

# Fast Underwater Image Enhancement for Improved Visual Perception

## 1 Introduction

### 1.1 Task Definition

Enhancing underwater images poses a unique challenge compared to terrestrial images. In deep water, red wavelengths are absorbed, resulting in a dominant green or blue hue. This causes irregular non-linear distortions, leading to underwater images that are often low-contrast, blurred, and color-degraded. To address this issue, a conditional Generative Adversarial Network (GAN)-based model [3] was designed specifically for enhancing underwater images. This model analyzes feasibility for real-time applications and formulates the problem as an image-to-image translation task [1]. It assumes the existence of a non-linear mapping between the distorted input images and the enhanced output images. The model was developed using the EUVP (Enhancement of Underwater Visual Perception) dataset and learns this mapping through adversarial training where the training was conducted in two modes:

1. Paired Training: The objective function guides the Generator to improve perceptual image quality, aiming for generated images that closely resemble the ground truth in high-level feature representation and global appearance. The Discriminator is used for rejecting images with inconsistent texture and style.
2. Unpaired Training: This mode does not enforce global similarity and content loss constraints due to the absence of pairwise ground truth. The focus is on learning the forward mapping and reconstruction while maintaining cycle-consistency.

### 1.2 Evaluation Protocol

State-of-the-art models, including the least-squared GAN and Wasserstein GAN, were compared to FUnIE-GAN using a mix of qualitative and quantitative performance evaluations. These comparisons showed that FUnIE-GAN could significantly improve the quality of underwater images, benefiting from both paired and unpaired training methods. The enhanced images notably improved the effectiveness of various underwater visual perception tasks, such as saliency prediction, object detection, and human pose estimation. These findings confirm that FUnIE-GAN is apt for real-time preprocessing within the autonomy pipeline of visually-guided underwater robots.

### 1.3 Dataset

The EUVP dataset comprises over 12K paired and 8K unpaired underwater images, collected from various oceanic explorations and human-robot cooperative experiments. Captured under diverse visibility conditions using different cameras like low-light USB cameras, GoPros, Aqua AUV's uEye cameras, and Trident ROV's HD camera [1], these images reflect a range of natural variability such as waterbody types and lighting conditions.

For the unpaired dataset (Figure 2), images of good and poor quality were separated based on visual inspection by six human participants, focusing on properties like contrast, sharpness, and foreground object identification. The paired dataset was prepared by distorting good quality images using a CycleGAN-based model trained on the unpaired data, creating corresponding pairs that facilitate the learning of domain transformation between good and poor-quality images.



Figure 1: Paired Instances: ground truth images and their respective distorted images.



Figure 2: Unpaired Instances: good and poor quality images are shown on the top and bottom row.

## 2 Neural Network / Machine Learning Model

### 2.1 Architecture

To perform automatic image enhancement, a conditional Generative Adversarial Network (GAN) model is used which consists of a generator and a discriminator. The generator progressively learns the transformation from the domain of poor quality images to enhanced images, while simultaneously engaging in a strategic iterative process with an adversarial discriminator, characterized by a min-max game.

The generator network (Figure 3) is based on the U-Net [2] structure. This network includes an encoder-decoder design with skip-connections that effectively facilitate image-to-image translation and quality enhancement. Skip connections are made across mirrored layers, connecting (e1, d5), (e2, d4), (e3, d2), and (e4, d4) [1]. To enable quicker inference, the FUnIE-GAN model adopts a simplified architecture with fewer parameters. The encoder network takes  $256 \times 256 \times 3$  images and learns 256 feature-maps of size  $8 \times 8$  and it collects features from the underwater image at various degrees of abstraction. The decoder then employs these feature-maps and inputs from the skip-connections to generate an enhanced image. They enable the decoder to use low-level encoder features. This is critical for retaining image structure and features during the enhancing process. This network is fully convolutional, using 2D convolutions with Leaky-ReLU [4] non-linearity and Batch Normalization [5] at each layer.

