| Team Number: | apmcm24209453 |
|---|---|
| Problem Chosen: | A |

2025 APMCM summary sheet

This paper addresses the challenges of blurring, low contrast, and color distortion in images captured in complex underwater environments. By applying multi-angle statistical analysis techniques, we classify the images and establish a degradation model for underwater imagery. To correct color cast issues, we employ semi-physical model parameter adjustment methods for color correction. For complex scenes, we design and implement a Convolutional Neural Network (CNN) model to achieve image enhancement. Additionally, we perform comparative analyses of the proposed methods and provide feasibility assessments and prospects for future applications of underwater enhancement technologies.

In **Question 1**, we applied statistical analysis techniques to the images, conducting multi-angle analysis and classification based on the extracted information. We first extracted and categorized image features, then constructed a feature-based, rule-driven scoring system for classification. Utilizing this scoring system, we classified the images into: 55 with color cast, 61 with low illumination, and 284 blurred images.

In **Question 2**, we delved into the causes of underwater image degradation. Building upon the Jaffe-McGlamery model, we incorporated absorption of the direct component to explain color cast. We detailed four mechanisms of particle interactions with light: absorption, outward scattering, emission, and internal scattering. Focusing on Rayleigh and Mie scattering, we employed the Henyey-Greenstein phase function to model complex scattering behaviors. Simultaneously, we utilized the Lambert-Beer law to describe the absorption characteristics of underwater light. By supplementing absorption and multiple scattering mechanisms, we optimized the traditional model, providing a more comprehensive and accurate underwater imaging degradation model.

In **Question 3**, we enhanced underwater images for a single scene using various semi-physical model parameter adjustment methods for color correction. These methods included white balance adjustment based on the gray-world assumption, perfect reflection white balance, underwater color correction based on Retinex theory, adaptive color correction, and color constancy correction. Experimental results, evaluated based on saturation improvement, color balance, and the Underwater Color Image Quality Evaluation (UCIQE) index, demonstrated that these methods effectively eliminated color cast issues.

In **Question 4**, we designed a lightweight and efficient CNN model for underwater image enhancement in complex scenes. The proposed model adopts a multi-branch parallel structure to effectively extract multi-level features. To enhance generalization capability, we expanded the training set through data augmentation and introduced multi-scale processing strategies to adapt to different depths and lighting conditions. Experimental results indicate that the proposed CNN model provides an efficient and robust solution for underwater image enhancement.

In **Question 5**, we conducted comparative analyses between various underwater enhancement methods in specific scenes and single enhancement techniques in complex scenes. Based on previous analyses and comparisons of evaluation metrics, we drew corresponding conclusions. Building

upon our analyses and methodological innovations, we proposed feasible suggestions for the practical application of underwater image enhancement technologies, including customizing solutions according to specific needs, integrating multi-sensor data, and promoting technological innovation through interdisciplinary collaboration.

The innovation of this paper lies in optimizing and extending traditional models to model and analyze underwater image enhancement from multiple levels and perspectives. Based on multi-angle statistical analysis, classification, and degradation cause modeling, we explored physical methods to handle images with color cast. For complex scenes, we employed CNNs for image restoration and enhancement, achieving significant results. The proposed models exhibit high adaptability and accuracy, providing effective solutions for underwater image enhancement.

**Keywords:** underwater image enhancement CNN image degradation model color correction

# Contents

# I. Introduction

## 1.1 Question Background

As a key information carrier in ocean exploration and development, the quality of underwater images directly affects multiple fields such as seabed resource exploration, scientific research, archaeology, and aquaculture. However, due to the unique nature of underwater environments, light propagation in water is affected by absorption and scattering, leading to issues like low contrast, blurry details, and color distortion in underwater images. degradation phenomena severely limit the effectiveness of underwater images in practical applications. Therefore, how to enhance and restore degraded underwater images through post-processing algorithms has become a hot topic of common concern in academia and industry.

In recent years, with the rapid development of deep learning technology, significant progress has been made in underwater image enhancement and restoration techniques based on deep learning. These technologies improve image quality by constructing complex network models and learning the mapping relationship from degraded images to clear images. Compared with traditional physical or non physical model methods, deep learning methods can better handle the complex degradation problem of underwater images, especially in dealing with color distortion, contrast enhancement, and detail restoration.

Based on the above background, this article aims to establish an effective mathematical model tailored to the characteristics of underwater image degradation and propose a new method for underwater image enhancement and restoration by integrating deep learning techniques. We first analyzed the main causes of underwater image degradation and established the corresponding mathematical model. Then, we designed an improved neural network architecture to enhance and restore images. Finally, we verified the effectiveness of the proposed method by comparing it with existing methods and discussed its potential in practical applications.

## 1.2 Restatement and analysis of Problem

Considering the background information and restricted conditions in the problem statement, the following problems should be solved:

### 1.2.1 *Problem 1: Image Classification and Analysis*

- **Task Description:** Utilize image statistical analysis techniques to conduct a detailed multi-angle analysis of the underwater images provided in Attachment 1, identifying and categorizing the types of image degradation.
- **Classification Requirements:** Classify the images into three main categories: color bias, low illumination, and blur. The classification should be based on statistical features of the images, such as color distribution, brightness levels, and clarity indicators.
- **Output Requirements:** Fill in the file names classified into the three categories at the specified locations in the ”Answer.xls” attachment, and provide a brief explanation for each category, explaining why specific images are assigned to specific categories.

- **Our Analysis and Approach:** The question requires an analysis of the images in Attachment 1. We will score each image based on three dimensions: color cast, low light, and blur, and provide corresponding weights to determine which category it belongs to. Classifying images into single scenes can make subsequent optimizations more targeted.

#### 1.2.2 *Problem 2: Constructing Underwater Image Degradation Model*

- **Task Description:** Based on the degradation types identified in Problem 1, construct an underwater scene image degradation model using the provided underwater imaging model, and attach examples of degraded images.
- **Analysis Requirements:** Analyze the degradation causes of underwater images captured in different scenarios, including but not limited to color bias, low illumination, etc., and discuss the similarities or differences of these degradation models in terms of color, lighting, and clarity.
- **Output Requirements:** Provide a detailed description of the degradation model and image examples, as well as an in-depth analysis of the degradation causes.
- **Our Analysis and Approach:** Building a degradation model can be based on the existing Jaffe McGlamery underwater imaging model, and further detailed exploration based on physical models can be carried out to explain and analyze all the causes of degradation as much as possible. In addition, analyzing the similarity of these degraded models is beneficial for the uniformity of subsequent optimization (i.e., merging optimizations to save steps), and analyzing differences can make optimization more detailed and targeted.

#### 1.2.3 *Problem 3: Proposing Single-Scene Image Enhancement Method*

- **Task Description:** Based on the underwater scene image degradation model established in Problem 2, propose an underwater image enhancement method for a single scene (e.g., color bias, blur, low illumination).
- **Validation Requirements:** Validate the proposed enhancement method using the image data provided in the attachment, and include the enhancement results and corresponding evaluation indicators of the test images in Attachment 2 in the thesis.
- **Output Requirements:** Calculate and present the evaluation indicators such as PSNR, UCIQE, UIQM for the output images, and fill in the results in the table provided in the ”Answer.xls” attachment.
- **Our Analysis and Approach:** The enhancement method for a single scene can be achieved through supervised machine learning. Train a unique model to optimize this type of photo using deep learning on a benchmark dataset, in order to achieve universal optimization for this type of scene. In addition, using Attachment 2 for evaluation and calculating evaluation metrics such as PSNR, UCIQE, UIQM of the output image will also be beneficial for our team to further understand and improve the performance of the model.

#### 1.2.4 *Problem 4: Proposing Complex Scene Image Enhancement Model*

- **Task Description:** Propose an underwater image enhancement model for complex scenes that can handle multiple complex underwater image degradation issues simultaneously.

- **Enhancement Requirements:** Include the enhancement results and corresponding evaluation indicators of the test images in Attachment 2 in the thesis, and calculate the PSNR, UCIQE, UIQM, and other evaluation indicators for the output images.
- **Output Requirements:** Fill in the results in the table provided in the ”Answer.xls” attachment, and discuss the adaptability and effectiveness of the model.
- **Our Analysis and Approach:** This question is based on question 3 and requires us to train a more comprehensive model by synchronously optimizing different aspects of different scenarios. In addition, the optimization order of various problems may also affect the final optimization results, and our team has conducted further analysis and research on it. Similarly, we calculate and output evaluation metrics such as PSNR, UCIQE, UIQM for the output image to quantitatively assess the performance of the optimization algorithm.

#### 1.2.5 *Problem 5: Comparing Enhancement Techniques and Making Suggestions*

- **Task Description:** Compare various enhancement techniques in specific scenarios with a single enhancement technique in complex scenarios, and make suggestions for the practical application of underwater visual enhancement.
- **Comparison Requirements:** Based on the results of Problems 3 and 4, analyze the performance of different enhancement techniques in specific and complex scenarios, and propose suggestions for practical applications.
- **Output Requirements:** Provide a detailed comparison and analysis, and suggestions to help users understand the advantages and limitations of different techniques, and guide the selection of techniques in practical applications.
- **Our Analysis and Approach:** Comparing the results of question 3 and question 4, it is verified that multi-faceted enhancement results are superior to unilateral enhancement. In addition, we should propose more practical and feasible suggestions to further align algorithm models with applications.

# II. Model Hypothesis

## 2.1 Ignoring the emission of underwater particles

In underwater imaging and optical transmission simulation, the rationality of ignoring particle emission is mainly based on the dominant role of scattering processes in the underwater environment, which makes the impact of particle emission relatively small. In addition, ignoring particle emission can significantly improve computational efficiency and simplify the model without significantly affecting the accuracy of the results in many cases. Experimental data and theoretical analysis also support the strategy of ignoring particle emission under specific conditions, such as clear water or in application scenarios where scattering characteristics are more critical. Therefore, this simplified method provides a practical and effective approach for underwater optical research while maintaining simulation accuracy.

## 2.2 Only consider the collective statistical behavior of particles

We presume that the scattering medium water(what we are considering in this paper) can be depicted as an ensemble of minuscule particles. Given their minuscule scale and random distribution, we refrain from modeling each specific particle in our light propagation simulations. Instead, we focus on the collective statistical behavior of light[1] as it traverses the medium. Additionally, it is postulated that these particles are significantly separated from one another in comparison to their own dimensions. This supposition indicates that when a photon passes through the medium and engages with a particle, the outcome of this interaction is probabilistically distinct from the results of any subsequent encounters.

## 2.3 The average values of the GB channels in underwater images should be close to each other

According to a widely accepted theory in the field of automatic white balance, any image with a rich variety of colors will have its red, green, and blue (RGB) components' average values converge towards equilibrium. In underwater settings, it is observed that the average values of the blue and green components tend to equalize, while the red component is treated differently due to its unique characteristics under water. The analysis of the RGB data from 400 images in Attachment 1, as depicted in the figure below, supports this hypothesis. This observation is crucial for understanding color balance in underwater photography and can inform the development of more accurate automatic white balance algorithms tailored for such environments.

Although there is a significant difference between the red (R), green (G), and blue (B) channels in underwater environments, common sense and literature review indicate that red should not be present underwater; even a small amount of red is intolerable. Therefore, even though we initially intended to compensate for the R channel, we found that the results were not satisfactory. In the end, we processed the R channel according to its original values, ensuring that the red channel values remain lower than those of the green and blue channels. This approach guarantees that the processed image does not contradict common sense and the characteristics of the actual underwater environment.

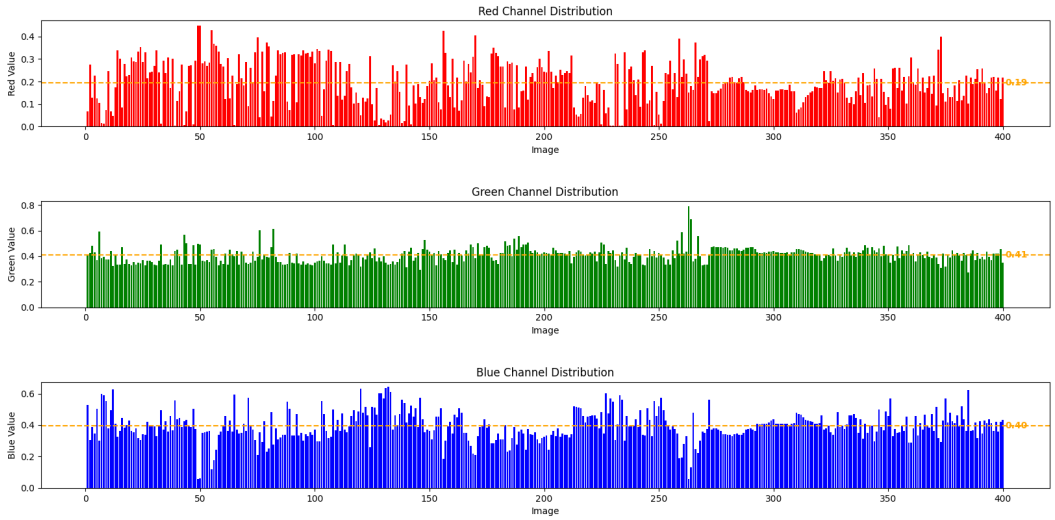



**Figure 1 RGB Component Analysis: GB Color Balance in Underwater Environments with 400 Images**

## 2.4 The brightest point in an image should be white

According to physical reflection characteristics: In the real world, white objects can reflect light of all wavelengths, so under any lighting conditions, the reflected light from a white object should be white or nearly white. The perfect reflection algorithm is based on this physical property, considering that the brightest point in the image (i.e., the point with the most reflected light) should be white.

# III. Notations

**Notations of different Symbols**

| Symbol | Description | Unit |
|---|---|---|
| $avg$ | Channel average values | - |
| $\eta_c$ | Dominant color ratio | - |
| $s$ | Saturation | - |
| $v$ | Brightness | - |
| $\eta_d$ | Dark pixel ratio | - |
| $\delta_v$ | Brightness skewness | - |
| $\mathcal{L}[v]$ | Laplacian variance | - |
| $\mathcal{G}$ | Sobel kernels | - |
| $G$ | Gradient magnitude | - |
| $I$ | The grayscale image | - |
| $\Gamma_c$ | Color cast score | [0-100] |
| $\Gamma_l$ | Low light score | [0-100] |
| $\Gamma_b$ | Blur score | [0-100] |
| $\sigma_a$ | Absorption coefficient | $m^{-1}$ |
| $\sigma_s$ | Scattering coefficient | $m^{-1}$ |
| $\sigma_t$ | Extinction coefficient | $m^{-1}$ |
| $\rho$ | Albedo | - |
| $p$ | Phase function | $sr^{-1}$ |
| $A$ | Absorbance | - |
| $\varepsilon$ | Molar absorptivity | $L \cdot mol^{-1} \cdot cm^{-1}$ |
| $c$ | Concentration of the absorbing species | $mol/L$ |
| $l$ | Path length of light in medium | $cm$ |
| $F$ | The output after mapping | - |
| $H$ | Learning function | - |
| $M$ | The image obtained through learning | - |

**Notations of different Operators**

| Operator | Description | Function |
|---|---|---|
| $\langle ... \rangle$ | Average of something | $\langle x \rangle = \frac{1}{N} \sum_{i=1}^{N} x_i$ |
| $\nabla$ | The gradient of a scalar field | $\nabla f(x,y,z) = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$ |
| $\Delta$ | Laplacian | $\Delta f = \nabla^2 f$ |
| $*$ | Convolution | $f(t) * g(t) = \int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau$ |
| $\eta$ | the ratio of something | $\eta = \frac{\text{Specific variables}}{\text{Sum of all variables}} \times 100\%$ |
| $\Gamma$ | Score normalized to [0-100] | Different weights |
| $\mu_n$ | N-th order central moment | $\mu_n(x) = \langle (x_i - \langle x \rangle)^n \rangle$ |
| $\sigma$ | standard deviation | $\sigma(x) = \sqrt{\mu_2} = \sqrt{\langle (x_i - \langle x \rangle)^2 \rangle}$ |
| $\delta$ | Skewness | $\delta(x) = \frac{\mu_3(x)}{\sigma(x)^3}$ |

# IV. Model Establishment and Solution of Problem 1

For Question 1, we extracted features from the images and transformed them into intelligent information.Subsequently, we developed an image quality assessment model to score and classify the images. Specifically, we constructed a feature-based rule-driven scoring system to categorize the images.

