low-light-enhancement

November 20, 2024

Zero-Reference Deep Curve Estimation or **Zero-DCE** formulates **low-light image enhancement** as the task of estimating an image-specific **tonal curve** with a deep neural network. Zero-DCE takes a low-light image as input and produces high-order tonal curves as its output. These curves are then used for pixel-wise adjustment on the dynamic range of the input to obtain an enhanced image. The curve estimation process is done in such a way that it maintains the range of the enhanced image and preserves the contrast of neighboring pixels. This curve estimation is inspired by curves adjustment used in photo editing software such as Adobe Photoshop where users can adjust points throughout an image's tonal range.

Zero-DCE is appealing because of its relaxed assumptions with regard to reference images: it does not require any input/output image pairs during training. This is achieved through a set of carefully formulated non-reference loss functions, which implicitly measure the enhancement quality and guide the training of the network.

In this notebook, we will train a lightweight deep network, **DCE-Net**, to estimate pixel-wise and high-order tonal curves for dynamic range adjustment of a given image using **Tensorflow**.

[2]:
```
from IPython.display import clear_output

!pip install -q tensorflow==2.4.1

clear_output()
```

[3]:
```
!pip install wandb
```

```
Requirement already satisfied: wandb in /opt/conda/lib/python3.10/site-packages
(0.18.3)
Requirement already satisfied: click!=8.0.0,>=7.1 in
/opt/conda/lib/python3.10/site-packages (from wandb) (8.1.7)
Requirement already satisfied: docker-pycreds>=0.4.0 in
/opt/conda/lib/python3.10/site-packages (from wandb) (0.4.0)
Requirement already satisfied: gitpython!=3.1.29,>=1.0.0 in
/opt/conda/lib/python3.10/site-packages (from wandb) (3.1.43)
Requirement already satisfied: platformdirs in /opt/conda/lib/python3.10/site-
packages (from wandb) (3.11.0)
Requirement already satisfied: protobuf!=4.21.0,!=5.28.0,<6,>=3.19.0 in
/opt/conda/lib/python3.10/site-packages (from wandb) (3.20.3)
Requirement already satisfied: psutil>=5.0.0 in /opt/conda/lib/python3.10/site-
packages (from wandb) (5.9.3)
```

```
Requirement already satisfied: pyyaml in /opt/conda/lib/python3.10/site-packages
(from wandb) (6.0.2)
Requirement already satisfied: requests<3,>=2.0.0 in
/opt/conda/lib/python3.10/site-packages (from wandb) (2.32.3)
Requirement already satisfied: sentry-sdk>=1.0.0 in
/opt/conda/lib/python3.10/site-packages (from wandb) (2.15.0)
Requirement already satisfied: setproctitle in /opt/conda/lib/python3.10/site-
packages (from wandb) (1.3.3)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.10/site-
packages (from wandb) (70.0.0)
Requirement already satisfied: six>=1.4.0 in /opt/conda/lib/python3.10/site-
packages (from docker-pycreds>=0.4.0->wandb) (1.16.0)
Requirement already satisfied: gitdb<5,>=4.0.1 in
/opt/conda/lib/python3.10/site-packages (from gitpython!=3.1.29,>=1.0.0->wandb)
(4.0.11)
Requirement already satisfied: charset-normalizer<4,>=2 in
/opt/conda/lib/python3.10/site-packages (from requests<3,>=2.0.0->wandb) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-
packages (from requests<3,>=2.0.0->wandb) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.10/site-packages (from requests<3,>=2.0.0->wandb)
(1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.10/site-packages (from requests<3,>=2.0.0->wandb)
(2024.8.30)
Requirement already satisfied: smmap<6,>=3.0.1 in
/opt/conda/lib/python3.10/site-packages (from
gitdb<5,>=4.0.1->gitpython!=3.1.29,>=1.0.0->wandb) (5.0.1)
```

[4]:
```python
import os
import random
import numpy as np
from glob import glob
from PIL import Image, ImageOps
from kaggle_secrets import UserSecretsClient

import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid

import wandb
from wandb.integration.keras import WandbCallback


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, losses, optimizers, callbacks
```

```
AUTOTUNE = tf.data.AUTOTUNE
```

