Multi-view underwater image enhancement method via embedded fusion mechanism

Group 47:

Sander Britton	5110947	s.britton@student.tudelft.nl
Kasper Vaessen	4994760	k.e.vaessen@student.tudelft.nl
Niels Mateijsen	4803760	n.mateijsen@student.tudelft.nl
Wouter Büthker	5061717	w.buthker@student.tudelft.nl

Introduction

Underwater imaging presents unique challenges due to the complex interactions of light with water, resulting in degraded image quality characterized by low contrast, color distortion, and reduced visibility. In recent years, researchers have proposed various methods to address these challenges and enhance the quality of underwater images. One such method is the "Multi-view Underwater Image Enhancement Method via Embedded Fusion Mechanism," introduced in a paper by Jingchun Zhou, Jiaming Sun, Weishi Zhang and Zifan Lin (2021) [5].

Reproducing a deep learning paper holds great value in ensuring the credibility and reliability of scientific research. By replicating the findings and methodologies of existing papers, researchers can validate the robustness and generalizability of the proposed approaches, uncover potential limitations or biases, and facilitate the adoption of best practices within the community.

In this blog post we'll discuss the process of reproducing the results of this paper. More specifically Table 1 and Fig 7 to Fig 15 [5] (see Fig 10 to Fig 18 in the appendix) is aimed to be reproduced. We'll explore the key components and discuss the challenges encountered during the reproduction process, insights gained from experimenting with different parameters.

Dataset

For training and testing of the model, we used the Underwater Image Enhancement Benchmark (UIEB) dataset which has been made specifically for the purpose of evaluating different image enhancement models [1]. It contains 950 real-world images of underwater environments. 890 of these images have reference images which are already enhanced versions of the raw images. The remaining 60 images do not have such reference images and are seen as challenging data.

Fig. 1. Sampling images from UIEB. Top row: raw underwater images taken in diverse underwater scenes; Bottom row: the corresponding reference results.

Methodology

This section outlines the methodology employed in our reproducibility project. Due to the unavailability of code, we implemented the entire model from scratch using the PyTorch library. Our approach focused on closely adhering to the architecture, training regimen, and evaluation criteria specified in the original paper. Not all details are fully enclosed in the paper and thus some assumptions have been made, which are clearly stated in the respective section.

Pre-processing

Instead of using a single image as an input, the paper suggests using two image enhancement algorithms to transform the raw image:

1. White-Balance
2. CLAHE

These transformed raw images are then saved and to be used as input for the final model.

White-Balance

The first algorithm is the White-Balance (WB) algorithm as proposed in [2]. This operation is used to adjust the color of each channel of an image by removing any color cast caused by the lighting conditions. This effect is present a lot in underwater imaging as it suffers a lot due to underwater medium scattering and absorption.

WB works on the principle that, in a well-balanced image, the average intensity of each channel should be alike. However, due to these lighting conditions, the channels might have different average intensities. This leads to a color cast on the image. The algorithms can be summarized by the following formula:

$$\mu_i = 0.5 + \lambda \mu_{sce}$$

Where i is the channel index (typically ranging from 1 to 3 for RGB), μ_i is the new white-balanced pixel value per scene, μ_{sce} is the overall mean of all the channels in the scene and λ is a parameter to control the extent of the white balancing effect.

Since both the network paper and the original WB paper got the best results for a λ value of 0.2, this was also used in our reproduction. An example of the enhancement can be seen in Fig 2.

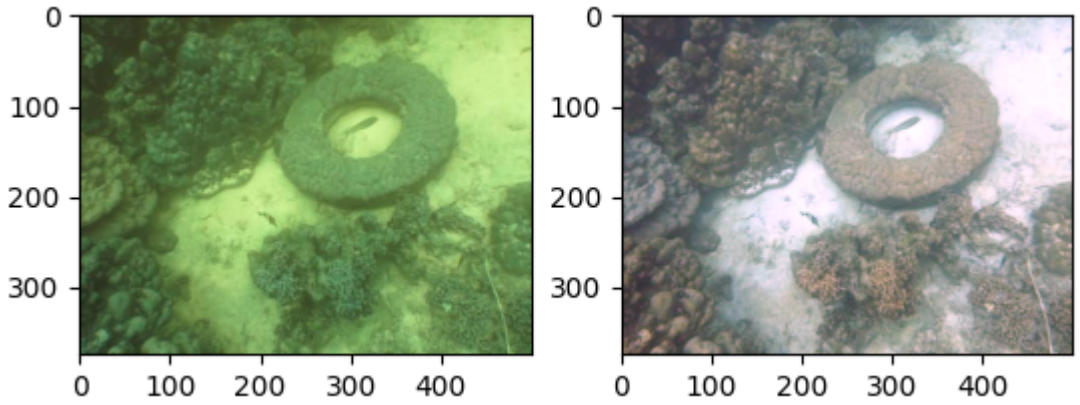

Fig. 2: Example raw image (left) and it white-balanced result (right) with λ=0.2.

