Enhancing Low-Light Images Using the Zero-DCE Model

- **Swayam Pal**

21411036

Abstract

The use and assessment of the Zero-DCE model for improving low-light photos are presented in this report. A new method called Zero-DCE, or Zero-Reference Deep Curve Estimation, uses deep learning to improve image quality without the need for paired or unpaired training data. TensorFlow and Keras were used to implement the model, which was trained using the LoL (Low-Light) dataset. The Peak Signal-to-Noise Ratio (PSNR) metric and a number of bespoke loss functions were used to assess the model's performance. The outcomes showed that the visual quality of photos taken in low light had significantly improved.

Methodology

Data Loading and Processing

The LoL dataset was used for training and validation. This dataset consists of low-light images and their corresponding normal-light versions. The images were resized to 256×256 pixels and normalized to a range of [0, 1]. The dataset was divided into training and validation sets with a ratio of 4:1.

```
def load_data(image_path):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_png(img, channels=3)
    img = tf.image.resize(images=img, size=[IMAGE_SIZE, IMAGE_SIZE])
    img = img / 255.0  #scaling between [0,1]
    return img
```

```
def data_generator(low_light_images):
    data = tf.data.Dataset.from_tensor_slices((low_light_images))
    data = data.map(load_data, num_parallel_calls=tf.data.AUTOTUNE)
    data = data.batch(BATCH_SIZE, drop_remainder=True)
    return data
```

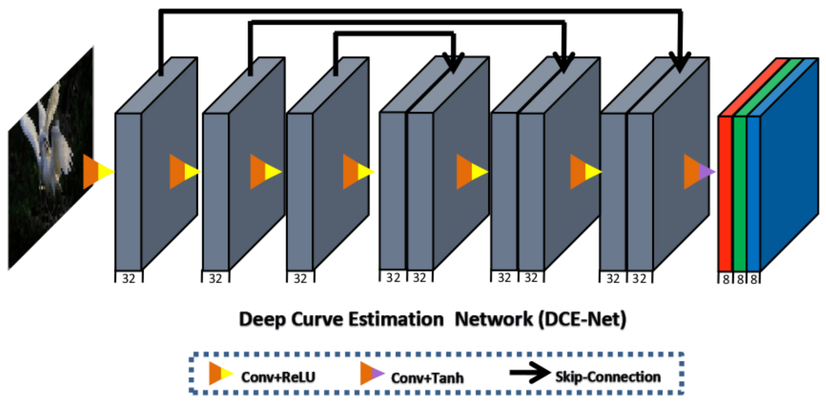
DATASET PREPARATION

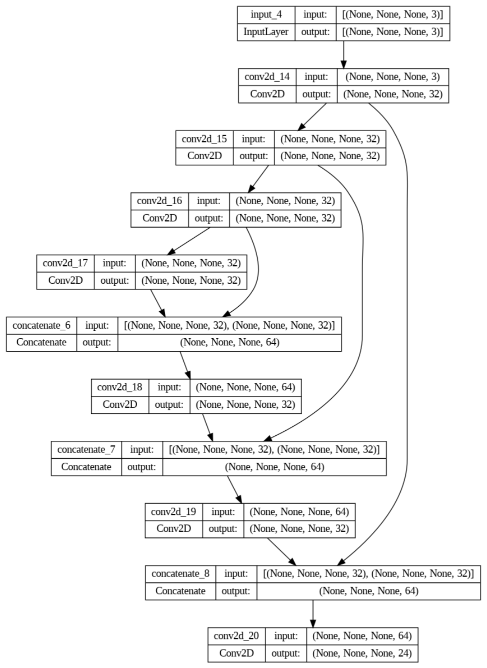
```
train_low_light_images = sorted(glob("/content/drive/MyDrive/lol_dataset/our485/low/*"))[:MAX_TRAIN_IMAGES]
val_low_light_images = sorted(glob("/content/drive/MyDrive/lol_dataset/our485/low/*"))[MAX_TRAIN_IMAGES:]

test_low_light_images = sorted(glob("/content/drive/MyDrive/lol_dataset/eval15/low/*"))
test_high_light_images = sorted(glob("/content/drive/MyDrive/lol_dataset/eval15/high/*"))


train_dataset = data_generator(train_low_light_images)
val_dataset = data_generator(val_low_light_images)

print("Train Dataset:", train_dataset)
print("Validation Dataset:", val_dataset)
```