[5]:
```
# Experiment Configs

SEED = 10
random.seed(SEED)
tf.random.set_seed(SEED)


IMAGE_SIZE = 256
MAX_TRAIN_IMAGES = 400
MAX_VAL_IMAGES = 485
BATCH_SIZE = 16
TRAIN_VAL_IMAGE_DIR = "/kaggle/input/lol-dataset/lol_dataset/our485/low"
TEST_IMAGE_DIR = "/kaggle/input/dark-face-dataset/image"
LEARNING_RATE = 1e-4
LOG_INTERVALS = 10
EPOCHS = 60
```

[6]:
```
train_val_image_files = glob(os.path.join(TRAIN_VAL_IMAGE_DIR, "*.png"))
test_image_files = glob(os.path.join(TEST_IMAGE_DIR, "*.png"))
train_image_files = train_val_image_files[:MAX_TRAIN_IMAGES]
val_image_files = train_val_image_files[MAX_TRAIN_IMAGES:MAX_VAL_IMAGES]

print("Number of Training Images:", len(train_image_files))
print("Number of Validation Images:", len(val_image_files))
print("Number of Test Images from LOL Dataset:", len(test_image_files))
```

```
Number of Training Images: 400
Number of Validation Images: 85
Number of Test Images from LOL Dataset: 6000
```

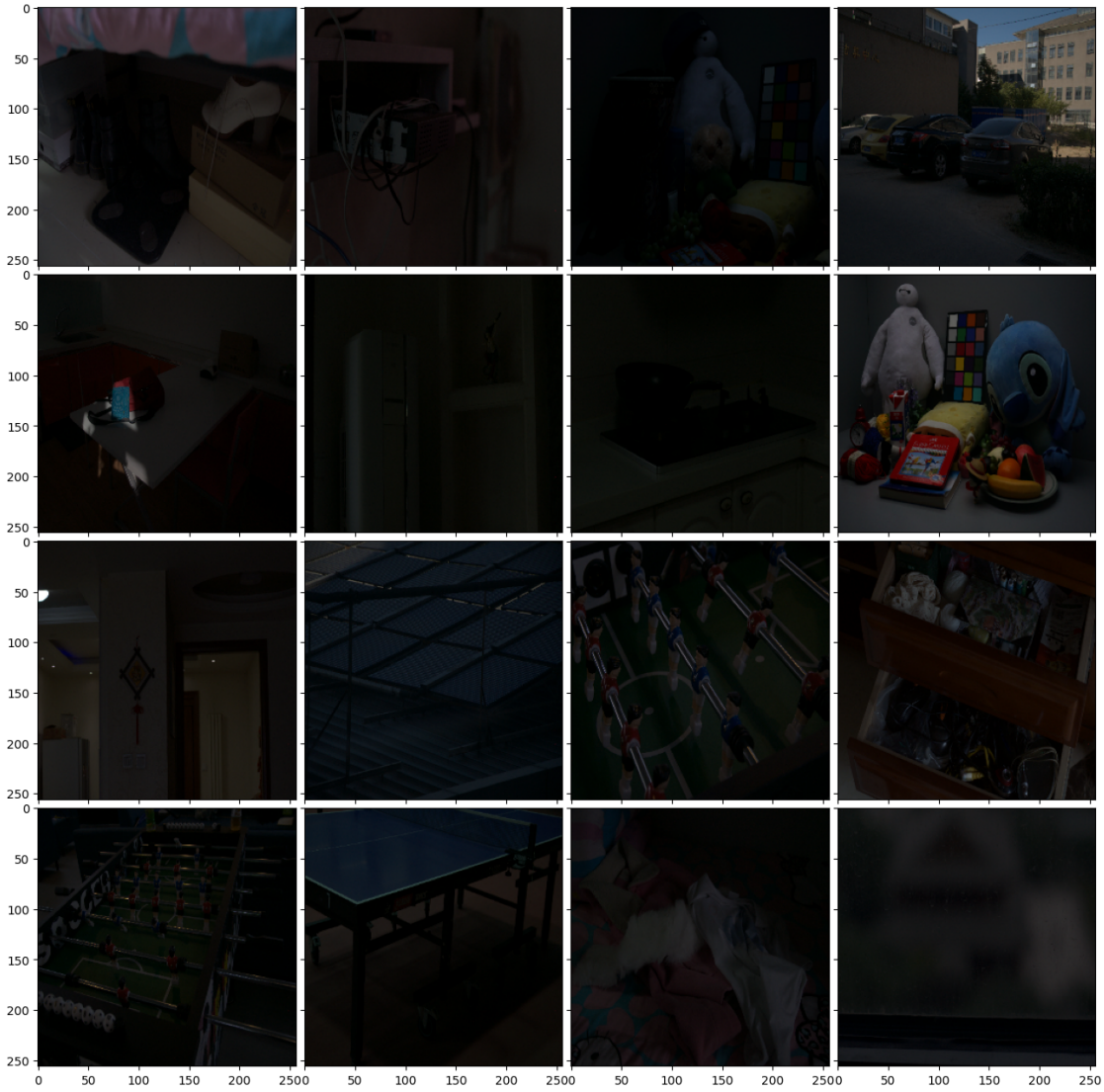
[7]:
```
def load_data(image_path):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_png(image, channels=3)
    image = tf.image.resize(images=image, size=[IMAGE_SIZE, IMAGE_SIZE])
    image = image / 255.0
    return image


def get_dataset(images):
    dataset = tf.data.Dataset.from_tensor_slices((images))
    dataset = dataset.map(load_data, num_parallel_calls=AUTOTUNE)
    dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)
    dataset = dataset.prefetch(AUTOTUNE)
```

```
    return dataset


train_dataset = get_dataset(train_image_files)
val_dataset = get_dataset(val_image_files)
```

