Project Report

Low Light Image Enhancement

By: Shoilayee Chaudhuri

21112102

The Problem:

Light is of paramount importance to photography. Night and low light place very demanding constraints on photography due to very limited photon count and inescapable noise. One natural reaction is to gather more light by, e.g., enlarging aperture setting, lengthening exposure time and opening flashlight. However, each method brings a trade-off–large aperture incurs small depth of field and is usually unavailable in smartphone cameras; long exposure can induce blur due to scene variations or camera motions; flash can cause colour aberrations and is useful only for nearby objects. Therefore, denoising in extremely low-light conditions has become an important research direction in the low-level image processing community, aiming to restore details in images captured in extremely low-light scenes and enhance visual quality.

The Solution:

The objective was to develop and evaluate an approach for enhancing images captured in low light. Two methods were implemented:

1. Using Convolutional Neural Network
2. Implementation of a research paper: LIME (Low-light Image Enhancement via Illumination Map Estimation)

Dataset used:

The dataset had 2 folders:

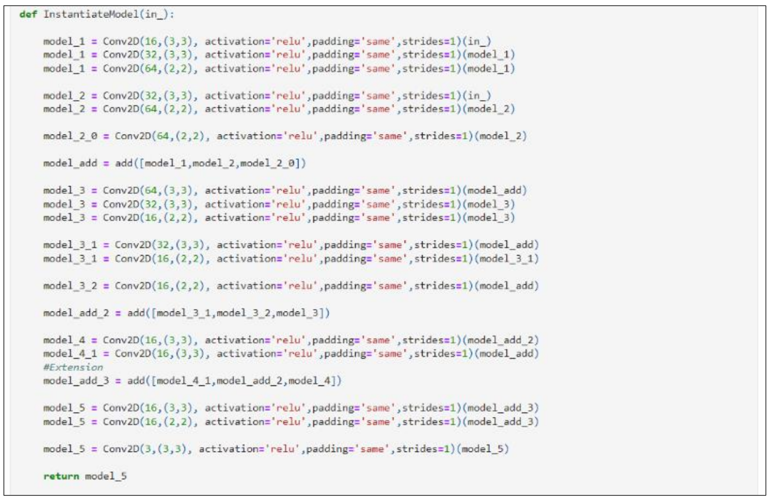
a) High: images captured in good lighting
b) Low: same images with low lighting

Details:

Approach – 1: Using CNN:

This method leverages convolutional neural networks (CNNs) for enhancing low-light images. The pipeline includes data preprocessing, model instantiation, training, and testing.

a. Pre-processing: A function read images from the specified directories, converted them to RGB format, resized to 500x500 pixels, and returned arrays of low-light and high-light images.
b. Model: A CNN model is created with Keras. It model consists of a few convolutional layers with strides of 1 and the output layer uses a ReLU activation function to generate the enhanced image. It was compiled with 'adam' optimiser.
c. After training, the PSNR value is still not satisfactory, so we perform histogram equalisation. The cumulative probability distribution of the pixel intensities is calculated, the zero intensities are masked to avoid division by 0, and then cdf is normalised.

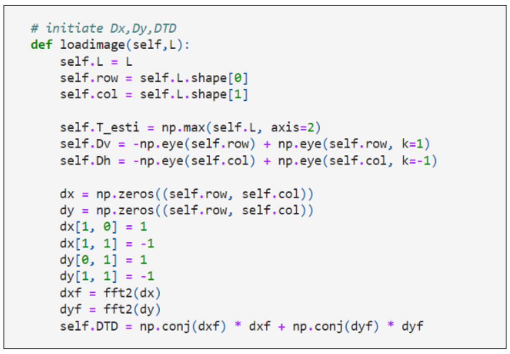


```python
def InstantiateModel(in_):

    model_1 = Conv2D(16,(3,3), activation='relu',padding='same',strides=1)(in_)
    model_1 = Conv2D(32,(3,3), activation='relu',padding='same',strides=1)(model_1)
    model_1 = Conv2D(64,(2,2), activation='relu',padding='same',strides=1)(model_1)

    model_2 = Conv2D(32,(3,3), activation='relu',padding='same',strides=1)(in_)
    model_2 = Conv2D(64,(2,2), activation='relu',padding='same',strides=1)(model_2)

    model_2_0 = Conv2D(64,(2,2), activation='relu',padding='same',strides=1)(model_2)

    model_add = add([model_1,model_2,model_2_0])

    model_3 = Conv2D(64,(3,3), activation='relu',padding='same',strides=1)(model_add)
    model_3 = Conv2D(32,(3,3), activation='relu',padding='same',strides=1)(model_3)
    model_3 = Conv2D(16,(2,2), activation='relu',padding='same',strides=1)(model_3)

    model_3_1 = Conv2D(32,(3,3), activation='relu',padding='same',strides=1)(model_add)
    model_3_1 = Conv2D(16,(2,2), activation='relu',padding='same',strides=1)(model_3_1)

    model_3_2 = Conv2D(16,(2,2), activation='relu',padding='same',strides=1)(model_add)

    model_add_2 = add([model_3_1,model_3_2,model_3])

    model_4 = Conv2D(16,(3,3), activation='relu',padding='same',strides=1)(model_add_2)
    model_4_1 = Conv2D(16,(3,3), activation='relu',padding='same',strides=1)(model_add)
    #Extension
    model_add_3 = add([model_4_1,model_add_2,model_4])

    model_5 = Conv2D(16,(3,3), activation='relu',padding='same',strides=1)(model_add_3)
    model_5 = Conv2D(16,(2,2), activation='relu',padding='same',strides=1)(model_add_3)

    model_5 = Conv2D(3,(3,3), activation='relu',padding='same',strides=1)(model_5)

    return model_5
```