The discriminator (Figure 4) is built using the Markovian PatchGAN [5] architecture. This design operates under the principle that pixels are independent once they exceed a certain patch size, meaning the discriminator focuses on smaller segments of the image for its analysis. This method is particularly effective in identifying and emphasizing high-frequency details such as texture and style in local areas of the image. Additionally, this patch-level discrimination is more computationally efficient because it requires fewer parameters than if the entire image were to be analyzed in one go. The discriminator's structure includes four convolutional layers that process both real and artificially generated images, each sized at  $256 \times 256 \times 6$ ,

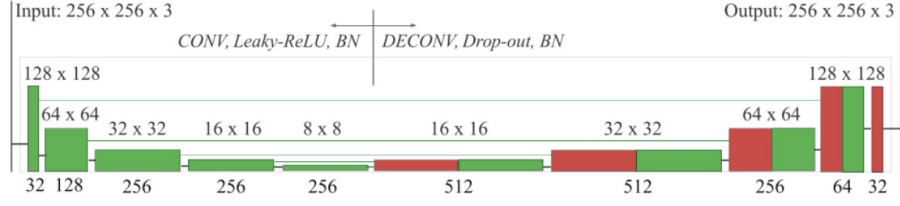


Figure 3: Generator architecture

and converts them into a  $16 \times 16 \times 1$  output. This output essentially averages the realness responses of the discriminator. Within these layers,  $3 \times 3$  convolutional filters are employed, with a stride of 2. Following this, like the generator, Leaky-ReLU non-linearity, and Batch Normalization (BN) are applied to further process the images.

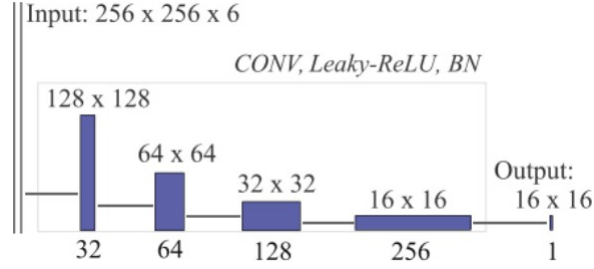


Figure 4: Discriminator architecture

## 2.2 Loss Function

In a typical conditional GAN-based model, the learning process involves mapping  $G : \{X, Z\} \rightarrow Y$ . Here,  $X$  represents the source domain,  $Y$  is the target domain, and  $Z$  stands for random noise. The model uses a conditional adversarial loss function to guide this process, which is expressed as equation 1. Generator  $G$  aims to reduce the conditional adversarial loss, whereas the discriminator  $D$  seeks to increase it.

$$L_{cGAN}(G, D) = E_{X,Y} [\log D(Y)] + E_{X,Y} [\log(1 - D(X, G(X, Z)))] \quad (1)$$

**Objective Function:** The objective function in FUNIE-GAN consists of three extra components to better assess the perceptual quality of images. These components are global similarity, image content, and local texture and style.

1. **Global Similarity:** Existing methods demonstrate that integrating an L1 (or L2) loss into the objective function helps the generator ( $G$ ) to produce samples that are globally similar to the target in the context of L1 (or L2) metrics. The L1 loss is preferred as it tends to cause less blurring. Therefore, the L1 loss included in the objective function is:

$$L_1(G) = E_{X,Y,Z} [\|Y - G(X, Z)\|_1] \quad (2)$$

2. **Image Content:** To ensure that the generated images closely match the target images in terms of content, a content loss term is added in the objective function. This term encourages the generator  $G$  to produce enhanced images that share similar features with the real target images. The image content function  $\Phi(\cdot)$  has been defined by the high-level features obtained from the block5 conv2 layer of a pre-trained VGG-19 [1] network. The content loss is:

$$L_{con}(G) = E_{X,Y,Z} [\|\Phi(Y) - \Phi(G(X, Z))\|_2] \quad (3)$$

3. Local Texture and Style: Markovian PatchGANs excel at capturing high-frequency details related to local texture and style. Therefore, we depend on the discriminator (D) to maintain consistency in local texture and style through adversarial methods.

