Reproduction Paper Underwater Image Enhancement

Dave Hogendoorn - D.Hogendoorn@tudelft.nl - 4880137
Simon Molenkamp - S.H.Molenkamp@tudelft.nl - 5163234
Thomas Cramer - T.C.W.M.Cramer@tudelft.nl - 4861779
Maurits Heemskerk - M.J.Heemskerk@tudelft.nl - 5055725

April 14, 2024

1. Introduction

For this blog post, we have tried to reproduce the paper: 'Multi-view underwater image enhancement method via embedded fusion mechanism' (Zhou et al. [2023]). As the paper does not provide any code, we have successfully reproduced the code to achieve similar results as the paper states. We have used Pytorch (Paszke et al. [2019]) as main library for the implementation. During this process, we have encountered several challenges for a successful reproduction of the code. This blog aims to provide the reader with an insight into our struggles and solutions we found to those challenges. In doing so, we hope the reader has the means to reproduce this paper with less effort than we had to exert. Sections 2-4 are organised to provide clarity on the following questions:

1. How does this part work?
2. What was unclear or vague from the paper?
3. What did we do to clarify/solve this problem?

Afterwards, we discuss the loss function and with it how we trained our model, of which the latter is scarcely talked about in the paper. Lastly, we will present our results and discuss how successful we were in our reproduction of this paper.

2. Image pre-processing

In the paper, the authors utilise two crucial pre-processing techniques, White Balance (WB) and Contrast Limited Adaptive Histogram Equalisation (CLAHE), to enhance the quality of input images for their underwater object detection model.

2.1 White Balance

White Balance (WB) correction is employed to rectify the object tones and counteract the colour bias induced by the underwater environment's absorption of various light wavelengths. By restoring the original colour balance of the image, WB minimises the impact of colour variations across the scene.

Reproducing the White Balance (WB) algorithm presented several challenges. Firstly, the authors incorporated variables and formulas from multiple papers in their formulation, leading to inconsistencies and confusion in variable definitions. This complex referencing made it difficult and time-consuming to get a clear understanding of the algorithm's components. Secondly, the authors directly copied text from another paper, which did not pose a problem for reproducing the paper but it did raise concerns regarding academic integrity. Additionally, there was no way to verify our algorithm's output against the input used in the paper, leaving us with uncertainties early on in the reproduction process.

Given the paper's lack of clarity on white balance and referencing inconsistencies, our implementation process began with research on Gray-scale white balancing from online sources to understand the basics and identify where to integrate Equation 1 into our code. Although we could not directly compare our results with the paper's, insights from referenced literature helped us understand common issues, especially in underwater imaging where poor illumination estimation causes reddish colour shifts. Using these insights, we focused on testing Equation 1's parameter λ mentioned in the paper. This parameter was said to minimise colour shifts. When we implemented Equation 1 with a λ value of 0.2, as suggested, we noticed fewer colour shifts in the resulting images. This confirmed the effectiveness of λ in reducing colour deviations and assured us that we were implementing the correct approach.

2.2 CLAHE

The CLAHE algorithm is utilised to address challenges posed by the turbid underwater environment, characterised by light scattering due to numerous tiny particles. This phenomenon often leads to low contrast, blurring, and other image quality issues. CLAHE works by enhancing contrast and clarity in the image. It achieves this by redistributing pixel values across the histogram to ensure consistent histogram distribution while limiting noise amplification and maintaining smooth transitions between edges.

Reimplementing the Contrast Limited Adaptive Histogram Equalisation (CLAHE) algorithm proved to be challenging, as the paper referenced another paper that provided a step-by-step implementation but lacked specific parameters. In light of this, we opted to use the standard parameters from the OpenCV library, which already included a CLAHE algorithm, significantly easing the implementation process.

However, validating the implementation posed a challenge. The referenced paper did not specify which images were taken from the dataset, requiring a manual comparison of results. This would entail adjusting parameters in the OpenCV function iteratively until satisfactory results were achieved, a time-consuming process. Considering the time constraints and the satisfactory results obtained with the default parameters, we decided against manual validation.