Based on the factors influencing image quality, we selected and defined features and calculated the scores of each image under these features.

## 4.1 Color Cast Score Calculation:

The main features included in the calculation of the color cast score are: the mean values of each color channel, the standard deviation of the color channel means, the proportion of the dominant color, and the average saturation.

By integrating these features, appropriate mathematical methods are used to standardize each feature to eliminate the influence of dimensions. Combined with weights, a weighted summation is performed to finally obtain the normalized color deviation score of the image, serving as the evaluation result in this scoring system.

- Channel average values:

$$avg_r = \langle r_i \rangle = \frac{1}{N}\sum_{i=1}^{N} r_i \tag{1}$$

$$avg_g = \langle g_i \rangle = \frac{1}{N}\sum_{i=1}^{N} g_i \tag{2}$$

$$avg_b = \langle b_i \rangle = \frac{1}{N}\sum_{i=1}^{N} b_i \tag{3}$$

$$\langle avg \rangle = \frac{avg_r + avg_g + avg_b}{3} \tag{4}$$

- Standard deviation of channel average values:

$$\sigma(\langle avg \rangle) = \sqrt{\frac{1}{3}\sum_{c\in\{r,g,b\}} (\text{avg}_c - \langle avg \rangle)^2} \tag{5}$$

$$= \sqrt{\langle (avg_c - \langle avg \rangle)^2 \rangle} \tag{6}$$

- Dominant color ratio:

$$\eta_c = \frac{\max(\text{avg}_r, \text{avg}_g, \text{avg}_b)}{\text{avg}_r + \text{avg}_g + \text{avg}_b + 10^{-6}} \tag{7}$$

  in which $10^{-6}$ is added to prevent the denominator from being zero.

- Average saturation:

$$\langle s \rangle = \frac{1}{N}\sum_{i=1}^{N} s_i \tag{8}$$

  where $s$ is the saturation value of each point.

- Color cast score (normalized to [0-100]):

$$\Gamma_c = \frac{\sigma(\langle avg \rangle)}{127.5} \times 60 + \eta_c \times 20 + \frac{\langle s \rangle}{255} \times 20 \tag{9}$$

## 4.2 Low Light Score Calculation:

In the specific implementation, the calculation of the low light score mainly considers features such as average brightness, proportion of dark pixels, and brightness skewness.

We performed standardization on each feature to eliminate the influence of dimensions, and combined them with weights to calculate the final score.

- Assume that brightness of each point $i$ is $v_i$.
- Average brightness:

$$\langle v \rangle = \frac{1}{N} \sum_{i=1}^{N} v_i \tag{10}$$

- Dark pixel ratio:

$$\eta_d = \frac{\sum_{i=1}^{N} [v_i < 80]}{N} \tag{11}$$

  where $[v_i < 80]$ is an indicator function, taking 1 when $v_i < 80$, and 0 otherwise.
- Brightness skewness:

$$\delta_v = \delta(v) = \frac{\mu_3(v)}{\sigma(v)^3} \tag{12}$$

- Low light score (normalized to [0-100]):

$$\Gamma_l = (1 - \frac{\langle v \rangle}{255}) \times 50 + \eta_d \times 30 + max(0, \delta(v)) \times 20 \tag{13}$$

## 4.3 Blur Score Calculation:

Similarly, the calculation of the blur score is mainly based on features such as Laplacian variance and gradient mean.

We also standardized these features to eliminate the influence of dimensions, and combined them with weights to obtain the final score.

- Convert the image to the grayscale image $I$.
- Laplacian variance:

$$\mathcal{L}[v] = \mu_2(\nabla^2 I) \tag{14}$$

- Gradient magnitude mean: The Sobel kernels for computing the gradient are defined as follows:

$$\mathcal{G}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \mathcal{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{15}$$

$$G_i = \sqrt{(x_i * \mathcal{G}_x)^2 + (y_i * \mathcal{G}_y)^2} \tag{16}$$

$$\langle G \rangle = \frac{1}{N} \sum_{i=1}^{N} G_i \tag{17}$$

- Blur score (normalized to [0-100]):

$$\Gamma_b = 100 - \min\left(100, \frac{\mathcal{L}[v]}{500} \times 50 + \frac{\langle G \rangle}{100} \times 50\right) \tag{18}$$

## 4.4 Partial results presented

After separately calculating the scores of the images in the three classification categories, we assign each image to the category with the highest score. The distribution of the scores, along with some scores and classification results, are shown in the following table:

**Table 1 Partial Scores and Classification Results of Each Image**

| filename | category | color_cast_score | low_light_score | blur_score |
|---|---|---|---|---|
| image_273.jpg | blur | 42.07722561 | 25.40856785 | 90.0816605 |
| image_274.jpg | blur | 42.82545136 | 28.61212883 | 91.46205457 |
| image_275.jpg | blur | 41.57152686 | 33.6608241 | 93.01451697 |
| image_276.jpg | blur | 38.99320029 | 32.4783651 | 92.90215239 |
| image_277.jpg | blur | 37.93598891 | 30.92624769 | 89.72824132 |

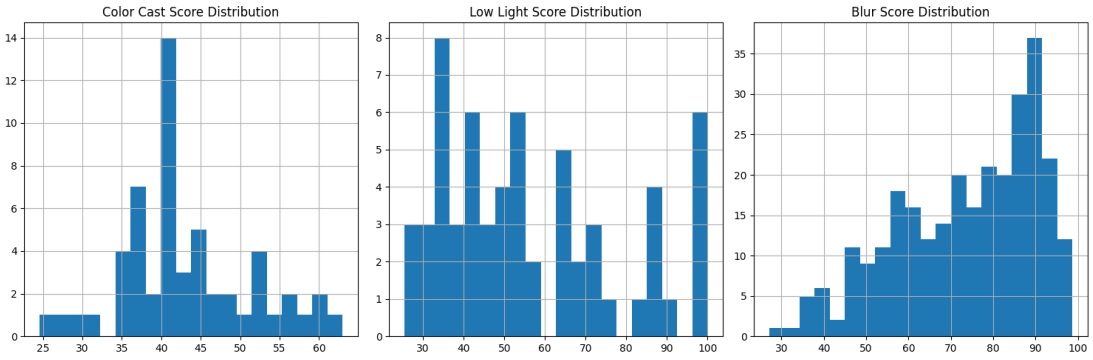



**Figure 2 The Distribution of Image Scores**

# V. Model Establishment and Solution of Problem 2

Based on the existing Jaffe McGlamery underwater imaging model[2], our team has redrawn it. The underwater image captured by the camera can be represented as a linear combination of three components: direct component, forward scattering component, and backward scattering component. Among them, the forward scattering component refers to the light that enters the imaging system after being scattered by suspended particles in water, which can cause the edges of the obtained image to be indistinct, that is, blurred. The backscattering component refers to the scattering of natural light by suspended particles after entering water, resulting in low contrast in the obtained image, that is, low illumination.

In addition, we have supplemented the part where the direct component is absorbed during propagation in the aqueous medium to improve the degradation model and explain color cast. (Of course, scattering also has a certain degree of influence on color cast) As shown in the following figure (where the absorption part is a unique part of our model).

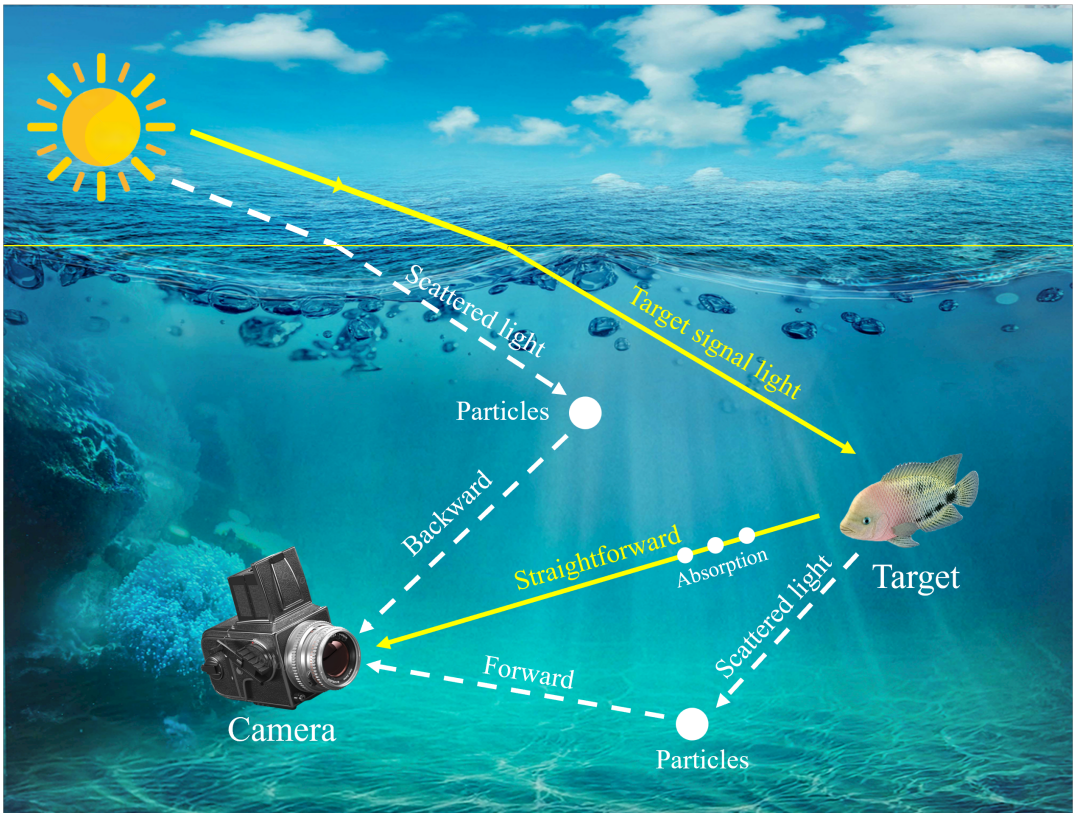



**Figure 3 Optimized Jaffe McGlamery underwater imaging model**

The above is a classification of different degraded scenes, which essentially refers to the influence of particles on light. When light passes through a medium, its physical reasons can be divided into the following four categories.

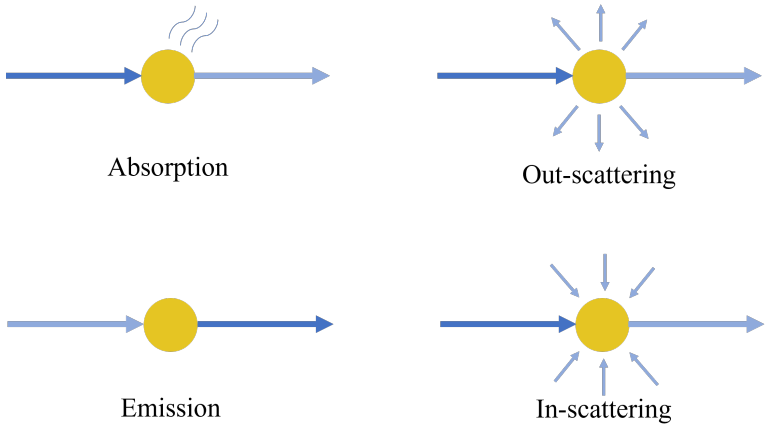



**Figure 4 Particle Model**

- **Absorption ($\sigma_a$ function)** - Photons are absorbed by matter in the medium and converted into heat or other forms of energy.
- **Out-scattering ($\sigma_s$ function)** - Photons are deflected by particles within the medium and scattered. The phase function $p$ describes the distribution of the directions of the scattered light.
- **Emission** - When a medium reaches a high temperature, such as the blackbody radiation of fire, light is emitted.
- **In-scattering ($\sigma_s$ function)[3]** - Photons can scatter into the current path of light from any direction after bouncing off particles, contributing to the final radiance. The amount of light scattered from a given direction also depends on the phase function $p$ for that direction of light.

In summary, the addition of photons along the path is a function of in-scattering $\sigma_s$ and emission. The function $\sigma_t = \sigma_a + \sigma_s$ represents absorption and out-scattering. From the radiative transfer equation, it is known that these coefficients represent the derivative of radiance with respect to $L(x, v)$ in the direction of position $x$ and $v$. Therefore, the values of these coefficients range within $[0, +\infty]$. The scattering and absorption coefficients * determine the albedo $\rho$ of the medium, defined as:

$$\rho = \frac{\sigma_s}{\sigma_s + \sigma_a} = \frac{\sigma_s}{\sigma_t} \tag{19}$$

$\rho$ indicates the importance of scattering relative to absorption within the visible spectrum considered for the medium, that is, the overall reflectivity of the medium. The value of $\rho$ ranges within [0,1]. A value close to 0 indicates that most of the light is absorbed, leading to a turbid medium, such as dark exhaust fumes. A value close to 1 indicates that most of the light is scattered rather than absorbed, resulting in a brighter medium, such as air, clouds, or very clear water.

For the underwater scenarios we need to analyze, in some cases, the albedo $\rho$ in deep water may be close to a lower value, indicating that the medium tends to absorb light rather than scatter it. For instance, deep or heavily polluted waters may have a lower albedo, leading to reduced light penetration and making the water appear more turbid. In contrast, in clear deep water environments, the albedo $\rho$ may be closer to 1, meaning that more light is scattered, thus making the water appear more transparent.

In deep water, as we discussed in subsection 2 . 1, we do not consider the emission part. Then,

what we need to explain the effect of particles on light can be divided into two parts: scattering and absorption.

## 5.1 Particle Scatter

Now let's analyze the part of light scattering. In the case of multiple scattering path tracking, many people have described the radiative transfer equation. Here, we will focus on single scattering, which only considers one rebound of light on the particles that make up the participating medium, while multiple scattering tracks multiple reflections of each optical path, making it much more complex and not considered here.

