Denoising Using the Zero-DCE Model

Abstract

This project 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 provided 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

Loading and Processing the provided DATA

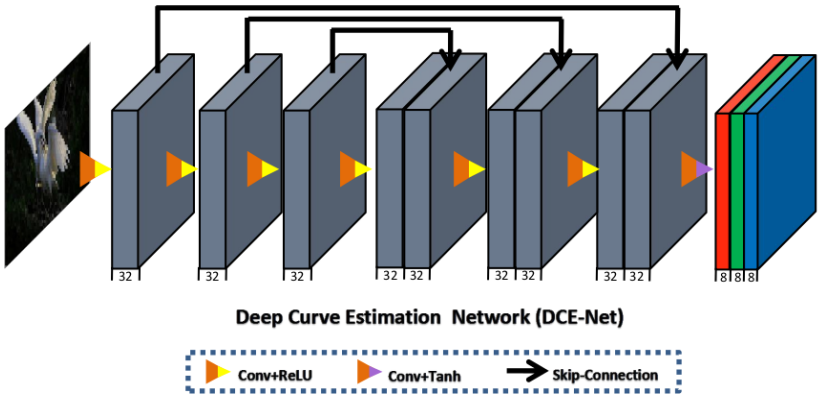
```
[ ]  def load_data(image_path):
         pic = tf.io.read_file(image_path)
         pic = tf.image.decode_png(pic, channels=3)
         pic = tf.image.resize(images=pic, size=[400, 400])
         pic = pic / 255.0  #scaling between [0,1]
         return pic
```

```
[ ]  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(16, drop_remainder=True)
         return data
```

The dataset was employed in both validation and training. 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 resized to 400 by 400 pixels. A 3:1 ratio was used to split the dataset into training and validation sets.

```
[ ]  # Load and sort image files for different datasets
     train_images_low_light = sorted(glob("/content/drive/MyDrive/Train/low/*"))[:300]
     validation_images_low_light = sorted(glob("/content/drive/MyDrive/Train/low/*"))[300:400]
     test_images_low_light = sorted(glob("/content/drive/MyDrive/Train/low/*"))[400:485]
     test_images_high_light = sorted(glob("/content/drive/MyDrive/Train/high/*"))[400:485]

     # Generate datasets using the data generator
     train_set = data_generator(train_images_low_light)
     validation_set = data_generator(validation_images_low_light)

     # Print the datasets
     print("Training Dataset:", train_set)
     print("Validation Dataset:", validation_set)
```

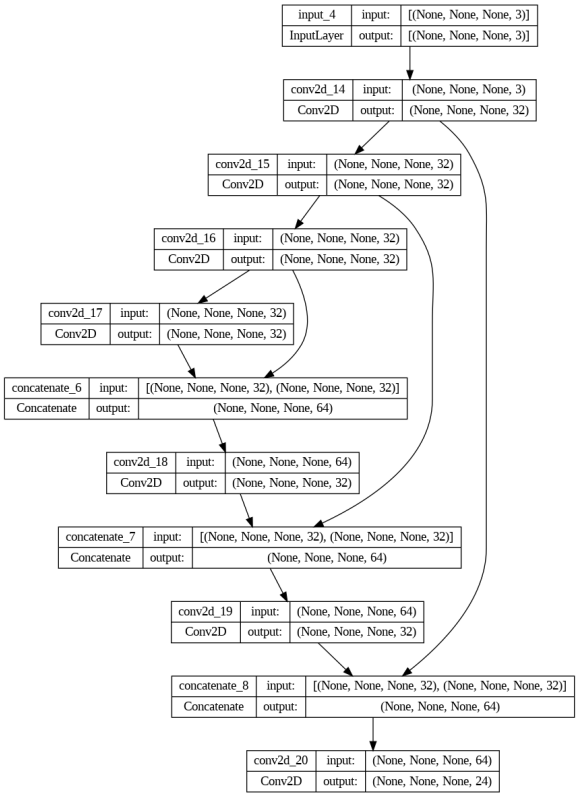
Model Architecture



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 Create_DCE_Network():
    input_image = Input(shape=[None, None, 3])
    layer1 = Conv2D(128, (3, 3), strides=(1, 1), activation="relu", padding="same")(input_image)
    layer2 = Conv2D(128, (3, 3), strides=(1, 1), activation="relu", padding="same")(layer1)
    layer3 = Conv2D(64, (3, 3), strides=(1, 1), activation="relu", padding="same")(layer2)
    layer4 = Conv2D(64, (3, 3), strides=(1, 1), activation="relu", padding="same")(layer3)

    concat1 = Concatenate(axis=-1)([layer4, layer3])
    layer5 = Conv2D(32, (3, 3), strides=(1, 1), activation="relu", padding="same")(concat1)

    concat2 = Concatenate(axis=-1)([layer5, layer2])
    layer6 = Conv2D(32, (3, 3), strides=(1, 1), activation="relu", padding="same")(concat2)

    concat3 = Concatenate(axis=-1)([layer6, layer1])
    output_layer = Conv2D(24, (3, 3), strides=(1, 1), activation="tanh", padding="same")(concat3)