3. Modules

3.1 Residual-enhancement module

The REM module performs the initial feature extraction on the original image, the white balance adjusted image, and the CLAHE-enhanced image. The original module displayed in the paper consists of 8 convolutional layers and 6 GELU activation functions. In the paper, it is stated that the size of the convolutional kernel is set to 3x3, and the stride is set to 1 which we also used. Since the paper does not state all the hyperparameters of the convolutional layers, some assumptions were made. To maintain the width and height of the images we added a padding of 1 and we assumed that the number of channels of the output was the same as at the input. This module in itself was functional. However, when implementing this module in the final network we encountered a problem. The network presented in the paper contains concatenations which increase the number of channels of an image. This in and of itself is not a problem. However, in two places, after concatenating and passing the concatenated image through a REM module the paper suggests doing a pixel-wise addition with the original image, which has only 3 channels, this is not possible as the processed image has more than 3 channels. This problem could be solved in multiple ways. You could for example remove the pixel-wise addition, however, in doing so the risk of exploding gradients increases. Therefore, we decided on adding an extra convolutional layer to these two REM modules that have an output of 3 channels with the kernel size set to 1, see fig 1. This extra convolutional layer was only added to the REM modules where pixel-wise addition wouldn't have been possible otherwise.

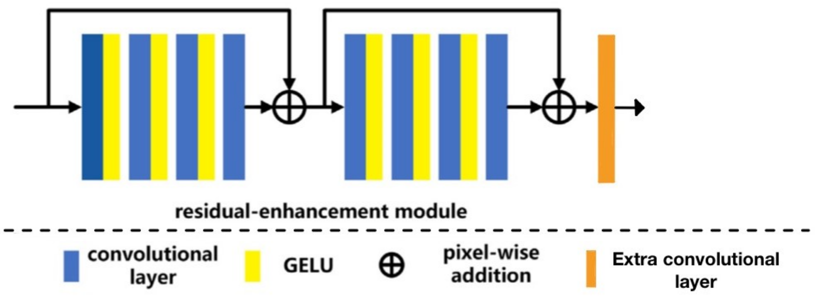

Figure 1: REM Module with an extra convolutional module

3.2 Multi-feature fusion module

The MFF module is designed to fuse information from features obtained at the first stage of processing, aiming to incorporate diversity in feature content. Each path in the MFF module has passed through the Residual-enhancement Module, generating features. These features are then processed through 1x1, 3x3, and 5x5 convolution operations, respectively, to capture different perceptual fields. The resulting features are then concatenated into a comprehensive feature. This utilises the GELU activation function to facilitate interactions between the multi-form features. Residual connectivity mechanisms are introduced at each layer to efficiently extract features and prevent gradient vanishing, this is the pixel-wise addition in the MFF schematic. Finally, a channel shuffle strategy is employed to promote cross-channel information exchange and enhance feature representation, followed by a 3x3 convolution to obtain the same number of channels as the input features.

The schematic of this module in the paper is easily reproduced and has a 3x3-channel input and a 3-channel output. The paper lacks explicit details of the channel-shuffle. In traditional CNNs, convolutional layers preserve spatial information while operating independently on each channel. However, they may not effectively utilise the information across different channels. The channel shuffle operation aims to address this limitation by facilitating communication between channels.

Here's how the channel shuffle operation works:

- Grouping Channels: First, the input feature maps are divided into multiple groups along the channel dimension. Each group contains a subset of the input channels.
- Shuffling Channels within Groups: Within each group, the channels are shuffled or rearranged. This means that channels from different parts of the input feature maps are mixed within each group.
- Concatenating Groups: Finally, the shuffled groups are concatenated along the channel dimension to form the output feature maps.

In our code, we use the channel-shuffle function from torch to address this problem. However, the paper does not provide any information about the number of groups used in the channel shuffle.

Therefore, a value of 1 was chosen.