The scattering integral of point light sources in a scene at a given position $x$ and direction $v$ can be represented as follows[4]:

$$L_{\text{scat}}(x, v) = \pi \sum_{i=1}^{n} p(v, l_{c_i}) v(x, p_{\text{light}_i}) c_{\text{light}_i}(\|x - p_{\text{light}_i}\|) \tag{20}$$

where $n$ is the number of light sources, $p()$ is the phase function, $v()$ is the visibility function, $l_{c_i}$ is the direction vector of the $i$-th light, and $p_{\text{light}_i}$ is the position of the $i$-th light. Additionally, $C_{\text{light}_i}$ is the radiance of the $i$-th light source as a function of its distance from the position, using the inverse square law. The visibility function $v(x, p_{\text{light}_i})$ represents the ratio of light from $p_{\text{light}_i}$ that reaches position $x$:

$$v(x, p_{\text{light}_i}) = \text{shadowMap}(x, p_{\text{light}_i}) \cdot \text{volShad}(x, p_{\text{light}_i}) \tag{21}$$

where $\text{volShad}(x, p_{\text{light}_i}) = T_r(x, p_{\text{light}_i})$.

Below, we analyze the phenomena of light scattering and extinction. Assume $\sigma_s = (0.5, 1, 2)$ and $\sigma_a = (0, 0, 0)$. In the shorter optical paths within the medium, in-scattering events will dominate over extinction. For example, at shallow depths, $T_r \approx 1$, and out-scattering becomes the primary event. Due to the highest value of $\sigma_s$ in this channel, the material will appear blue. The deeper the light penetrates the medium, the fewer photons pass through, which is a result of extinction.

The participating medium is composed of particles with different radii. The size distribution of these particles will affect the probability of light scattering in a specific direction relative to the direction of light propagation. When calculating in scattering, use the phase function to macroscopically describe the probability and distribution of scattering direction, as shown in formula 20.

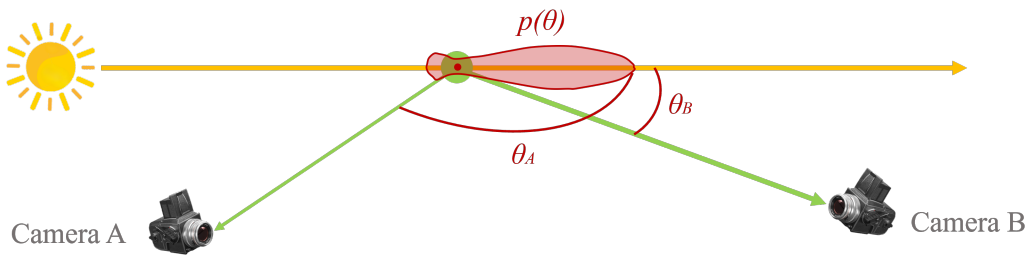



**Figure 5 The phase function (red) and its effect on scattered light (green) are functions of θ**

The red phase function is represented with parameter $\theta$, and the yellow line represents the angle between the direction of light propagation and the green direction of $v$. Note that in this phase function example, there are two main lobes: a small backward-scattering lobe in the opposite direction of the light path and a large forward-scattering lobe. Camera B is positioned in the direction of the large forward-scattering lobe, so it will receive more scattered radiance than Camera A. This is in accordance with the conservation of energy, which dictates that the integral of the phase function over a unit sphere must be equal to 1.

The phase function will modify the in-scattering at a point based on the directional radiance information arriving at that point. The simplest function is isotropic: light scatters uniformly in all directions, and this ideal but unrealistic behavior is expressed as:

$$p(\theta) = \frac{1}{4\pi} \tag{22}$$

Here, $\theta$ is the angle between the incident light and the out-scattering direction, and $4\pi$ is the surface area of a unit sphere.

The physically based phase function depends on the relative size of the particle $s_p$:

$$s_p = \frac{2\pi r}{\lambda} \tag{23}$$

where $r$ is the particle radius and $\lambda$ is the wavelength of the light considered:

- When $s_p \ll 1$, Rayleigh scattering occurs.
- When $s_p \approx 1$, Mie scattering occurs.
- When $s_p \gg 1$, geometric scattering occurs.

### 5.1.1 *Rayleigh scattering*

Lord Rayleigh[5] derived the formula for the scattering of light from air molecules. This phase function has two lobes, as shown in the figure, which are the backward and forward scattering relative to the direction of light.

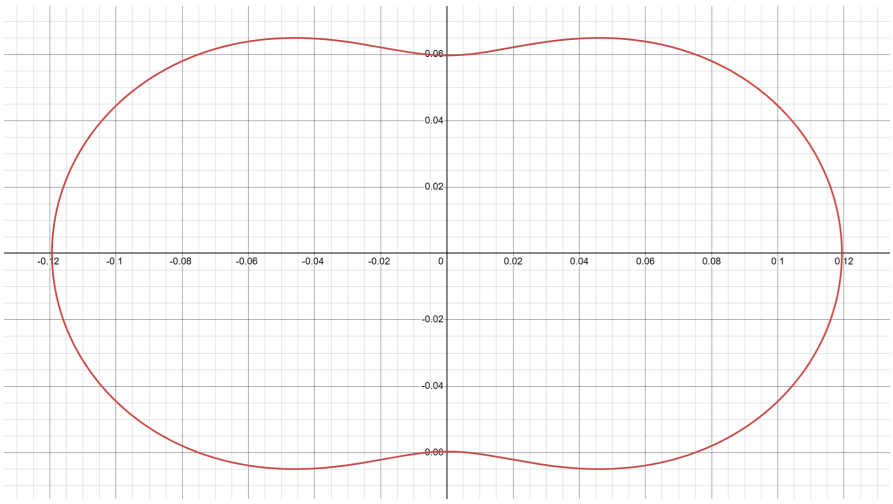



**Figure 6 Polar plot of Rayleigh phase function with θ**

Rayleigh's phase function is represented in polar coordinates. Light enters horizontally from the left, and the relative intensity is displayed as angle $\theta = 0$, measured counterclockwise from the x-axis. The chances of scattering forward and backward are the same.

This function takes its value at $\theta$, which is the angle between the incident light and the out-scattering direction. The function is:

$$p(\theta) = \frac{3}{16\pi}(1 + \cos^2\theta) \tag{24}$$

Rayleigh scattering is highly dependent on wavelength. When considering the function of the wavelength $\lambda$, the scattering coefficient $\sigma_s$ of Rayleigh scattering is proportional to the inverse fourth power of the wavelength:

$$\sigma_s(\lambda) \propto \frac{1}{\lambda^4} \tag{25}$$

This relationship implies that short-wavelength blue or violet light scatters more easily than long-wavelength red light. The spectral distribution obtained from Equation 25 can be converted to RGB using the spectral color matching function (Section 8.1.3): $\sigma_s = (0.490, 1.017, 2.339)$. This value is normalized to a brightness of 1 and should be scaled according to the desired scattering intensity. This explains why blue light scatters more in the ocean, resulting in visual effects.

### 5.1.2 *Mie scattering*

Mie scattering[6] is a model that can be used when the size of the particles is approximately the same as the wavelength of light. This type of scattering is independent of the wavelength. MiePlot software can be used to simulate this phenomenon. The phase function for a specific particle size is usually a complex distribution with a strong and sharp directional lobe, meaning that relative to the direction of photon movement, the probability of photons scattering in a specific direction is high. Calculating such a phase function for volume rendering is costly, but fortunately, it is rarely needed. Typical media have a continuous distribution of particle sizes, and averaging all these different sizes of the Mie phase function can yield a smooth average phase function for the entire medium. Therefore, a relatively smooth phase function can be used to represent Mie scattering.

A commonly used phase function is the Henyey-Greenstein (HG) phase function, which was initially used to model light scattering in interstellar dust. This function cannot capture every complex scattering behavior in the real world, but it can well represent a phase function lobe, that is, in the direction of the main scattering. It can be used to represent any type of participating medium like smoke, fog, or dust. Such media can exhibit strong backward or forward scattering, leading to large visual halos around light sources. Examples include spotlights in fog and the strong silver-lining effect on the edges of clouds in the direction of the sun.

The HG phase function can represent more complex behaviors than Rayleigh scattering and is given by:

$$p_{hg}(\theta, g) = \frac{1 - g^2}{4\pi(1 + g^2 - 2g\cos\theta)^{1.5}} \tag{26}$$

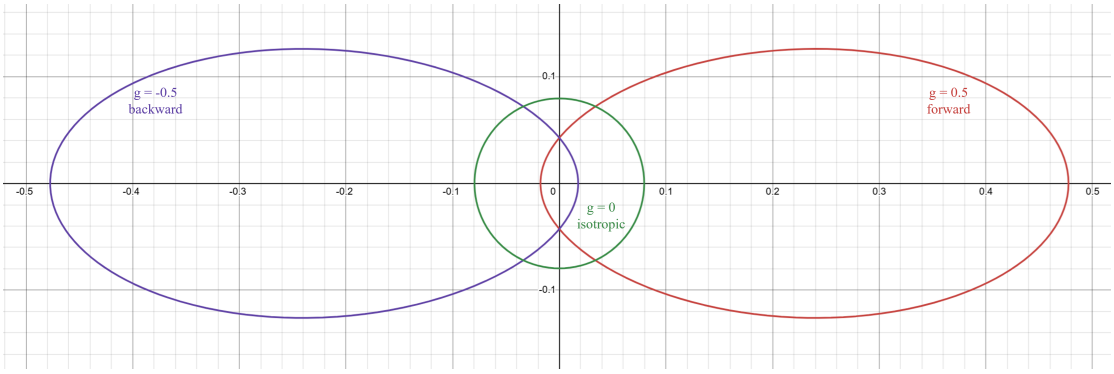



**Figure 7 Mie Scattering when g=0(green), g=-0.5(puple) and g=0.5(red)**

As we can see in fig.7, the phase function can produce different shapes, and the $g$ parameter can be used to represent backward ($g < 0$), isotropic ($g = 0$), or forward ($g > 0$) scattering, where $g$ is in the range [-1, 1].

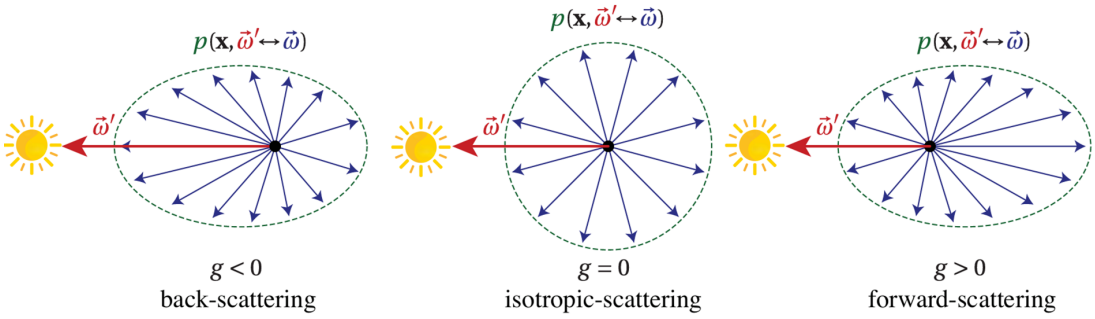



**Figure 8 Mie Scattering**

And we can also use a faster method if we simplify Equation26 into an approximate equation:

$$p(\theta, k) = \frac{1 - k^2}{4\pi(1 + k\cos\theta)^2} \tag{27}$$

where:

$$k \approx 1.55g - 0.55g^3 \tag{28}$$

It does not include any complex power functions, but merely a square, which makes for faster computation. To map this function to the original HG phase function, it is necessary to calculate the $k$ parameter from $g$. For a participating medium with a constant $g$ value, this needs to be done only once. In practical applications, the Schlick phase function serves as a good approximation for energy conservation.

It is also possible to combine multiple HG or Schlick phase functions to represent a more complex range of general phase functions. This allows us to simultaneously represent phase functions with strong forward and backward scattering lobes.

### 5.1.3 *geometric scattering*

Geometric scattering[7] occurs on particles significantly larger than the wavelength of light. In this case, light can refract and reflect within each particle. This behavior may require a complex scattering phase function to simulate it at a macroscopic level. The polarization of light can also affect this type of scattering.

Scattering is dominated by **Rayleigh scattering and Mie scattering** underwater, so we will not introduce geometric scattering in detail here.

### 5.1.4 *Possible Raman scattering*

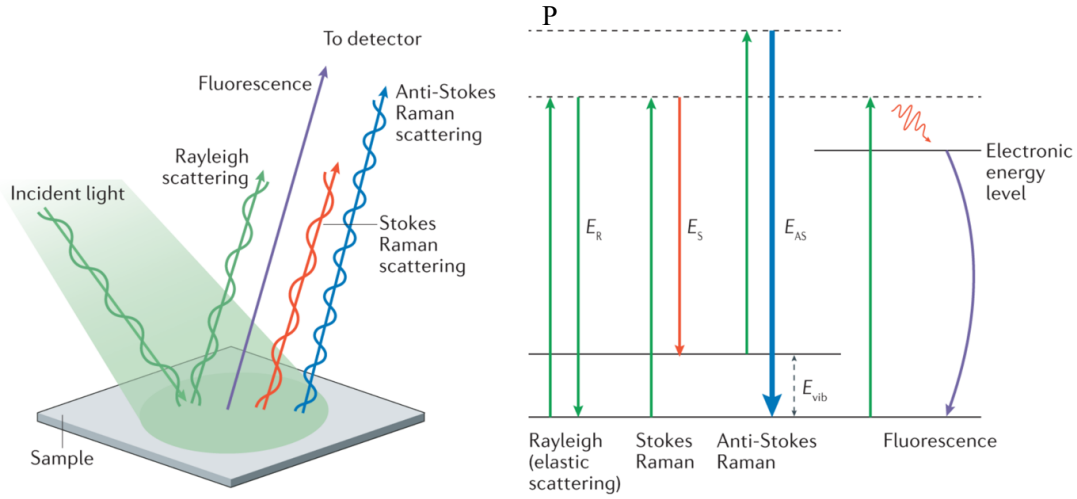



**Figure 9 Raman Scattering**

When light acts on a sample, the sample material will produce scattered light. In the scattered light, besides the Rayleigh light with the same frequency as the incident light, there are a series of other frequencies of light on both sides of the Rayleigh light. This scattered light is named Raman light[8].

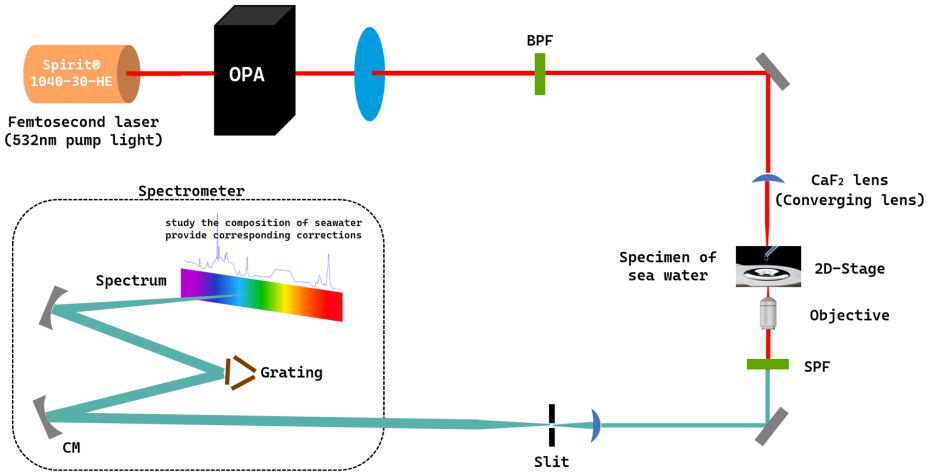



**Figure 10 Experimental instruments of Raman Scattering to investigate the components of seawater**

The above is a diagram of a Raman apparatus proposed by the author for the analysis of seawater samples. The author has even built the relevant instrument, which is capable of analyzing the components of seawater. This could further facilitate more targeted modeling, although such an approach might not be necessary.