CLAHE

Contrast-Limited Adaptive Histogram Equalization (CLAHE) is an image enhancement algorithm that is designed to enhance the contrast of an image [3]. The enhancement works on the principle that a good image will have pixel values stretched across the entire domain of values. For example, brighter images will have high values for most pixels. Histogram Equalization will stretch these values over the entire domain. CLAHE divides the image in small blocks ("tiles") and performs Histogram Equalization on these tiles. If a histogram bin is larger than a specified contrast limit, those pixels are clipped and distributed uniformly to other bins before applying the equalization.

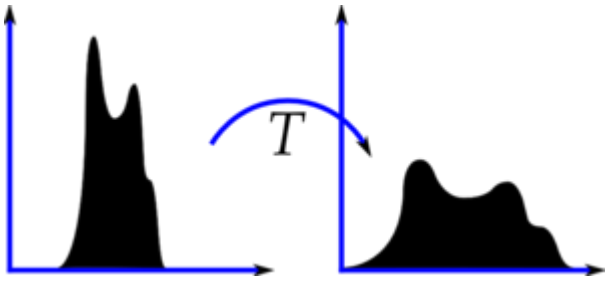

Fig 3: A Histogram Equalization transformation [4].

To use this enhancement in our model, we can use the OpenCV library which imports useful image transformation operations such as CLAHE. Since the main paper does not specify the exact parameters used, we used the default parameters by OpenCV: a tile-size of 8x8 and a clip limit of 40.

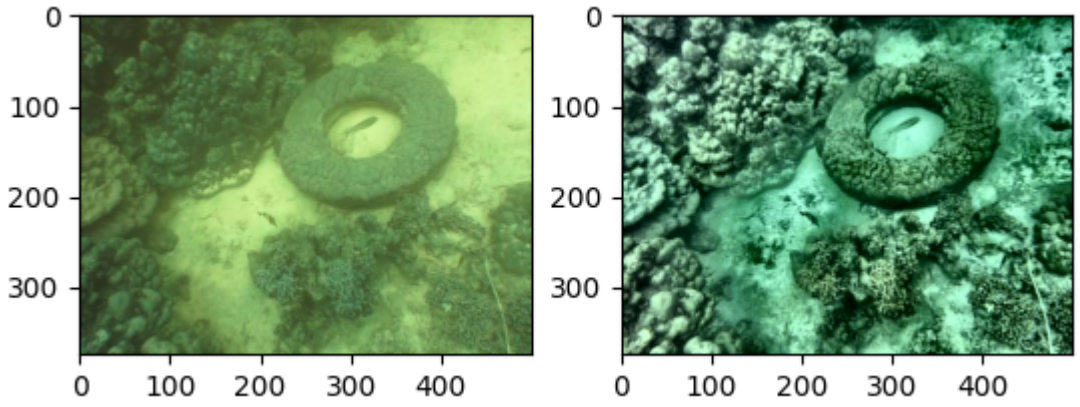

Fig 4: Example raw image (left) and its CLAHE result (right) with tile size 8x8 and clip limit 40.

Modules

The architecture of the Multi-Feature Underwater Image Enhancement Method via Embedded Fusion Mechanism (MFEF) is given in Fig 5. As mentioned in the previous section the input images are processed with the White Balance and CLAHE. These, together with the original image, are the inputs of the network. The main components are the Residual-Enhancement Module (REM), Multi-Feature Fusion (MFF), Downsampling and Dense Connections (DSDC) and Pixel-weighted Channel Attention Module (PCAM).

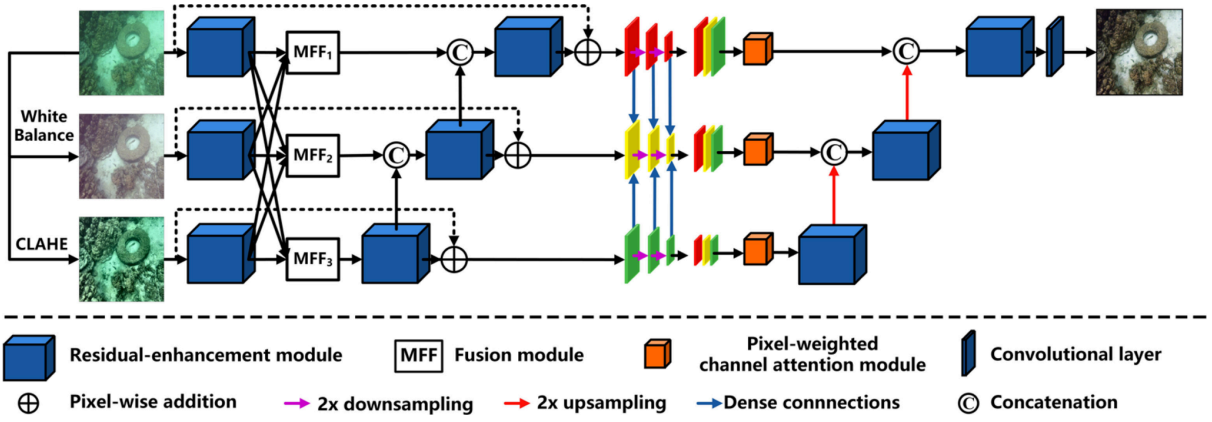

Fig 5: Overview of the MFEF architecture [5].

