

Edge-Preserving Generative Adversarial Networks for Enhanced Underwater Image Restoration

Algorithms of each module of our proposed work is as follows:

Algorithm 1 Training Procedure

1. **Start** with paired training samples.
2. Initialize generator , multi-scale discriminator , and set hyperparameters .
3. Set optimizer: and and learning rate and parameters .
4. Set noise .
5. Initialize .

Training Loop

6. For epoch = 1 to N do
7. Set generator and discriminator to training mode. Initialize epoch generator loss .
8. For each minibatch do

Discriminator Update (every 25 batches)

9. If (batch index mod 25) = 0 then,
10. Freeze generator: compute fake image without gradient: (no gradient).
11. Add Gaussian noise to real and fake images: .
12. Compute discriminator outputs (multi-scale): . Each branch returns a probability in .
13. Binary cross-entropy losses:
14. Total discriminator loss: .
15. Update discriminator parameters via gradient descent: .
16. End if.

Generator Update (every batch)

17. Compute generated image with gradient: .
18. Content loss:
19. Adversarial loss (generator tries to fool):
20. Perceptual loss (simplified MSE): .
21. Edge loss (edge consistency via Canny): , where is the same Canny operator used for input.
22. Weighted total loss: .
23. Update generator parameters: .
24. End for (training minibatch loop).

Validation Phase

25. Set generator and discriminator to evaluation mode.
26. Initialize empty lists for PSNR and SSIM values.
27. For each validation pair do:
 - i. Compute enhanced output: .
 - ii. Compute .
 - iii. Compute SSIM per standard formula:
 - iv. Append PSNR and SSIM to their respective lists.
28. End for loop.
29. Compute mean PSNR and mean SSIM: .

Model Checkpointing

30. If validation PSNR > best PSNR then, save as best weights and set best PSNR to current PSNR.
31. End if
32. End for (epoch loop).

33. End.

Algorithm 2: Edge Map Extraction

- 1. Start** with RGB image , or
2. Detach tensor from autograd graph to avoid gradient tracking:
3. Determine if input is batched:
 - a. If has 4 dimensions batched input
 - b. Else single image.

Case 1: Batched Input (B, 3, H, W)

4. Initialize empty list for edge maps: .
- 5. For each image in batch :**
6. Convert PyTorch tensor to NumPy array and reorder channels:
7. Undo normalization from :
8. Convert RGB to grayscale using simple averaging:
9. Apply Canny edge detector with Gaussian smoothing :
- 10. Convert edge map to PyTorch tensor on original device:**
11. Append to list .
- 12. End for.**
13. Stack edge maps into batch form and add channel dimension:
14. Return .

Case 2: Single Image (3, H, W)

15. Convert tensor to NumPy array and reorder channels:
16. Undo normalization from to :
17. Convert to grayscale:

18. Apply Canny edge detector:
19. Convert to PyTorch tensor and add channel dimension:
20. Return .
21. **End**

Algorithm 3: Deformable Convolution Block

1. **Start** with input feature map .
2. Compute learned sampling offset using a convolution: , where represents offsets for each of the kernel positions.
3. Apply deformable convolution using offsets:
, which samples spatially shifted positions where .
4. Apply Group Normalization: .
5. Apply LeakyReLU activation .
6. Return output feature map .
7. **End.**

Algorithm 4: Residual Block

1. **Start** with input feature map .
2. Apply identity skip connection: .
3. Apply first convolution with kernel and padding = 1: .
4. Apply Group Normalization: .
5. Apply LeakyReLU activation with slope 0.2: .
6. Apply the second convolution with kernel and padding = 1: .
7. Apply Group Normalization: .
8. Add skip connection (residual addition): .

9. Return output feature map .

10. End.

Algorithm 5: Attention Module

1. **Start** with input feature map , where is the number of the channels.

2. Apply convolution to compute attention logits:.

3. Apply element-wise sigmoid activation to form attention weights: , where .

4. Apply channel-wise and spatial-wise modulation by multiplying input and attention map element-wise: . Where denotes element-wise multiplication.

5. Return attention-refined feature map .

6. End.

Algorithm 6: Multi-Scale Discriminator Forward Pass

1. **Start** with input image .

2. Apply first convolution with stride 2:

3. Apply second convolution:

4. Apply final convolution to reduce to 1 channel:

5. Output of branch 1 is .

6. Downsample input using average pooling:.

7. Apply first convolution on downsampled input:

8. Apply second convolution:

9. Apply final convolution to reduce to 1 channel:.

10. Output of branch 2 is .

11. Return the list of discriminator outputs at two scales: .

12. End.

Algorithm 7: Enhanced Generator Architecture

1. **Start** with input tensor x formed by concatenating RGB distorted image with its edge map.

Encoder Path

2. Apply first convolution (downsample to 128×128):
3. Apply Deformable Block:
4. Apply third encoder convolution (downsample to 64×64):

Bottleneck

5. Initialize bottleneck representation: z .
6. Pass through 6 Residual Blocks:
7. Apply Attention Module to refine spatial focus:

Decoder Path

8. Apply first transposed convolution (upsample to 128×128):
9. Apply Deformable Block:

Output Layer

10. Apply σ activation to constrain output to the range $[0, 1]$.
11. Return enhanced output image $\hat{y}$.
12. **End.**