## 5.2 Particle Absorption

For particle absorption, we adopted the classical Lambert Beer law[9].

The Lambert-Beer Law, also known as Beer-Lambert Law or Beer's Law, is a fundamental principle in spectroscopy and analytical chemistry that describes the relationship between the absorption of light by a solution and the properties of the solution. It only considers the collective statistical behavior of particles, as we mentioned in subsection 2.2.

The law states that the absorbance $A$ of a solution is directly proportional to the concentration $c$ of the absorbing species and the path length $l$ that the light travels through the solution. Mathematically, it is expressed as:

$$A = \varepsilon \cdot c \cdot l \tag{29}$$

- The absorbance $A$ is a measure of how much light is absorbed by a material.
- The molar absorptivity $\varepsilon$ is a property of a substance and is measured in units of $L{\cdot}mol^{-1}{\cdot}cm^{-1}$.
- The concentration $c$ of the absorbing species is expressed in moles per liter (mol/L).
- The path length $l$ of light in the medium is the distance that the light travels through the material and is measured in centimeters (cm).

The law is derived from the assumption that the light is absorbed exponentially as it passes through the solution. The fraction of light transmitted through the solution decreases exponentially with increasing concentration and path length, which can be described by the equation:

$$I = I_0 \cdot e^{-\varepsilon \cdot c \cdot l} \tag{30}$$

where $I$ is the intensity of light after passing through the solution, and $I_0$ is the initial intensity of light.

# VI. Model Establishment and Solution of Problem 3

In this section, we employed a variety of advanced semi-physical model parameter adjustment methods to achieve color bias correction in images. The specific methods include white balance adjustment based on the gray world assumption, which posits that the average values of the RGB channels in natural images should be similar; perfect reflection white balance, which is based on the assumption that the brightest points in an image should be white; underwater environment color correction based on the Retinex theory, which corrects colors by simulating the human visual system's adaptability to changes in lighting; adaptive color correction, which dynamically adjusts colors based on image content; and color constancy correction, aimed at simulating the human eye's stability in color perception under different lighting conditions. By integrating these methods, we successfully corrected the color bias in images, significantly improving the color cast issue. Experimental results demonstrate that the corrected images exhibit more natural and accurate colors, with a markedly better performance compared to traditional methods. These findings provide new perspectives and solutions in the field of image color correction.

## 6.1 White balance adjustment

White balance adjustment[10] based on the gray world assumption is an image processing technique whose core concept is the assumption that the average values of the RGB channels in natural images should tend towards the same gray value K. This assumption is based on the observation that the average reflection of light by natural scenes is approximately "gray" overall. The color correction algorithm can be described as follows:

1. Calculate the average values of each color channel:

$$\langle R\rangle = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} R(i,j) \tag{31}$$

$$\langle G\rangle = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} G(i,j) \tag{32}$$

$$\langle B\rangle = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} B(i,j) \tag{33}$$

   where $M$ and $N$ are the height and width of the image, respectively.

2. Find a reference value $K$, typically the geometric mean of the channel means:

$$K = \sqrt[3]{\langle R\rangle \cdot \langle G\rangle \cdot \langle B\rangle} \tag{34}$$

3. Calculate the gain values for each channel:

$$R_{\text{gain}} = \frac{K}{\langle R\rangle} \tag{35}$$

$$G_{\text{gain}} = \frac{K}{\langle G\rangle} \tag{36}$$

$$B_{\text{gain}} = \frac{K}{\langle B\rangle} \tag{37}$$

4. Correct each pixel in the image by multiplying with the corresponding gain value:

$$R_{\text{corrected}}(i,j) = R(i,j) \cdot R_{\text{gain}} \tag{38}$$

$$G_{\text{corrected}}(i,j) = G(i,j) \cdot G_{\text{gain}} \tag{39}$$

$$B_{\text{corrected}}(i,j) = B(i,j) \cdot B_{\text{gain}} \tag{40}$$

## 6.2 Perfect reflection

Perfect reflection[11] is based on the hypothesis that the brightest pixels in an image are equivalent to points on shiny or mirror-like surfaces of objects, which provide a lot of information about the scene's illumination conditions. If there are pure white parts in the scene, then the light source information can be directly extracted from these pixels. This is because mirror-like or shiny surfaces do not absorb light themselves, so the color they reflect is the true color of the light source. This is due to the fact that the reflection ratio function of mirror-like or shiny surfaces remains constant over a broad range of wavelengths. The perfect reflection method utilizes this characteristic to adjust the image. During the execution of the algorithm, it detects the brightest pixel in the image and uses it as a reference white point. Methods based on this concept are collectively referred to as the perfect reflection method, also known as the mirror method. In simple terms, the brightest point in the entire image is either white or reflected by a mirror, meaning that the brightest point represents the properties of the light source. However, this point should itself be a white point, and based on this, gain values can be calculated for correction.

1. Image Related Information Statistics:

$$\begin{cases} R_{\text{max}} = \max(R_{ij}) \quad (i = 1 \cdots N,\ j = 1 \cdots M) \\ G_{\text{max}} = \max(G_{ij}) \quad (i = 1 \cdots N,\ j = 1 \cdots M) \\ B_{\text{max}} = \max(B_{ij}) \quad (i = 1 \cdots N,\ j = 1 \cdots M) \end{cases} \tag{41}$$

2. RGB Channel Gain Calculation:

$$\begin{cases} \text{Gain}_{R_{\text{max}}} = \max(R_{\text{max}}, G_{\text{max}}, B_{\text{max}})/R_{\text{max}} \\ \text{Gain}_{G_{\text{max}}} = \max(R_{\text{max}}, G_{\text{max}}, B_{\text{max}})/G_{\text{max}} \\ \text{Gain}_{B_{\text{max}}} = \max(R_{\text{max}}, G_{\text{max}}, B_{\text{max}})/B_{\text{max}} \end{cases} \tag{42}$$

3. The correction process: The correction process is applied to each pixel in the image for each color channel. The gain is applied, and the values are clipped to ensure they fall within the valid range of 0 to 255.

Take the red channel as an example, the correction is as follows:

$$R'_{\text{max}} = \begin{cases} R \times \text{Gain}_{R_{\text{max}}} & \text{if } R \times \text{Gain}_{R_{\text{max}}} < 255 \\ 255 & \text{if } R \times \text{Gain}_{R_{\text{max}}} > 255 \end{cases} \tag{43}$$

The same principle applies to the green and blue channels.

## 6.3 Combination of Gray World and Perfect Reflector Methods

The combination of the Gray World assumption and the Perfect Reflector method involves merging these two approaches. The specific formulas are as follows:

$$u^r R_{\text{ave}}^2 + v^r R_{\text{ave}} = K_{\text{ave}}; \tag{44}$$

$$u^r R_{\text{max}}^2 + v^r R_{\text{max}} = K_{\text{max}}; \tag{45}$$

$$K_{\text{ave}} = \frac{R_{\text{ave}} + G_{\text{ave}} + B_{\text{ave}}}{3}; \tag{46}$$

$$K_{\text{max}} = \frac{R_{\text{max}} + G_{\text{max}} + B_{\text{max}}}{3}; \tag{47}$$

By solving the above system of equations, we can then correct the original pixel values:

$$R_{\text{new}} = u^r R_{\text{ave}}^2 + v^r R_{\text{ave}} \tag{48}$$

where $R_{\text{new}}$ is the corrected red channel value, $R_{\text{ave}}$ is the average value of the red channel, and $u^r$ and $v^r$ are the correction parameters for the red channel.

## 6.4 Retinex Image Enhancement Algorithm

Principles of Retinex Theory：[12]

1. The color of an object is determined by its reflective properties to long-wave (red), medium-wave (green), and short-wave (blue) light, rather than by the absolute value of the reflected light intensity.
2. The color of an object remains consistent and is not affected by the non-uniformity of illumination.
3. The Retinex theory is based on the principle of color perception constancy.

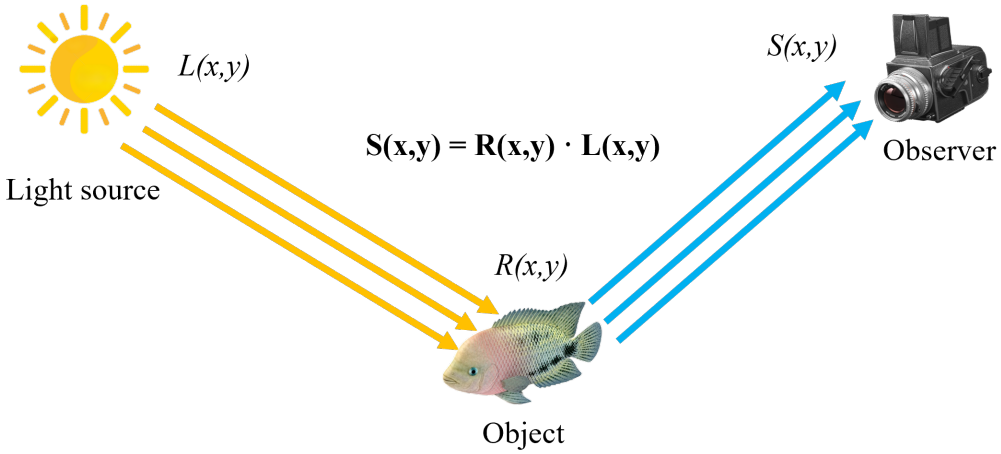



**Figure 11 Schematic Diagram of Retinex Theory**

As shown in the figure above, the image $S(x, y)$ observed by the viewer is composed of the incident image $L(x, y)$ and the reflected image $R(x, y)$. That is, the incident light shines on the

reflecting object, and through the reflection of the object, the reflected light enters the human eye, forming an image. The formula is represented as follows:

$$S(x,y) = L(x,y) \cdot R(x,y) \tag{49}$$

The core of the Retinex algorithm is to estimate the incident image $L(x,y)$ from the observed image $S(x,y)$, to estimate the $L(x,y)$ component, and to remove the $L(x,y)$ component to obtain the reflection image $R(x,y)$. The specific operations are as follows:

1. Take the logarithm of both sides:

$$\log[R(x,y)] = \log[S(x,y)] - \log[L(x,y)] \tag{50}$$

2. Since $L(x,y)$ can only be approximated, it is represented by convolving $S(x,y)$ with a Gaussian kernel. The formula is as follows:

$$\log[R(x,y)] = \log[S(x,y)] - \log[S(x,y) * G(x,y)] \tag{51}$$

where $*$ denotes the convolution operation, and $G(x,y)$ represents the Gaussian kernel.

3. Map the obtained $R(x,y)$ to the range (0, 255). The quantization formula is:

$$R(x,y) = \frac{Value - \text{Min}}{\text{Max} - \text{Min}} \times (255 - 0) \tag{52}$$

## 6.5 Other optimizations

We also definite other three color correction functions, each targeting different image processing needs.These functions optimize the color representation of images through different methods, aiming to improve visual quality and color accuracy.

### 6.5.1 *underwater color correction*

The underwater color correction function is specifically designed for color correction of underwater images. It restores the natural colors of the images by estimating underwater lighting and transmission maps, utilizing physical models and image statistical characteristics to simulate the propagation of light in water and the reflection from objects.

### 6.5.2 *adaptive color correction*

The adaptive color correction function implements adaptive color correction. It automatically adjusts correction parameters based on the color distribution of the image. By analyzing the a and b channels in the LAB color space, it calculates the degree of color shift and applies CLAHE (Contrast Limited Adaptive Histogram Equalization) to enhance the image's contrast.

### 6.5.3 *color constancy*

The color constancy function simulates the color perception capability of the human visual system. It calculates the mean values of each channel in the image and applies color constancy correction, making the image colors more consistent with human visual perception.

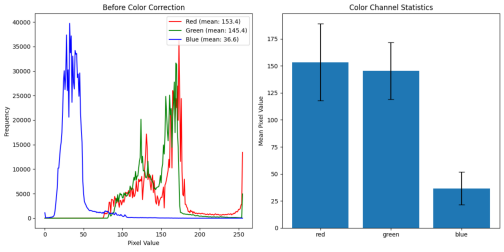
## 6.6 Evaluation of Image Enhancement

To comprehensively assess the effectiveness of our image enhancement algorithm, we conducted a series of quantitative analyses. Specifically, we calculated the following key metrics:

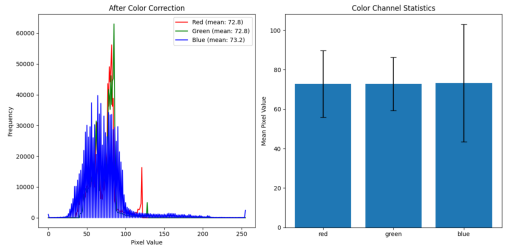
(1) **Saturation Improvement**: By comparing the saturation before and after enhancement, we can quantify the effect of color enhancement. An increase in saturation typically signifies more vivid and lively colors in the image.

(2) **Color Balance**: This metric measures the equilibrium of the distribution of the RGB channels in the enhanced image. A balanced color distribution contributes to a more natural and realistic visual effect.

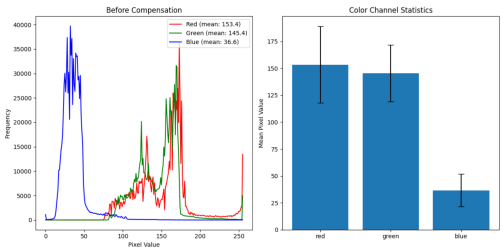
(3) **UCIQE Metric (Universal Image Quality Evaluator)**: This is a comprehensive image quality assessment indicator that takes into account the human visual perception of image quality. An improvement in the UCIQE score indicates that the enhanced image is visually closer to a high-quality image.



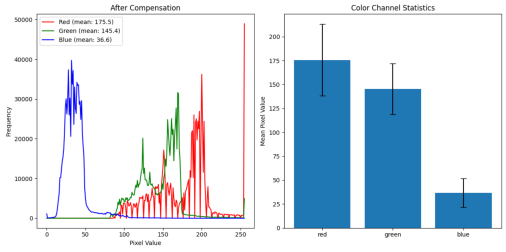

**(a) Before Color Correction**




**(b) After Color Correction**




**(c) Before Compensation**




**(d) After Compensation**

In the comparison before and after color correction, significant improvements are evident. Prior to correction, the average pixel value of the blue channel was 36.6, which was substantially lower than the red channel's 153.4 and the green channel's 145.4, resulting in a cold color bias in the image. After color correction, the average pixel value of the blue channel notably increased to 73.2, aligning more closely with the red channel's 72.8 and the green channel's 72.8, achieving a color balance. Furthermore, the histograms show that the distribution of blue pixel values is more uniform after correction, consistent with the distribution trends of the red and green channels, indicating that the image's colors are more natural and in better accordance with human visual perception of natural colors. These specific numerical changes clearly demonstrate the effectiveness of color correction technology in enhancing image quality.