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

The simplified architecture of the model shown with concatenations:



Our defined LOSS FUNCTIONS

The training procedure was guided by three bespoke loss functions: color balance loss, brightness loss and smoothness loss:

Brightness Control Loss (Exposure)

```
def brightness_loss(input_image, target_level=0.6):
    avg_intensity = tf.reduce_mean(input_image, axis=3, keepdims=True)
    pooled_mean = tf.nn.avg_pool2d(avg_intensity, ksize=16, strides=16, padding="VALID")
    loss_exposure = tf.reduce_mean(tf.square(pooled_mean - target_level))
    return loss_exposure
```

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

This ensures that the enhanced image has an appropriate exposure level.

FORMULA:

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

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.

Color Balance Loss

The implementation for the color balance loss function is as follows:

```python
def color_balance_loss(input_image):

    mean_channels = tf.reduce_mean(input_image, axis=(1, 2), keepdims=True)

    mean_red, mean_green, mean_blue = (
        mean_channels[:, :, :, 0],  # Mean value for the red channel
        mean_channels[:, :, :, 1],  # Mean value for the green channel
        mean_channels[:, :, :, 2],  # Mean value for the blue channel
    )
    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)

    # Compute the color constancy loss as the square root of the sum of squared differences
    loss_color_balance = tf.sqrt(tf.square(diff_red_green) + tf.square(diff_red_blue) + tf.square(diff_green_blue))

    return loss_color_balance
```

By doing this, it is guaranteed that the colors in the improved image match those in the original.

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

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

Smoothness Loss

Following is the implementation of smoothness loss functions

```python
def smoothness_loss(input_tensor):
    num_batches = tf.shape(input_tensor)[0]
    height = tf.shape(input_tensor)[1]
    width = tf.shape(input_tensor)[2]
    num_h_elements = (width - 1) * tf.shape(input_tensor)[3]
    num_w_elements = width * (tf.shape(input_tensor)[3] - 1)
    vertical_diff = tf.reduce_sum(tf.square(input_tensor[:, 1:, :, :] - input_tensor[:, :height - 1, :, :]))
    horizontal_diff = tf.reduce_sum(tf.square(input_tensor[:, :, 1:, :] - input_tensor[:, :, :width - 1, :]))
    num_batches = tf.cast(num_batches, dtype=tf.float32)
    num_h_elements = tf.cast(num_h_elements, dtype=tf.float32)
    num_w_elements = tf.cast(num_w_elements, dtype=tf.float32)
    return 2 * (vertical_diff / num_h_elements + horizontal_diff / num_w_elements) / num_batches
```

To maintain the monotonicity relations among adjacent pixels, we incorporate an illumination smoothness loss into every 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 A stands for the enhancement curves, ∇x and ∇y for the horizontal and vertical gradient operations, and N is the number of iterations.

Evaluation Metrics Given: PSNR Ratio

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:

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

Below is the summary of the final losses and PSNR values attained after 40 training epochs:

```
def compute_psnr(original_image, enhanced_image):
    original_image = np.array(original_image)
    enhanced_image = np.array(enhanced_image)
    mse = np.mean((original_image - enhanced_image) ** 2)
    psnr = 10 * np.log10(255 ** 2 / mse)
    return psnr
```

```
psnr_ratio = []
for i in range(len(test_images_low_light)):
    low_light_image = Image.open(test_images_low_light[i])
    enhanced_image = enhance_image(low_light_image)
    psnr = compute_psnr(low_light_image, enhanced_image)
    psnr_ratio.append(psnr)
```

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

```
27.966147730734242
```

PSNR - `27.966147730734242`

Final Results

The improved pictures showed notable gains in contrast, color accuracy, and all-around aesthetic appeal. The model successfully preserved the natural aspect of the scenes while improving the visibility of details in low light.

References

https://openaccess.thecvf.com/content_CVPR_2020/papers/Guo_Zero-Reference_Deep_Curve_Estimation_for_Low-Light_Image_Enhancement_CVPR_2020_paper.pdf

- Akshant Prakash

  21119004, Q7