Pix2Pix is as follows:

$$\Omega_{Pix2Pix} = \min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda_{L1} \mathcal{L}_{L1}(G) \tag{3.23}$$

Hyperparameter	Value	Description
Generator Model	ResNet 9-Block	Architecture used for the generator
Discriminator Model	70×70 PatchGAN	Architecture used for the discriminator, an N-layer discriminator, featuring 3 layers.
Optimizer	Adam	The type of optimizer used for each generator and discriminator. Initialized with a first moment and second moment of 0.50 and 0.99, respectively.
Batch Size	1	The number of images processed on each iteration.
l Pool Size	0	The number of previous outputs the model keeps in memory.
Learning Rate	0.0002	The initial learning rate of the model.
GAN Mode	Vanilla	Means that the discriminator loss uses BCE with Logits loss on the discriminator score.
Learning Rate Policy	Linear	The method employed for decaying the learning rate. It is kept constant for the first 100 epochs, and then follows a linear decay over the last 100 epochs.
Total Epochs	200	The total number of epochs in the model's training phase.

Table 3.7: Table showing pre-training hyperparameters for Pix2Pix.

where λ_{L1} is the weight for L1 loss, set as 100. Doing so, gives the L1 loss 100× more importance than the adversarial loss. This can be broken down into the following equations:

$$\mathcal{L}_{L1}(G, D, X, Y) = \sum \mathbb{E}_{x \in X, y \in Y} \Big[||y - G(x)|| \Big] \tag{3.24}$$

$$\mathcal{L}_{cGAN}(G, D, X, Y) = 0.5 \cdot \mathbb{E}_{x \in X, y \in Y} \Big[\mathcal{L}_{BCE}(D(y), 1) + \mathcal{L}_{BCE}(D(G(x), 0) \Big] \tag{3.25}$$

It does this using a cross-entropy loss wrapped in a sigmoid function, which is calculated as follows:

$$\mathcal{L}_{BCE}(x, y) = -\Big[y \cdot \log(x) + (1 - y) \cdot \log(1 - x) \Big] \tag{3.26}$$

3.3.4 Pix2Pix with Depth Loss

With the intent of keeping outputs of similar scenes consistent, the method used for Pix2Pix is then extended to provide the model with feedback on how its transformation affects the depth information of the image. The goal of this is to give the model a higher loss when it alters the depth information of the image. This is implemented by extending the dataset to contain RGB-D images of the real images, by using Depth Anything by Yang et al. [53]. The model from the previous section is then loaded for a second training phase. In this phase, it gains a new loss function, that calculates an L1 loss between the real depth map and fake depth map:

$$\mathcal{L}_{\text{depth}}(G, X, Y) = \mathbb{E}_{x \in X, y \in Y} \big| F_{\text{depth}}(y) - F_{\text{depth}}(G(x)) \big| \tag{3.27}$$

where F_{depth} is the DepthAnything-Large model. However, on the NYUD dataset, DepthAnything-Large is not required, as the dataset uses the pre-defined depth maps. Using this the models objective function is updated to be as follows:

$$\Omega_{\text{Pix2Pix-D}} = \min_G \max_D \mathcal{L}_{cGAN}(G, D, X, Y) + \lambda_{L1} \mathcal{L}_{L1}(G, X) + \lambda_D \mathcal{L}_{\text{depth}}(G, X) \tag{3.28}$$

This loss is experimented with a weight, λ_{depth}, of 1, 5, and 10.

3.3.5 Pix2Pix with Video Loss

On the same notion as in 3.3.4, Pix2Pix is given a different loss function to try and encourage temporal smoothing. The dataset is altered to contain TIFF images containing 5 consecutive video frames. This dataset did not include in-air images as they have no sequential nature. The model has the parameters loaded from 3.3.3, and is provided with a new loss function, as follows:

$$\mathcal{L}_{\text{video}} = \mathbb{E}_{x \in X, y \in Y} \Big[\frac{1}{5} \sum_{i=1}^{5} \big| y_i - G(x_i) \big| \Big] \tag{3.29}$$

$$\text{where } \{x_1, x_2, x_3, x_4, x_5\} = x \in X \text{ and } y_{\times 5} = \{y_1, y_2, y_3, y_4, y_5\} \text{ where } y \in Y$$

This is taking an average of the L1 loss over the 5 frames, which should encourage the model to be more consistent over a video clip. This new loss is experimented with values of $\lambda_{\text{video}} = 1, 10, 50$. This addition means the objective function is now defined as follows:

$$\Omega_{\text{Pix2Pix-V}} = \min_G \max_D \mathcal{L}_{cGAN}(G, D, X, Y) + \lambda_{L1}\mathcal{L}_{L1}(G, X) + \lambda_{\text{video}}\mathcal{L}_{\text{video}}(G, X) \tag{3.30}$$

3.4 Overall Contributions

Since the chapter is coming to an end, this section will include a summary of the contributions made in this chapter. The contributions of this project include:

- Gathering a diverse unpaired dataset for UIE.
- Generating a synthetic dataset for UIE by using state-of-the-art attenuation calculations [2] with a state-of-the-art monocular depth estimation model [53] on a hybrid of underwater and in-air images.
- Comprehensively studying the performance of image-to-image translation models on UIE.
- Modifying CycleGAN architecture to use 15 channel input rather than the regular 3, therefore resulting in video-to-video and video-to-image translations.
- Altering the objective function of Pix2Pix in an attempt to improve UIE and temporal consistency.

Chapter 4

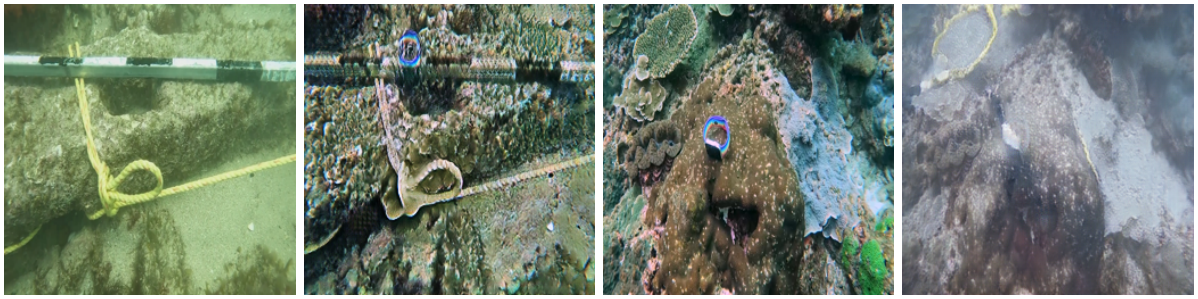
Critical Evaluation and Discussion

UIE is an ill-posed problem. The goal is undefined, and as a result, it cannot be determined exactly what defines a good enhancement. A typical underwater video will appear to have a blue or green tinge, with haze and inconsistent lighting worsening visibility. In contrast, perfect underwater images are often best likened to images taken in-air; they should have no loss of visibility, and no color degradation. Additionally, the attenuation underwater is quasi-periodic, that means models need to be adaptive to new environments in order to maintain temporal consistency. For paired training, quantitative measures such as mean squared error (MSE) can be applied to compare the result with the ground truth. However, in the the absence of ground truth, there are fewer metrics, and so comparisons will be more intuitive, featuring visualizations which will be compared qualitatively.

Due to the complexity of the project, the evaluation of tests will be separated into 3 distinct sections. Firstly, there will be qualitative analysis on the bad section of the unpaired dataset. Subsequently, the models will also have their perceptual quality compared with each other and other models using the SAUD benchmark dataset [27]. Finally, there will be an analysis of each model's effect on temporal consistency for videos.

4.1 Dataset

Compiling the dataset was a large portion of this project, whether it was finding and classifying videos for unpaired training, or creating synthetic data for paired training. In this section, the successes and shortcomings of the dataset from throughout the project will be discussed.



(a) Real A (b) Fake B (c) Real B (d) Fake A

Figure 4.1: 4.1a and 4.1b show a CycleGAN $A \rightarrow B$ generator hallucinating an Apple watch into existence. In contrast, 4.1c and 4.1d show the reverse generator mapping the characteristic rope and pole from 4.1a into the output image.