Residual Enhancement Module (REM)

The REM is used three times in the network, all for different purposes. The first time is straight after the pre-processing and functions as encoding of features. This enables the network to learn meaningful representations of the input data, leading to improved performance. The second time is after the MFF modules, which make all features interact, and its goal is to extract deeper features by once again abstracting the features. The encoded features must of course be reconstructed to images, which is what the REM is used for the third time.

The architecture of the module is given in Fig 6. The convolutional layer can be easily implemented using the “Conv2d” PyTorch module. The size of the convolutional kernel is set

to 3 × 3, the stride is set to 1 and padding to ‘same’. PyTorch’s GELU can be used for nonlinearity.

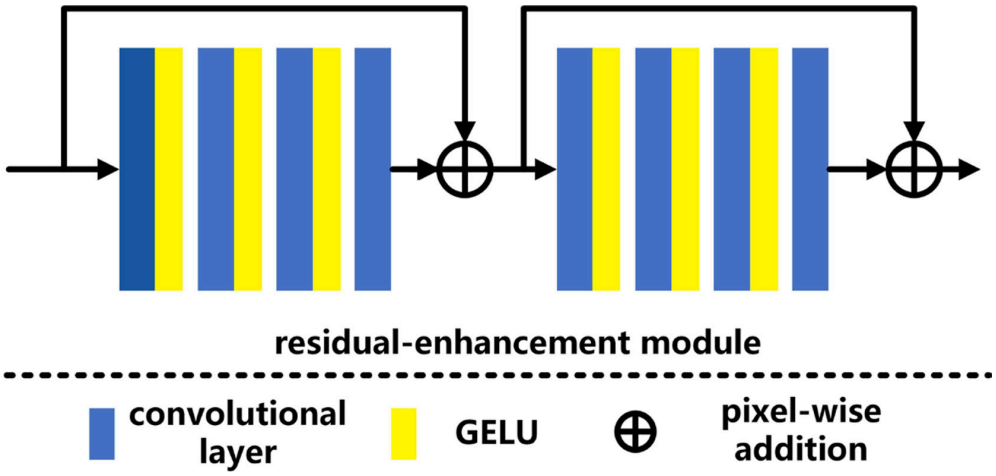

Fig 6: The diagram of the Residual Enhancement Module (REM) [5].

Multi-Feature Fusion (MFF)

The MFF makes all the features from different flows interact, as mentioned in the previous section. The architecture is given in Fig 7. The module aims to fuse features from each path with differing perceptual fields, which is why the kernel size of the convolutional layers differ. The GELU is then used to implement the interactions between the multi-form features. The original features are then added and all streams are concatenated using a channel shuffle strategy to fully interact with multiple forms of features [5].

The “ChannelShuffle” PyTorch module was used to implement the channel shuffle block. This module requires a number of groups to divide channels in. This was not specified in the paper, but was assumed to be three, so that the R, G and B channels are shuffled. However, due to a problem in the implementation of the backwards method of this function, it was not used during training. The last convolutional layer reduces the amount of channels from nine, caused by the concatenation, to three. All convolutional layers have stride of 1 and padding of ‘same’.

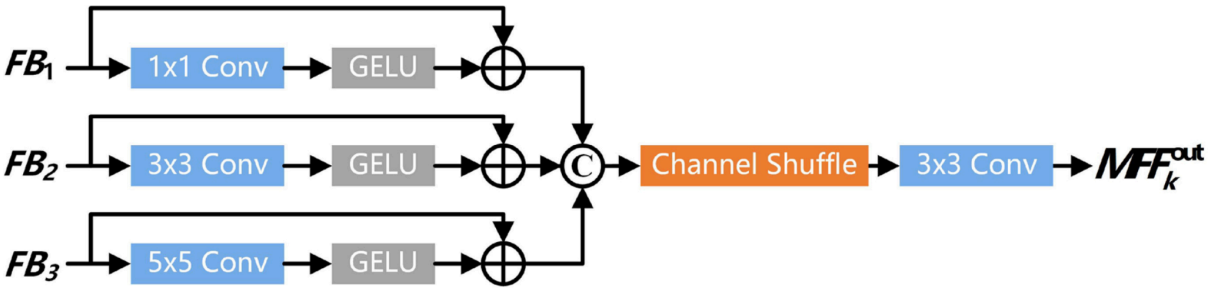

Fig 7: The diagram of the Multi-Feature Fusion (MFF) module [5].