**Paired Training:** The objective function directs the generator (G) to enhance the overall perceptual quality of images. Conversely, the discriminator (D) is tasked with rejecting any generated image that displays inconsistencies in local texture and style [1]. The objective function is as follows:

$$G^* = \arg \min_G \max_D L_{\text{cGAN}}(G, D) + \lambda_1 L_1(G) + \lambda_c L_{\text{con}}(G) \quad (4)$$

**Unpaired Training:** Since we do not have pairwise ground truth available, we do not apply constraints for global similarity and content loss [1]. Our goal here is to learn two mappings simultaneously: the forward mapping  $G_F : \{X, Z\} \rightarrow Y$  and the backward reconstruction  $G_R : \{Y, Z\} \rightarrow X$ . We achieve this by ensuring cycle-consistency in the process. The objective function is:

$$(G_F^*, G_R^*) = \arg \max_{G_F, G_R} \min_{D_Y, D_X} L_{\text{cGAN}}(G_F, D_Y) + L_{\text{cGAN}}(G_R, D_X) + \lambda_{\text{cyc}} L_{\text{cyc}}(G_F, G_R) \quad (5)$$

Here cycle-consistency loss as follows:

$$L_{\text{cyc}}(G_F, G_R) = E_{X,Y,Z} [\|X - G_R(G_F(X, Z))\|_1] + E_{X,Y,Z} [\|Y - G_F(G_R(Y, Z))\|_1] \quad (6)$$

## 2.3 Hyperparameters

The following hyperparameters were used while training the model:

- Data: Path to the image dataset.
- Architecture: There exist 2 versions of the generator architecture that was proposed by the authors.
- Workers: Number of data loading workers.
- Epochs: Number of total epochs to run.
- Batch-size: The total batch size of all GPUs on the current node.
- Learning-rate: Initial learning rate.
- gen-resume: path to latest generator checkpoint.
- dis-resume: path to latest discriminator checkpoint.
- save-path: Path to save the model.

## 2.4 Preprocessing of the Datasets

We have 3 Paired datasets: underwater.imagenet, underwater.scenes, and underwater dark. In these datasets, the validation data is not paired. Therefore, we merge all three datasets and redistribute the data, allocating 90% of the total pairs for training purposes and reserving the remaining 10% for validation.

## 2.5 Early stopping Regularization

The early stopping mechanism is a technique used in training machine learning models which involves halting the training process of a model before it has fully completed its training iterations. It monitors the model's performance on a validation set at regular intervals during training. If the model's performance on the validation set stops improving or starts to deteriorate, training is halted. This point is considered the optimal state of the model. This process prevents overfitting of the model, enhances efficiency, and helps in capturing the optimal performance of the model.

## 3 Experiment Design

### 3.1 Research Question

Can we enhance the quality of the images generated by tuning the hyperparameters and preprocessing the underwater datasets to achieve results comparable to the original model? Additionally, is it possible to obtain faster real-time results by employing early stopping regularization?

### 3.2 Proposed Modifications

- We are preprocessing the underwater datasets and experimenting with hyperparameters to train the conditional GAN model, aiming to achieve results comparable to the original model.
- Additionally, we are applying early stopping regularization to enhance the efficiency and capture the optimal performance of the model.

### 3.3 Hypothesis

Based on our understanding of the conditional GAN model, we hypothesize that by modifying the hyperparameters during training and testing and preprocessing the dataset, we can train a model that achieves comparable results to the original conditional GAN model. Additionally, we believe that our trained model will achieve optimal performance and prevent overfitting of the model with the help of early stopping regularization.

### 3.4 Independent Variables

1. Architecture: The architecture of the GAN model, which determines the structure of the generator and discriminator. The generator has 2 types of architecture: v1 and v2.
2. Learning rate: This affects how quickly or slowly a model learns.
3. Epochs: Total number of training cycles.
4. Batch size: Size of the data batches used during training.
5. `gen.resume` and `dis.resume`: Paths to resume training from a specific checkpoint for the generator and discriminator, respectively.
6. `train_loader` and `valid_loader`: Data loaders for training and validation datasets, which include aspects like data shuffling and batch sizes.