4.1.1 Collecting Data

Creating a dataset for both good and bad underwater images was an arduous task. Early in model training, there was a realization that the dataset must be very diverse if the model was going to be robust. Figure 4.1 shows the output from CycleGAN trained on the earlier dataset, only including videos

from Table 3.1. In a large proportion of the good videos, there is a distinct, "rainbow wrist-banded" Apple watch. Also, in the "bad" videos, there is often a characteristic black and white pole with a yellow rope. Since the model only contained a limited number of scenes, it wrongly learnt that all good images probably contain an Apple watch, whereas the bad ones are likely to contain a yellow rope and stripey pole. As a result, outputs from the model would contain hallucinations of these objects. Onwards from this point, the data was collected with a large concern for the variety of scenarios being showcased in the images. As a result, the dataset for bad images ended up being quite diverse, but would have benefited from being larger. On the other hand, the good dataset was large enough, but a lot of the data came from a small set of long videos, which potentially made the models less robust. The inclusion of in-air photos in the synthetic dataset helped the model to pick up better color corrections, but could have benefited from more in-air photos.

4.1.2 Synthetic Data

In this project, the synthetic dataset was an extremely useful tool as it allowed both paired training, and the usage of quantitative measures for the evaluation. Furthermore, it permitted the inclusion of in-air images, which gives some of the data an absolute ground truth, with the assumption that in-air images have good enough visibility and color balances. Since it allowed paired training, it opened up avenues for the project to experiment with a more extensive range of model architectures. Examples of the synthetic images can be seen in Figure 4.2, showing images being created with a diverse range of water conditions being emulated.

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

Figure 4.2: Images showcasing good raw images, both in-air and underwater, as well as their synthetically degraded counterparts - top and bottom, respectively.

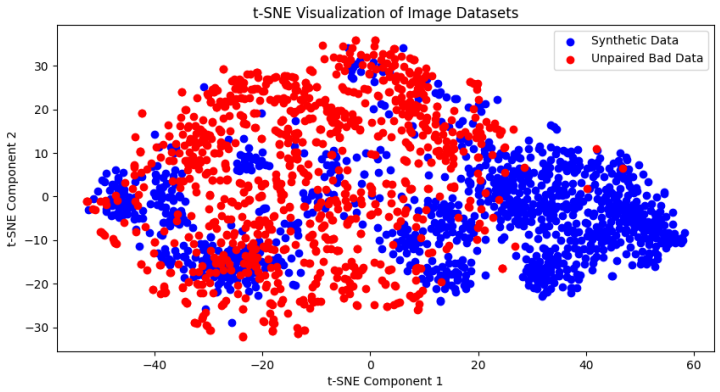


Figure 4.3: t-SNE results comparing the distribution of features in a sample of the synthetic dataset, and a sample of the bad domain from the unpaired dataset.

To evaluate the success of the synthetic dataset, t-distributed stochastic neighbour embedding (t-SNE) was used. This is a method for visualizing high dimensional data, such as images, by giving each data point a location in a 2D map. By visualizing the data in this way, it offers an insight into the structure of data, even when at different scales. It was proposed by Maaten et al. in 2008, and was offered as a useful way of representing relations between data that is related in some low-level way. It does this by calculating pair-wise similarities, which involves defining an underlying probability distribution corresponding to the likelihood that one point would pick the other point as its neighbour. Then it uses Kullback-Leibler divergence to minimize between the constructed conditional probability distributions, resulting in a lower-dimensional view of the data, in which more similar points are plotted close together. This method was used to evaluate the success of the image synthesis, comparing the synthetic dataset with the bad images from the unpaired dataset. The result of t-SNE can be seen in Figure 4.3, and shows a clear correlation between the synthetic and real bad dataset. This is a strong indicator that the synthetic dataset was a good representation for bad underwater data. However, there is a clear region with lack of overlap beyond x=30, indicating that there is room for improvement in either the data selection or the synthesizing method [45].

4.2 Qualitative Analysis

As UIE is an ill-posed problem, each solution is best judged by a qualitative analysis. This section contains an overview of each model's qualitative performance on the test dataset. To evaluate the qualitative performance, each model was applied to every image in the unpaired test set, and results were hand-picked to display both the strengths and weaknesses of the respective model.

4.2.1 CycleGAN Image-to-Image

The outputs after using CycleGAN on the test set can be viewed in Figure 4.4, and these show that for the most part, the model learns good color corrections. This is exhibited in almost all of the images, but is especially evident in 4.4b, as the yellow rope and black and white pole display stronger colors. Furthermore, the model appears to cope well with poorly lighted images, as can be seen in 4.4h as it makes the details in the image far more visible. However, 4.4e is an example of CycleGAN struggling to restore the image, as it merely changes the shade of blue, whilst introducing noise to the scene as it tries to de-haze. Furthermore, in these kinds of scenes, there is more uncertainty for the model, which results in temporal inconsistencies when applied to videos. This effect is exhibited in Figure 4.5, and is something other models will hopefully improve upon.

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

Figure 4.4: Results of CycleGAN processing.

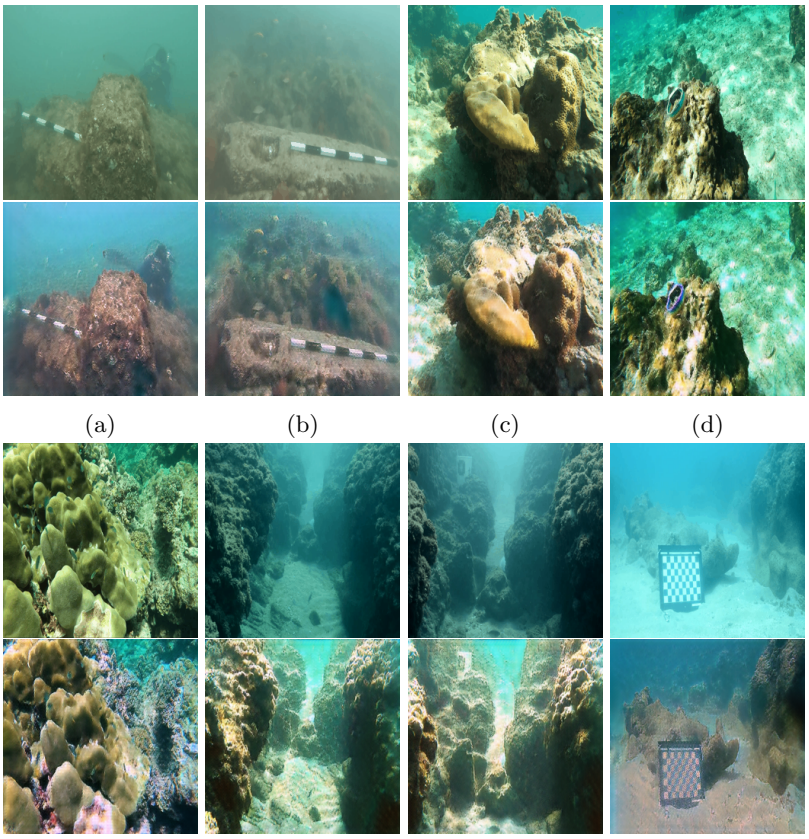
Figure 4.5: Set of frames from SMD's dataset, taken less than 1 second apart, being processed by CycleGAN model exhibiting lack of temporal consistency

4.2.2 CycleGAN Video-to-Video

Extending CycleGAN to generate and discriminate videos was, in hind-sight, an ambitious attempt at achieving temporal consistency. The results, shown in Figure 4.6, are the result of performing a video-to-video translation, and taking the middle frame of the input and output. This is done as a workaround to enable showing some results in the report, which does not support videos. The results show that while the model learns some good color corrections, it becomes very generative, resulting in hallucinations of structures that are not there. For example, in 4.6b the model color corrects and de-hazes, but with no consideration for the spatial features of the images, resulting in rock structures that occlude the haze. The shortcomings of this model could potentially have been improved by making architectural changes, such as giving the discriminator more layers, due to the increased complexity of the inputs and outputs.

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

Figure 4.6: Outputs of CycleGAN video-to-video on the test set.



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

(e) (f) (g) (h)

Figure 4.7: Diagrams showing the middle frame of the input, alongside the singular-frame output for the CycleGAN video-to-image model.

4.2.3 CycleGAN Video-to-Image

After deeming video generation to be too ambitious with the current architecture, the model was altered back to image output, whilst still taking video as input. In this configuration, the model is aiming to enhance the middle (or $3rd$) frame of the input by considering the two frames either side of it in the video. In doing so, the model is able to better understand the scene, and average out any changes in lighting that occur. An example of the results from the test set are shown in Figure 4.7, and this shows that the model was able to perform strong color corrections and improve lighting, with especially good de-hazing. In particular, 4.7f shows a large improvement in lighting and image sharpness, with the depth of the image being maintained. Furthermore, 4.7b shows the model successfully improving the contrast in the image, and improving the definition of previously subtle objects. In summary, this model shows significant strength at both de-hazing and color correction, and exhibits the potential for usage of a video generator and discriminator for UIE.

