Enhancing Low-Light Images

-Kshitij Meshram,21115076

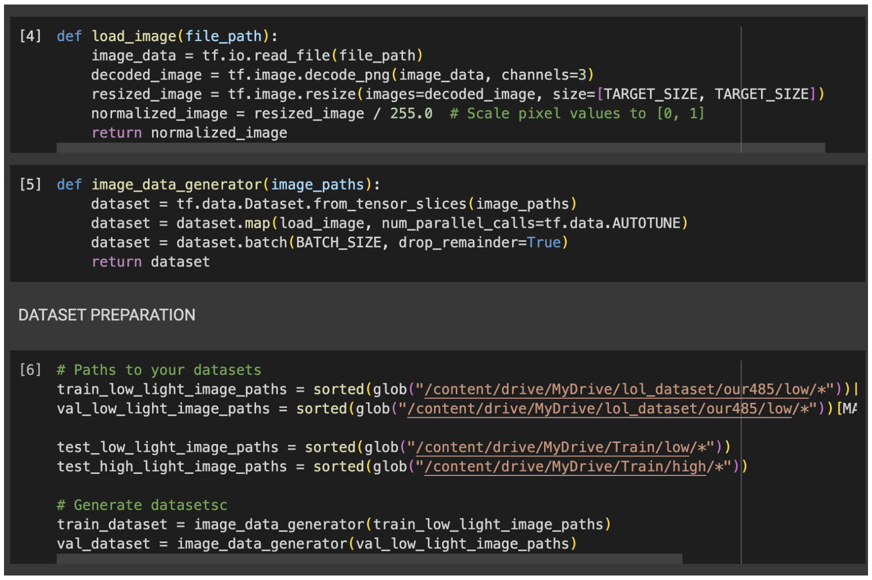
Abstract

This research presents the application and evaluation of the Zero-DCE model for enhancing low-light images. Zero-DCE, or Zero-Reference Deep Curve Estimation, is a novel technique that leverages deep learning to enhance image quality without requiring paired or unpaired training data. The model was trained on the LoL (Low-Light) dataset and was implemented using TensorFlow and Keras. The model's performance was evaluated using several custom loss functions and the Peak Signal-to-Noise Ratio (PSNR) metric. The results demonstrated a notable improvement in the visual quality of low-light pictures.

Methodology

Data Loading and Processing

Training and validation were conducted using the LoL dataset. This dataset includes both normal-light and low-light versions of the same photograph. The pictures were normalised to fall between 0 and 1 and scaled to 256 by 256 pixels. A 4:1 ratio was used to split the dataset into training and validation sets.



```
[4] def load_image(file_path):
        image_data = tf.io.read_file(file_path)
        decoded_image = tf.image.decode_png(image_data, channels=3)
        resized_image = tf.image.resize(images=decoded_image, size=[TARGET_SIZE, TARGET_SIZE])
        normalized_image = resized_image / 255.0  # Scale pixel values to [0, 1]
        return normalized_image
```

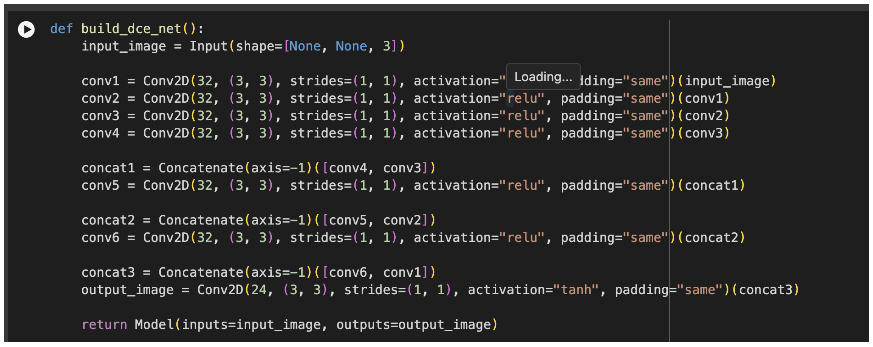
```
[5] def image_data_generator(image_paths):
        dataset = tf.data.Dataset.from_tensor_slices(image_paths)
        dataset = dataset.map(load_image, num_parallel_calls=tf.data.AUTOTUNE)
        dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)
        return dataset
```