[8]:
```
print("\033[94m")
print("Train Data Elements:", train_dataset.element_spec)
print("Validation Data Elements:", val_dataset.element_spec)
```

```
Train Data Elements: TensorSpec(shape=(16, 256, 256, 3), dtype=tf.float32,
name=None)
Validation Data Elements: TensorSpec(shape=(16, 256, 256, 3), dtype=tf.float32,
name=None)
```

[9]:
```
images = next(iter(train_dataset)).numpy()
fig = plt.figure(figsize=(16, 16))
grid = ImageGrid(fig, 111, nrows_ncols=(4, 4), axes_pad=0.1)
random_images = images[np.random.choice(np.arange(images.shape[0]), 16)]
for ax, image in zip(grid, images):
    image = image * 255.0
    ax.imshow(image.astype(np.uint8))
plt.title("Sample Images from Tiny-NeRF Data")
plt.show()
```



Zero-DCE can cope with diverse lighting conditions including nonuniform and poor lighting cases for low-light image enhancement. Instead of performing image-to-image mapping, in the case of Zero-DCE, the problem is reformulated as an image-specific curve estimation problem. In particular, the proposed method takes a low-light image as input and produces high-order curves as its output. These curves are then used for pixel-wise adjustment on the dynamic range of the input to obtain an enhanced image. A unique advantage of this approach is that its **zero-reference**, i.e., **it does not require any paired or even unpaired data** in the training process as in existing CNN-based and GAN-based methods.

0.1 Understanding light-enhancement curves

A ligh-enhancement curve is a kind of curve that can map a low-light image to its enhanced version automatically, where the self-adaptive curve parameters are solely dependent on the input image. When designing such a curve, three objectives should be taken into account:

- Each pixel value of the enhanced image should be in the normalized range `[0,1]`, in order to avoid information loss induced by overflow truncation.
- It should be monotonous, to preserve the contrast between neighboring pixels.
- The shape of this curve should be as simple as possible, and the curve should be differentiable to allow backpropagation.

The light-enhancement curve is separately applied to three RGB channels instead of solely on the illumination channel. The three-channel adjustment can better preserve the inherent color and reduce the risk of over-saturation.

1 The Zero-DCE Net

The **DCE-Net** is a lightweight deep neural network that learns the mapping between an input image and its best-fitting curve parameter maps. The input to the DCE-Net is a low-light image while the outputs are a set of pixel-wise curve parameter maps for corresponding higher-order curves. It is a plain CNN of seven convolutional layers with symmetrical concatenation. Each layer consists of 32 convolutional kernels of size 3×3 and stride 1 followed by the ReLU activation function. The last convolutional layer is followed by the Tanh activation function, which produces 24 parameter maps for 8 iterations, where each iteration requires three curve parameter maps for the three channels.

