Low light Image Enhancement-VLG Open Project 2024: -

Name: - Dikshant Kansal (22117047), IIT Roorkee

Background

In many practical scenarios, image quality is severely compromised due to poor lighting conditions. This poses a significant challenge for various applications that rely on clear and detailed visual information. Low light environments can lead to images that are noisy, poorly illuminated, and lack essential details, making it difficult to extract meaningful information. Enhancing images captured in such conditions is crucial for improving visibility and accuracy in tasks that require reliable image analysis

Problem Statement

The primary objective of this project is to develop an effective method for enhancing images captured in low light conditions. These enhancements aim to improve the visibility and clarity of the images, thereby making them more useful for various applications. Poor lighting can obscure critical details, affecting everything from personal photography to professional applications in surveillance, navigation, and medical imaging. Traditional methods often fail to produce satisfactory results, necessitating more advanced solutions like deep learning-based techniques.

Objectives

The key goals of this project are:

1. To develop a deep learning-based approach for enhancing low light images using a deep curve estimation network.

2. To evaluate the effectiveness of the proposed method in improving image quality in adverse lighting conditions.
3. To explore practical applications of the enhanced images in various fields such as security, navigation, and healthcare.

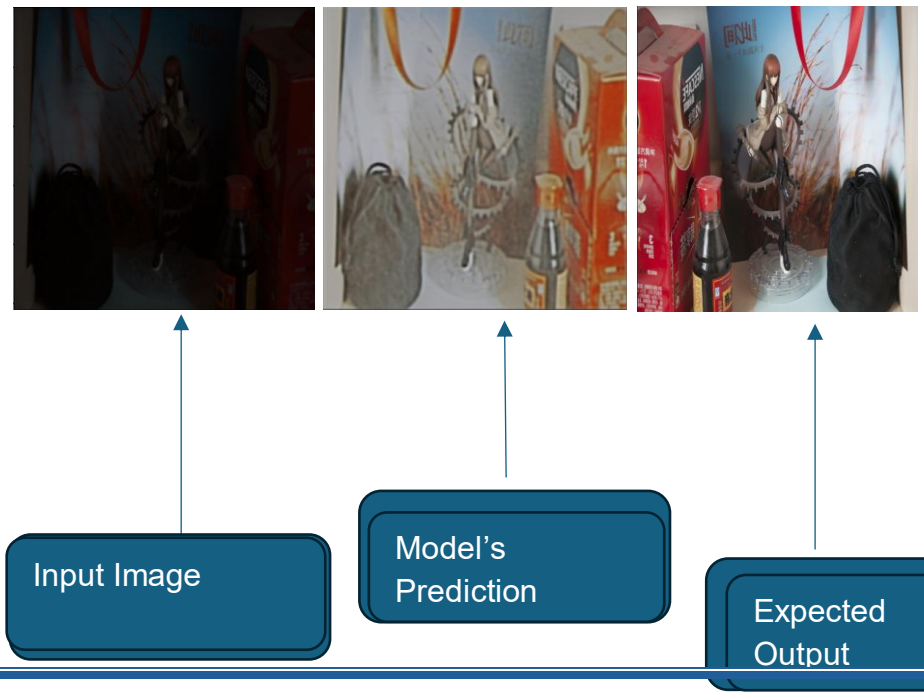
Literature Review

Recent advancements in deep learning have introduced powerful methods for image enhancement, particularly in low light environments. Traditional image processing techniques often fall short in dealing with the complexity and variability of low light conditions. The use of deep curve estimation networks offers a promising alternative by learning adaptive mappings that can enhance the visibility and detail of images.

Methodology: -

Initially, a basic CNN encoder-decoder network was tested to understand the intrinsic features of low light images. However, this approach yielded suboptimal results, as illustrated in the figure below. The network architecture, also depicted below, highlights the simplicity that limited its effectiveness in enhancing image quality.

Below shown is an example of input when passed through the trained model vs the expected output. As can be seen, the expected output does not match the given output. This architecture achieved a PNSR score of 21.029.