Downsampling and Dense Connections (DSDC)

This module was not described as a module in the paper but was implemented as one. The downsampling is performed to obtain shallow features. Features of the same size are concatenated in the “Dense Connection”.

The architecture is given in Fig 8. The downsampling is implemented by simply using slicing to select every second pixel. It is important to use padding when the width or height is not even.

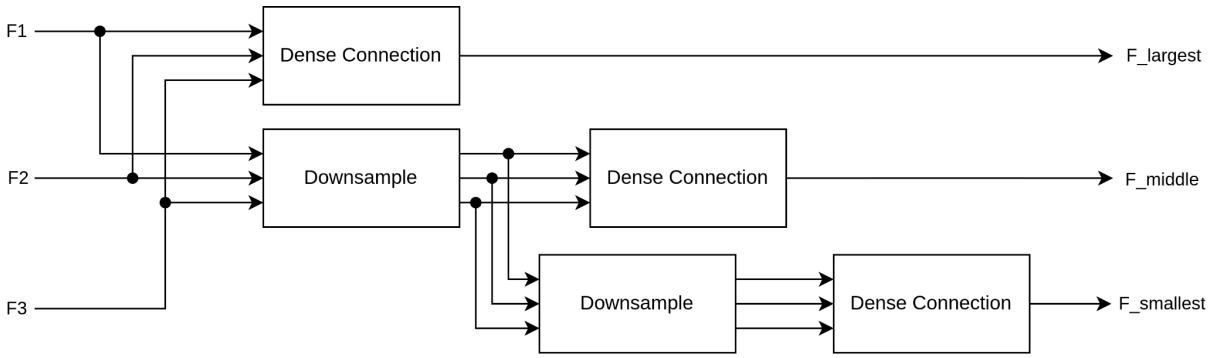

Fig 8: Diagram of the Downsampling and Dense Connections (DSDC).

Pixel-weighted Channel Attention Module (PCAM)

The goal of the PCAM is to re-calibrate the critical features of the images. The architecture is given in Fig 9. The 3x3 convolution and GELU is used to enrich the feature content context of information and get the feature localization. Considering the spatial information of the features, 1 × 1 convolution and 3 × 3 parallel convolution is employed to obtain the spatial dependence of the features, learning with the sigmoid activation function to get the spatial weights of the features, and multiplying element by element to get the recalibrated features [5]. The global average pooling and fully connected layers are used to give greater weights to the critical features [5].

All convolutional layers have stride set to 1. The 3x3 layers have padding of 1 and the 1x1 layer has padding of 0. GAP is implemented using the “AdaptiveAvgPool2d” PyTorch module with output size (1,1) and the fully connected layers with the “Linear” PyTorch module.

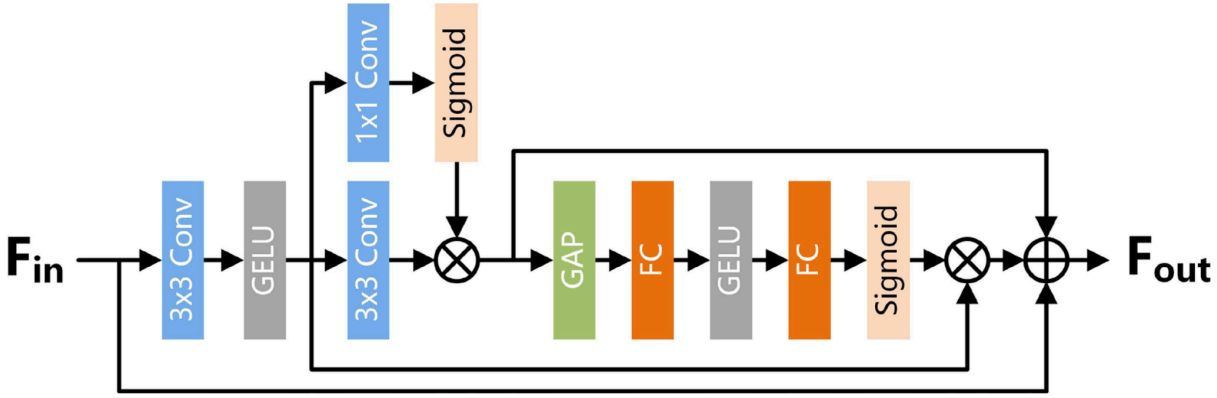

Fig 9: Diagram of the Pixel-weighed Channel Attention Module (PCAM) [5].

Final Network

The final network as described in Fig 5 is put together using the modules described in the previous sections. The upsampling is implemented using the “Upsample” module from PyTorch, with a scale factor of 2. The type of upsampling is not defined in the paper, but it is assumed to be the ‘nearest’ mode.