[10]:
```
def build_dce_net(image_size=None) -> keras.Model:
    input_image = keras.Input(shape=[image_size, image_size, 3])
    conv1 = layers.Conv2D(
        32, (3, 3), strides=(1, 1), activation="relu", padding="same"
    )(input_image)
    conv2 = layers.Conv2D(
        32, (3, 3), strides=(1, 1), activation="relu", padding="same"
    )(conv1)
    conv3 = layers.Conv2D(
        32, (3, 3), strides=(1, 1), activation="relu", padding="same"
    )(conv2)
    conv4 = layers.Conv2D(
        32, (3, 3), strides=(1, 1), activation="relu", padding="same"
    )(conv3)
    int_con1 = layers.Concatenate(axis=-1)([conv4, conv3])
    conv5 = layers.Conv2D(
        32, (3, 3), strides=(1, 1), activation="relu", padding="same"
    )(int_con1)
    int_con2 = layers.Concatenate(axis=-1)([conv5, conv2])
    conv6 = layers.Conv2D(
        32, (3, 3), strides=(1, 1), activation="relu", padding="same"
    )(int_con2)
    int_con3 = layers.Concatenate(axis=-1)([conv6, conv1])
    x_r = layers.Conv2D(24, (3, 3), strides=(1, 1), activation="tanh",␣
↪padding="same")(
        int_con3
    )
```

```
    return keras.Model(inputs=input_image, outputs=x_r)
```

2 Non-Reference Loss Functions

To enable zero-reference learning in DCE-Net, the Zero-DCE framework employs a set of differentiable non-reference losses that allow us to evaluate the quality of enhanced images. By defining a non-reference loss function that **does not requires a ground truth image** allows us to train their model on the low-light images only. We would discuss and implement the four types of losses are adopted by the Zero-DCE framework to train the DCE-Net.

2.1 Color Constancy Loss

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

[11]:
```
def color_constancy_loss(x):
    mean_rgb = tf.reduce_mean(x, axis=(1, 2), keepdims=True)
    mean_red = mean_rgb[:, :, :, 0]
    mean_green = mean_rgb[:, :, :, 1]
    mean_blue = mean_rgb[:, :, :, 2]
    diff_red_green = tf.square(mean_red - mean_green)
    diff_red_blue = tf.square(mean_red - mean_blue)
    diff_green_blue = tf.square(mean_blue - mean_green)
    return tf.sqrt(
        tf.square(diff_red_green) + tf.square(diff_red_blue) + tf.
↪square(diff_green_blue)
    )
```

2.2 Exposure Control Loss

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

[12]:
```
def exposure_loss(x, mean_val=0.6):
    x = tf.reduce_mean(x, axis=3, keepdims=True)
    mean = tf.nn.avg_pool2d(x, ksize=16, strides=16, padding="VALID")
    return tf.reduce_mean(tf.square(mean - mean_val))
```

2.3 Illumination Smoothness Loss

In order to preserve the monotonicity relations between neighboring pixels, the Zero-DCE frameworkd employs an illumination smoothness loss to each curve parameter map. The Illumination Smoothness Loss for a given curve parameter map $\mathcal{A}$ is given by:

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

where ∇_x and ∇_y represent the horizontal and vertical gradient operations, respectively.

[13]:
```
def illumination_smoothness_loss(x):
    batch_size = tf.shape(x)[0]
    height_x = tf.shape(x)[1]
    width_x = tf.shape(x)[2]
    count_height = (tf.shape(x)[2] - 1) * tf.shape(x)[3]
    count_width = tf.shape(x)[2] * (tf.shape(x)[3] - 1)
    height_total_variance = tf.reduce_sum(
        tf.square((x[:, 1:, :, :] - x[:, : height_x - 1, :, :]))
    )
    width_total_variance = tf.reduce_sum(
        tf.square((x[:, :, 1:, :] - x[:, :, : width_x - 1, :]))
    )
    batch_size = tf.cast(batch_size, dtype=tf.float32)
    count_height = tf.cast(count_height, dtype=tf.float32)
    count_width = tf.cast(count_width, dtype=tf.float32)
    return 2 * (
        height_total_variance / count_height + width_total_variance /␣
↪count_width
    ) / batch_size
```

2.4 Spatial Constancy Loss

The spatial consistency loss encourages spatial coherence of the enhanced image through preserving the difference of neighboring regions between the input image and its enhanced version. The Spatial Constancy Loss is given by:

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

where

- $\Omega(i)$ is the four neighboring regions (top, down, left, right) centered at the region i
- Y and I denote the average intensity value of the local region in the enhanced version and input image, respectively