Architecture: -

```
model = Sequential([
                    # encoder network
                    Conv2D(256, 3, activation='relu', padding='same', input_shape=(256, 256, 3)),
                    # MaxPooling2D(2, padding='same'),
                    Conv2D(64, 3, activation='relu', padding='same'),
                    # MaxPooling2D(2, padding='same'),
                    # decoder network
                    Conv2D(64, 3, activation='relu', padding='same'),
                    # UpSampling2D(2),
                    Conv2D(256, 3, activation='relu', padding='same'),
                    # UpSampling2D(2),
                    # output layer
                    Conv2D(3, 3, activation='sigmoid', padding='same')
])

model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()
```

Initially, the architecture was too simplistic to capture the intricate details of the images, as evident in the results. Even after removing the Max Pooling layers to retain more information, there was no significant improvement in performance. To address this, I experimented with several complex architectures, but they tended to overfit the data.

To mitigate overfitting, I opted to augment the images, which introduced its own set of challenges. The most significant challenge was ensuring that identical augmentations were applied to both the high and low light images. To solve this, I concatenated the images along the RGB axis, transforming their dimensions from (256,256,3) to (256,256,6). This approach allowed the same transformations to be applied uniformly to both images. After augmentation, I split the images back into their original RGB channels. This technique helped in balancing the data and preserving consistency across the augmented image pairs.

Despite this solution, I faced limitations in computational resources, which prevented me from training the model on this large, augmented dataset. Consequently, I had to explore alternative methods that would be more efficient and feasible with the available resources.

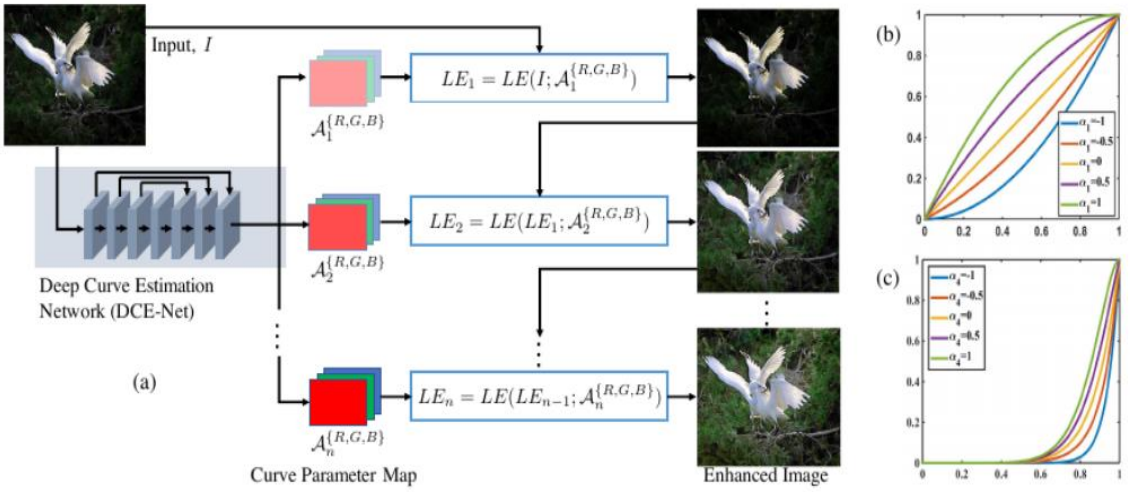
Due to the constraints imposed by my computational resources, I could not train the model effectively on the large, augmented dataset. This challenge led me to explore alternative methods, ultimately guiding me to the Deep Curve Estimation (DCE) Network. The DCE Network stood out for several reasons: it was significantly more efficient in terms of computational requirements, and it provided a logical and open framework for enhancing images captured in low light conditions.

The DCE method utilizes a learned curve estimation to dynamically adjust pixel values, which aligns with the inherent challenges posed by low light imaging. This adaptive approach allows for precise enhancement of image details while preserving the original colour and texture. The simplicity and effectiveness of the DCE framework made it an ideal choice for my project, addressing both the computational limitations and the need for high-quality image enhancement.

Inspired by the ease of implementation and the promising results reported in the DCE paper, I decided to integrate this technique into my project. The details of the DCE methodology, including its theoretical foundations and practical applications, are provided below. This integration marked a significant improvement in the ability to enhance low light images, offering a more robust and scalable solution compared to the previous approaches.

DCE Deep Curve Estimation Network: -

Deep Curve Estimation (DCE) networks are a class of deep learning models specifically designed for tasks that require adaptive image enhancement, such as improving visibility in low light conditions. Unlike traditional image enhancement techniques that apply uniform adjustments across an image, DCE networks learn to apply context-specific modifications by estimating curves that adjust pixel values dynamically.