The images that are fed into the network have three channels (R, G and B), but due to the concatenations happening in several places in the network, the dimensions are not the same everywhere. This is no issue, unless two different types of dimensions are attempted to be added together. This is the case for the dotted lines in Fig 5. To fix this a channel reduction has been added by the means of two 3x3 convolutional layers, one for the original image path and one for the white balance path. The original image path needs to be reduced from 9 to 3 and the white balance path from 6 to 3. A similar channel reduction is done for the final image, but then done using the final 3x3 convolutional layer before outputting the image. This reduces the channels from 27 to 3. All these layers have a stride of 1 and 'same' padding.

Training

As described in the paper, the training is performed using the Adaptive Moment Estimation Weight Decay Optimizer (AdamW) with a learning rate of 1e-4. For this the corresponding PyTorch module is used.

According to the paper, a batch size of 16 is used for training. Using batches requires the images to have equal resolution, which does not hold for the input images. J. Zhou et. al. did not describe how they addressed this problem. One solution would be to crop or interpolate the images to have the same resolution, but since this changes the data, this could hurt the performance of the model. Therefore, we decided to keep a batch size of 1.

For the loss function only the L1 loss is used, instead of the proposed combination of the L1 loss and the perceptual loss function given by:

$$L = L_1 + \eta L_{per}$$

Where L_{per} is given by:

$$L_{per} = \sum_{x=1}^{H} \sum_{y=1}^{W} |\phi_j(J)(x,y) - \phi_j(R)(x,y)|$$

This is because L_{per} requires the VGG-16 network indicated above by ϕ. To reduce the complexity of our model, this was removed.

The data is split into a training and test set of sizes 800 and 90 respectively. The model was trained on a RTX 4070 for 33 epochs in approximately 6 hours. Further improvements might be possible with further training. When training further, more video memory is recommended.

Results

Quantitative comparison

Table 1. PSNR, SSIM, PCQI, UCIQE and UIQM assessments of the paper and our reproduced model. (higher is better)

	PSNR(dB)	**SSIM**	**PCQI**	**UCIQE**	**UIQM**
Ucolor	21.188	0.873	0.741	0.556	1.187
Paper	23.352	0.910	0.881	0.602	1.427
Reproduced	20.692	0.843	0.707	0.561	1.266

As is visible from Table 1, the results of our reproduced network are not as high as the ones presented in the paper. Causes of this might be a different setup or hyperparameters. Results might also be improved further by additional training of the network. Nevertheless, the results are still impressive and comparable to other state of the art techniques such as Ucolor.