(a) Before Color Correction

(b) After Color Correction

(a) Before Color Correction

(b) After Color Correction

(a) Before Color Correction

(b) After Color Correction

(a) Before Color Correction

(b) After Color Correction

Through the calculation and analysis of these metrics, we have validated the effectiveness of our image enhancement algorithm in improving the quality of underwater images. These quantitative evaluation results provide objective validation for our algorithm, ensuring that the enhanced images are not only more visually appealing but also significantly enhanced in terms of color accuracy and overall quality.

# VII. Model Establishment and Solution of Problem 4

In this section, convolutional neural networks are primarily utilized to optimize and enhance underwater image processing techniques in complex scenarios.

We propose a network architecture based on a multi-branch structure, which includes three parallel branches: the Color Enhancement Branch, the Illumination Branch, and the Clarity Branch.

The outputs from these three branches are integrated through a feature fusion layer, resulting in the final enhanced image.

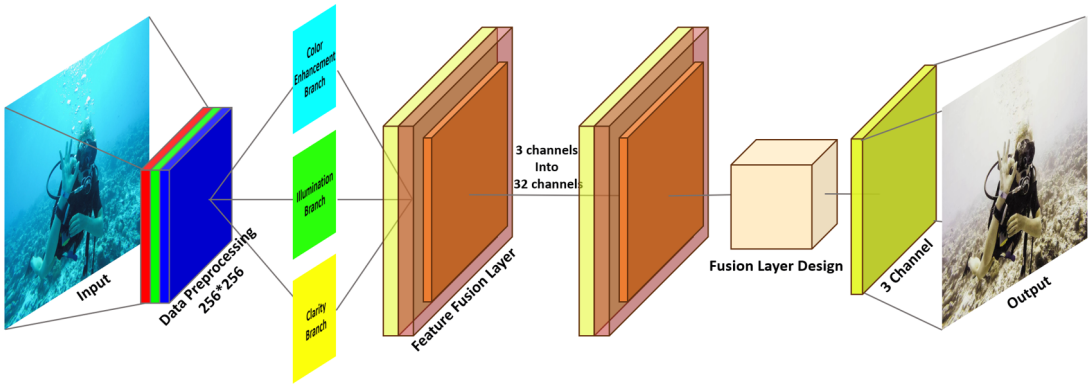



**Figure 17 Deep Neural Network Architecture Diagram**

## 7.1 Data Preprocessing

Representing an image in tensor form:

$$I \in \mathbb{R}^{C \times H \times W}$$

where:

- $C$ is the number of color channels ($C = 3$ for RGB images).
- $H$ and $W$ are the height and width of the image, respectively.

The image is then resized to a uniform size. Data normalization and standardization are performed to facilitate processing.

## 7.2 Construction of Convolutional Neural Network

1 **Feature Fusion**

- **Color Enhancement Branch:**

$$F_{color}(x) = x + H_{color}(x)$$

  - $x$: Input image
  - $H_{color}$: Residual learning function
  - $F_{color}$: Output after color enhancement

This branch employs a residual structure, which offers several key advantages. It enables the network to learn the residual mapping for color correction, effectively addressing the task of enhancing image colors.

Additionally, the use of skip connections mitigates the vanishing gradient problem, facilitating more stable and efficient training. Furthermore, the residual structure preserves the original image information, ensuring that the enhancements do not lead to excessive modifications or distortions of the input image.

- **Illumination Branch:**

$$F_{illum}(x) = x * M(x)$$

$M(x)$: The learned illumination map and it is normalized to the range $[0, 1]$ using the sigmoid function.

The model generates a single-channel illumination map that represents the illumination intensity at each pixel, achieving adaptive illumination adjustment through element-wise multiplication. The use of sigmoid activation ensures smoothness in the illumination adjustment process.

- **Clarity Branch:**

$$F_{clarity}(x) = x + H_{clarity}(x)$$

The structure is similar to the Color Enhancement Branch, utilizing residual connections as well. It focuses on extracting and enhancing the detailed features of the image.

- **Feature Fusion Layer:**

$$F_{fusion}(x) = Conv(Concat[F_{color}(x), F_{illum}(x), F_{clarity}(x), x])$$

2 **Convolution Layer Design:**

- Each branch adopts a similar convolutional structure:
  Input Layer: Transforms 3 channels into 32 channels.
  Intermediate Layer: Maintains 32 channels.
  Output Layer: Adjusts to the corresponding number of channels based on the branch's specific function.
- The parameter settings are designed to optimize performance, with a 32-channel intermediate layer providing sufficient feature extraction capability. The use of a smaller $3 \times 3$ convolution kernel balances computational efficiency and receptive field size, while a consistent padding strategy ($padding = 1$) ensures that the feature map dimensions remain unchanged throughout the process.

3 **Training Strategy**

- **Loss Function**
  The Mean Squared Error (MSE) loss is used:

$$L = ||F(x) - y||^2$$

where $F(x)$ is the network output and $y$ is the target enhanced image.

- **Optimizer Configuration**

The Adam optimizer is employed with a learning rate of 0.001. It is chosen for its adaptive learning rate adjustment and effectiveness in handling high-dimensional, non-convex optimization problems, such as image processing tasks.

- **Data Preprocessing**
  The preprocessing pipeline includes the following steps:
  1. Resizing images to a resolution of $256 \times 256$.
  2. Normalizing pixel values to the $[0, 1]$ range. 3. Converting images to tensor format.

4 **Network Parameter Configuration**

- **Convolution Layer Design:**
  Each branch uses a consistent structure:
  - Input: 3 channels → 32 channels.
  - Intermediate: 32 channels → 32 channels.
  - Output: Adjusted based on branch functionality.
- **Activation Functions**
  ReLU: Used in all layers except the final layer for efficient, non-linear transformations. Sigmoid: Applied only in the illumination branch output to normalize the illumination map to $[0, 1]$, ensuring smooth adjustments.

$$M(x) = Sigmoid(F(x))$$

- **Fusion Layer Design**
  The fusion layer integrates 12 input channels (3 branch outputs + the original input), processes through 32 channels, and outputs 3 channels (RGB).

Compared to traditional CNN methods, one of the innovative aspects of this model is its ability to perform multi-task parallel processing. During the image enhancement process, the model simultaneously addresses color, illumination, and clarity, with each branch independently learning different features.

Additionally, feature fusion is achieved through concatenation and convolution, which preserves the original image information. This method is specifically optimized for underwater image enhancement tasks, resulting in a lightweight and efficient dedicated network structure.

Each branch operates independently, and through parallel processing and feature fusion, the model effectively enhances underwater images.

## 7.3 Analysis of Key Metrics

We evaluated three critical indicators: PSNR (Peak Signal-to-Noise Ratio), UCIQE (Underwater Color Image Quality Evaluation), and UIQM (Underwater Image Quality Measure).

### 7.3.1 *PSNR Analysis*

- **Range:** 9.59 – 40.34 dB
- **Highest:** `test_001.png` (40.34 dB) indicates the best enhancement with minimal distortion.
- **Lowest:** `test_003.png` (9.59 dB) shows significant distortion.

- **Average:** Approximately 19.08 dB, generally low.
- Only three images (`test_001`, `test_008`, `test_009`) exceed 20 dB, suggesting that the CNN enhancement may introduce considerable noise in most images.

#### 7.3.2 *UCIQE Analysis*

- **Range:** 0.47 – 0.56
- **Highest:** `test_007.png` (0.559)
- **Lowest:** `test_008.png` (0.477)
- Values exhibit minor variations, indicating relatively stable color quality.
- Most images have UCIQE values between 0.48 and 0.52, reflecting moderate overall color quality.

#### 7.3.3 *UIQM Analysis*

- **Range:** 3.30 – 62.83
- **Highest:** `test_004.png` (62.83)
- **Lowest:** `test_012.png` (3.30)
- Significant fluctuations indicate large differences in overall image quality.
- Outliers such as `test_004.png` suggest potential over-enhancement.

#### 7.3.4 *Comprehensive Analysis*

- **Inconsistent Quality:** `test_001.png` achieves the highest PSNR but only moderate UCIQE and UIQM scores. Conversely, `test_004.png` has the highest UIQM but a low PSNR, indicating possible over-enhancement.
- **Potential Issues:** Low PSNR across most images implies that the CNN enhancement may introduce excessive noise. While UCIQE remains stable, the wide variance in UIQM suggests the need for model parameter adjustments to ensure consistent quality.

#### 7.3.5 *Improvement Recommendations*

- Optimize the CNN model to enhance overall PSNR values.
- Examine the processing parameters of `test_001.png` for potential application to other images.
- Investigate the causes of UIQM fluctuations and consider adopting different processing strategies for various image types.

#### 7.3.6 *Special Considerations*

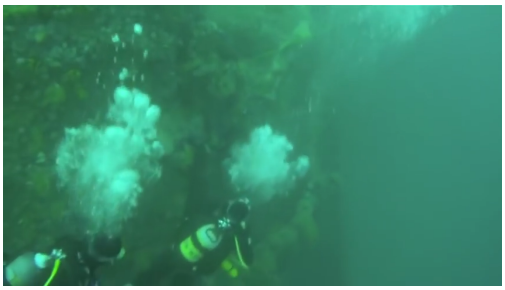
- **`test_001.png`:** Demonstrates the best overall performance and should serve as a reference case.
- **`test_004.png`:** Requires further analysis to understand its exceptionally high UIQM score.
- **`test_012.png`:** Its extremely low UIQM score necessitates a detailed investigation.

**Table 2 The Calculated Evaluation Metric Values After Image Processing**

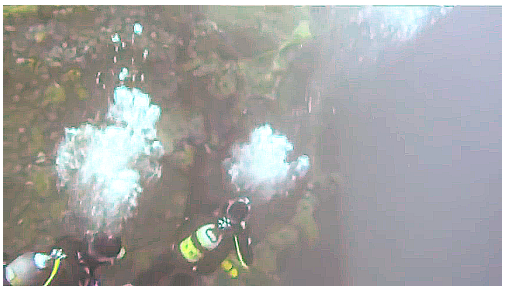
| image file name | PSNR | UCIQE | UIQM |
|---|---|---|---|
| test_001.png | 40.3421 | 0.490058 | 38.10944152 |
| test_002.png | 13.9512 | 0.542959 | 43.3445317 |
| test_003.png | 9.5921 | 0.503575 | 22.71714728 |
| test_004.png | 10.8094 | 0.556342 | 62.83140714 |
| test_005.png | 17.5604 | 0.487713 | 27.78781124 |
| test_006.png | 16.9692 | 0.481681 | 31.38875972 |
| test_007.png | 19.923 | 0.55875 | 44.92898355 |
| test_008.png | 29.9547 | 0.477239 | 14.54101328 |
| test_009.png | 25.6213 | 0.491767 | 44.12333489 |
| test_010.png | 12.4631 | 0.503667 | 14.54812355 |
| test_011.png | 13.1673 | 0.508144 | 29.82360941 |
| test_012.png | 18.7135 | 0.481987 | 3.298234863 |

**(a) Before Deep learning Correction**

**(b) After Deep learning Correction**

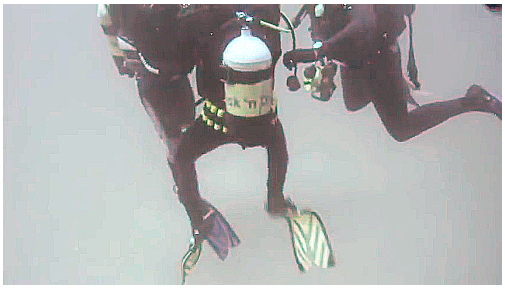


**(a) Before Deep learning Correction**



**(b) After Deep learning Correction**

**(a) Before Deep learning Correction**



**(b) After Deep learning Correction**

**(a) Before Deep learning Correction**

**(b) After Deep learning Correction**

# VIII. Comparison and Solution of Problem 5

Based on the evaluation metrics calculated from the processed images in Questions 3 and 4, we visualized them in the following charts.

## 8.1 Summary of Results Based on Visual Analysis

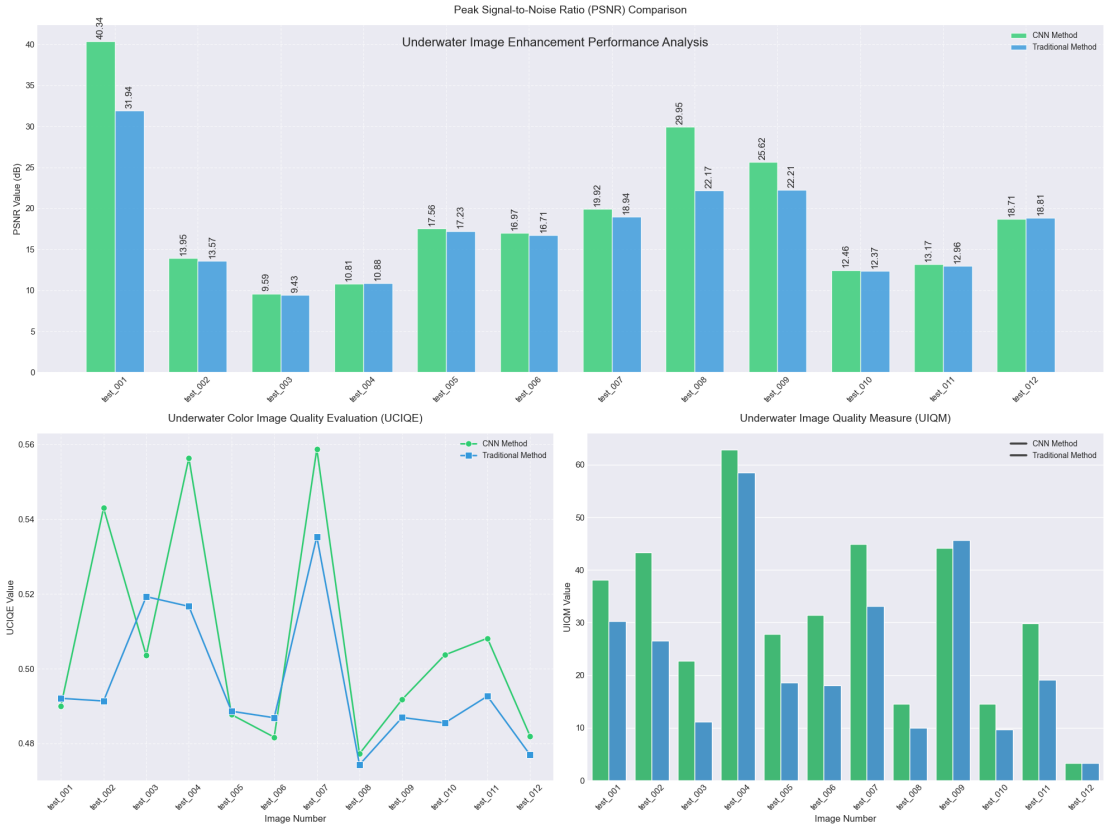



**Figure 22 Comparative Analysis**

### 8.1.1 *PSNR (Peak Signal-to-Noise Ratio) Analysis*

- **Overall Performance**: The CNN method outperforms traditional methods overall, especially for test images 001 and 008.
- **Specific Case**: In image 001, the PSNR value of the CNN method reaches 40.34dB, which is about 8.4dB higher than that of traditional methods.
- **General Level**: Most images have PSNR values between 10-20dB, indicating that there is still significant room for improvement in overall enhancement effects.
- **Specific Scenarios**: In certain images (such as 004, 012), the PSNR difference between the two methods is small, suggesting that traditional methods still perform well in specific scenarios.