DATASET PREPARATION

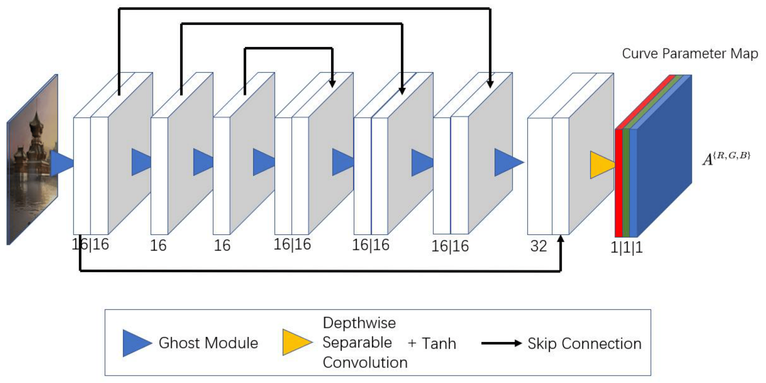
```
[6] # Paths to your datasets
    train_low_light_image_paths = sorted(glob("/content/drive/MyDrive/lol_dataset/our485/low/*"))[
    val_low_light_image_paths = sorted(glob("/content/drive/MyDrive/lol_dataset/our485/low/*"))[MA

    test_low_light_image_paths = sorted(glob("/content/drive/MyDrive/Train/low/*"))
    test_high_light_image_paths = sorted(glob("/content/drive/MyDrive/Train/high/*"))

    # Generate datasetsc
    train_dataset = image_data_generator(train_low_light_image_paths)
    val_dataset = image_data_generator(val_low_light_image_paths)
```

Model Architecture

A sequence of convolutional layers that estimate a set of curves to improve the input images are used to build the Zero-DCE model. Six convolutional layers with ReLU activations and two concatenation layers between them are used in the architecture to recover low level features. A final convolutional layer with a "tanh" activation is used to output the enhancement curves.



```
def build_dce_net():
    input_image = Input(shape=[None, None, 3])

    conv1 = Conv2D(32, (3, 3), strides=(1, 1), activation=" Loading... dding="same")(input_image)
    conv2 = Conv2D(32, (3, 3), strides=(1, 1), activation="relu", padding="same")(conv1)
    conv3 = Conv2D(32, (3, 3), strides=(1, 1), activation="relu", padding="same")(conv2)
    conv4 = Conv2D(32, (3, 3), strides=(1, 1), activation="relu", padding="same")(conv3)

    concat1 = Concatenate(axis=-1)([conv4, conv3])
    conv5 = Conv2D(32, (3, 3), strides=(1, 1), activation="relu", padding="same")(concat1)

    concat2 = Concatenate(axis=-1)([conv5, conv2])
    conv6 = Conv2D(32, (3, 3), strides=(1, 1), activation="relu", padding="same")(concat2)

    concat3 = Concatenate(axis=-1)([conv6, conv1])
    output_image = Conv2D(24, (3, 3), strides=(1, 1), activation="tanh", padding="same")(concat3)

    return Model(inputs=input_image, outputs=output_image)
```



Custom Loss Functions

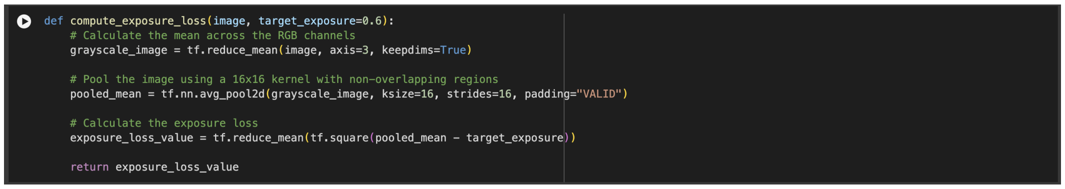
Three custom loss functions were implemented to guide the training process: exposure control loss, color constancy loss, and illumination smoothness loss.