Model Architecture

The Zero-DCE model is constructed using a series of convolutional layers that estimate a set of curves to enhance the input images. The architecture consists of six convolutional layers with ReLU activations having 2 concatenation layers in between to retrieve the low level features, followed by a final convolutional layer with a ‘tanh’ activation to output the enhancement curves.

```
def Build_DCE_NET():
    input_img = Input(shape=[None, None, 3])
    x1 = Conv2D(32, (3, 3), strides=(1, 1), activation="relu", padding="same")(input_img)
    x2 = Conv2D(32, (3, 3), strides=(1, 1), activation="relu", padding="same")(x1)
    x3 = Conv2D(32, (3, 3), strides=(1, 1), activation="relu", padding="same")(x2)
    x4 = Conv2D(32, (3, 3), strides=(1, 1), activation="relu", padding="same")(x3)

    int_x1 = Concatenate(axis=-1)([x4, x3])
    x5 = layers.Conv2D(32, (3, 3), strides=(1, 1), activation="relu", padding="same")(int_x1)

    int_x2 = Concatenate(axis=-1)([x5, x2])
    x6 = Conv2D(32, (3, 3), strides=(1, 1), activation="relu", padding="same")(int_x2)

    int_x3 = Concatenate(axis=-1)([x6, x1])
    y = Conv2D(24, (3, 3), strides=(1, 1), activation="tanh", padding="same")(int_x3)

    return Model(inputs=input_img, outputs=y)
```




input_4 | input: | [(None, None, None, 3)]
InputLayer | output: | [(None, None, None, 3)]
conv2d_14 | input: | (None, None, None, 3)
Conv2D | output: | (None, None, None, 32)
conv2d_15 | input: | (None, None, None, 32)
Conv2D | output: | (None, None, None, 32)
conv2d_16 | input: | (None, None, None, 32)
Conv2D | output: | (None, None, None, 32)
conv2d_17 | input: | (None, None, None, 32)
Conv2D | output: | (None, None, None, 32)
concatenate_6 | input: | [(None, None, None, 32), (None, None, None, 32)]
Concatenate | output: | (None, None, None, 64)
conv2d_18 | input: | (None, None, None, 64)
Conv2D | output: | (None, None, None, 32)
concatenate_7 | input: | [(None, None, None, 32), (None, None, None, 32)]
Concatenate | output: | (None, None, None, 64)
conv2d_19 | input: | (None, None, None, 64)
Conv2D | output: | (None, None, None, 32)
concatenate_8 | input: | [(None, None, None, 32), (None, None, None, 32)]
Concatenate | output: | (None, None, None, 64)
conv2d_20 | input: | (None, None, None, 64)
Conv2D | output: | (None, None, None, 24)

Custom Loss Functions

Three custom loss functions were implemented to guide the training process: exposure control loss, color constancy loss, and illumination smoothness loss.

Exposure Control Loss

Ensures that the enhanced image has an appropriate exposure level.

$$L_{exp} = \frac{1}{M} \sum\nolimits_{k=1}^{M} |Y_k - E|,$$

Where M represents the number of non-overlapping local regions of size 16×16, Y is the average intensity value of a local region in the enhanced image.

We set E to 0.6 in our experiments although we do not find much performance difference by setting E within [0.4, 0.7].

```
def exposure_loss(x, E=0.6):  # E is the grey level in RGB color generally taken as 0.6 for experiments
    x = tf.reduce_mean(x, axis=3, keepdims=True)
    mean = tf.nn.avg_pool2d(x, ksize=16, strides=16, padding="VALID") #averaging over 16x16 non-overlapping regio
    L_exp = tf.reduce_mean(tf.square(mean - E)) #took the mean of all the values
    return L_exp
```

Color Constancy Loss

Ensures that the colors in the enhanced image are consistent with the input image.

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

Where Jp denotes the average intensity value of p channel in the enhanced image, (p,q) represents a pair of channels.

Following is the implementation for color constancy loos function:

```
def color_constancy_loss(x):
    mean_rgb = tf.reduce_mean(x, axis=(1, 2), keepdims=True)
    jr, jg, jb = (
        mean_rgb[:, :, :, 0],
        mean_rgb[:, :, :, 1],
        mean_rgb[:, :, :, 2],
    )
    #ji denotes average intensity of ith channel
    #pairwise taking squares
    diff_rg = tf.square(jr - jg)
    diff_rb = tf.square(jr - jb)
    diff_gb = tf.square(jb - jg)

    L_col = tf.sqrt(tf.square(diff_rg) + tf.square(diff_rb) + tf.square(diff_gb))
    return  L_col
```

Illumination Smoothness Loss

To preserve the mono-tonicity relations between neighboring pixels, we add an illumination smoothness loss to each curve parameter map