### 8.1.2 *UCIQE (Underwater Color Image Quality Evaluation) Analysis*

- **Overall Range**: The UCIQE values for both methods range between 0.47-0.56, with relatively small differences.

- **Outstanding Performance**: The CNN method shows significantly higher UCIQE values in images 002, 004, and 007, indicating better color enhancement capability in these scenarios.
- **Individual Advantage**: In some images (such as 003), the traditional method's UCIQE value is slightly higher than that of the CNN method.
- **Overall Trend**: Overall, the CNN method shows a slight advantage in maintaining color quality.

#### 8.1.3 *UIQM (Underwater Image Quality Measure) Analysis*

- **Comprehensive Advantage**: The CNN method significantly outperforms the traditional method in terms of UIQM values for most images.
- **Key Cases**: Particularly in images 002, 003, 005, 006, 007, and 011, the UIQM values of the CNN method are 30%-50% higher than those of the traditional method.
- **High-Performance Images**: In image 004, both methods achieve high UIQM values, indicating a good overall enhancement effect for this image.
- **Low-Performance Images**: In image 012, both methods have low UIQM values, possibly due to poor quality of the original image or the specific characteristics of the scene.

#### 8.1.4 *Overall Conclusion*

1. **Adaptability and Stability**: The CNN method demonstrates superior adaptability and stability in complex scenes, especially in the PSNR and UIQM metrics.
2. **Color Quality**: In terms of color quality (UCIQE), the difference between the two methods is small, but the CNN method still has a slight edge.
3. **Potential of Traditional Methods**: Despite the superior performance of the CNN method, traditional methods can still achieve good results in specific scenarios.
4. **Possible Reasons**: The advantage of the CNN method may stem from its ability to learn more generalized enhancement strategies from large datasets, whereas the parameters of traditional methods are usually better suited for specific scenarios.

#### 8.1.5 *Recommendations*

1. **Prefer CNN Methods for Complex Scenes**: For complex and variable underwater scenes, it is recommended to prioritize using the CNN method.
2. **Optimize Traditional Methods for Specific Scenarios**: In specific scenarios, optimized traditional methods can be considered to improve processing speed and efficiency.
3. **Develop Hybrid Strategies**: It is recommended to combine the strengths of both methods to develop a hybrid enhancement strategy for better overall results.

### 8.2 Feasibility Suggestions for Underwater Visual Enhancement in Practical Applications

- **Selecting Appropriate Image Enhancement Methods Based on Specific Application Requirements:**

Traditional methods such as histogram equalization and contrast stretching are suitable for scenarios with high real-time performance demands but may have limited effectiveness in complex environments. Deep learning-based methods, such as Generative Adversarial Networks (GANs), excel in handling complex scenarios but often require more computational resources. In practical applications, analyzing and customizing image enhancement strategies according to specific needs can help achieve a better balance between resource efficiency and performance. Additionally, these strategies can be further tailored to improve the adaptability of visual enhancement within the target application context.

- **Integrating Data from Other Sensors:** Combining visual enhancement with data from other sensors, such as sonar or LiDAR, can improve the accuracy and reliability of environmental perception. This multi-sensor fusion approach has been proven effective in underwater target detection and could be further explored and validated in future research.
- **Promoting Interdisciplinary Collaboration:** Advancements in underwater visual enhancement can benefit from interdisciplinary efforts involving optics, computer science, materials science, and other fields. Such collaboration can drive innovation and breakthroughs in the development of underwater visual enhancement technologies.

# IX. Conclusions

## 9.1 Conclusions of the problem

- Successfully extracted and classified image features from underwater scenes, achieving robust identification of color casts, low illumination, and blurring through a rule-driven scoring system.
- Enhanced traditional underwater image degradation models by incorporating absorption and various scattering mechanisms, providing a more comprehensive model for underwater image degradation.
- Achieved effective color correction for single scene underwater images using semi-physical model parameter adjustment methods, validating the effectiveness through multiple evaluation metrics.
- Designed a lightweight CNN model with improved generalization capabilities for underwater image enhancement, proving its efficiency and robustness through experiments.

## 9.2 Methods used in our models

- Rule-driven scoring system for image classification based on extracted features, including color cast, illumination, and sharpness.
- Enhanced Jaffe-McGlamery model by adding detailed absorption and scattering mechanisms, incorporating Rayleigh and Mie scattering with the Henyey-Greenstein phase function.
- Color correction techniques based on semi-physical models, such as white balance adjustment (gray-world hypothesis, perfect reflector), Retinex theory, adaptive color correction, and color constancy.
- Convolutional Neural Network (CNN) architecture designed for extracting multi-layer features and adapting to different underwater conditions through multi-scale strategies and data augmentation.

## 9.3 Applications of our models

- Effective image classification and preprocessing in underwater environments with complex lighting conditions, facilitating further analysis or enhancement.
- Improved underwater imaging systems for scientific exploration and marine biology research by providing a more precise degradation model.
- Enhanced visual quality of underwater images for single-scene analysis, contributing to applications like underwater monitoring and inspection.
- Robust and lightweight underwater image enhancement solutions for practical applications, suitable for underwater robotics, ROVs, and autonomous underwater vehicles.

# X. Future Work

Although the proposed lightweight convolutional neural network has achieved significant results in underwater image enhancement, there remains ample room for further optimization and expansion. Future research can explore the following areas:

1. **Model Optimization:** Further streamline the network architecture to reduce the number of parameters and computational complexity, thereby enhancing the model's efficiency on resource-constrained devices and enabling broader real-time applications.
2. **Diverse Environmental Adaptation:** Extend the model's adaptability to handle more complex and variable underwater environments, including different types of water bodies, pollution levels, and various lighting conditions, thereby improving the model's generalization performance.
3. **Integration with Advanced Technologies:** Investigate the combination of the proposed method with advanced techniques such as Generative Adversarial Networks (GANs), attention mechanisms, or transfer learning to further enhance image enhancement quality and model robustness.
4. **Dataset Expansion:** Develop larger and more diverse underwater image datasets that encompass a wider range of real-world scenarios, thereby enhancing the model's training effectiveness and performance in practical applications.
5. **Application Expansion:** Apply the underwater image enhancement model to a broader range of practical scenarios, such as underwater robot navigation, marine ecological monitoring, and underwater archaeology, to validate its effectiveness and reliability in real-world applications.
6. **Real-Time Performance Improvement:** Research real-time underwater image enhancement techniques to ensure rapid response in dynamic underwater environments, meeting the demands of real-time monitoring and decision-making.

By continuously exploring and innovating in these directions, it is expected that the performance and application scope of underwater image enhancement technology will be further enhanced, providing a more solid technical foundation for the development of related fields.

# XI. References

[1] Khanom, F.Mohamed, N.Lopushenko, I.Sdobnov, A.Doronin, A.Bykov, A.Rafailov, E.Meglinski, I. (2024). Twists through turbidity: propagation of light carrying orbital angular momentum through a complex scattering medium. Scientific Reports, 14, Article 20662. Advance online publication.

[2] McGlamery, B. (1980). A Computer Model For Underwater Camera Systems. In Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series.

[3] Preisendorfer, R. W. (1965). Radiative Transfer on Discrete Spaces. New York: Pergamon Press.

[4] van de Hulst, H. C. (1957). Light Scattering by Small Particles. New York: Wiley.

[5] Strutt, J. W. *[Lord Rayleigh]* (1871). On the scattering of light by small particles. *Philosophical Magazine*, **4**(41), 447–454.

[6] Mie, G. (1908). Beiträge zur Optik trüber Medien, speziell kolloidaler Metallösungen. *Annalen der Physik*, **330**(3), 377–445.

[7] Brillouin, L. (1922). Diffusion de la lumière et des Rayons X par un corps transparent homogène. *Annales de Physique*, 9, 88–122.

[8] Raman, C. V., & Krishnan, K. S. (1928). A new type of secondary radiation. *Nature*, 121, 501–502.

[9] Lambert, J. H. (1760). Photometria sive de mensure et gradibus luminis colorum et umbra. Basel.

[10] Land, E. H. (1971). Color constancy from reciprocal reflectance. Journal of the Optical Society of America, 61(1), 11-20.

[11] Brainard, D. H., & Freeman, W. T. (1997). Bayes color constancy. Journal of the Optical Society of America A, 14(10), 2393-2406.

[12] Land, E. H. (1971). Lightness and retinex theory. Journal of the Optical Society of America, 61(1), 1-11.

[13] Li Z, Liu F, Yang W, et al. A survey of convolutional neural networks: analysis, applications, and prospects[J]. IEEE transactions on neural networks and learning systems, 2021, 33(12): 6999-7019.

# XII. Appendix

## 12.1 Code for Problem 1

```
import cv2
import numpy as np
import os
from pathlib import Path
import matplotlib.pyplot as plt
from scipy.stats import skew
import pandas as pd

def load_and_analyze_image(image_path):
  """Load an image and perform basic analysis"""
  img = cv2.imread(str(image_path))
  if img is None:
      raise ValueError(f"Unable to read image: {image_path}")
  # Convert to RGB color space
  img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  # Convert to HSV color space
  img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
  return img, img_rgb, img_hsv

def analyze_color_cast(img_rgb, img_hsv):
  """Analyze color cast"""
  # Split RGB channels
  r, g, b = cv2.split(img_rgb)
  # Calculate the average of each channel
  avg_r, avg_g, avg_b = np.mean(r), np.mean(g), np.mean(b)
  # Calculate the standard deviation of channel averages
  channel_std = np.std([avg_r, avg_g, avg_b])
  # Split the saturation channel from HSV
  _, s, _ = cv2.split(img_hsv)
  # Calculate the average saturation
  saturation_mean = np.mean(s)
  # Calculate the ratio of dominant color
  dominant_color_ratio = max(avg_r, avg_g, avg_b) / (avg_r + avg_g + avg_b + 1e-6)

  # Normalized score (0-100)
  # Mathematical expression:
  # color_cast_score = ( (channel_std / 127.5) * 60 ) + (dominant_color_ratio * 20 ) +
      ( (saturation_mean / 255) * 20 )
  color_cast_score = (
      (channel_std / 127.5 * 60) + # Contribution of channel standard deviation
      (dominant_color_ratio * 20) + # Contribution of dominant color
      (saturation_mean / 255 * 20) # Contribution of saturation
  )

  return min(100, color_cast_score)
```

```
def analyze_low_light(img_hsv):
  """Analyze insufficient lighting"""
  # Extract the brightness channel
  _, _, v = cv2.split(img_hsv)
  # Calculate the average brightness
  mean_brightness = np.mean(v)
  # Calculate the ratio of dark pixels (pixels with brightness less than 80)
  dark_ratio = np.sum(v < 80) / v.size
  # Calculate the skewness of the brightness distribution
  brightness_skewness = skew(v.ravel())

  # Normalized score (0-100), the higher the score, the more insufficient the lighting
  # Mathematical expression:
  # low_light_score = ( (100 - (mean_brightness / 255) * 100) * 0.5 ) + (dark_ratio *
      100 * 0.3 ) + ( max(0, brightness_skewness) * 10 )
  low_light_score = (
      (100 - mean_brightness / 255 * 100) * 0.5 + # Contribution of average brightness
      (dark_ratio * 100) * 0.3 +             # Contribution of dark pixel ratio
      max(0, brightness_skewness) * 20       # Contribution of skewness
  )

  return min(100, low_light_score)

def analyze_blur(img):
  """Analyze blurriness"""
  # Convert to grayscale
  img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  # Calculate the variance of the Laplacian operator
  laplacian_var = cv2.Laplacian(img_gray, cv2.CV_64F).var()
  # Calculate the mean of the Sobel gradient
  sobelx = cv2.Sobel(img_gray, cv2.CV_64F, 1, 0, ksize=3)
  sobely = cv2.Sobel(img_gray, cv2.CV_64F, 0, 1, ksize=3)
  gradient_magnitude = np.sqrt(sobelx**2 + sobely**2)
  gradient_mean = np.mean(gradient_magnitude)

  # Normalized score (0-100), the higher the score, the blurrier the image
  # Mathematical expression:
  # blur_score = 100 - min(100, ( (laplacian_var / 500) * 50 ) + ( (gradient_mean /
      100) * 50 ) )
  blur_score = 100 - min(100, (
      (laplacian_var / 500) * 50 +   # Contribution of Laplacian variance
      (gradient_mean / 100) * 50     # Contribution of gradient mean
  ))

  return blur_score

def classify_single_image(image_path):
  """Classify a single image"""
```

```
    try:
        # Load the image
        img, img_rgb, img_hsv = load_and_analyze_image(image_path)

        # Calculate the scores for the three indicators
        color_cast_score = analyze_color_cast(img_rgb, img_hsv)
        low_light_score = analyze_low_light(img_hsv)
        blur_score = analyze_blur(img)

        # Get the category with the highest score
        scores = {
            'color_cast': color_cast_score,
            'low_light': low_light_score,
            'blur': blur_score
        }

        # Find the category with the highest score
        # Mathematical expression:
        # category = argmax_{k  {'color_cast', 'low_light', 'blur'}} scores[k]
        max_category = max(scores.items(), key=lambda x: x[1])

        return {
            'filename': Path(image_path).name,
            'category': max_category[0],
            'color_cast_score': color_cast_score,
            'low_light_score': low_light_score,
            'blur_score': blur_score
        }
    except Exception as e:
        print(f"Error processing image {image_path}: {e}")
        return None

def process_dataset(image_folder):
    """Process the entire dataset"""
    results = []
    categories = {'color_cast': [], 'low_light': [], 'blur': []}

    # Get all image files
    image_files = list(Path(image_folder).glob('*.jpg')) +
         list(Path(image_folder).glob('*.png'))
    total_files = len(image_files)
    print(f"Found {total_files} image files")

    # Process each image
    for i, image_path in enumerate(image_files, 1):
        if i % 10 == 0: # Show progress every 10 images
            print(f"Processing progress: {i}/{total_files}")

        result = classify_single_image(image_path)
```

```
        if result:
            results.append(result)
            categories[result['category']].append(result['filename'])

    # Create DataFrame
    df = pd.DataFrame(results)

    # Print category statistics
    print("\nCategory statistics:")
    print(df['category'].value_counts())

    return df, categories

def save_results(df, categories, output_folder):
    """Save classification results"""
    os.makedirs(output_folder, exist_ok=True)

    # Save detailed results to Excel
    df.to_excel(f"{output_folder}/classification_results.xlsx", index=False)

    # Save the list of images for each category
    for category, files in categories.items():
        with open(f"{output_folder}/{category}_images.txt", 'w') as f:
            f.write('\n'.join(files))

    # Create distribution plots
    plt.figure(figsize=(15, 5))

    plt.subplot(131)
    df[df['category'] == 'color_cast']['color_cast_score'].hist(bins=20)
    plt.title('Color Cast Score Distribution')

    plt.subplot(132)
    df[df['category'] == 'low_light']['low_light_score'].hist(bins=20)
    plt.title('Low Light Score Distribution')

    plt.subplot(133)
    df[df['category'] == 'blur']['blur_score'].hist(bins=20)
    plt.title('Blur Score Distribution')

    plt.tight_layout()
    plt.savefig(f'{output_folder}/score_distributions.png')
    plt.close()

def main():
    # Set paths
    image_folder = 'APMCM Problem A\\Attachment\\Attachment 1' # Replace with actual path
    output_folder = 'APMCM Problem A\\Attachment\\Attachment 1' # Replace with actual path
```

```
# Process the dataset
print("Starting to process images...")
df, categories = process_dataset(image_folder)

# Save results
print("Saving results...")
save_results(df, categories, output_folder)

# Print some examples of images from each category
print("\nSome examples of images from each category:")
for category, files in categories.items():
    print(f"\n{category} category (total {len(files)} images):")
    print('\n'.join(files[:5])) # Only show the first 5 examples

if name == "main":
    main()
```