Key Features of Zero-DCE

1. Image-Specific Curve Estimation: Zero-DCE uses a deep network to estimate curves specific to each input image for dynamic range adjustment. These curves are designed to be pixel-wise and high-order, allowing for detailed and localized enhancements.
2. No Reference Images Needed: The method does not require any paired or unpaired reference data during training. This is achieved through carefully designed non-reference loss functions that implicitly evaluate the quality of the enhancement.
3. Dynamic Range Adjustment: The estimated curves can effectively map input pixel values to an enhanced output over a wide dynamic range, improving image brightness and contrast.

Pipeline of Zero-DCE

1. Framework Overview

(a) DCE-Net Architecture: The DCE-Net (Deep Curve Estimation Network) is trained to estimate Light-Enhancement (LE) curves that can be applied iteratively to enhance the input image.

- Input: A given image.
- Output: An enhanced image produced by applying the LE curves iteratively.

The network learns to produce the parameters for these LE curves, which are then used to adjust the pixel values dynamically.

(b) LE-Curve Dynamics: The LE curves are defined as:

$$LE(I(x); \alpha) = I(x) + \alpha I(x)(1 - I(x))$$

Where:

- I(x) is the input pixel value normalized to the range [0, 1].
- α₩alpha is the adjustment parameter that determines the extent of enhancement.

The LE curve is designed to be simple yet effective in enhancing the image by boosting pixel values while maintaining the original image structure.

(c) Iterative Application: The curves are applied iteratively to the image, and each iteration enhances the dynamic range further:

- The parameter α₩alphaα can be varied to control the enhancement effect.
- Multiple iterations (n) allow for fine-tuning the enhancement.

In the subfigures:

- The horizontal axis represents the input pixel values.
- The vertical axis represents the output pixel values after applying the LE curves.

Detailed Breakdown of Components

1. DCE-Net Structure

- **Lightweight Network**: DCE-Net is designed to be efficient, requiring minimal computational resources while effectively estimating the enhancement curves.
- **Pixel-Wise Estimation**: The network predicts parameters for each pixel, allowing for localized enhancements that adapt to different regions of the image.
- **High-Order Curves**: By iterating the application of the LE curves, DCE-Net can simulate higher-order polynomial mappings, enabling complex transformations that are not possible with linear adjustments.

2. LE Curves and Their Application

- **Curve Definition**: The LE curve formula $LE(I(x); \alpha)=I(x)+\alpha I(x)(1-I(x))$ is designed to be monotonic and differentiable, ensuring smooth and continuous enhancement.
- **Adjustable Parameter (α₩alphaα)**: The parameter α₩alphaα controls the degree of enhancement. Positive values increase brightness, while negative values can decrease it.
- **Iterative Enhancement**: Repeatedly applying the curve with different parameters allows for gradual and controlled enhancement. For example, setting α₩alphaα to a negative value and applying the curve four times results in significant changes to the pixel values.

3. Non-Reference Loss Functions

The training of DCE-Net is guided by non-reference loss functions that do not require reference images. These losses ensure that the network learns to enhance the image based on perceptual and structural criteria. The primary loss functions used include:

1. ***Exposure Control Loss***: - To manage under- and over-exposed areas in images, an exposure control loss is designed to regulate exposure levels effectively. This loss quantifies the deviation between the average intensity of a local region and a desired well-exposed level (EEE). By minimizing this loss, the

method ensures that the image maintains consistent and optimal exposure, avoiding excessively dark or bright areas and enhancing overall visual quality. This approach helps achieve balanced exposure across the image, ensuring that details are preserved in both shadows and highlights.

$$L_{exp} = \frac{1}{M}\sum_{k=1}^{M}|Y_k - E|, \qquad (5)$$

where M represents the number of nonoverlapping local regions of size 16×16, Y is the average intensity value of a local region in the enhanced image.

```python
def exposure_loss(image, desired_mean=0.6):


    # Calculate the mean intensity across the color channels for each pixel
    mean_intensity = tf.reduce_mean(image, axis=3, keepdims=True)

    # Perform average pooling to get the local average intensity values
    local_mean = tf.nn.avg_pool2d(mean_intensity, ksize=16, strides=16, padding="VALID

    # Compute the exposure loss as the mean squared difference from the desired mean v
    loss = tf.reduce_mean(tf.square(local_mean - desired_mean))

    return loss
```