### 3.5 Control Variables

1. Dataset: Using the same datasets that were used for training for testing and evaluation.
2. Preprocessing: The preprocessing steps performed on the images before training should be consistent across all the experiments. This includes merging all three datasets and redistributing the data, allocating 90% of the total pairs for training purposes and reserving the remaining 10% for validation.
3. Seed variables like `np.random.seed`, `torch.manual.seed`, and `torch.cuda.manual.seed`: These seeds ensure reproducibility by controlling the randomness in the data shuffling and model initialization.
4. `is_cuda`: Indicates whether CUDA is available and used, which can significantly affect training speed.
5. `dis_criterion` and `gen_criterion`: These variables store the discriminator loss and generator loss, respectively.
6. Adam optimizer: The optimizer algorithm and its parameters, which remain constant throughout training.

7. `max_epochs_with_no_better_performance`: The threshold for early stopping.
8. **Model Architecture**: The structure of the generators and discriminators, including their internal configurations.

### 3.6 Dependant Variables

1. **Validation loss of the generator and discriminator**: These are the performance metrics evaluated during validation. They are directly affected by changes in independent variables.
2. `gen.losses` and `dis.losses`: These variables track the average loss of the generator and discriminator during training epochs.
3. **Early stopping behavior**: Variables like `no_better_performance_epoch` and `best_loss_value` are dependent on the validation losses and determine when the training stops.
4. **Evaluation Variables**: Variables such as `ssims`, `psnrs`, and `uqims` store the calculated SSIM, PSNR, and UIQM values, respectively, for each image pair in the dataset. These are directly influenced by independent variables, including image resolution and the selected datasets.

### 3.7 Methodology

The conditional GAN model was trained on the EUVP dataset. We first preprocessed the datasets by combining three paired datasets before proceeding with the training. Since the validation data were not paired, we split the data so that 90% of the total pairs were used for training and the remaining 10% for validation. This split information was saved in a `splits.json` file.

We then trained the model using different hyperparameter values for both model architectures. In our experiments, we varied the number of epochs between 30 and 80, varied the batch size between 4 and 16, and varied the learning rate between  $1e-4$  and  $1e-2$ . After each epoch, we implemented early stopping regularization based on the generator's validation loss. The training would halt if there was no improvement in the validation loss for ten consecutive epochs. We also saved the generator and discriminator checkpoints after training each epoch and recorded the epoch with the lowest validation loss as the best generator model.

Following the training phase, we used the best generator model for making inferences on the input images. The inference phase concluded with the generation of enhanced images from the testing inputs which were of very low quality with lot of green and blue hues. The inferences images from the trained model were then validated based on the UIQM, SSIM, and PSNR metrics. PSNR, or Peak Signal-to-Noise Ratio, approximates the reconstruction quality of a generated image compared to its ground truth based on their Mean Squared Error (MSE). SSIM, or Structural Similarity, compares image patches based on contrast, luminance, and structure. UIQM, or Underwater Image Quality Measure, quantifies underwater image colorfulness, contrast, and sharpness. These metrics help us determine how effectively the model enhances low-quality images, which can then be utilized for tasks such as object detection, pose estimation, and others.

## 4 Experimental Results and Discussion

### 4.1 Training

Since the models were built using Pytorch and would train faster with GPU, the CS servers such as granger, weasley and lovegood were used to train the model.