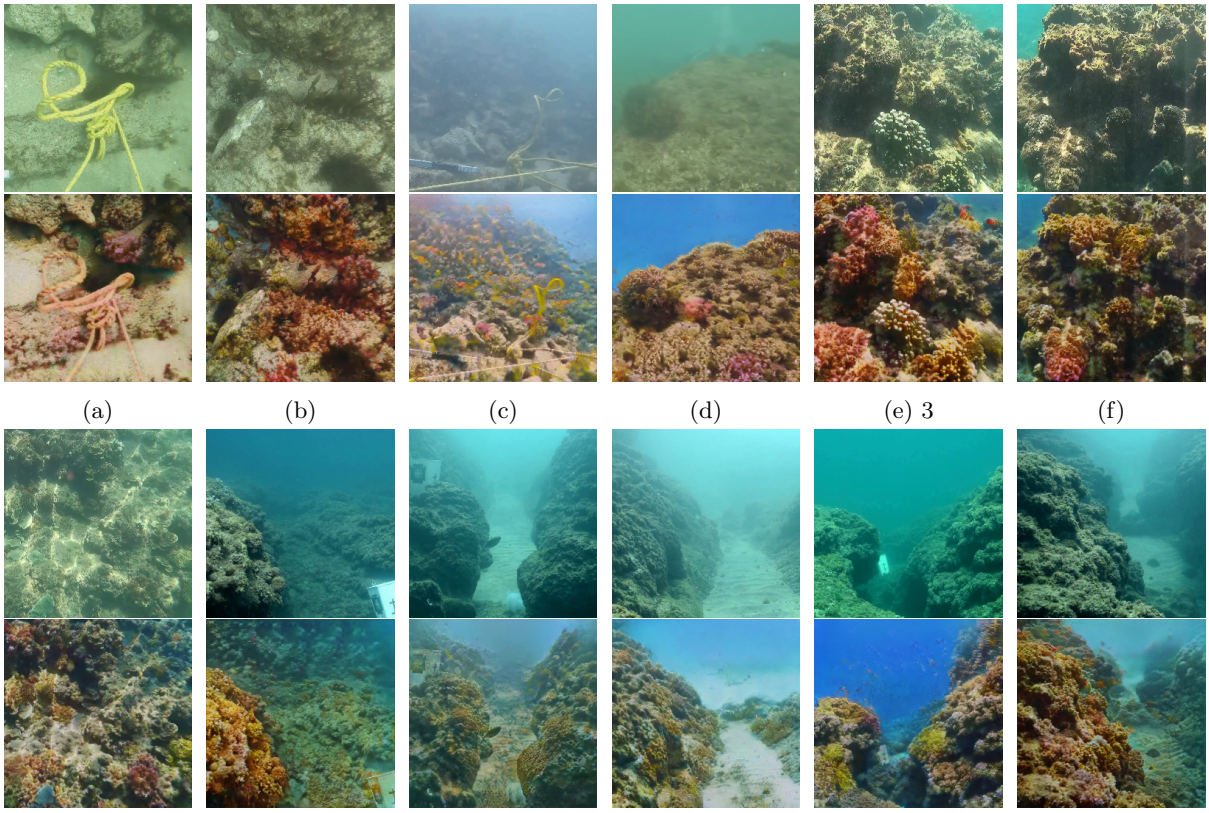


Figure 4.8: Diagrams showing inputs and outputs of UVCGAN from the unpaired testing dataset.

4.2.4 UVCGAN

The shortcomings of CycleGAN were likely due to an overly simplistic architecture or objective function. For this reason, UVCGAN was trained with the same goal as CycleGAN. These models are both designed for unpaired image-to-image translations, but UVCGAN is given the advantage of having a more modern architecture, such as by employing a vision transformer. UVCGAN also utilizes a pre-training phase that should have improved its weight initialization for the main training phase. These hypotheses proved to be correct as the model performed far better on all kinds of scenes. The results on the unpaired test set can be seen in Figure 4.8, and for the most part these show far more natural colors, with almost all haze removed. Pair 4.8j shows a prime example of haze removal, as it improves the contrast and colors in the foreground, whilst pushing back the background haze. Additionally, pair 4.8b shows the model performing very effective color corrections on the underwater wildlife. However, some images such as in 4.8a and 4.8c, exhibit bad behaviours of the model. For example, in 4.8a the model actually overcompensates in the color alterations, making what is clearly a yellow rope into a red one. Furthermore, in 4.8c, the model becomes too generative in trying to de-haze, and ends up translating what appears to be noise into rocks

and fish. Moreover, whilst color is being attributed to the noise, it is causing the main features of the image, i.e. the rope and pole, to lose their distinctness. Overall, UVCGAN performed well on the test set, as it showed the most significant changes in color, but did not always respect the spatial information in images.

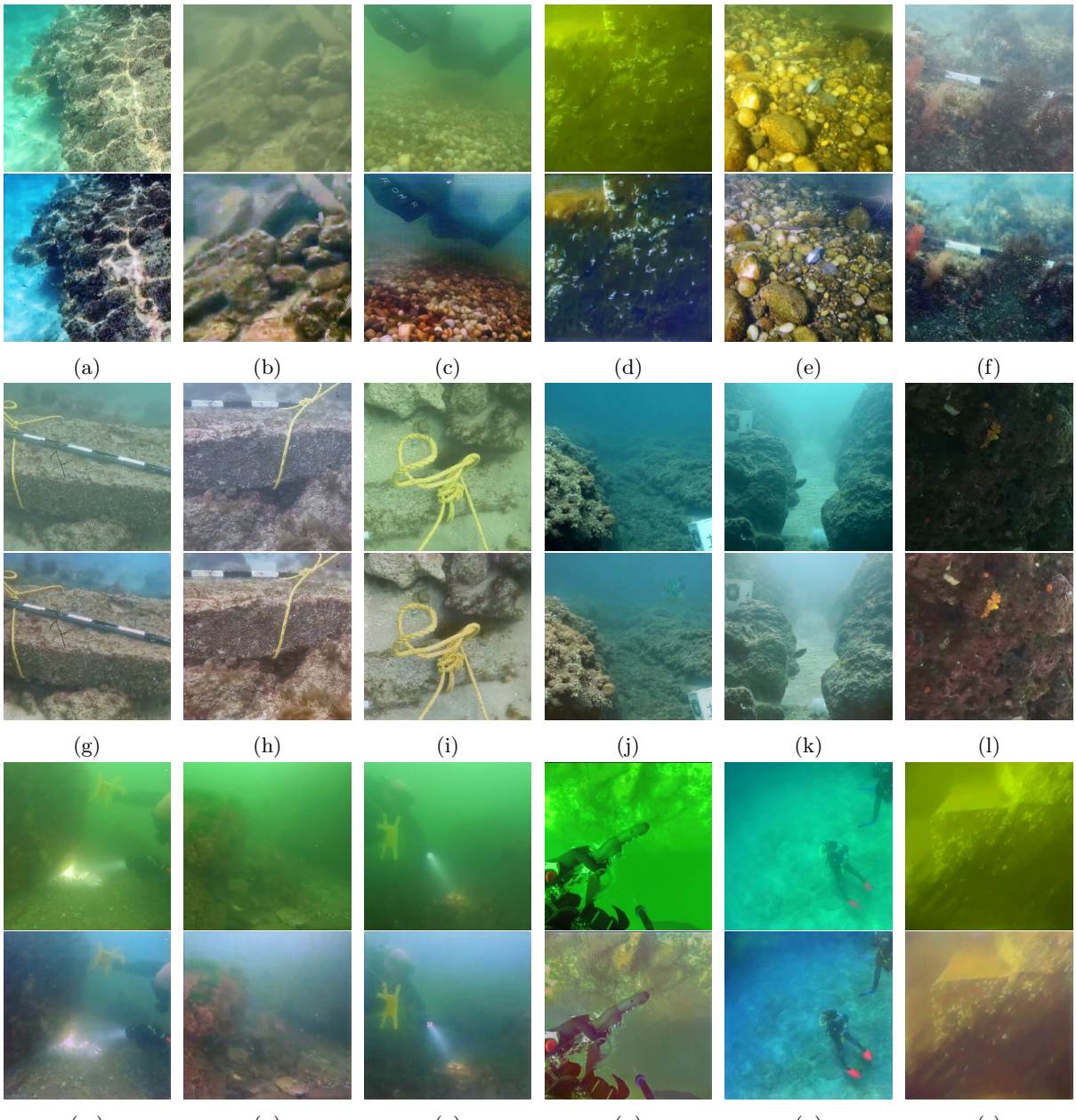


Figure 4.9: A large set of images from the unpaired testing dataset, showcasing the outputs from Pix2Pix when trained on the synthetic dataset.