3.3 Pixel-weighted channel attention module

The Pixel-weighted Channel Attention Module (PCAM) is introduced to re-calibrate critical features in underwater images with varying degrees of degradation. It consists of several key steps. First, the input features undergo 3x3 convolution and GELU activation functions to enrich their content and obtain feature localisation. Spatial dependence of the features is then considered by employing 1x1 convolution and parallel 3x3 convolutions. These convolutions are followed by a sigmoid activation function to obtain spatial weights for the features. The re-calibrated features are obtained by concatenation of the spatial weights with the feature localisation. Global average pooling is employed to obtain the attention of each channel, and channel descriptors are computed accordingly. Sets of different weights are assigned to each feature map, and these weights are combined using convolution and fully connected layers. The final attention map is obtained by concatenating the final weight with the obtained feature localisation. The final reconstructed feature output is obtained by adding the attention map to the input features and the re-calibrated features.

The paper's schematic is sufficient in reproducing the architecture of this module, which we successfully do in our code. The paper lacks detailed explanations regarding the intuition behind certain design choices.

Furthermore, the paper's schematic does not indicate any change in dimensions after layers, which makes reproduction much more difficult than it could be.

4. Down-sampling and up-sampling

Down-sampling helps in extracting higher-level features by progressively reducing the spatial dimensions of the feature maps. Additionally, it introduces a degree of translation invariance, making the network more robust to small shifts in the input data.

The paper states this purpose but shows a vague schematic of how this is introduced in the network. It seems the paper utilises 2x down-sampling across 3 channels and concatenates the features of the same size. We interpreted this as down-sampling to half the size of the original feature map. The paper's schematic's flow is deceptive and shows something different. Basically, from the 3 branches we down-sample the features twice to create 3 different sized features and we are left with 9 channels. We then concatenate the features of the same size and continue with 3 branches again putting the largest features in the top branch going down. The down-sampling is achieved by a convolutional layer with kernel size 2 and stride 2.

Up-sampling is the inverse operation of down-sampling. It is used to increase the spatial dimensions of feature maps while retaining or interpolating the information present in the original feature map. 2x up-sampling, as stated in the paper, is interpreted by us as increasing the size of the feature map to twice as large. The paper, again, lacks any explanation about up-sampling. From their schematic, it is somewhat clear however that we up-sample to be able to concatenate the feature maps with the same size of the branch above. This is done twice to get the original dimensions of the image.

5. Loss-function

The loss function is explained well in the paper and thus was not very hard to reproduce. The paper states a loss function that consists of adding two different loss functions. The first loss function is an L1 loss function that comes with PyTorch and can be expressed as the distance between the reconstructed image result and the ground truth. The second loss function utilises a VGG-16 network pre-trained on ImageNet. The parameters for the pre-trained VGG-16 network were obtained via a dataset on Kaggle.

6. Training our model

The training settings of the model are described quite concise in the paper. This lead us to the following issues:

6.1 Batch size

The paper states that ”the batch size is set to 16”. However, each image has different dimensions, which means they cannot be put in batches in a straightforward manner. Namely, pytorch tensors complain about different sizes in a certain dimension. As the paper discusses no mitigation of this issue, we decided to instead use a batch size of 1 to circumvent this issue.

6.2 Train/test split

The training procedure, as described in the paper, states that 800 images were used for training. However, it stated that they were selected ”randomly”, without elaborating on what randomization method is used, or what seed used, so that the experiment and model could never be exactly replicated.
Additionally, it states that the remaining 90 images should be used for testing. This is quite clear.
A critical note is that no validation set is used in this research. This limits the models’ ability for hyperparameter tuning.

6.3 Optimizer

The optimizer used was AdamW. AdamW is in the standard library of pytorch, and could thus be found and implemented. The paper specified a learning rate of 1e-4. The other parameters of AdamW were not specified, so the assumption of default parameters for the rest was made. Again, the paper could have been clearer here by adding one sentence, stating what values were used for the other parameters, instead of leaving it up to the reader. Since it was not stated, we made use of the default parameter values in the pytorch implementation of AdamW.