## 12.2 Code for Problem 3 (assumption)

```
from PIL import Image
import numpy as np
import os
import matplotlib.pyplot as plt

directory = os.getcwd()
Num = 400
# Initialize RGB distribution list to store the RGB distribution of each image
rgb_distributions = {'R': [], 'G': [], 'B': []}

# Loop through images numbered from 1 to 400
for i in range(1, Num+1):
    # Construct filename, try both jpg and png formats
    filename_jpg = f"image_{i:03}.jpg"
    filename_png = f"image_{i:03}.png"
    file_path_jpg = os.path.join(directory, filename_jpg)
    file_path_png = os.path.join(directory, filename_png)

    # Check if file exists
    if os.path.exists(file_path_jpg):
        file_path = file_path_jpg
    elif os.path.exists(file_path_png):
        file_path = file_path_png
    else:
        print(f"File {filename_jpg} or {filename_png} does not exist.")
        continue

    # Open the image
    image = Image.open(file_path).convert('RGB')

    # Convert the image to a numpy array
    pixels = np.array(image)

    # Get the image dimensions
    width, height = image.size

    # Initialize the RGB distribution dictionary
    rgb_distribution = {'R': 0, 'G': 0, 'B': 0}

    # Loop through each pixel
    for j in range(width):
        for k in range(height):
            r, g, b = pixels[k][j] # Get RGB values
            rgb_distribution['R'] += r
            rgb_distribution['G'] += g
            rgb_distribution['B'] += b
```

```
    # Calculate the average values
    total_pixels = width * height
    rgb_distribution['R'] /= total_pixels
    rgb_distribution['G'] /= total_pixels
    rgb_distribution['B'] /= total_pixels
    total_rgb = rgb_distribution['R']+rgb_distribution['G']+rgb_distribution['B']

    # Add the results to the list
    rgb_distributions['R'].append(rgb_distribution['R']/total_rgb)
    rgb_distributions['G'].append(rgb_distribution['G']/total_rgb)
    rgb_distributions['B'].append(rgb_distribution['B']/total_rgb)

    print(f"image_{i:03}", rgb_distribution)
# Calculate the average of RGB distributions
average_red = sum(rgb_distributions['R']) / len(rgb_distributions['R'])
average_green = sum(rgb_distributions['G']) / len(rgb_distributions['G'])
average_blue = sum(rgb_distributions['B']) / len(rgb_distributions['B'])

# Plot a bar chart
plt.figure(figsize=(10, 6))

# Plot the red channel bar chart
plt.subplot(3, 1, 1)
plt.bar(range(1, Num + 1), rgb_distributions['R'], color='red')
plt.title('Red Channel Distribution')
plt.xlabel('Image')
plt.ylabel('Red Value')
plt.axhline(y=average_red, color='orange', linestyle='--') # Add a horizontal dashed
    line
plt.text(Num + 1, average_red, f'{average_red:.2f}', va='center', ha='left',
    color='orange', fontweight='bold') # Add a label for the average red value

# Plot the green channel bar chart
plt.subplot(3, 1, 2)
plt.bar(range(1, Num + 1), rgb_distributions['G'], color='green')
plt.title('Green Channel Distribution')
plt.xlabel('Image')
plt.ylabel('Green Value')
plt.axhline(y=average_green, color='orange', linestyle='--') # Add a horizontal dashed
    line
plt.text(Num + 1, average_green, f'{average_green:.2f}', va='center', ha='left',
    color='orange', fontweight='bold') # Add a label for the average green value

# Plot the blue channel bar chart
plt.subplot(3, 1, 3)
plt.bar(range(1, Num + 1), rgb_distributions['B'], color='blue')
plt.title('Blue Channel Distribution')
plt.xlabel('Image')
plt.ylabel('Blue Value')
```

```
plt.axhline(y=average_blue, color='orange', linestyle='--') # Add a horizontal dashed
    line
plt.text(Num + 1, average_blue, f'{average_blue:.2f}', va='center', ha='left',
    color='orange', fontweight='bold') # Add a label for the average blue value

# Adjust subplot spacing
plt.tight_layout()

# Display the chart
plt.show()
```

## 12.3 Code for Problem 3 (Color Correction)

```
import cv2
import numpy as np
from skimage import color
from scipy import ndimage
import os
import pandas as pd
from pathlib import Path
import logging
import json
from datetime import datetime

class ColorCorrectionEnhancer:
    """Color Correction Enhancer Class"""
    def __init__(self):
        self.metrics = {}

    def white_balance_grayworld(self, img):
        """
        White balance using the gray world assumption
        Based on the assumption that the average values of the RGB channels in natural
            images should be similar
        """
        B, G, R = cv2.split(img.astype('float32'))

        # Calculate the average value of each channel
        B_avg = np.mean(B)
        G_avg = np.mean(G)
        R_avg = np.mean(R)

        # Calculate the average value of all channels
        avg = (B_avg + G_avg + R_avg) / 3

        # Calculate the gain factor for each channel
        B_gain = avg / B_avg if B_avg != 0 else 1
        G_gain = avg / G_avg if G_avg != 0 else 1
        R_gain = avg / R_avg if R_avg != 0 else 1

        # Apply gain
        B = np.clip(B * B_gain, 0, 255)
        G = np.clip(G * G_gain, 0, 255)
        R = np.clip(R * R_gain, 0, 255)

        return cv2.merge([B.astype('uint8'), G.astype('uint8'), R.astype('uint8')])

    def white_balance_perfect_reflector(self, img):
        """
        Perfect reflector white balance
```

```
    Based on the assumption that the brightest point in the image should be white
    """
    B, G, R = cv2.split(img.astype('float32'))

    # Get the top 1% of the brightest pixels in each channel
    percentile = 99
    B_max = np.percentile(B, percentile)
    G_max = np.percentile(G, percentile)
    R_max = np.percentile(R, percentile)

    # Calculate the gain factor (assuming the brightest point should be 255)
    B_gain = 255.0 / B_max if B_max != 0 else 1
    G_gain = 255.0 / G_max if G_max != 0 else 1
    R_gain = 255.0 / R_max if R_max != 0 else 1

    # Apply gain
    B = np.clip(B * B_gain, 0, 255)
    G = np.clip(G * G_gain, 0, 255)
    R = np.clip(R * R_gain, 0, 255)

    return cv2.merge([B.astype('uint8'), G.astype('uint8'), R.astype('uint8')])

# [Keep other methods unchanged...]
def color_balance_retinex(self, img):
    """
    Color balance based on Retinex theory
    Considers the adaptive characteristics of the human visual system
    """
    log_img = np.log1p(np.array(img, dtype="float") / 255.0)

    for ch in range(3):
        blur_radius = 5
        gaussian = cv2.GaussianBlur(log_img[:,:,ch], (0, 0), blur_radius)
        log_img[:,:,ch] = cv2.subtract(log_img[:,:,ch], gaussian)

    result = np.expm1(log_img) * 255.0
    return np.uint8(np.clip(result, 0, 255))

def underwater_color_correction(self, img):
    """
    Underwater environment-specific color correction
    Combines physical models and image statistical characteristics
    """
    # 1. Estimate underwater lighting
    dark_channel = np.min(img, axis=2)
    A = np.percentile(dark_channel, 99)

    # 2. Estimate transmission map
    omega = 0.95
```

```
        transmission = 1 - omega * dark_channel / A

        # 3. Refine transmission map
        transmission = cv2.GaussianBlur(transmission, (0,0), 3)

        # 4. Recover image
        result = np.empty_like(img, dtype=np.float64)
        for ch in range(3):
            result[:,:,ch] = (img[:,:,ch].astype(np.float64) - A) / \
                             np.maximum(transmission, 0.1) + A

        return np.uint8(np.clip(result, 0, 255))

    def adaptive_color_correction(self, img):
        """
        Adaptive color correction
        Automatically adjusts parameters based on image features
        """
        # 1. Analyze image color distribution
        lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
        l, a, b = cv2.split(lab)

        # 2. Calculate the degree of color offset
        a_mean = np.mean(a)
        b_mean = np.mean(b)
        color_shift = np.sqrt(a_mean**2 + b_mean**2)

        # 3. Adaptive parameter adjustment
        correction_strength = np.clip(color_shift / 128.0, 0.5, 1.5)

        # 4. Apply CLAHE
        clahe = cv2.createCLAHE(clipLimit=correction_strength*3.0,
                                tileGridSize=(8,8))
        l = clahe.apply(l)

        # 5. Color correction
        a = cv2.subtract(a, np.uint8((a_mean - 128) * correction_strength))
        b = cv2.subtract(b, np.uint8((b_mean - 128) * correction_strength))

        # 6. Merge channels
        result = cv2.merge([l, a, b])
        return cv2.cvtColor(result, cv2.COLOR_LAB2BGR)

    def color_constancy(self, img):
        """
        Color constancy correction
        Simulates the color perception ability of the human visual system
        """
        result = np.zeros_like(img, dtype=np.float32)
```

```
        # Calculate the mean of each channel
        b_avg, g_avg, r_avg = np.mean(img, axis=(0,1))

        # Calculate adjustment coefficients
        k = np.sqrt((b_avg**2 + g_avg**2 + r_avg**2) / 3.0)

        # Apply correction
        result[:,:,0] = np.minimum(img[:,:,0] * (k / b_avg), 255)
        result[:,:,1] = np.minimum(img[:,:,1] * (k / g_avg), 255)
        result[:,:,2] = np.minimum(img[:,:,2] * (k / r_avg), 255)

        return np.uint8(result)

    def evaluate_correction(self, original, corrected):
        """
        Evaluate color correction effect
        """
        # Calculate color saturation change
        hsv_orig = cv2.cvtColor(original, cv2.COLOR_BGR2HSV)
        hsv_corr = cv2.cvtColor(corrected, cv2.COLOR_BGR2HSV)
        sat_improvement = np.mean(hsv_corr[:,:,1]) - np.mean(hsv_orig[:,:,1])

        # Calculate color balance
        rgb_std_orig = np.std(cv2.mean(original)[:-1])
        rgb_std_corr = np.std(cv2.mean(corrected)[:-1])
        color_balance = rgb_std_orig - rgb_std_corr

        # Calculate UCIQE metric
        hsv = cv2.cvtColor(corrected, cv2.COLOR_BGR2HSV)
        chroma = np.std(hsv[:,:,1])
        saturation = np.mean(hsv[:,:,1])
        contrast = np.std(hsv[:,:,2])
        uciqe = 0.4680 * chroma + 0.2745 * contrast + 0.2576 * saturation

        return {
            'saturation_improvement': sat_improvement,
            'color_balance_improvement': color_balance,
            'UCIQE': uciqe
        }

class ColorCorrectionSystem:
    """Color Correction System Class"""
    def __init__(self, input_folder, output_folder, result_file=None):
        """
        Initialize the color correction system

        Args:
            input_folder: Input image folder path
```

```
        output_folder: Output image folder path
        result_file: Result recording file path (optional)
    """
    self.input_folder = Path('C:/Users/Wang/Desktop/APMCM/Attachment 2')
    self.output_folder = Path('C:/Users/Wang/Desktop/APMCM/Attachment 2 enhanced2')
    self.result_file = Path(result_file) if result_file else None
    self.results = []
    self.setup_logging()

    # Ensure the output directory exists
    self.output_folder.mkdir(parents=True, exist_ok=True)

    # Create an enhancer instance
    self.enhancer = ColorCorrectionEnhancer()

def setup_logging(self):
    """Configure the logging system"""
    log_folder = self.output_folder / 'logs'
    log_folder.mkdir(parents=True, exist_ok=True)

    logging.basicConfig(
        level=logging.INFO,
        format='%(asctime)s - %(levelname)s - %(message)s',
        handlers=[
            logging.FileHandler(log_folder /
                f'color_correction_{datetime.now().strftime("%Y%m%d_%H%M%S")}.log'),
            logging.StreamHandler()
        ]
    )

def process_single_image(self, image_path, correction_methods=None):
    """
    Process a single image

    Args:
        image_path: Image file path
        correction_methods: List of correction methods to use, defaults to all
            methods if not specified
    """
    if correction_methods is None:
        correction_methods = [
            'white_balance_grayworld',
            'white_balance_perfect_reflector',
            'color_balance_retinex',
            'underwater_color_correction',
            'adaptive_color_correction',
            'color_constancy'
        ]
```

```
    logging.info(f"Processing image: {image_path}")

    # Read the original image
    original = cv2.imread(str(image_path))
    if original is None:
        logging.error(f"Failed to load image: {image_path}")
        return None

    results = {}
    for method in correction_methods:
        try:
            # Get the correction method
            correction_func = getattr(self.enhancer, method)

            # Apply color correction
            corrected = correction_func(original.copy())

            # Evaluate the result
            metrics = self.enhancer.evaluate_correction(original, corrected)

            # Save the result
            output_path = self.output_folder /
                f"{image_path.stem}_{method}{image_path.suffix}"
            cv2.imwrite(str(output_path), corrected)

            results[method] = {
                'metrics': metrics,
                'output_path': str(output_path)
            }

            logging.info(f"Successfully applied {method} to {image_path.name}")

        except Exception as e:
            logging.error(f"Error processing {image_path} with {method}: {str(e)}")
            results[method] = {'error': str(e)}

    return results

def batch_process(self, file_pattern="*.png"):
    """
    Batch process images

    Args:
        file_pattern: File matching pattern, defaults to processing all PNG images
    """
    image_files = list(self.input_folder.glob(file_pattern))
    logging.info(f"Found {len(image_files)} images to process")

    all_results = []
```

```
        for image_path in image_files:
            results = self.process_single_image(image_path)
            if results:
                all_results.append({
                    'image_name': image_path.name,
                    'results': results
                })

        # Save processing results
        self.save_results(all_results)

        return all_results

    def save_results(self, results):
        """
        Save processing results

        Args:
            results: List of processing results
        """
        # Save detailed JSON results
        json_path = self.output_folder / 'detailed_results.json'
        with open(json_path, 'w', encoding='utf-8') as f:
            json.dump(results, f, indent=4)

        # If a result file is specified, create a summary in Excel format
        if self.result_file:
            df_rows = []
            for item in results:
                for method, result in item['results'].items():
                    if 'metrics' in result:
                        row = {
                            'Image': item['image_name'],
                            'Method': method,
                            'UCIQE': result['metrics']['UCIQE'],
                            'Saturation_Improvement':
                                result['metrics']['saturation_improvement'],
                            'Color_Balance_Improvement':
                                result['metrics']['color_balance_improvement'],
                            'Output_Path': result['output_path']
                        }
                        df_rows.append(row)

            df = pd.DataFrame(df_rows)
            df.to_excel(self.result_file, index=False)
            logging.info(f"Results saved to {self.result_file}")

def main():
    """Main function example"""
```

```
    # Set paths
    input_folder = "input_images" # Modify to your input folder path
    output_folder = "output_images" # Modify to your output folder path
    result_file = "color_correction_results.xlsx" # Modify to your desired result file
        name

    # Create the processing system
    system = ColorCorrectionSystem(
        input_folder=input_folder,
        output_folder=output_folder,
        result_file=result_file
    )

    # Batch process images
    results = system.batch_process()

    print(f"Processing complete. Results saved in {output_folder}")

if __name__ == "__main__":
    main()
```