4.2.5 Pix2Pix Image-to-Image

Creating a synthetic dataset enabled the possibility for paired training. For this reason, Pix2Pix was utilized due to its similarity to CycleGAN. When training Pix2Pix, there was a hypothesis that it would perform better due to the more quantitatively defined goal. The results from training Pix2Pix on the paired synthetic data can be seen in Figure 4.9, and demonstrate the (qualitatively) best results of all the experimented models so far. It appears to be very strong at color correcting, even in extreme cases of light attenuation such as in 4.9p. In close-up images, such as 4.9i, it could be mistakenly concluded that the image was taken in-air, with no obvious signs of attenuation. Alternatively, in images that span large

areas, the haze is dealt with suitably. For example, in 4.9m, there is significant hazing, but the model is able to improve the definition where possible, without being too generative. This model has shown the best overall UIE, as it consistently provides outputs with strong color corrections, and without altering the image's spatial information.

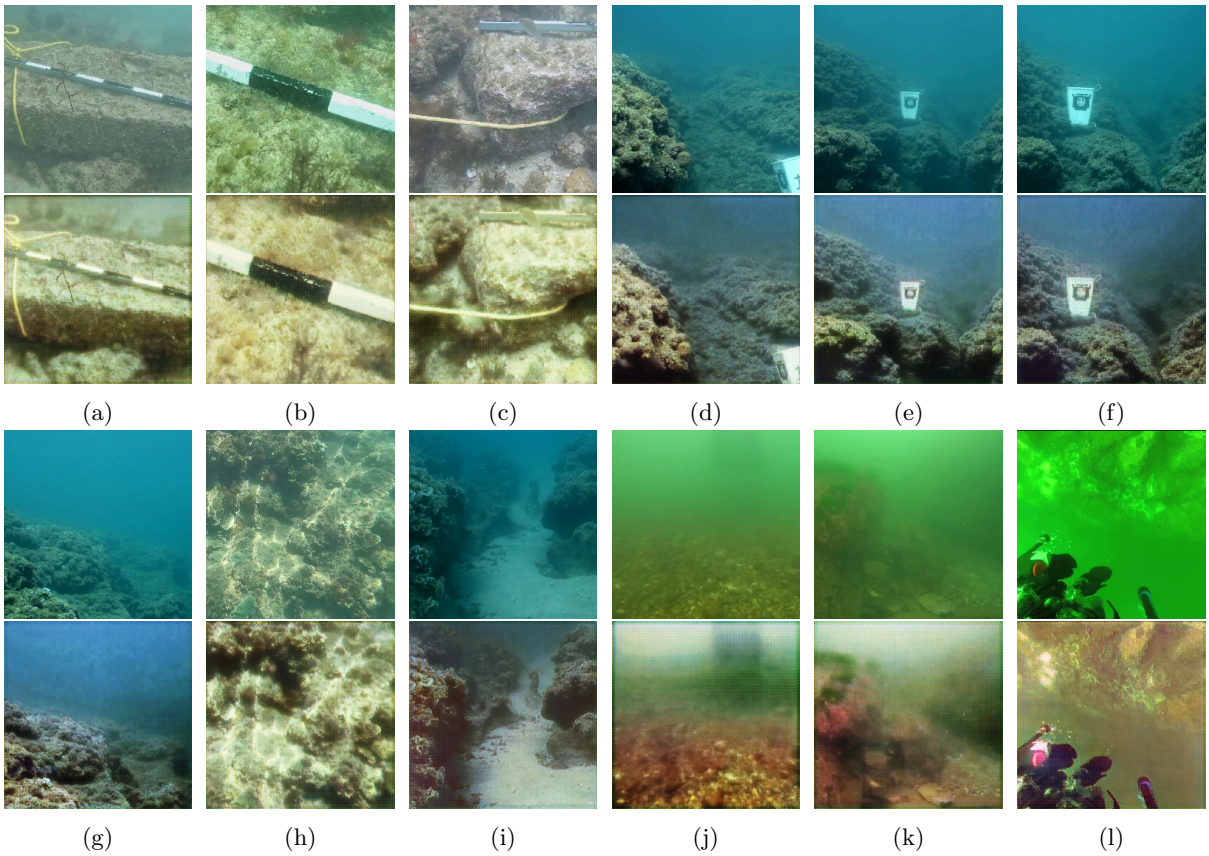


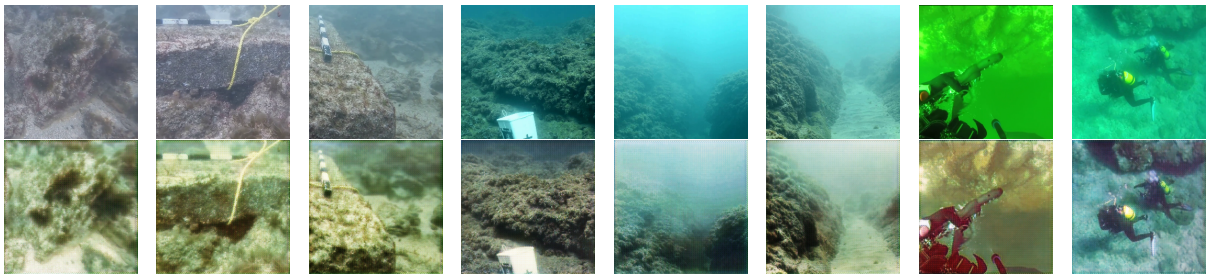
Figure 4.10: Diagrams showing inputs and outputs of Pix2Pix with depth loss from the unpaired testing dataset.

4.2.6 Pix2Pix with Depth Loss

Adding a depth loss function to Pix2Pix was done with the intention of allowing the model to gain a better gauge for the spatial information of the scene. The results of this model can be seen in Figure 4.10, and show that the model excelled at de-hazing images. In each of the images, but especially 4.10j and 4.10k, the model removes the thick haze, and improves visibility of previously faint objects. Unlike the other model's such as CycleGAN, this model also knows when to stop, as in all of the images the de-hazing seems accurate, whereas CycleGAN would hallucinate in order to de-haze. However, in some images, such as 4.10a and 4.10c, the image appears to become a bit blurred, which is probably due to the model trying to color correct them to be overly yellow, which is causing a loss of definition for the true yellow objects, like the rope. In summary, the depth loss definitely shows potential for de-hazing, but in future should be implemented to keep the color correction and image sharpness a priority.

4.2.7 Pix2Pix with Video Loss

Supplementing Pix2Pix with a video loss function was done with the aim of improving the model's temporal consistency. The results of the model on the test-set are shown in Figure 4.11, and display results similar to with the depth loss. For example, the model is frequently able to color correct, but often makes outputs too yellow, such as in 4.11a, 4.11b, and 4.11c. However, images 4.11d and 4.11e show the model's strength in de-hazing images. Similarly to the model with depth loss, this model shows an overall strength in de-hazing, but in future should be altered to give more significance to color correction.



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

Figure 4.11: Diagrams showing inputs and outputs of Pix2Pix with video loss from the unpaired testing dataset.

4.3 Quantitative Analysis

Without access to paired data, the methods used for analyzing the model's performance must be less direct. With paired data, the model could have a direct comparison with the ground truth, by using pixel-wise comparisons such as mean squared error or L1 loss. Since the primary goals of the model are to improve the image quality whilst maintaining the spatial information of the image, the analysis must use methods that can assess the image quality and measure the consistency between the input and output.