[14]:
```
class SpatialConsistencyLoss(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
        )

    def call(self, y_true, y_pred):

        original_mean = tf.reduce_mean(y_true, 3, keepdims=True)
        enhanced_mean = tf.reduce_mean(y_pred, 3, keepdims=True)
        original_pool = tf.nn.avg_pool2d(
            original_mean, ksize=4, strides=4, padding="VALID"
        )
        enhanced_pool = tf.nn.avg_pool2d(
            enhanced_mean, ksize=4, strides=4, padding="VALID"
        )

        d_original_left = tf.nn.conv2d(
            original_pool, self.left_kernel, strides=[1, 1, 1, 1],␣
↪padding="SAME"
        )
        d_original_right = tf.nn.conv2d(
            original_pool, self.right_kernel, strides=[1, 1, 1, 1],␣
↪padding="SAME"
        )
        d_original_up = tf.nn.conv2d(
            original_pool, self.up_kernel, strides=[1, 1, 1, 1], padding="SAME"
        )
        d_original_down = tf.nn.conv2d(
            original_pool, self.down_kernel, strides=[1, 1, 1, 1],␣
↪padding="SAME"
        )

        d_enhanced_left = tf.nn.conv2d(
            enhanced_pool, self.left_kernel, strides=[1, 1, 1, 1],␣
↪padding="SAME"
        )
        d_enhanced_right = tf.nn.conv2d(
            enhanced_pool, self.right_kernel, strides=[1, 1, 1, 1],␣
↪padding="SAME"
        )
        d_enhanced_up = tf.nn.conv2d(
```

```
            enhanced_pool, self.up_kernel, strides=[1, 1, 1, 1], padding="SAME"
        )
        d_enhanced_down = tf.nn.conv2d(
            enhanced_pool, self.down_kernel, strides=[1, 1, 1, 1],␣
↪padding="SAME"
        )

        d_left = tf.square(d_original_left - d_enhanced_left)
        d_right = tf.square(d_original_right - d_enhanced_right)
        d_up = tf.square(d_original_up - d_enhanced_up)
        d_down = tf.square(d_original_down - d_enhanced_down)
        return d_left + d_right + d_up + d_down
```

[15]:
```
class ZeroDCE(keras.Model):
    def __init__(self, **kwargs):
        super(ZeroDCE, self).__init__(**kwargs)
        self.dce_model = build_dce_net()

    def compile(self, learning_rate, **kwargs):
        super(ZeroDCE, self).compile(**kwargs)
        self.optimizer = optimizers.Adam(learning_rate=learning_rate)
        self.spatial_constancy_loss = SpatialConsistencyLoss(reduction="none")

    def summary(self, *args, **kwargs):
        self.dce_model.summary(*args, **kwargs)

    def get_enhanced_image(self, data, output):
        x = data
        for i in range(0, 3 * 8, 3):
            r = output[:, :, :, i: i + 3]
            x = x + r * (tf.square(x) - x)
        return x

    def call(self, data):
        dce_net_output = self.dce_model(data)
        return self.get_enhanced_image(data, dce_net_output)

    def compute_losses(self, data, output):
        enhanced_image = self.get_enhanced_image(data, output)
        loss_illumination = 200 * illumination_smoothness_loss(output)
        loss_spatial_constancy = tf.reduce_mean(
            self.spatial_constancy_loss(enhanced_image, data)
        )
        loss_color_constancy = 5 * tf.
↪reduce_mean(color_constancy_loss(enhanced_image))
        loss_exposure = 10 * tf.reduce_mean(exposure_loss(enhanced_image))
        total_loss = (
```

```
            loss_illumination
            + loss_spatial_constancy
            + loss_color_constancy
            + loss_exposure
        )
        return {
            "total_loss": total_loss,
            "illumination_smoothness_loss": loss_illumination,
            "spatial_constancy_loss": loss_spatial_constancy,
            "color_constancy_loss": loss_color_constancy,
            "exposure_loss": loss_exposure,
        }

    def train_step(self, data):
        with tf.GradientTape() as tape:
            output = self.dce_model(data)
            losses = self.compute_losses(data, output)
        gradients = tape.gradient(
            losses["total_loss"], self.dce_model.trainable_weights
        )
        self.optimizer.apply_gradients(zip(gradients, self.dce_model.
↪trainable_weights))
        return losses

    def test_step(self, data):
        output = self.dce_model(data)
        return self.compute_losses(data, output)

    def save_weights(self, filepath, overwrite=True, save_format=None,␣
↪options=None):
        self.dce_model.save_weights(
            filepath, overwrite=overwrite, save_format=save_format,␣
↪options=options
        )

    def load_weights(self, filepath, by_name=False, skip_mismatch=False,␣
↪options=None):
        self.dce_model.load_weights(
            filepath=filepath,
            by_name=by_name,
            skip_mismatch=skip_mismatch,
            options=options,
        )
```