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

Where N is the number of iteration, ∇x and ∇y represent the horizontal and vertical gradient operations, respectively and A represents the enhancement curves

Following is the implementation of the above written formula:

```python
def illumination_smoothness_loss(x):
    batch_size = tf.shape(x)[0]
    h_x = tf.shape(x)[1]
    w_x = tf.shape(x)[2]
    count_h = (w_x - 1) * tf.shape(x)[3]
    count_w = w_x * (tf.shape(x)[3] - 1)
    h_tv = tf.reduce_sum(tf.square((x[:, 1:, :, :] - x[:, : h_x - 1, :, :])))
    w_tv = tf.reduce_sum(tf.square((x[:, :, 1:, :] - x[:, :, : w_x - 1, :])))
    batch_size = tf.cast(batch_size, dtype=tf.float32)
    count_h = tf.cast(count_h, dtype=tf.float32)
    count_w = tf.cast(count_w, dtype=tf.float32)
    return 2 * (h_tv / count_h + w_tv / count_w) / batch_size
```

Evaluation Metrics

The performance of the model was evaluated based on the total loss and its components. Additionally, the Peak Signal-to-Noise Ratio (PSNR) metric was used to quantitatively assess the quality of the enhanced images. PSNR is defined as:

$$\mathrm{PSNR} = 20 \log_{10} \left(\frac{\mathrm{MAX}_I}{\sqrt{\mathrm{MSE}}} \right)$$

Quantitative Results

The final losses and PSNR values achieved after 100 epochs of training are summarized in Table.

📌 PSNR calculated over the test dataset - `28.32682145082687`

```python
def calculate_psnr(original_image, enhanced_image):
  # Convert the images to numpy arrays.
  original_image = np.array(original_image)
  enhanced_image = np.array(enhanced_image)
  # Calculate the mean squared error (MSE) between the two images.
  mse = np.mean((original_image - enhanced_image) ** 2)
  # Calculate the peak signal-to-noise ratio (PSNR).
  psnr = 10 * np.log10(255 ** 2 / mse)
  return psnr
```

```
psnr_ratio = []
for i in range(len(test_low_light_images)):
    low_light_image = Image.open(test_low_light_images[i])
    enhanced_image = low_to_high_light(low_light_image)
    psnr = calculate_psnr(low_light_image, enhanced_image)
    psnr_ratio.append(psnr)
    #print(f"PSNR for {test_low_light_images[i]}: {psnr}")
```

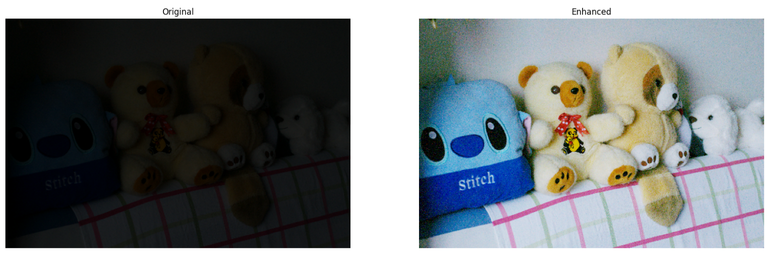
```
np.average(psnr_ratio)
```

```
28.32682145082687
```

Metric	Training Loss	Validation Loss
Total Loss	`0.7599`	`0.6394`
Illumination Smoothness	`0.0281`	`0.0426`
Color Constancy	`0.0959`	`0.1441`
Exposure Control	`0.6360`	`0.4527`

Qualitative Results

The enhanced images exhibited significant improvements in brightness, color accuracy, and overall visual appeal. The model was able to effectively enhance the visibility of details in low-light conditions while preserving the natural appearance of the scenes.



References

Guo, C., Li, C., Guo, J., Loy, C. C., Hou, J., Kwong, S., & Cong, R. (2020). Zero-Reference Deep Curve Estimation for Low-Light Image Enhancement. *BIIT Lab, Tianjin University; City University of Hong Kong; Nanyang Technological University; Beijing Jiaotong University*. Retrieved from https://openaccess.thecvf.com/content_CVPR_2020/papers/Guo_Zero-Reference_Deep_Curve_Estimation_for_Low-Light_Image_Enhancement_CVPR_2020_paper.pdf

HTX's COE. (May 26, 2023). Breaking Through the Darkness: How to Enhance Low-Light Images with Deep Learning Techniques. *Medium*. Retrieved from https://medium.com/htx-s-s-coe/breaking-through-the-darkness-how-to-enhance-low-light-images-with-deep-learning-techniques-257c98daf96f.