4.3.1 Metrics

Mean squared error (MSE) is a very popular method in paired machine learning for measuring the difference between the model output and the ground truth. It is commonly used in cases where the model is specifically wanted to penalize outputs far from the correct output. It is defined as follows:

$$\text{MSE(x,y)} = \mathbb{E}_{i \in x, j \in y} |i - j|^2 \tag{4.1}$$

where X and Y are images. Peak signal-to-noise ratio (PSNR) is a popular metric that utilizes MSE in order to help find the perceptive quality of an image or video. It is defined by the following:

$$\text{PSNR}(x, y) = 10 \cdot \log\Big(\frac{(I_{max} - I_{min})^2}{\sqrt{\text{MSE}(x, y)}}\Big) \tag{4.2}$$

where I_{min} and I_{max} is the minimum and maximum possible pixel value in the image, respectively [20]. Generally, it is preferable for an image to have a high PSNR score, as it means that the MSE was lower. Structural similarity index (SSIM) is an accepted metric for measuring the quality of an image by making use of three components: luminance, contrast, and structure. It is defined as follows:

$$\text{SSIM}(x, y) = [l(x, y)]^{\alpha} [c(x, y)]^{\beta} [s(x, y)]^{\gamma} \tag{4.3}$$

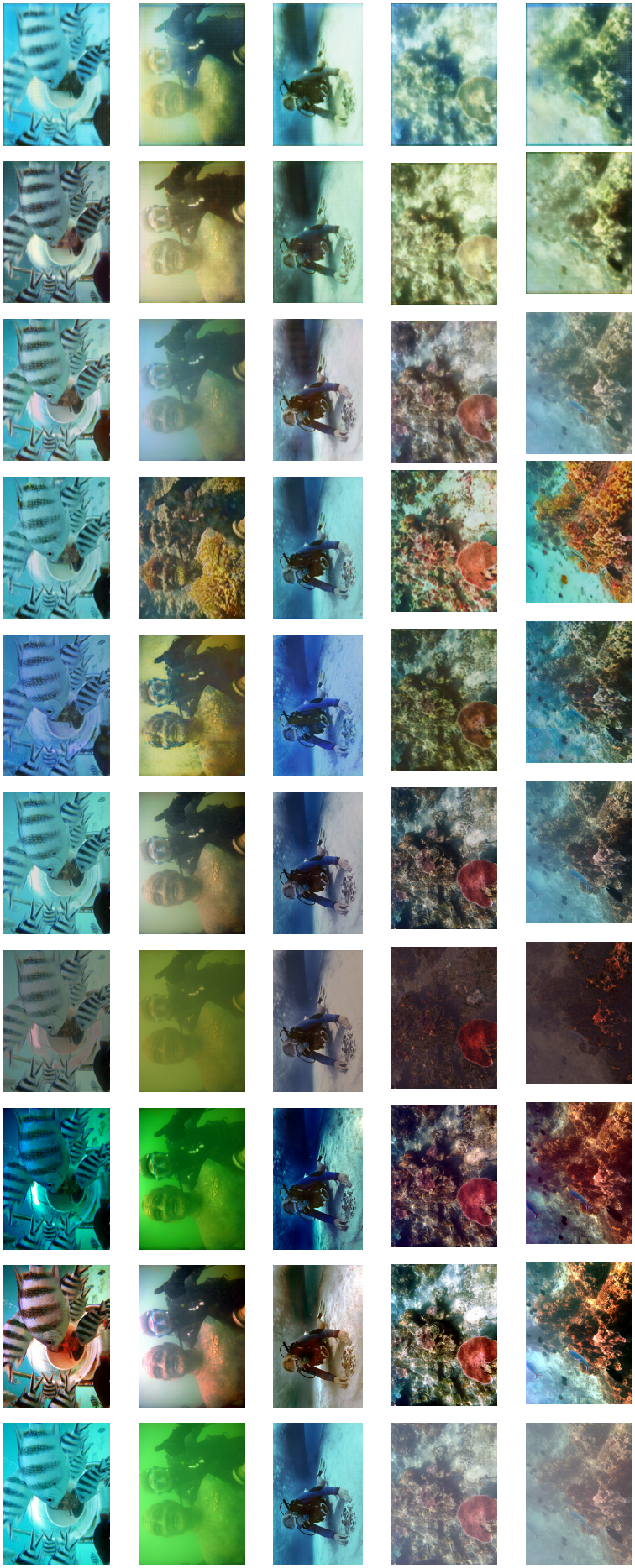
where

$$l(x, y) = \frac{2\mu_x \mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_2}, \tag{4.4}$$

$$c(x, y) = \frac{2\sigma_x \sigma_y + C_1}{\sigma_x^2 + \sigma_y^2 + C_2}, \tag{4.5}$$

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x \sigma_y + C_3}, \tag{4.6}$$

In these equations μ_x and μ_y represent the means of the images, σ_x and σ_y represent the standard deviations of the images, and C_1, C_2, and C_3 are constants introduced to make sure the denominator doesn't reach zero [13].



(a) Original (b) HP (c) UDCP (d) UWCNN (e) Water-Net (f) CycleGAN (g) UVCGAN (h) Pix2Pix (i) P2P-Depth (j) P2P-Video

Figure 4.12: Display of multiple models on benchmark dataset [27].

Model	Benchmark Dataset	
	PSNR ↑	SSIM ↑
CycleGAN	15.52	0.62
CycleGAN Video-to-Image	-	-
CycleGAN Video-to-Video	-	-
UVCGAN	20.78	0.72
Pix2Pix	17.59	0.74
Pix2Pix + Depth	14.64	0.63
Pix2Pix + Video Loss	14.47	0.62
Histogram Prior	13.30	0.64
UDCP	13.53	0.63
UWCNN	12.87	0.74
Water-Net	17.26	0.85

Table 4.1: Table showing quantitative model results on the benchmark dataset. Note that the video models are unable to produce scores for this, as there are only image inputs available. Also, higher PSNR and SSIM scores indicate a better image enhancement, but are not to be completely trusted, so shall be used alongside qualitative analysis.

4.3.2 Results

Up until this point, the models had only been tested on the unpaired dataset. This has likely resulted in a bias in the test results, with the unpaired models more likely to perform well due to the similarity between the training dataset and testing dataset, because they came from the same set of videos. Whilst less of an issue for the paired models, it is still a useful comparative technique to test each model's performance on a benchmark dataset. Unfortunately, at this point both video-input CycleGAN models will be unable to enter the comparison, because the benchmark dataset only contains images. This section will detail each model's quantitative and qualitative performance on the subjectively annotated underwater dataset (SAUD) [27], which was created by Jiang et al. Using this dataset provides a useful framework for comparing the results of the models, with other algorithms. Furthermore, it allows other researchers to reproduce the model's results on a standardized dataset. SAUD contains a range of image types, featuring scenes that have different water types, and different levels of attenuation. The diversity of the dataset means that any model or algorithm will have to be robust to different types of image for it to be successful. Table 4.1 shows the PSNR and SSIM scores for each model on the benchmark dataset.

Figure 4.12 contains 5 images from the benchmark dataset, each alongside the processed versions from each model. The images were selected to qualitatively evaluate the performance of the model on a range of images, with some easier and some harder. The set contains a variety of water-types and haziness, whilst the contents of the image range from rocks to humans. The first image contains a number of fish in the foreground, which partially occlude the reflection of a man's face. Due to the unique-ness of the image, it was deemed very difficult, and only the HP-based, UVCGAN, Pix2Pix, and Pix2Pix-Depth were able to improve the man's skin-tone. The HP-based model was the best at color correcting the fish, but ended up making the man's face overly red as a result. The second image contains a diver being photographed with a diver, in water causing a green tinge, with significant haze. All of the models from this project are able to remove the green tinge, but only HP-Based and Water-Net are able to from the comparison models. However, in this case UVCGAN was majorly at fault as it hallucinated colorful corals onto the image. CycleGAN and Pix2Pix + Depth appear have produced a good output, but they are overly yellow in comparison to Pix2Pix. The next image in the set contains a diver in water with a strong blue tint, with a background that spans a long distance. Pix2Pix was the best performer for this image, as it restores natural looking colors to the image, whilst all others do not. CycleGAN and UVCGAN make very little changes to the image, which suggests they can't properly comprehend this image. Penultimately, the fourth image contains a distant view of some underwater rocks, and has poor contrast. Almost models performed well on this image, which indicates it is an easier scene. UVCGAN, Pix2Pix, and HP-Based were the standout models for this image, as they displayed the best color restoration and contrast improvement. Finally, the last image contained a similar scene to the fourth, but with some fish in the foreground. CycleGAN and Pix2Pix performed well on this image, whilst UVCGAN was arguably too generative. Whilst UVCGAN's output looks pleasing, it is

inconclusive as to whether it would be a practical output.