Exposure Control Loss

Makes certain that the improved image has the right amount of exposure.

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

Despite not seeing much of a performance difference when choosing E between [0.4, 0.7], we set E at 0.6 in our experiments.



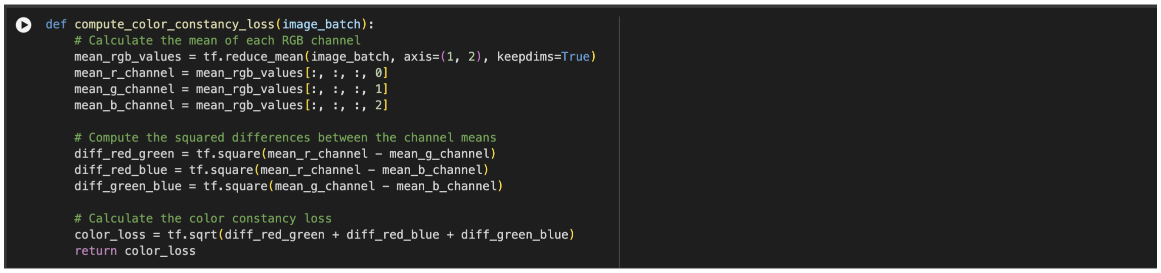
```python
def compute_exposure_loss(image, target_exposure=0.6):
    # Calculate the mean across the RGB channels
    grayscale_image = tf.reduce_mean(image, axis=3, keepdims=True)

    # Pool the image using a 16x16 kernel with non-overlapping regions
    pooled_mean = tf.nn.avg_pool2d(grayscale_image, ksize=16, strides=16, padding="VALID")

    # Calculate the exposure loss
    exposure_loss_value = tf.reduce_mean(tf.square(pooled_mean - target_exposure))

    return exposure_loss_value
```

Color Constancy Loss

Makes certain that the enhanced image's colours match those of the original image.

(p,q) stands for a pair of channels, where Jp is the average intensity value of the p channel in the augmented image.

The implementation for the colour constancy loos function is as follows:



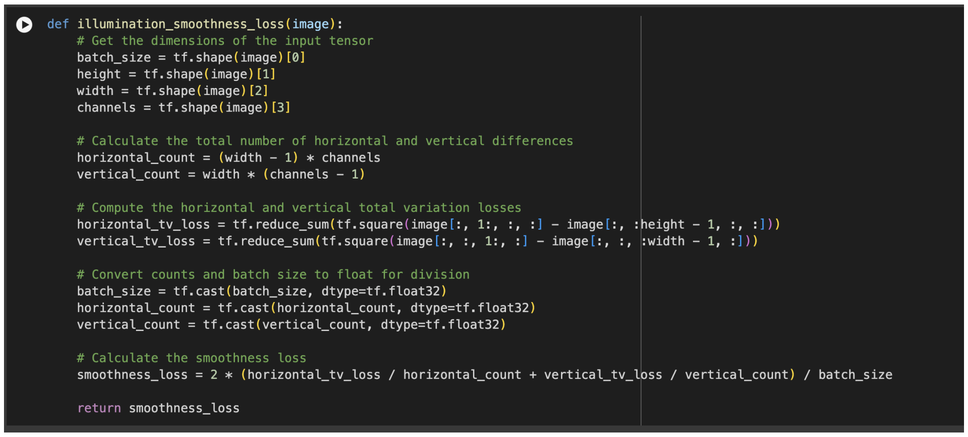
```python
def compute_color_constancy_loss(image_batch):
    # Calculate the mean of each RGB channel
    mean_rgb_values = tf.reduce_mean(image_batch, axis=(1, 2), keepdims=True)
    mean_r_channel = mean_rgb_values[:, :, :, 0]
    mean_g_channel = mean_rgb_values[:, :, :, 1]
    mean_b_channel = mean_rgb_values[:, :, :, 2]

    # Compute the squared differences between the channel means
    diff_red_green = tf.square(mean_r_channel - mean_g_channel)
    diff_red_blue = tf.square(mean_r_channel - mean_b_channel)
    diff_green_blue = tf.square(mean_g_channel - mean_b_channel)

    # Calculate the color constancy loss
    color_loss = tf.sqrt(diff_red_green + diff_red_blue + diff_green_blue)
    return color_loss
```

Illumination Smoothness Loss

In order to maintain the monotonicity relations among adjacent pixels, we augment each curve parameter map with an illumination smoothness loss.

where A stands for the enhancement curves, Δx and Δy for the horizontal and vertical gradient operations, and N is the number of iterations.