[16]:
```
zero_dce_model = ZeroDCE()
zero_dce_model.summary()
```

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, None, None, 3)	0	-
conv2d (Conv2D)	(None, None, None, 32)	896	input_layer[0][0]
conv2d_1 (Conv2D)	(None, None, None, 32)	9,248	conv2d[0][0]
conv2d_2 (Conv2D)	(None, None, None, 32)	9,248	conv2d_1[0][0]
conv2d_3 (Conv2D)	(None, None, None, 32)	9,248	conv2d_2[0][0]
concatenate (Concatenate)	(None, None, None, 64)	0	conv2d_3[0][0], conv2d_2[0][0]
conv2d_4 (Conv2D)	(None, None, None, 32)	18,464	concatenate[0][0]
concatenate_1 (Concatenate)	(None, None, None, 64)	0	conv2d_4[0][0], conv2d_1[0][0]
conv2d_5 (Conv2D)	(None, None, None, 32)	18,464	concatenate_1[0]…
concatenate_2 (Concatenate)	(None, None, None, 64)	0	conv2d_5[0][0], conv2d[0][0]
conv2d_6 (Conv2D)	(None, None, None, 24)	13,848	concatenate_2[0]…

```
Total params: 79,416 (310.22 KB)

Trainable params: 79,416 (310.22 KB)

Non-trainable params: 0 (0.00 B)
```

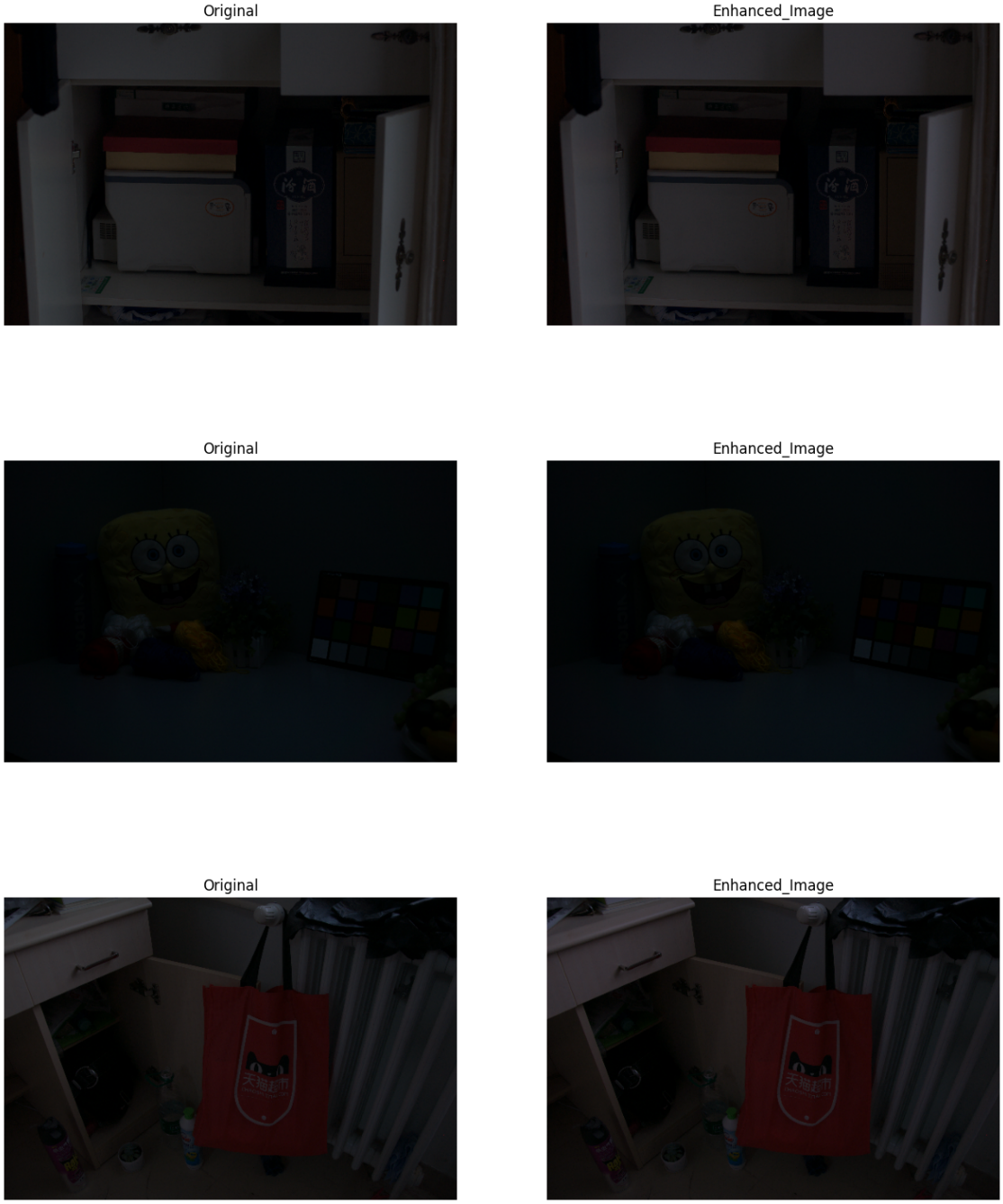
[17]:
```
def plot_results(images, titles, figure_size=(12, 12)):
    fig = plt.figure(figsize=figure_size)
    for i in range(len(images)):
        fig.add_subplot(1, len(images), i + 1).set_title(titles[i])
        _ = plt.imshow(images[i])
        plt.axis("off")
    plt.show()


def infer(original_image):
    image = keras.preprocessing.image.img_to_array(original_image)
    image = image[:, :, :3] if image.shape[-1] > 3 else image
    image = image.astype("float32") / 255.0
    image = np.expand_dims(image, axis=0)
    output_image = zero_dce_model(image)
    output_image = tf.cast((output_image[0, :, :, :] * 255), dtype=np.uint8)
    output_image = Image.fromarray(output_image.numpy())
    return output_image
```