Benchmark Qualitative Ranking

Model	Image					Average Ranking ↓
	1	2	3	4	5	
Original	7	9	9	7	6	7.6
HP-Based	3	**1**	2	2	2	**2.0**
UDCP	10	7	10	5	9	8.2
UWCNN	9	8	6	10	10	8.6
Water-Net	5	3	4	4	4	4.0
CycleGAN	6	6	8	6	**1**	5.4
UVCGAN	4	10	7	**1**	5	5.4
Pix2Pix	2	2	**1**	3	3	2.2
Pix2Pix + Depth	**1**	4	5	8	7	5.0
Pix2Pix + Video	8	5	3	9	9	6.6

Table 4.2: Table showing qualitative rankings for images in Figure 4.12. This includes a mean average of the ranks. For this, a lower ranking is better.

Furthermore, these benchmark images were qualitatively ranked, and summarized into Table 4.2. From the table, it can be seen that the HP-Based model performed best, closely followed by Pix2Pix. However, the rest of the models had a larger range of results, resulting in lower rankings. For example, UVCGAN performed admirably, but due to a terrible performance on image 2, it had its score significantly worsened. Overall, the qualitative results in Table 4.2 mostly concur with the quantitative scores in Table 4.1. The HP-Based method was a large outlier in this correlation, as it had one of the worst quantitative performances, but ranked highest qualitatively.

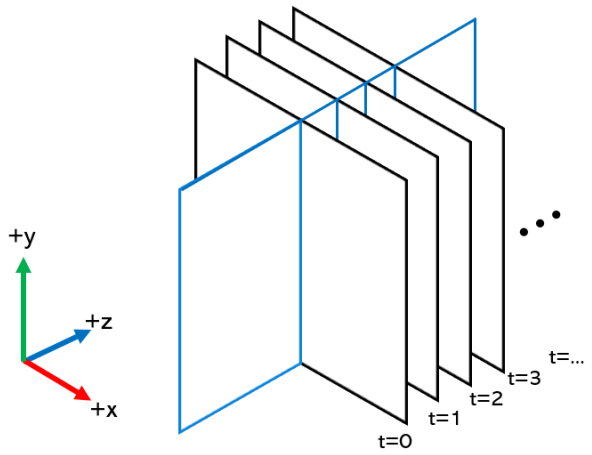
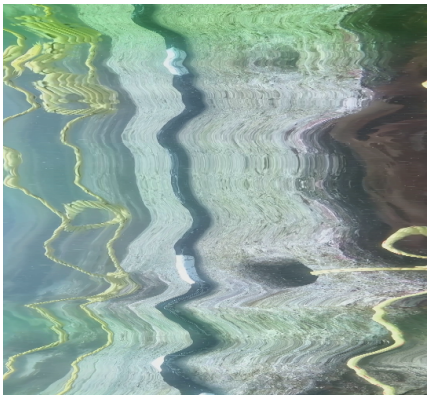


Figure 4.13: Diagram showing how the YZ plane is used to visualize temporal consistency.

4.4 Temporal Consistency

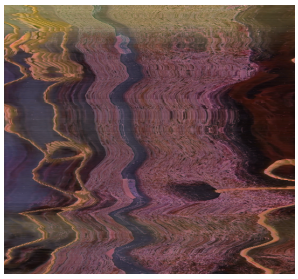
Applying a UIE algorithm to a monocular image has been done frequently, but with the spatial and temporal inconsistencies of underwater scenes, when they are applied to videos they are not able to provide consistent results. As discussed, the algorithm will need to be able to adapt to a sudden change in lighting, or similar. Temporal consistency is best visualized in video format, however visualizing temporal consistency in an image format is a unique task. In this section, a method similar to in [3] by Anantrasirichai, 2023, is deployed, in which an image of the YZ plane of a video is created, where the Z-axis is the temporal axis. This process is shown in Figure 4.13, which visualizes the process. To compile these images, the middle column of each image is taken, and used as a row for the new image. These rows are stacked vertically to produce the images in Figure 4.14. The results effectively display each model's ability to stay consistent when applied to A2 from the DRUVA dataset (see Table 3.1).

Importantly, this video was not included in the training data to try and remove bias. The reason for this video was due to the fact it contains an artifact (black and white pole) that stays in frame for the entirety of the video, which results in a clear consistent feature when plotting the YZ plane. Furthermore, this video has quasi-periodic lighting, as the video fluctuates between a green and blue tinge in non-linear periods. This can be seen in Figure 4.14a, which exhibits two green stripes, one at the top, and the other near the bottom. These results can be analyzed by looking at the colors, the color's consistency, as well as looking for the smoothness of vertical edges as an indicator of temporal consistency.

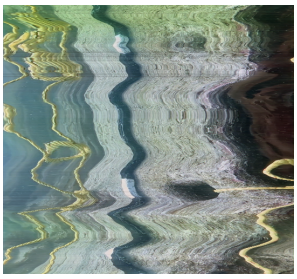


(a) Original

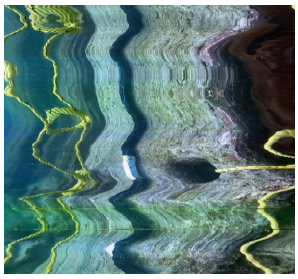
Figure 4.14: YZ plane image created by stacking the middle column, x=270 of each video frame, in original image. The same process is used on the processed images to produce the rest of Figure 4.14.



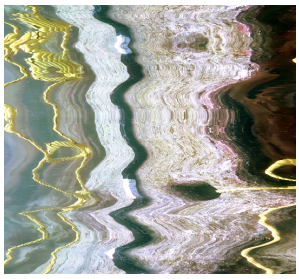
(b) UWCNN



(c) Water-Net



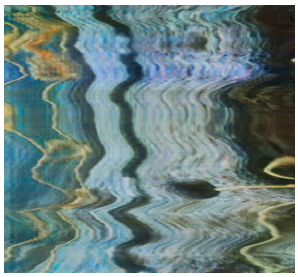
(d) UDCP



(e) Histogram Prior

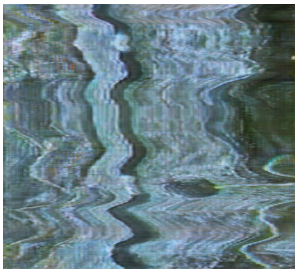
Figure 4.14: YZ plots for alternative UIE models.

Using Figure 4.14 as reference, the temporal consistency can be analyzed. Beginning, with the original image, 4.14a, the haze causes a low contrast in the colors, as well as periodic blue and green tinges which can be seen in horizontal stripes. The edges in the image are reasonably sharp, but appear to jitter which results in a less smooth video. Then looking at the comparison models, UDCP, see 4.14d, shows clear lack of consistency, as it is unable to counteract the green-tinged area. UWCNN, see 4.14b, stands out from the rest due to its unique color correction, which makes the image very dark and red. Additionally, it fails to counteract the green tinge, which results in two yellow stripes on the YZ plane image. Water-Net, seen in 4.14c, has the same issues with the stripes, but shows far better color corrections than the others mentioned. Finally, there is the HP model, 4.14e, which appears performs excellent color corrections and keeps a consistent tone throughout. However, looking at the vertical edges in the images, they are fairly sharp but appear to jitter, which is an indicator that the video will not be as smooth.