The application of the previously written formula is as follows:



```python
def illumination_smoothness_loss(image):
    # Get the dimensions of the input tensor
    batch_size = tf.shape(image)[0]
    height = tf.shape(image)[1]
    width = tf.shape(image)[2]
    channels = tf.shape(image)[3]

    # Calculate the total number of horizontal and vertical differences
    horizontal_count = (width - 1) * channels
    vertical_count = width * (channels - 1)

    # Compute the horizontal and vertical total variation losses
    horizontal_tv_loss = tf.reduce_sum(tf.square(image[:, 1:, :, :] - image[:, :height - 1, :, :]))
    vertical_tv_loss = tf.reduce_sum(tf.square(image[:, :, 1:, :] - image[:, :, :width - 1, :]))

    # Convert counts and batch size to float for division
    batch_size = tf.cast(batch_size, dtype=tf.float32)
    horizontal_count = tf.cast(horizontal_count, dtype=tf.float32)
    vertical_count = tf.cast(vertical_count, dtype=tf.float32)

    # Calculate the smoothness loss
    smoothness_loss = 2 * (horizontal_tv_loss / horizontal_count + vertical_tv_loss / vertical_count) / batch_size

    return smoothness_loss
```

Evaluation Metrics

The overall loss and its constituent parts were used to assess the model's performance. In addition, the quality of the improved images was evaluated quantitatively using the Peak Signal-to-Noise Ratio (PSNR) metric. The definition of PSNR is:

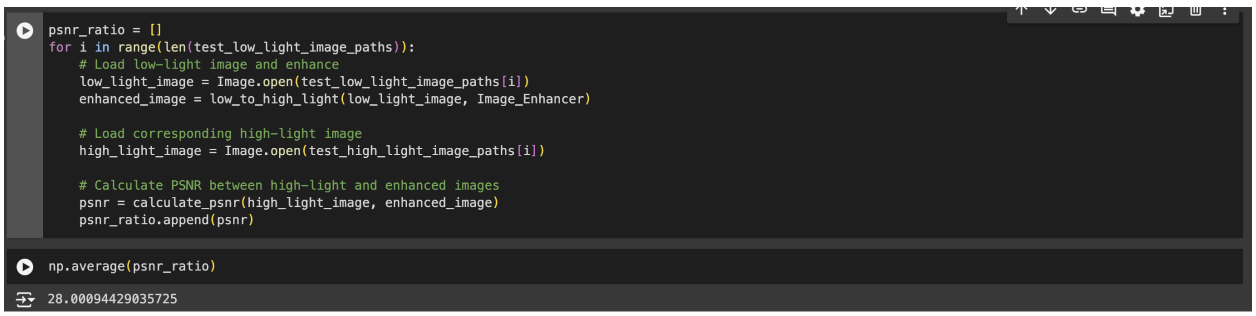
$$\mathrm{PSNR} = 20 \log_{10}\left(\frac{\mathrm{MAX}_I}{\sqrt{\mathrm{MSE}}}\right)$$

Quantitative Results

The final losses and PSNR values achieved after 50 epochs of training are summarized in Table.

💡 PSNR calculated over the test dataset -

`28.00094429035725`



```
psnr_ratio = []
for i in range(len(test_low_light_image_paths)):
    # Load low-light image and enhance
    low_light_image = Image.open(test_low_light_image_paths[i])
    enhanced_image = low_to_high_light(low_light_image, Image_Enhancer)

    # Load corresponding high-light image
    high_light_image = Image.open(test_high_light_image_paths[i])

    # Calculate PSNR between high-light and enhanced images
    psnr = calculate_psnr(high_light_image, enhanced_image)
    psnr_ratio.append(psnr)
```

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

```
28.00094429035725
```

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