Visual comparisons

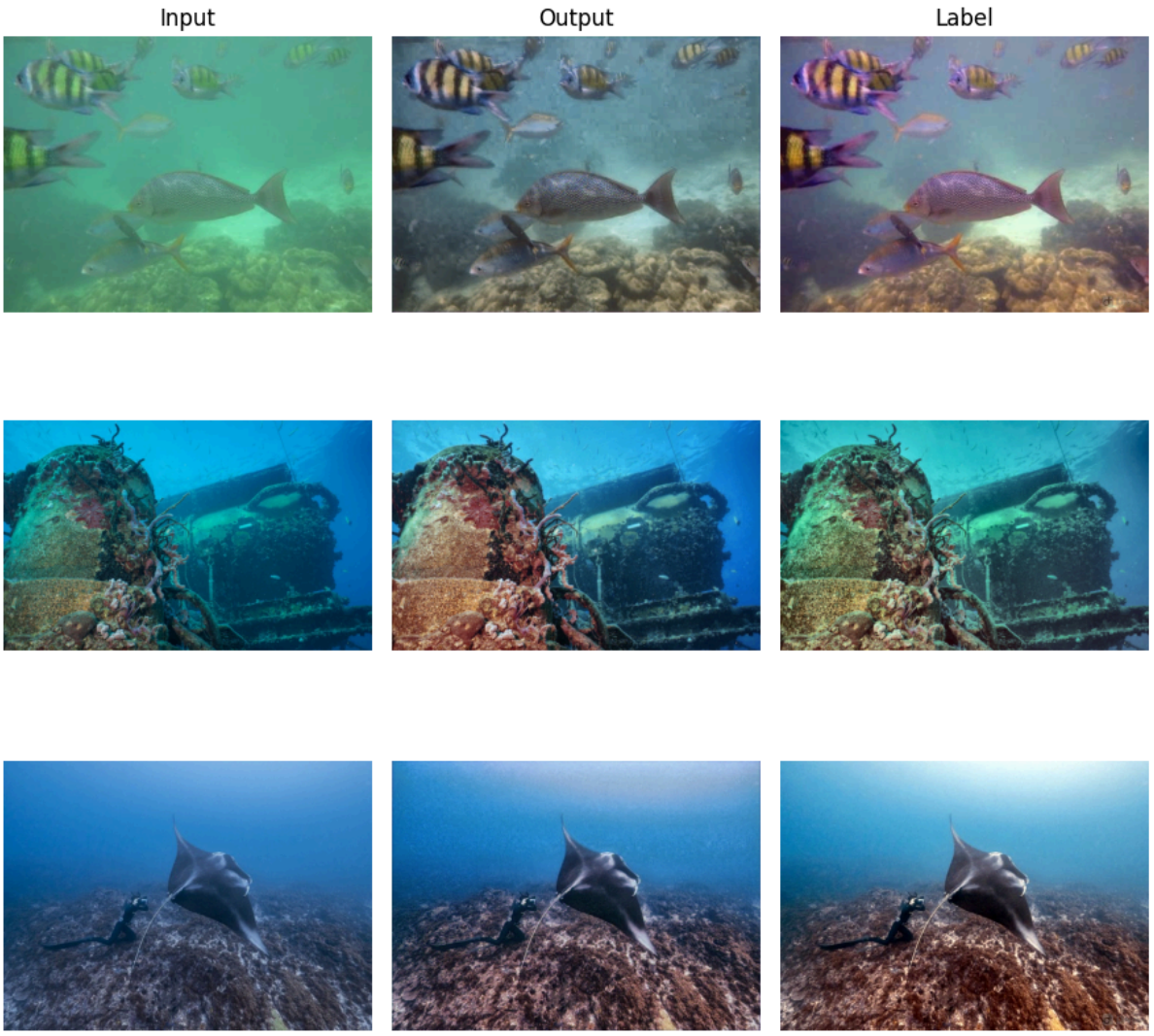

Fig 10: results from our reproduced network

As is visible in Fig 10, resulting images from our reproduced network do look very good. Unfortunately we were not able to compare our output images to the ones presented in the paper as not all images in the paper seem to be available in our dataset.

Conclusion

Based on the quantitative results, it can be determined our results still lag behind the results presented in the paper. This is likely due to the different setup and parameters used caused by the lack of details in the paper. Results might be improved by doing hyperparameter optimization. Further training of the network (requiring additional resources) also has the possibility to improve the results. However, even with these limitations, the resulting images appear to be very close to the label images, sometimes even arguably better. Considering these factors, we can conclude the authors have presented a novel and improved way of image enhancement for underwater images.

Recommendation

A recommendation for further study is to train with a batch size of 16. Currently, training was done with batch size of one, to prevent issues with differing resolutions. Another recommendation is to use VGG-16 in the loss function. Both of these suggestions would result in a closer representation of the paper.

Quite some details were left out in the paper, such as how the valuewise addition of tensors with different amounts of channels are handled or what the parameters of the CLAHE preprocessing steps are. Some research could be conducted in the best way to implement this. One can also contact the author of the original paper to ask for clarification.
Lastly, the model can be trained for more epochs to possibly improve results.

Contributions

All four team members contributed to the full re-implementation. Since there was no pre-existing code and the paper left out some important details, the re-implementation took a lot of time and no further study was attempted. Pre-processing (CLAHE and WB) was done by Kasper. REM was done by Kasper. MFF was done by Niels. DSDC was done by Sander. PCAM was done by Niels. Final network was put together by Sander and Wouter. The training setup was done by Kasper and the further training and results analysis was done by Wouter. The blog was written by all team members.

Appendix

Fig 10. Visual comparison of yellow-toned underwater images (figure 7 from [5]).

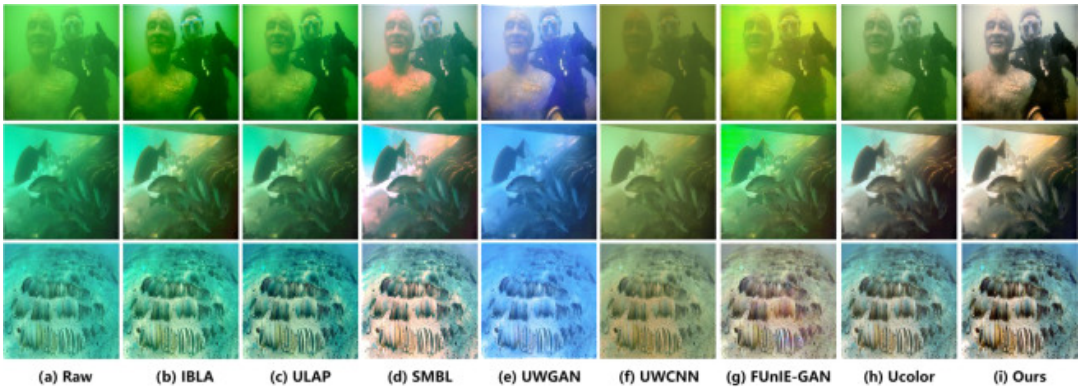

Fig 11. Visual comparison of green-toned underwater images (figure 8 from [5]).

Fig 12. Visual comparison of blue-toned underwater images (figure 9 from [5]).