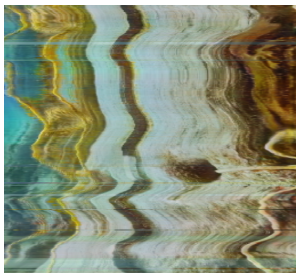


(f) CycleGAN Image-to-Image

(g) CycleGAN Video-to-Image

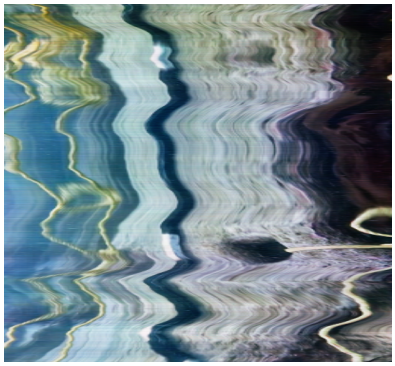


(h) CycleGAN Video-to-Video



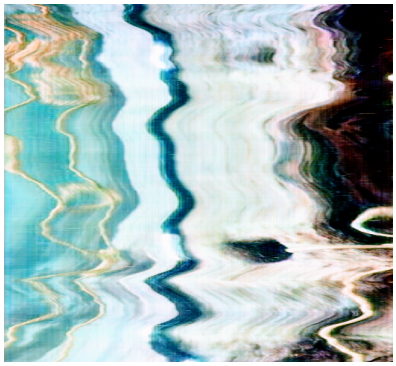
(i) UVCGAN Image-to-Image

The unpaired models from this project exhibited a variety in performances on this test. The original CycleGAN model, as seen in 4.14f, increases contrast, but remains with a significant blue tinge, which changes in intensity over time. The change in intensity can be seen as a stronger blue stripe that appears near the top of the image. Furthermore, the edges become less sharp, which indicates that it makes temporal consistency worse. Moreover, the CycleGAN video-to-video model, seen in 4.14h, is the worst performing of all the models, as it makes the whole image more blue, and makes edges less sharp. In contrast, CycleGAN video-to-image, seen in 4.14g results in an image with better color corrections as well as sharp edges, that are smooth throughout the image. This indicates that the model was able to use the information from neighbouring frames to inform its enhancement on the target frame. However, the model exhibits worse contrast than alternatives, as the intensity of the black and white pole are not as apparent. Similarly, UVCGAN appears to have smooth, sharp edges for the most part, but has other shortcomings. As seen in 4.14i, the yellow rope loses its distinctness as the model makes the surrounding rock overly yellow. As well as this, there are a few rows of the image that do not follow the pattern at all, indicating that some frames had an incohesive output.



(j) Pix2Pix Image-to-Image

(k) Pix2Pix with Depth Loss



(l) Pix2Pix with Video Loss

Continuing its performance from Section 4.3.2, Pix2Pix, see 4.14j, showed a strong sense of color correction, as for the most part colors are less blue, and have a higher contrast. Moreover, the model exhibits strong temporal consistency, which can be seen through the sharp vertical edges, and lack of periodic horizontal stripes (like in 4.14a). When extended to include depth loss, as in 4.14k, the model maintains good color corrections, but makes objects more distinguishable from each other. This can be observed by looking at the rope, which is sharper, and contrasts more from the rock. Notably, the resulting YZ plane for this model has less vertical edges, which could lead to conclude both that there is a loss of detail, and that the video is more temporally smooth. Furthermore, it is similar for the Pix2Pix with video loss model, which does not contain as many sharp edges, but also gives the image an overall higher intensity, with especially more blue colors, and introduces noise onto the pole. Overall, it seems that the model with depth loss is the best performing for this video because it provides the best balance of color restoration and temporal smoothing.

4.5 Run-time Analysis

Whilst perceptual image quality is the primary goal for this project, it is also important to consider the computational overhead involved with each model. In a practical scenario, the use-cases of UIE can vary

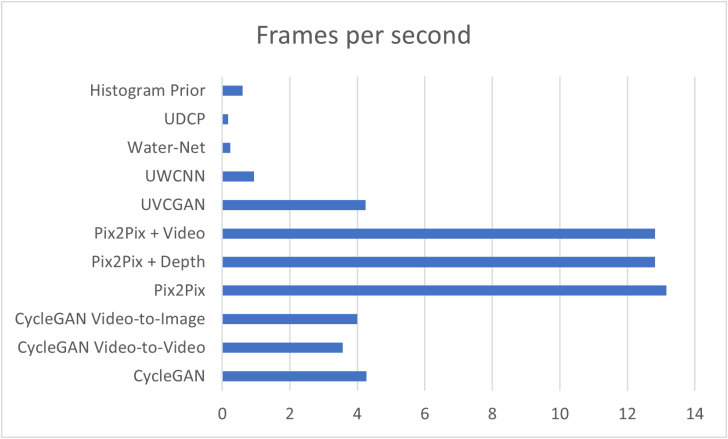


Figure 4.15: A bar chart comparing the FPS for processing with each model. This was calculated by timing the process of creating the YZ plane, dividing by the number of images processed (498), and taking the reciprocal of this value.

from post-processing to real-time processing. In the case of real-time, the algorithm needs to be very quick, so that there is minimal latency in the video. To assess the speed of each algorithm, a timer was used to measure the length of time required to process 500 frames for the YZ plane in Figure 4.14. This value was then divided by the number of frames, to get time per frame, and then the reciprocal was used to calculate frames per second (FPS). Figure 4.15 shows the results of measuring the FPS of each model, and shows that all of the model's from this project are able to run significantly quicker than the current alternatives. The models with Pix2Pix based architecture were able to run at an FPS greater than 12. The CycleGAN and UVCGAN models were also quick relative to the comparison models. For a video enhancement algorithm to run in real-time, it must run at the FPS of observed video, which in most cases is 30-60 FPS. The level of efficiency shown by Pix2Pix means that it is not far from being usable for real-time applications, unlike the others. Moreover, all of the FPS results were performed on my personal computer, which contains an 11th Gen Intel 8-Core i7-1165G7 @ 2.80GHz CPU and an NVIDIA GeForce MX350 GPU. Therefore with some improved hardware, it would likely be able to run quick enough for real-time applications. Ultimately, this means that the Pix2Pix models would be suitable for usage on ROVs and AUVs, which could be a step towards enhanced underwater exploration.

Chapter 5

Conclusion

The final chapter of this project consists of two sections. Firstly, there is a summary of the project, outlining the successes of the process in relation to the initial goals. Subsequently, there is a discussion of project completeness, and what potential there is for future work on this task.

5.1 Summary

This project set out with the broad, exploratory goal of training a generative deep learning model capable of underwater image enhancements, with a particular concern for temporal consistency. Both paired and unpaired models were explored, which meant there was a requirement for both types of dataset. The unpaired dataset came from a range of sources, resulting in a diverse range of underwater data, which allowed strong performance from the unpaired models. The original unpaired architectures were extended for video inputs, and as a result exhibited improved temporal consistency. Additionally, the paired dataset utilized a state-of-the-art method for synthesizing data, which led to a realistic synthesis from a hybrid of underwater and in-air images. Moreover, the paired models were able to match the performance of comparable modern UIE solutions whilst having less of a computational overhead, which makes it suitable for real-time usage. Furthermore, the model's produced in this project are able to display improved temporal consistency in relation to the alternatives that were explored, making them better suited for video enhancement. Ultimately, this project has been a success in both creating a strong UIE model, and exploring methods for improving temporal consistency.

5.2 Future Work

Whilst the project was an overall success, there were a number of avenues for future work that were identified throughout the process. Due to their simplicity, the focus of this project's exploration was with Pix2Pix and CycleGAN. However, there has since been a proliferation of numerous more complex architectures for image-to-image translations. One of these, UVCGAN, was explored, and showed potential but a lack of versatility. Despite this, there is room for more experimentation, as it is likely that with more strict restraints, it could produce state-of-the-art results. On the same notion, UVCGAN was only explored for its intended unpaired training, however, it seems a very logical idea to explore paired training using the Pix2Pix architecture equipped with the ViT transformer utilized at the bottleneck.

Additionally, this project has potential for extension by using a more sophisticated loss function with the depth and video loss. Similarly, a more complex method for ensuring temporal consistency could be used. Other research has shown the suitability of other methods, such as approximating optical flow [56], for implementing temporal consistency, and could be used as a parameter to improve the performance of the models.

Finally, despite the video-input models showing moderate success, their performance was likely hindered by the architectural simplicity, and a lack of data. Considering the model had a relatively similar number of parameters to an image-to-image translation model, it seems that if allowed more complexity it would be able to pick up more complex relationships. Moreover, since each iteration requires five times as much data, the model would probably benefit from a larger dataset. For this reason, with improved implementation, video generators and discriminators could have great potential for both image enhancements and temporal consistency.

Bibliography

[1] Derya Akkaynak and Tali Treibitz. A revised underwater image formation model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[2] Derya Akkaynak and Tali Treibitz. Sea-thru: A method for removing water from underwater images. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1682–1691, 2019. doi:10.1109/CVPR.2019.00178.

[3] Nantheera Anantrasirichai. Atmospheric turbulence removal with complex-valued convolutional neural network. *Pattern Recognition Letters*, 171:69–75, 2023. URL: https://www.sciencedirect.com/science/article/pii/S0167865523001447, doi:10.1016/j.patrec.2023.05.017.

[4] James A Anderson. *An introduction to neural networks*. MIT press, 1995.

[5] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[6] Devansh Arpit, Víctor Campos, and Yoshua Bengio. How to initialize your network? robust initialization for weightnorm & resnets. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/e520f70ac3930490458892665cda6620-Paper.pdf.