For a batch size of 4, the time taken per epoch is around 50 mins approximately (Figure5).

```

Train: [2160/2573] Time 1.136 (1.126) Generator Loss 0.0754 (0.1056) Discriminator Loss 0.0072 (0.0279)
Train: [2180/2573] Time 1.143 (1.127) Generator Loss 0.1131 (0.1055) Discriminator Loss 0.0141 (0.0280)
Train: [2200/2573] Time 1.104 (1.126) Generator Loss 0.0730 (0.1054) Discriminator Loss 0.0654 (0.0280)
Train: [2220/2573] Time 1.365 (1.127) Generator Loss 0.1859 (0.1053) Discriminator Loss 0.0098 (0.0280)
Train: [2240/2573] Time 1.137 (1.127) Generator Loss 0.0839 (0.1052) Discriminator Loss 0.1250 (0.0279)
Train: [2260/2573] Time 1.104 (1.126) Generator Loss 0.0680 (0.1051) Discriminator Loss 0.0070 (0.0280)
Train: [2280/2573] Time 1.094 (1.127) Generator Loss 0.0885 (0.1051) Discriminator Loss 0.0066 (0.0279)
Train: [2300/2573] Time 1.166 (1.126) Generator Loss 0.0873 (0.1051) Discriminator Loss 0.0077 (0.0278)
Train: [2320/2573] Time 1.093 (1.127) Generator Loss 0.0743 (0.1050) Discriminator Loss 0.0171 (0.0278)
Train: [2340/2573] Time 1.093 (1.126) Generator Loss 0.0779 (0.1049) Discriminator Loss 0.0062 (0.0277)
Train: [2360/2573] Time 1.150 (1.127) Generator Loss 0.0844 (0.1048) Discriminator Loss 0.0061 (0.0275)
Train: [2380/2573] Time 1.114 (1.126) Generator Loss 0.0914 (0.1047) Discriminator Loss 0.0078 (0.0274)
Train: [2400/2573] Time 1.169 (1.127) Generator Loss 0.0882 (0.1046) Discriminator Loss 0.0056 (0.0274)
Train: [2420/2573] Time 1.007 (1.126) Generator Loss 0.0611 (0.1045) Discriminator Loss 0.0063 (0.0272)
Train: [2440/2573] Time 1.097 (1.127) Generator Loss 0.0991 (0.1044) Discriminator Loss 0.0059 (0.0271)
Train: [2460/2573] Time 1.214 (1.127) Generator Loss 0.0757 (0.1044) Discriminator Loss 0.0064 (0.0270)
Train: [2480/2573] Time 1.086 (1.127) Generator Loss 0.0690 (0.1044) Discriminator Loss 0.3465 (0.0270)
Train: [2500/2573] Time 1.078 (1.127) Generator Loss 0.0892 (0.1042) Discriminator Loss 0.0058 (0.0269)
Train: [2520/2573] Time 1.217 (1.127) Generator Loss 0.0973 (0.1043) Discriminator Loss 0.0080 (0.0269)
Train: [2540/2573] Time 1.101 (1.127) Generator Loss 0.0781 (0.1043) Discriminator Loss 0.0121 (0.0269)
Train: [2560/2573] Time 1.206 (1.127) Generator Loss 0.1392 (0.1043) Discriminator Loss 0.0124 (0.0268)
Valid: [ 0/286] Time 0.617 (0.617) Generator Loss 0.1822 (0.1822) Discriminator Loss 0.3577 (0.3577)
Valid: [ 20/286] Time 0.232 (0.309) Generator Loss 0.1462 (0.1803) Discriminator Loss 0.3573 (0.3605)
Valid: [ 40/286] Time 0.234 (0.272) Generator Loss 0.1501 (0.1791) Discriminator Loss 0.3545 (0.3585)
Valid: [ 60/286] Time 0.231 (0.259) Generator Loss 0.1140 (0.1720) Discriminator Loss 0.3540 (0.3586)
Valid: [ 80/286] Time 0.246 (0.255) Generator Loss 0.1491 (0.1723) Discriminator Loss 0.3656 (0.3595)
Valid: [100/286] Time 0.261 (0.255) Generator Loss 0.1414 (0.1743) Discriminator Loss 0.3517 (0.3602)
Valid: [120/286] Time 0.263 (0.255) Generator Loss 0.1591 (0.1741) Discriminator Loss 0.3794 (0.3608)
Valid: [140/286] Time 0.248 (0.255) Generator Loss 0.1873 (0.1723) Discriminator Loss 0.3472 (0.3609)
Valid: [160/286] Time 0.256 (0.254) Generator Loss 0.1494 (0.1704) Discriminator Loss 0.3558 (0.3646)
Valid: [180/286] Time 0.260 (0.254) Generator Loss 0.1240 (0.1684) Discriminator Loss 0.3481 (0.3666)
Valid: [200/286] Time 0.250 (0.254) Generator Loss 0.1402 (0.1673) Discriminator Loss 0.3613 (0.3714)
Valid: [220/286] Time 0.260 (0.254) Generator Loss 0.2412 (0.1665) Discriminator Loss 0.3518 (0.3734)
Valid: [240/286] Time 0.246 (0.254) Generator Loss 0.1243 (0.1642) Discriminator Loss 0.3651 (0.3746)
Valid: [260/286] Time 0.249 (0.254) Generator Loss 0.1023 (0.1605) Discriminator Loss 0.3435 (0.3753)
Valid: [280/286] Time 0.257 (0.254) Generator Loss 0.1122 (0.1573) Discriminator Loss 0.3423 (0.3745)
>>> Save generator to model/underwater_combo/v2/50e/1_gen.pth.tar
>>> Save discriminator to model/underwater_combo/v2/50e/1_dis.pth.tar