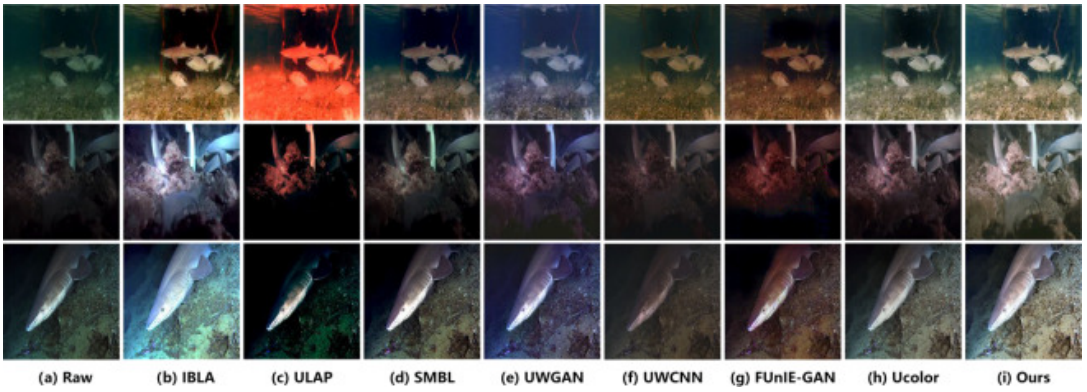

Fig 13. Visual comparison of low-light underwater images (figure 10 from [5]).

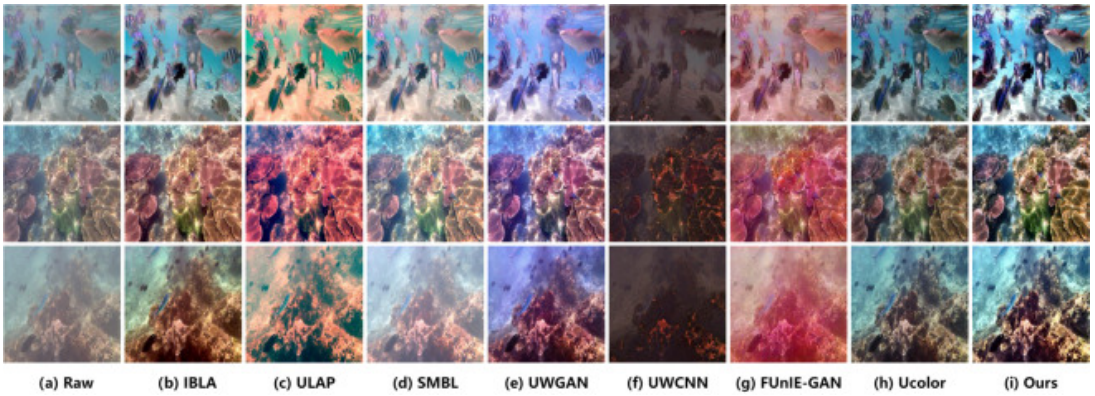

Fig 14. Visual comparison of images in shallow water areas (figure 11 from [5]).

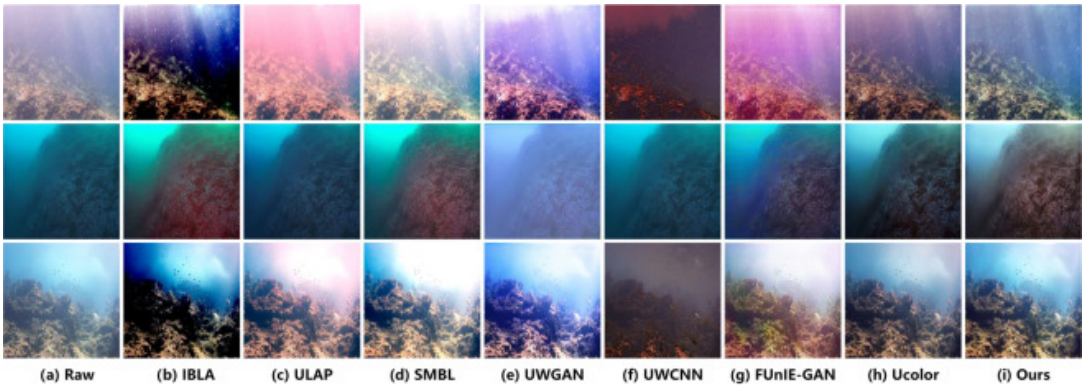

Fig 15. Visual comparison of images in deep water areas (figure 12 from [5]).