[7] S. S. Beauchemin and J. L. Barron. The computation of optical flow. *ACM Comput. Surv.*, 27(3):433–466, sep 1995. doi:10.1145/212094.212141.

[8] Brian Bingham, Brendan Foley, Hanumant Singh, Richard Camilli, Katerina Delaporta, Ryan Eustice, Angelos Mallios, David Mindell, Christopher Roman, and Dimitris Sakellariou. Robotic tools for deep water archaeology: Surveying an ancient shipwreck with an autonomous underwater vehicle. *Journal of Field Robotics*, 27(6):702–717, 2010. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20350, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20350, doi:10.1002/rob.20350.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[10] Xuelei Chen, Pin Zhang, Lingwei Quan, Chao Yi, and Cunyue Lu. Underwater image enhancement based on deep learning and image formation model, 2021. arXiv:2101.00991.

[11] Peng Dai, Xin Yu, Lan Ma, Baoheng Zhang, Jia Li, Wenbo Li, Jiajun Shen, and Xiaojuan Qi. Video demoireing with relation-based temporal consistency, 2022. arXiv:2204.02957.

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. arXiv:2010.11929.

[13] Richard Dosselmann and Xue Dong Yang. A comprehensive assessment of the structural similarity index. *Signal, Image and Video Processing*, 5:81–91, 2011.

[14] P. Drews Jr, E. do Nascimento, F. Moraes, S. Botelho, and M. Campos. Transmission estimation in underwater single images. In *2013 IEEE International Conference on Computer Vision Workshops*, pages 825–830, 2013. doi:10.1109/ICCVW.2013.113.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[16] Yecai Guo, Hanyu Li, and Peixian Zhuang. Underwater image enhancement using a multiscale dense generative adversarial network. *IEEE Journal of Oceanic Engineering*, 45(3):862–870, 2020. doi:10.1109/JOE.2019.2911447.

[17] Kaiming He, Jian Sun, and Xiaoou Tang. Single image haze removal using dark channel prior. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1956–1963, 2009. doi:10.1109/CVPR.2009.5206515.

[18] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. arXiv:1606.08415.

[19] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116, apr 1998. doi:10.1142/S0218488598000094.

[20] Alain Horé and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369, 2010. doi:10.1109/ICPR.2010.579.

[21] Johann Huber. Batch normalization in 3 levels of understanding. https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338, 2020.

[22] Veerle A.I. Huvenne, Katleen Robert, Leigh Marsh, Claudio Lo Iacono, Tim Le Bas, and Russell B. Wynn. *ROVs and AUVs*, pages 93–108. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-57852-1_7.

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL: https://proceedings.mlr.press/v37/ioffe15.html.

[24] Md Jahidul Islam, Marc Ho, and Junaed Sattar. Understanding human motion and gestures for underwater human–robot collaboration. *Journal of Field Robotics*, 36(5):851–873, 2019. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21837, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21837, doi:10.1002/rob.21837.

[25] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

[26] Jules S. Jaffe. Underwater optical imaging: The past, the present, and the prospects. *IEEE Journal of Oceanic Engineering*, 2015.

[27] Qiuping Jiang, Yuese Gu, Chongyi Li, Runmin Cong, and Feng Shao. Underwater image enhancement quality evaluation: Benchmark dataset and objective metric. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(9):5959–5974, 2022. doi:10.1109/TCSVT.2022.3164918.

[28] Junho Kim, Minjae Kim, Hyeonwoo Kang, and Kwang Hee Lee. U-gat-it: Unsupervised generative attentional networks with adaptive layer-instance normalization for image-to-image translation. In *International Conference on Learning Representations*, 2020. URL: https://openreview.net/forum?id=BJlZ5ySKPH.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. arXiv:1412.6980.

[30] JTO Kirk. Attenuation of light in natural waters. *Marine and Freshwater Research*, 28(4):497–508, 1977.

[31] Youssef Kossale, Mohammed Airaj, and Aziz Darouichi. Mode collapse in generative adversarial networks: An overview. In *2022 8th International Conference on Optimization and Applications (ICOA)*, pages 1–6, 2022. doi:10.1109/ICOA55659.2022.9934291.

[32] Chong-Yi Li, Ji-Chang Guo, Run-Min Cong, Yan-Wei Pang, and Bo Wang. Underwater image enhancement by dehazing with minimum information loss and histogram distribution prior. *IEEE Transactions on Image Processing*, 25(12):5664–5677, 2016. doi:10.1109/TIP.2016.2612882.

[33] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets, 2018. arXiv:1712.09913.

[34] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017. arXiv:1608.03983.

[35] Huimin Lu, Yujie Li, and Seiichi Serikawa. Computer vision for ocean observing. In H Lu and Y Li, editors, *Artificial intelligence and computer vision (Studies in Computational Intelligence, Volume 672)*, pages 1–16. Springer, Switzerland, 2017. URL: https://eprints.qut.edu.au/128521/.

[36] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks, 2017. arXiv:1611.04076.

[37] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014. arXiv:1411.1784.

[38] F. Moghimi M.K., Mohanna. Real-time underwater image enhancement: a systematic review. *J Real-Time Image Proc 18, 1509–1525*, (2021).

[39] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[40] Ori Nizan and Ayellet Tal. Breaking the cycle - colleagues are all you need. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2020.

[41] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. arXiv:1505.04597.

[42] Florian Shkurti, Anqi Xu, Malika Meghjani, Juan Camilo Gamboa Higuera, Yogesh Girdhar, Philippe Giguère, Bir Bikram Dey, Jimmy Li, Arnold Kalmbach, Chris Prahacs, Katrine Turgeon, Ioannis Rekleitis, and Gregory Dudek. Multi-domain monitoring of marine environments using a heterogeneous robot team. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1747–1753, 2012. doi:10.1109/IROS.2012.6385685.

[43] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[44] Dmitrii Torbunov, Yi Huang, Haiwang Yu, Jin Huang, Shinjae Yoo, Meifeng Lin, Brett Viren, and Yihui Ren. Uvcgan: Unet vision transformer cycle-consistent gan for unpaired image-to-image translation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 702–712, 2023.

[45] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[46] Nisha Varghese, Ashish Kumar, and A. N. Rajagopalan. Self-supervised monocular underwater depth recovery, image restoration, and a real-sea video dataset. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12248–12258, October 2023.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[48] Paul Viola and Michael Jones. Robust real-time object detection. volume 57, 01 2001.

[49] Junjie Wen, Jinqiang Cui, Zhenjun Zhao, Ruixin Yan, Zhi Gao, Lihua Dou, and Ben M Chen. Syreanet: A physically guided underwater image enhancement framework integrating synthetic and real images. *arXiv preprint arXiv:2302.08269*, 2023.

[50] Wikipedia contributors. Convolution — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Convolution&oldid=1212399231, 2024. [Online; accessed 8-April-2024].

[51] Jin Xu, Zishan Li, Bowen Du, Miaomiao Zhang, and Jing Liu. Reluplex made more practical: Leaky relu. In *2020 IEEE Symposium on Computers and communications (ISCC)*, pages 1–7. IEEE, 2020.

[52] Atsushi Yamashita, Megumi Fujii, and Toru Kaneko. Color registration of underwater images for underwater sensing with consideration of light attenuation. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4570–4575, 2007. doi:10.1109/ROBOT.2007.364183.

[53] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data, 2024. arXiv:2401.10891.

[54] Anqi Yi and Nantheera Anantrasirichai. A comprehensive study of object tracking in low-light environments, 2024. arXiv:2312.16250.

[55] Yonggyun Yu, Taeil Hur, and Jaeho Jung. Deep learning for determining a near-optimal topological design without any iteration. *Structural and Multidisciplinary Optimization*, 03 2019. doi:10.1007/s00158-018-2101-5.

[56] Fan Zhang, Yu Li, Shaodi You, and Ying Fu. Learning temporal consistency for low light video enhancement from single images. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4965–4974, 2021. doi:10.1109/CVPR46437.2021.00493.

[57] Shu Zhang, Ting Wang, Junyu Dong, and Hui Yu. Underwater image enhancement via extended multi-scale retinex. *Neurocomputing*, 245:1–9, 2017. URL: https://www.sciencedirect.com/science/article/pii/S0925231217305246, doi:10.1016/j.neucom.2017.03.029.

[58] Yihao Zhao, Ruihai Wu, and Hao Dong. Unpaired image-to-image translation using adversarial consistency loss. In *ECCV*, 2020.

[59] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.