## 12.4 Code for Problem 4 (Train)

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image
import os

# Define dataset class
class ColorCorrectionDataset(Dataset):
    def __init__(self, input_dir, target_dir, transform=None):
        self.input_images = sorted(os.listdir(input_dir))
        self.target_images = sorted(os.listdir(target_dir))
        self.input_dir = input_dir
        self.target_dir = target_dir
        self.transform = transform

    def __len__(self):
        return len(self.input_images)

    def __getitem__(self, idx):
        try:
            input_path = os.path.join(self.input_dir, self.input_images[idx])
            target_path = os.path.join(self.target_dir, self.target_images[idx])
            input_image = Image.open(input_path).convert('RGB')
            target_image = Image.open(target_path).convert('RGB')
            if self.transform:
                input_image = self.transform(input_image)
                target_image = self.transform(target_image)
            return input_image, target_image
        except Exception as e:
            print(f"Error loading image {input_path} or {target_path}: {e}")
            raise e

# Define a simplified U-Net model
class UNetSmall(nn.Module):
    def __init__(self, in_channels=3, out_channels=3):
        super(UNetSmall, self).__init__()

        def conv_block(in_channels, out_channels, use_batchnorm=True):
            layers = [
                nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
                nn.ReLU(inplace=True),
                nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
                nn.ReLU(inplace=True),
            ]
            if use_batchnorm:
                layers.insert(1, nn.BatchNorm2d(out_channels))
```

```
            layers.insert(-1, nn.BatchNorm2d(out_channels))
        return nn.Sequential(*layers)

    self.enc1 = conv_block(in_channels, 32)
    self.pool1 = nn.MaxPool2d(2)
    self.enc2 = conv_block(32, 64)
    self.pool2 = nn.MaxPool2d(2)
    self.enc3 = conv_block(64, 128)
    self.pool3 = nn.MaxPool2d(2)

    self.bottleneck = conv_block(128, 256)

    self.upconv3 = nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2)
    self.dec3 = conv_block(256, 128)
    self.upconv2 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2)
    self.dec2 = conv_block(128, 64)
    self.upconv1 = nn.ConvTranspose2d(64, 32, kernel_size=2, stride=2)
    self.dec1 = conv_block(64, 32)

    self.final_conv = nn.Conv2d(32, out_channels, kernel_size=1)

def forward(self, x):
    # Encoder
    enc1 = self.enc1(x)
    enc2 = self.enc2(self.pool1(enc1))
    enc3 = self.enc3(self.pool2(enc2))

    # Bottleneck
    bottleneck = self.bottleneck(self.pool3(enc3))

    # Decoder
    dec3 = self.upconv3(bottleneck)
    dec3 = torch.cat((dec3, enc3), dim=1)
    dec3 = self.dec3(dec3)

    dec2 = self.upconv2(dec3)
    dec2 = torch.cat((dec2, enc2), dim=1)
    dec2 = self.dec2(dec2)

    dec1 = self.upconv1(dec2)
    dec1 = torch.cat((dec1, enc1), dim=1)
    dec1 = self.dec1(dec1)

    return self.final_conv(dec1)

# Define training function
def train_model(model, dataloader, criterion, optimizer, num_epochs):
    model.train()
    try:
```

```
    for epoch in range(num_epochs):
        epoch_loss = 0
        for inputs, targets in dataloader:
            inputs = inputs.to(device, non_blocking=True)
            targets = targets.to(device, non_blocking=True)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
        avg_loss = epoch_loss / len(dataloader)
        print(f'Epoch {epoch+1}/{num_epochs}, Loss: {avg_loss:.4f}')
        # Save model after each epoch
        checkpoint_path = f'model_epoch_{epoch+1}.pth'
        torch.save(model.state_dict(), checkpoint_path)
        print(f"Model saved to {checkpoint_path}")
except KeyboardInterrupt:
    print("Training interrupted manually, saving model...")
finally:
    torch.save(model.state_dict(), 'color_correction_model.pth')
    print("Model saved.")
return model
```

## 12.5 Code for Problem 4 (Predict)

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image
import os
from tqdm import tqdm

# Load the trained model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = UNetSmall().to(device)
model.load_state_dict(torch.load('model_epoch_10.pth'))
model.eval()

# Define the dataset class for new images
class NewImagesDataset(Dataset):
    def __init__(self, image_dir, transform=None):
        self.image_files = sorted(os.listdir(image_dir))
        self.image_dir = image_dir
        self.transform = transform

    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, idx):
        try:
            image_name = self.image_files[idx]
            image_path = os.path.join(self.image_dir, image_name)
            image = Image.open(image_path).convert('RGB')
            if self.transform:
                input_image = self.transform(image)
            else:
                input_image = transforms.ToTensor()(image)
            return input_image, image_name
        except Exception as e:
            print(f"Error loading image {image_path}: {e}")
            raise e

# Helper function to adjust dimensions to be multiples of 8
def adjust_size(value, multiple=8):
    if value % multiple == 0:
        return value
    return (value // multiple + 1) * multiple

# Function to prepare image size
def adjust_image_size(image, target_width, target_height):
    # Get original size
```

```
    width, height = image.size
    # Adjust dimensions to be multiples of 8
    target_width = adjust_size(target_width)
    target_height = adjust_size(target_height)
    # Resize image while keeping aspect ratio, then pad/crop to the target size
    image = transforms.Resize((target_height, target_width))(image)
    return image, target_width, target_height

# Set up the dataset and dataloader
new_image_dir = 'C:/Users/Desktop/APMCM/Attachment 2'
output_dir = 'C:/Users/Desktop/APMCM/Attachment 2 new'
os.makedirs(output_dir, exist_ok=True)

# Perform inference and save images
with torch.no_grad():
    for image_name in tqdm(os.listdir(new_image_dir)):
        image_path = os.path.join(new_image_dir, image_name)
        try:
            # Load and adjust the image
            original_image = Image.open(image_path).convert('RGB')
            adjusted_image, adjusted_width, adjusted_height = adjust_image_size(
                original_image, original_image.width, original_image.height
            )

            # Transform the adjusted image to tensor
            transform = transforms.Compose([
                transforms.ToTensor()
            ])
            input_tensor = transform(adjusted_image).unsqueeze(0).to(device)

            # Forward pass
            output_tensor = model(input_tensor)

            # Resize the output to the adjusted size
            output_tensor = torch.nn.functional.interpolate(
                output_tensor, size=(adjusted_height, adjusted_width), mode='bilinear',
                    align_corners=False
            )

            # Convert tensor to image
            output_image = output_tensor.squeeze(0).cpu().clamp(0, 1)
            output_pil = transforms.ToPILImage()(output_image)

            # Save the enhanced image
            output_pil.save(os.path.join(output_dir, image_name))

        except Exception as e:
            print(f"Error processing image {image_path}: {e}")
```

## 12.6 Code for different metric calculations

```
import os
import cv2
import numpy as np
import pandas as pd

def calculate_psnr(img1, img2):
    """
    Calculate PSNR - Peak Signal to Noise Ratio
    PSNR = 20 * log10(MAX_I) - 10 * log10(MSE)
    """
    img1 = img1.astype(np.float64)
    img2 = img2.astype(np.float64)
    mse = np.mean((img1 - img2) ** 2)
    if mse == 0:
       return float('inf')
    max_pixel = 255.0
    psnr = 20 * np.log10(max_pixel / np.sqrt(mse))
    return psnr

def calculate_uciqe(img):
    """
    Calculate UCIQE (Underwater Color Image Quality Evaluation)
    UCIQE = c1 *  c + c2 * conl + c3 *  s
     c: Chroma standard deviation
    conl: Brightness contrast
     s: Mean saturation
    """
    # Convert to LAB color space
    lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
    l_chan, a_chan, b_chan = cv2.split(lab)

    # Calculate chroma
    chroma = np.sqrt(np.square(a_chan.astype(float)) + np.square(b_chan.astype(float)))

    # Calculate mean saturation  s
    sat_mean = np.mean(chroma)

    # Calculate chroma standard deviation  c
    chroma_std = np.std(chroma)

    # Calculate brightness contrast conl
    l_chan = l_chan.astype(float)
    l_max = np.max(l_chan)
    l_min = np.min(l_chan)
    conl = (l_max - l_min) / (l_max + l_min + 1e-4)

    # UCIQE coefficients
```

```
    c1 = 0.4680 # Weight of chroma standard deviation
    c2 = 0.2745 # Weight of brightness contrast
    c3 = 0.2576 # Weight of mean saturation

    uciqe = c1 * chroma_std + c2 * conl + c3 * sat_mean

    return uciqe

def calculate_uiqm(img):
    """
    Calculate UIQM (Underwater Image Quality Measure)
    UIQM = c1 * UICM + c2 * UISM + c3 * UIConM
    """
    def calculate_uicm(img):
        """Calculate UICM (Underwater Image Colorfulness Measure)"""
        # Normalize to [0,1]
        img_norm = img.astype(np.float32) / 255.0
        b, g, r = cv2.split(img_norm)

        # Calculate RG and YB contrast
        rg = r - g
        yb = (r + g) / 2 - b

        # Calculate mean and standard deviation
        rg_mean = np.mean(rg)
        yb_mean = np.mean(yb)
        rg_std = np.std(rg)
        yb_std = np.std(yb)

        # Calculate UICM
        uicm = -0.0268 * np.sqrt(np.abs(rg_mean**2 + yb_mean**2)) + \
               0.1586 * np.sqrt(rg_std**2 + yb_std**2)
        return uicm

    def calculate_uism(img):
        """Calculate UISM (Underwater Image Sharpness Measure)"""
        # Sobel edge detection
        def sobel_edge(channel):
            dx = cv2.Sobel(channel, cv2.CV_64F, 1, 0, ksize=3)
            dy = cv2.Sobel(channel, cv2.CV_64F, 0, 1, ksize=3)
            magnitude = np.sqrt(dx**2 + dy**2)
            return np.mean(magnitude)

        img_norm = img.astype(np.float32) / 255.0
        b, g, r = cv2.split(img_norm)

        # Calculate sharpness for each channel and take the average
        uism = (sobel_edge(r) + sobel_edge(g) + sobel_edge(b)) / 3.0
        return uism
```

```
    def calculate_uiconm(img):
        """Calculate UIConM (Underwater Image Contrast Measure)"""
        # Convert to Lab space and extract the lightness channel
        lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
        l_chan = lab[:,:,0].astype(np.float32)

        # Calculate local contrast
        window_size = 8
        h, w = l_chan.shape
        uiconm = 0

        for i in range(0, h-window_size+1, window_size):
            for j in range(0, w-window_size+1, window_size):
                window = l_chan[i:i+window_size, j:j+window_size]
                window_mean = np.mean(window)
                if window_mean > 0:
                    local_contrast = np.std(window) / window_mean
                    uiconm += local_contrast

        # Normalize
        num_windows = ((h // window_size) * (w // window_size))
        if num_windows > 0:
            uiconm /= num_windows

        return uiconm

    # Calculate the three components
    uicm = calculate_uicm(img)
    uism = calculate_uism(img)
    uiconm = calculate_uiconm(img)

    # UIQM weights
    c1 = 0.0282 # Weight of UICM
    c2 = 0.2953 # Weight of UISM
    c3 = 0.6765 # Weight of UIConM

    # Calculate UIQM
    uiqm = c1 * uicm + c2 * uism + c3 * uiconm
    return uiqm

def evaluate_images(folder1_path, folder2_path, output_excel):
    """Evaluate image quality and output to Excel"""
    results = []
    folder2_files = sorted(os.listdir(folder2_path))

    for img2_name in folder2_files:
        if not img2_name.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp')):
            continue
```

```
    img1_path = os.path.join(folder1_path, img2_name)
    img2_path = os.path.join(folder2_path, img2_name)

    if not os.path.exists(img1_path):
        print(f"Warning: Original image not found: {img2_name}")
        continue

    img1 = cv2.imread(img1_path)
    img2 = cv2.imread(img2_path)

    if img1 is None or img2 is None:
        print(f"Warning: Unable to read image: {img2_name}")
        continue

    if img1.shape != img2.shape:
        print(f"Warning: Image sizes do not match: {img2_name}")
        continue

    # Calculate metrics
    try:
        psnr = calculate_psnr(img1, img2)
        uciqe = calculate_uciqe(img2)
        uiqm = calculate_uiqm(img2)

        results.append({
            'Image_Name': img2_name,
            'PSNR': psnr,
            'UCIQE': uciqe,
            'UIQM': uiqm
        })

        print(f"Processed: {img2_name}")
        print(f"PSNR: {psnr:.4f}")
        print(f"UCIQE: {uciqe:.4f}")
        print(f"UIQM: {uiqm:.4f}")
        print("-" * 40)

    except Exception as e:
        print(f"Error processing image {img2_name}: {str(e)}")
        continue

if not results:
    raise ValueError("No images processed successfully")

# Create DataFrame and add averages
df = pd.DataFrame(results)
averages = pd.DataFrame([{
    'Image_Name': 'Average',
```

```
        'PSNR': df['PSNR'].mean(),
        'UCIQE': df['UCIQE'].mean(),
        'UIQM': df['UIQM'].mean()
    }])

    df = pd.concat([df, averages], ignore_index=True)

    # Save to Excel
    df.to_excel(output_excel, index=False, float_format='%.4f')
    print(f"\nResults saved to: {output_excel}")

    return df

if __name__ == "__main__":
    folder1_path = "C:/Users/Wang/Desktop/APMCM/Attachment 2" # Original images folder
    folder2_path = "C:/Users/Wang/Desktop/APMCM/Attachment 2 enhanced" # Enhanced
        images folder
    output_excel = "image_quality_metrics.xlsx"

    try:
        results_df = evaluate_images(folder1_path, folder2_path, output_excel)

        print("\nAverages:")
        print(f"Average PSNR: {results_df['PSNR'].iloc[-1]:.4f}")
        print(f"Average UCIQE: {results_df['UCIQE'].iloc[-1]:.4f}")
        print(f"Average UIQM: {results_df['UIQM'].iloc[-1]:.4f}")

    except Exception as e:
        print(f"Error: {str(e)}")
```