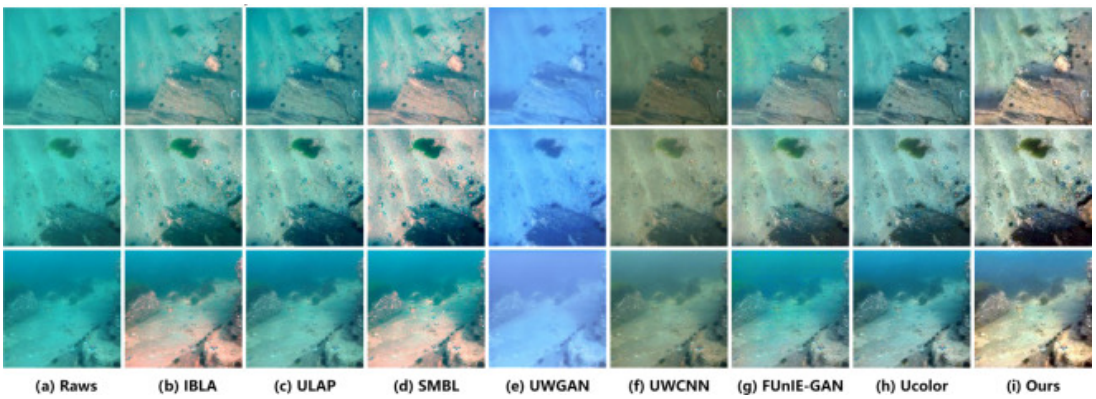

Fig 16. Visual comparison of blue-toned underwater images (figure 13 from [5]).

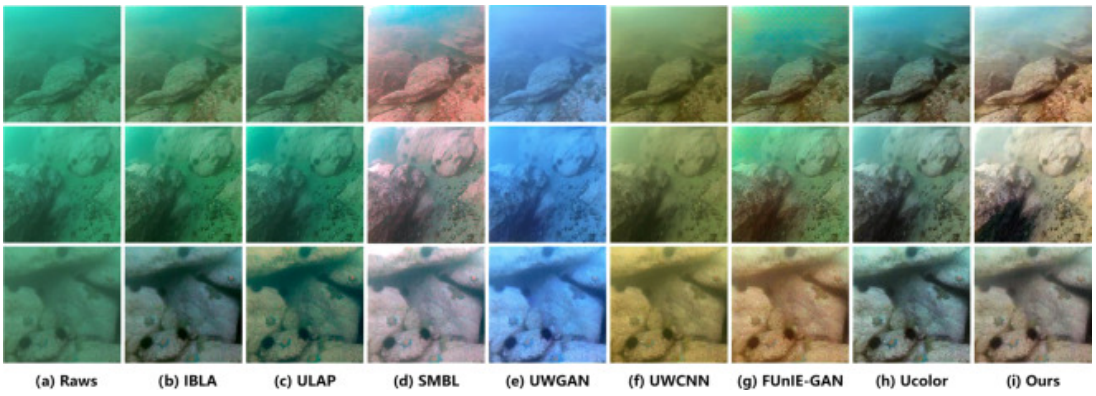

Fig 17. Visual comparison of blue-green-toned underwater images (figure 14 from [5]).

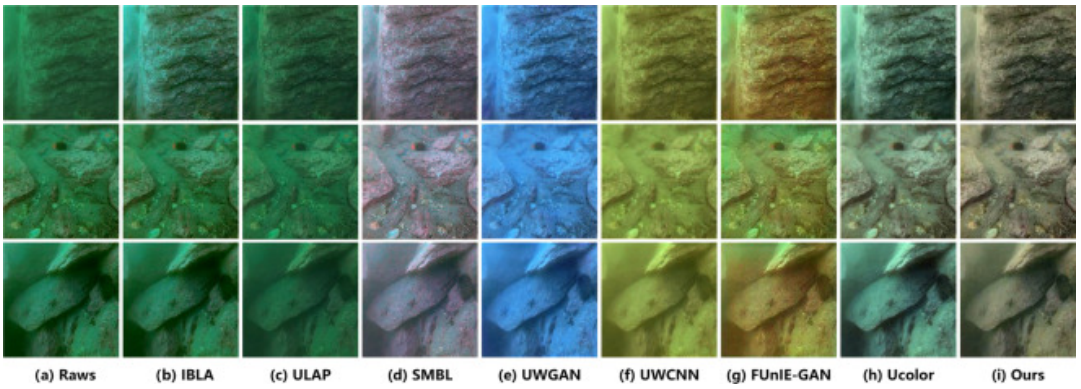

Fig 18. Visual comparison of green-toned underwater images (figure 15 from [5]).

References

[1] C. Li, C. Guo, W. Ren, R. Cong, J. Hou, S. Kwong, D. Tao, “An Underwater Image Enhancement Benchmark Dataset and Beyond,” IEEE Trans. Image Process., vol. 29, pp.4376-4389, 2019.

[2] C. Ancuti, C. O. Ancuti, T. Haber and P. Bekaert, "Enhancing underwater images and videos by fusion," *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Providence, RI, USA, 2012, pp. 81-88, doi: 10.1109/CVPR.2012.6247661.

[3] Zuiderveld, K., 1994. Contrast limited adaptive histogram equalization. Graphics Gems 474–485.

[4] *OpenCV: Histograms - 2: Histogram Equalization*. (n.d.). https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html

[5] Jingchun Zhou, Jiaming Sun, Weishi Zhang, Zifan Lin, “Multi-view underwater image enhancement method via embedded fusion mechanism,” Engineering Applications of Artificial Intelligence, Volume 121, 2023, https://www.sciencedirect.com/science/article/pii/S0952197623001306