2. ***Colour Preservation Loss***: According to the Gray-World colour constancy hypothesis, the average colour of each sensor channel (red, green, and blue) in an image should approximate a neutral Gray if the scene is viewed under normal lighting conditions. This principle suggests that the average colour intensity across an image should be the same for all channels, ensuring a balanced and natural colour appearance. To correct potential colour deviations in the enhanced image and to maintain the relationships among the three colour channels, a colour constancy loss (Lcol) is designed. This loss function measures the divergence from the Gray-world assumption, encouraging the colour channels to converge towards a neutral Gray balance. By minimizing Lcol the method corrects any imbalances and ensures colour fidelity, helping the enhanced image retain accurate and natural colour

representation. This approach effectively mitigates colour cast issues and maintains consistency across the image, resulting in visually pleasing and realistic enhancements.

$$L_{col} = \sum_{\forall (p,q) \in \varepsilon} (J^p - J^q)^2, \varepsilon = \{(R,G),(R,B),(G,B)\},$$

```python
def color_constancy_loss(image):

    # Calculate the mean value for each color channel (red, green, blue)
    mean_channels = tf.reduce_mean(image, axis=(1, 2), keepdims=True)
    mean_red = mean_channels[:, :, :, 0]
    mean_green = mean_channels[:, :, :, 1]
    mean_blue = mean_channels[:, :, :, 2]

    # Compute the squared differences between the mean values of the channels
    diff_red_green = tf.square(mean_red - mean_green)
    diff_red_blue = tf.square(mean_red - mean_blue)
    diff_green_blue = tf.square(mean_green - mean_blue)

    # Calculate the color constancy loss
    color_loss = tf.sqrt(tf.square(diff_red_green) + tf.square(diff_red_blue) + tf.squ

    return color_loss
```

3. *Spatial Consistency Loss:* The spatial consistency loss (Lspa) is crucial for maintaining spatial coherence in image enhancement processes. It ensures that the local structural differences between the original and the enhanced images are preserved. This loss function encourages the enhanced image to maintain consistent differences across neighbouring regions, like the input image.
    - Formula: Measures the difference in image gradients or other spatial features.

$$L_{spa} = \frac{1}{K} \sum_{i=1}^{K} \sum_{j \in \Omega(i)} (|(Y_i - Y_j)| - |(I_i - I_j)|)^2,$$

where K is the number of local regions, and Ω(I) is the four neighbouring regions (top, down, left, right) cantered at the region I. We denote Y and I as the average intensity value of the local region in the enhanced version and input image, respectively. We empirically set the size of the local region to 4×4. This loss is stable given other region sizes.

```
class SpatialConsistencyLoss(keras.losses.Loss):
    def __init__(self, **kwargs):
        super(SpatialConsistencyLoss, self).__init__(reduction="none")

        self.left_kernel = tf.constant(
            [[[0, 0, 0]], [[-1, 1, 0]], [[0, 0, 0]]], dtype=tf.float32
        )
        self.right_kernel = tf.constant(
            [[[0, 0, 0]], [[0, 1, -1]], [[0, 0, 0]]], dtype=tf.float32
        )
        self.up_kernel = tf.constant(
            [[[0, -1, 0]], [[0, 1, 0]], [[0, 0, 0]]], dtype=tf.float32
        )
        self.down_kernel = tf.constant(
            [[[0, 0, 0]], [[0, 1, 0]], [[0, -1, 0]]], dtype=tf.float32
        )
```

Loading...

4.*Illumination Smoothness Loss:* To maintain smooth and consistent illumination transitions between neighbouring pixels, we introduce an illumination smoothness loss for each curve parameter map AAA. This loss, denoted as LtvAL, ensures that the monotonicity of illumination values is preserved across adjacent pixels. The illumination smoothness loss LtvAL is specifically designed to penalize abrupt changes in illumination, encouraging gradual and smooth variations instead. By applying this loss, we can achieve a more visually coherent and natural-looking illumination across the image. The smoothness constraint helps in preventing sharp changes in brightness, thus maintaining the desired monotonicity and enhancing the overall quality of the illumination adjustment.

$$L_{tv_{\mathcal{A}}} = \frac{1}{N}\sum_{n=1}^{N}\sum_{c\in\xi}(|\nabla_x \mathcal{A}_n^c| + \nabla_y \mathcal{A}_n^c|)^2, \xi = \{R, G, B\}, \tag{7}$$

where N is the number of iteration, ∇_x and ∇_y represent the horizontal and vertical gradient operations, respectively.