```

Figure 5: Time taken for training the model

## 4.2 Result Table

We trained the FunIE-GAN model on the datasets separately and on the combined dataset with 50 epochs (with the proposed modifications) and the results can be seen in Table 1 below. The hyperparameters for the best models were FunIE-GAN architecture [v2], batch size = 4, learning rate = 1e-4, epochs = 50 (early stop resulted in break at 23 epochs)

Model	Dataset	UIQM	SSIM	PSNR
Paper Claim	EUVP	2.78 +/- 0.43	0.8876 +/- 0.068	21.92 +/- 1.07
Baseline (proposal)	Underwater.dark	2.62 +/- 0.49	0.74 +/- 0.086	20.52 +/- 3.35
Best Model	Underwater.imagenet	2.992 +/- 0.440	0.795 +/- 0.066	26.185 +/- 2.380
Best Model	Underwater.scenes	3.028 +/- 0.435	0.820 +/- 0.068	28.051 +/- 2.947
Best Model	Underwater.dark	2.817 +/- 0.501	0.751 +/- 0.069	26.159 +/- 2.820
Best Model	Combined_data	3.019 +/- 0.418	0.805 +/- 0.067	26.905 +/- 2.967

Table 1: UIQM, SSIM, and PSNR values for different models based on the dataset they were trained on.

## 4.3 Discussion

Our results support our hypothesis that the changes proposed have an impact on the training of the model and subsequently produce better results. With the use of early stopping, the models finished training around 14-15 epochs [as the loss did not get better than the best loss seen so far after 10 epochs] so the resource utilization was optimized while also achieving better results. With training on the combined dataset, the model was able to learn more features and with few hyperparameter tuning, the images produced were very similar to the ground truth images as can be seen in the visualization table (Figure 6) below.

### Limitations:

Since the cs machines have limited GPU memory, for the combined dataset model, we could only train with a batch size of 4. With higher batch size, the metrics of the images produced would likely be better.

### Future work:

- Including attention based layer in the architecture might help with producing better quality images
- Increasing the batch size per epoch and further tuning of hyperparameters may help as well.



#### 4.4 Visualization













Input Image	Generated Image	Ground Truth
		
		
		
		

Figure 6: Table of comparison between the input, generated and ground truth image



## References

- [1] Md Jahidul Islam, Youya Xia, and Junaed Sattar. Fast underwater image enhancement for improved visual perception. *IEEE Robotics and Automation Letters*, 5(2):3227–3234, 2020.
- [2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. pages 234–241, 2015.
- [3] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image to image translation with conditional adversarial networks. pages 1125–1134, 2017.
- [4] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. 30(1):3, 2013.
- [5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. pages 448–456, 2015.