6.4 Loss function

When implementing the loss function into the final network, an error occurred regarding the memory. Namely, loading the vgg16 network model with its entire feature map consumes a large amount of memory, which leads to implementation issues as the GPU’s available to us via kaggle did not allow this much memory allocation. Thus, unfortunately, we had to disregard the VGG16 part of the loss function due to running issues on our part. The paper lacks in specifying whether to use all features of vgg16 or whether to omit the fully connected layers at the end of that model, that lead to much memory space occupation.

7. Results and conclusion

When attempting to compare our results with those presented in figures 7 to 10 of the paper, we found that some of the images shown in these figures were not present in the dataset we had access to. Despite thorough checks, we were unable to locate these specific images, suggesting a discrepancy between the datasets used in the paper and our study.

Furthermore, even for the images that we could access, we noticed differences between our results and those depicted in the paper, see figures 2 and 3. These disparities may arise from various factors, including differences in the implementation of the algorithm, variations in dataset composition, or discrepancies in the overall architecture and methodology.

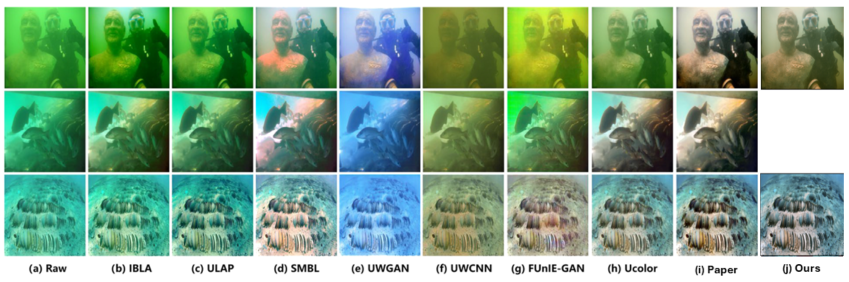

Figure 2: Green tint image comparison

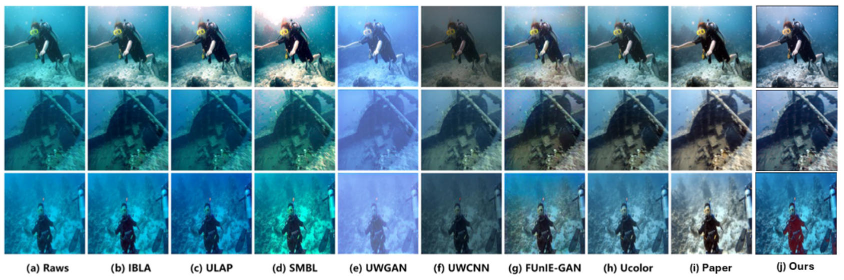

Figure 3: Blue tint image comparison

It's worth noting that due to the paper's lack of clarity and detailed information, particularly regarding the loss function and architectural nuances, we faced challenges in ensuring exact replication of the experimental setup. This uncertainty makes it difficult to ascertain whether our implementation accurately mirrors that of the paper.

While efforts were made to compare our results with those presented in the paper, these discrepancies highlight the importance of transparent reporting and thorough documentation in scientific research. Despite the limitations encountered, our study provides valuable insights and contributes to the broader understanding of the topic, albeit with certain caveats regarding result comparison and reproducibility.

8. Acknowledgements

As for the programming/code the following division of work was used:

- white balance module: Dave
- clahe module: Dave
- REM module: Thomas
- MFF module: Maurits
- PCAM module: Maurits
- Down/upsamle: code Simon/blog Maurits
- Loss function: Thomas
- Integrate to full network: Simon
- Training loop: Simon

As for this blogpost, every person wrote about the module they programmed,(except when mentioned otherwise) and the introduction and results were written and rewritten together.
Overall, we deemed the work division adequate. Most of the work was done sitting together at EEMCS.

8. References

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Jingchun Zhou, Jiaming Sun, Weishi Zhang, and Zifan Lin. Multi-view underwater image enhancement method via embedded fusion mechanism. *Engineering Applications of Artificial Intelligence*, 121:105946, 2023. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2023.105946. URL `https://www.sciencedirect.com/science/article/pii/S0952197623001306`.