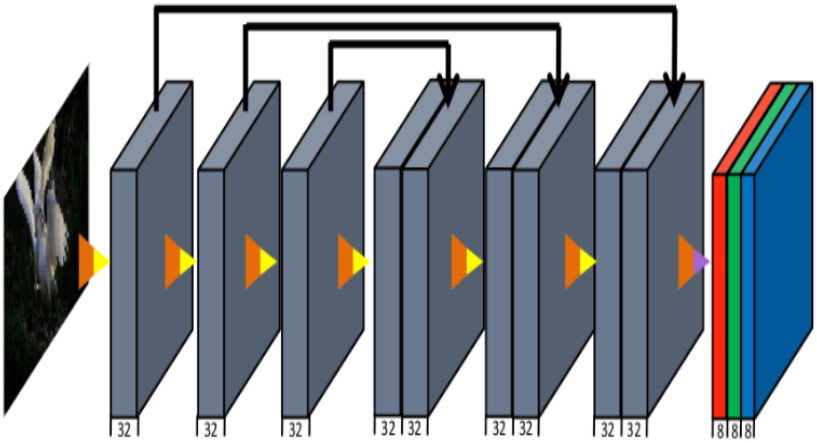
Total Loss. The total loss can be expressed as:

$$L_{total} = L_{spa} + L_{exp} + W_{col}L_{col} + W_{tv_{\mathcal{A}}}L_{tv_{\mathcal{A}}}, \tag{8}$$

where W_{col} and $W_{tv_{\mathcal{A}}}$ are the weights of the losses.

```python
def illumination_smoothness_loss(enhancement_map):
    batch_dim = tf.shape(enhancement_map)[0]
    height_dim = tf.shape(enhancement_map)[1]
    width_dim = tf.shape(enhancement_map)[2]

    # Compute the count of height and width elements for normalization
    height_elements = (tf.shape(enhancement_map)[2] - 1) * tf.shape(enhancement_map)[3
    width_elements = tf.shape(enhancement_map)[2] * (tf.shape(enhancement_map)[3] - 1)

    # Calculate the total variation across height and width dimensions
    height_variation = tf.reduce_sum(tf.square(enhancement_map[:, 1:, :, :] - enhancem
    width_variation = tf.reduce_sum(tf.square(enhancement_map[:, :, 1:, :] - enhanceme

    # Convert batch size and element counts to float for normalization
    batch_dim_float = tf.cast(batch_dim, dtype=tf.float32)
    height_elements_float = tf.cast(height_elements, dtype=tf.float32)
    width_elements_float = tf.cast(width_elements, dtype=tf.float32)

    # Compute the normalized smoothness loss
    smoothness_loss = 2 * (height_variation / height_elements_float + width_variation

    return smoothness_loss
```