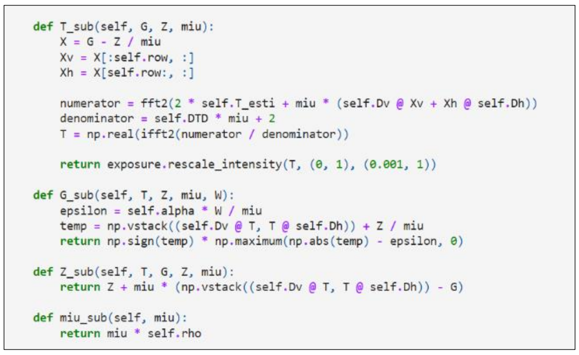
Approach – 2: Using LIME:

An illumination map is a representation of the varying illumination levels across an image. In the context of image processing, particularly for enhancing low-light images, the illumination map helps to separate the light component from the reflectance component of an image. This process involves iterative optimization to estimate the illumination map and enhance the image.

a. Pre-processing: The load method loads an image and initializes key variables such as:
    - T (Estimated illumination map)
    - G (Gradient update variable)
    - Z (Dual variable handling constraints)
    - µ (Penalty parameter)
b. The method iteratively refines the illumination map by updating T, G, Z and µ.
    - T_sub (Updating T): involves operations like Fast Fourier Transform for computational efficiency and regularization
    - G_sub (Updating G): Involves updating G to minimize the difference between T and the observed data.
    - Z_sub (Updating Z): Adjusts Z based on the residuals or differences between T, G, and the observed data.
    - miu_sub (Updating µ): Adjusts the penalty parameter µ to control the step size or convergence rate in the optimization process.



```python
# initiate Dx,Dy,DTD
def loadimage(self,L):
    self.L = L
    self.row = self.L.shape[0]
    self.col = self.L.shape[1]

    self.T_esti = np.max(self.L, axis=2)
    self.Dv = -np.eye(self.row) + np.eye(self.row, k=1)
    self.Dh = -np.eye(self.col) + np.eye(self.col, k=-1)

    dx = np.zeros((self.row, self.col))
    dy = np.zeros((self.row, self.col))
    dx[1, 0] = 1
    dx[1, 1] = -1
    dy[0, 1] = 1
    dy[1, 1] = -1
    dxf = fft2(dx)
    dyf = fft2(dy)
    self.DTD = np.conj(dxf) * dxf + np.conj(dyf) * dyf
```



```python
def T_sub(self, G, Z, miu):
    X = G - Z / miu
    Xv = X[:self.row, :]
    Xh = X[self.row:, :]

    numerator = fft2(2 * self.T_esti + miu * (self.Dv @ Xv + Xh @ self.Dh))
    denominator = self.DTD * miu + 2
    T = np.real(ifft2(numerator / denominator))

    return exposure.rescale_intensity(T, (0, 1), (0.001, 1))

def G_sub(self, T, Z, miu, W):
    epsilon = self.alpha * W / miu
    temp = np.vstack((self.Dv @ T, T @ self.Dh)) + Z / miu
    return np.sign(temp) * np.maximum(np.abs(temp) - epsilon, 0)

def Z_sub(self, T, G, Z, miu):
    return Z + miu * (np.vstack((self.Dv @ T, T @ self.Dh)) - G)

def miu_sub(self, miu):
    return miu * self.rho
```

The Evaluation:

CNN Approach:

- PSNR: 27.871782500208866
- MSE: 106.145628
- MAE: 130.416972

LIME Approach:

- PSNR: 28.458977457866048
- MSE: 92.7219625
- MAE: 94.39137638888889