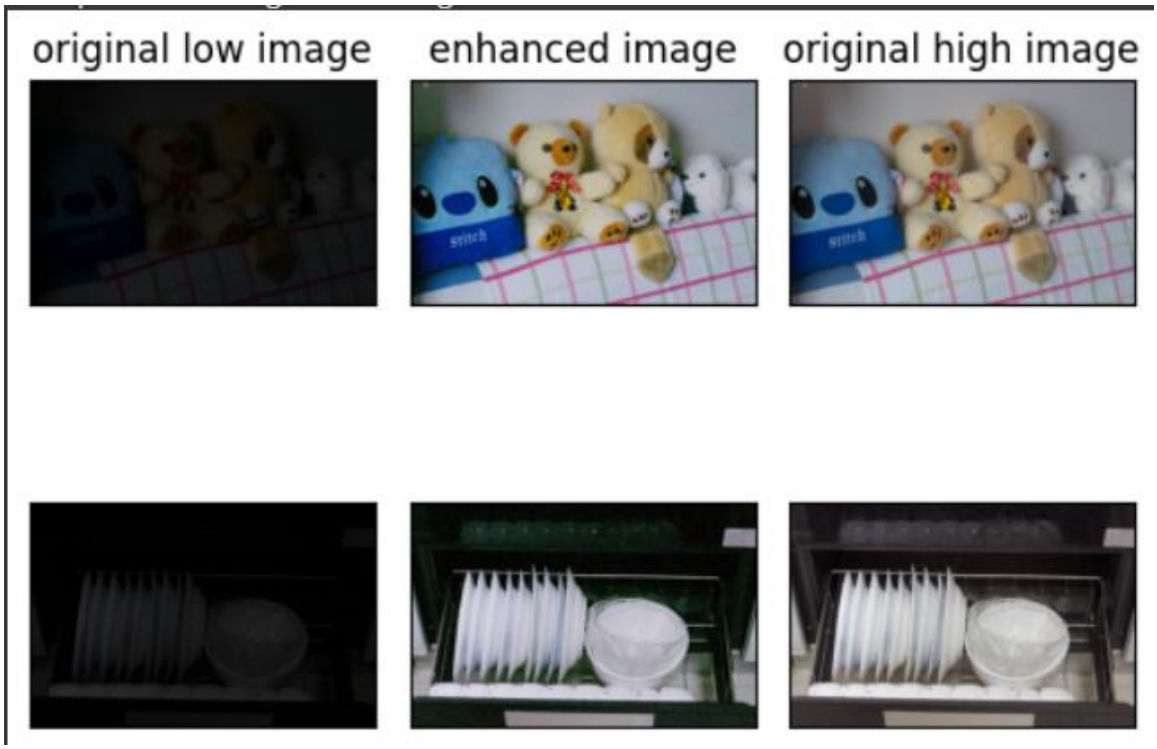
Architecture Of DCE Net: -



Deep Curve Estimation Network (DCE-Net)

```
conv_layer1 = layers.Conv2D(
    filters=32, kernel_size=(3, 3), strides=(1, 1), activation="relu", padding="sa
)(input_layer)

conv_layer2 = layers.Conv2D(
    filters=32, kernel_size=(3, 3), strides=(1, 1), activation="relu", padding="sa
)(conv_layer1)

conv_layer3 = layers.Conv2D(
    filters=32, kernel_size=(3, 3), strides=(1, 1), activation="relu", padding="sa
)(conv_layer2)

conv_layer4 = layers.Conv2D(
    filters=32, kernel_size=(3, 3), strides=(1, 1), activation="relu", padding="sa
)(conv_layer3)

# Concatenation of the third and fourth convolutional layers
concat1 = layers.Concatenate(axis=-1)([conv_layer4, conv_layer3])

# Fifth convolutional layer after concatenation
conv_layer5 = layers.Conv2D(
    filters=32, kernel_size=(3, 3), strides=(1, 1), activation="relu", padding="sa
)(concat1)

# Concatenation of the fifth layer with the second convolutional layer
concat2 = layers.Concatenate(axis=-1)([conv_layer5, conv_layer2])

# Sixth convolutional layer after the second concatenation
conv_layer6 = layers.Conv2D(
    filters=32, kernel_size=(3, 3), strides=(1, 1), activation="relu", padding="sa
)(concat2)
```

Result: -



PSNR Score: -

```python
def calculate_psnr(original_image, enhanced_image):
    """
    Calculate the Peak Signal-to-Noise Ratio (PSNR) between two images.

    Parameters:
    - original_image (np.ndarray): The original reference image.
    - enhanced_image (np.ndarray): The enhanced image to compare.

    Returns:
    - psnr (float): The PSNR value in decibels.
    """
    # Ensure the input images are in the correct format and dimensions
    original_image = np.array(original_image, dtype=np.float32)
    enhanced_image = np.array(enhanced_image, dtype=np.float32)

    if original_image.shape != enhanced_image.shape:
        raise ValueError("Input images must have the same dimensions and channels")

    # Compute the Mean Squared Error (MSE)
    mse = np.mean((original_image - enhanced_image) ** 2)

    if mse == 0:
        return float('inf')  # If the MSE is zero, the PSNR is infinite

    # Set the maximum pixel value
    max_pixel_value = 255.0

    # Compute the PSNR
    psnr = 10 * np.log10(max_pixel_value / np.sqrt(mse))

    return psnr
```

PSNR Score Achieved using Deep Curve Estimation= 27.602611571511055

Link to paper: -

https://li-chongyi.github.io/Proj_Zero-DCE.html

Some other references: -

https://youtu.be/9Rbi3f0Vg7k?si=ZsW7neicO9Gml8Fj

https://openaccess.thecvf.com/content/CVPR2022/papers/Zhang_Deep_Color_Consistent_Network_for_Low-Light_Image_Enhancement_CVPR_2022_paper.pdf