[18]:
```
class LogPredictionCallback(callbacks.Callback):

    def __init__(self, image_files, log_interval, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.image_files = image_files
        self.log_interval = log_interval

    def on_epoch_end(self, epoch, logs=None):
        if epoch % self.log_interval == 0:
            for image_file in self.image_files:
                original_image = Image.open(image_file)
                enhanced_image = infer(original_image)
                plot_results(
                    [original_image, enhanced_image],
                    ["Original", "Enhanced_Image"],
                    (15, 7),
                )
```

[19]:
```
zero_dce_model.compile(learning_rate=LEARNING_RATE)
history = zero_dce_model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=EPOCHS,
    callbacks=[
        LogPredictionCallback(
            image_files=random.sample(val_image_files, 4),
            log_interval=LOG_INTERVALS
        )
```

```
    ]
)
```

```
Epoch 1/60

WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1732016858.105189     122 service.cc:145] XLA service 0x7cf9300020d0
initialized for platform CUDA (this does not guarantee that XLA will be used).
Devices:
I0000 00:00:1732016858.105263     122 service.cc:153]   StreamExecutor device
(0): Tesla T4, Compute Capability 7.5
I0000 00:00:1732016858.105269     122 service.cc:153]   StreamExecutor device
(1): Tesla T4, Compute Capability 7.5

 1/25            5:53 15s/step -
color_constancy_loss: 0.0023 - exposure_loss: 2.9000 -
illumination_smoothness_loss: 3.6360 - spatial_constancy_loss: 8.6163e-06 -
total_loss: 6.5382

I0000 00:00:1732016870.420218     122 device_compiler.h:188] Compiled cluster
using XLA!  This line is logged at most once for the lifetime of the process.

25/25            0s 178ms/step -
color_constancy_loss: 0.0029 - exposure_loss: 2.9412 -
illumination_smoothness_loss: 1.8975 - spatial_constancy_loss: 1.0272e-05 -
total_loss: 4.8417
```




```
25/25              23s 344ms/step -
color_constancy_loss: 0.0029 - exposure_loss: 2.9359 -
illumination_smoothness_loss: 1.8728 - spatial_constancy_loss: 1.0553e-05 -
total_loss: 4.8115 - val_color_constancy_loss: 8.4301e-04 - val_exposure_loss:
2.9163 - val_illumination_smoothness_loss: 1.0878 - val_spatial_constancy_loss:
1.2486e-05 - val_total_loss: 4.0050
Epoch 2/60
```

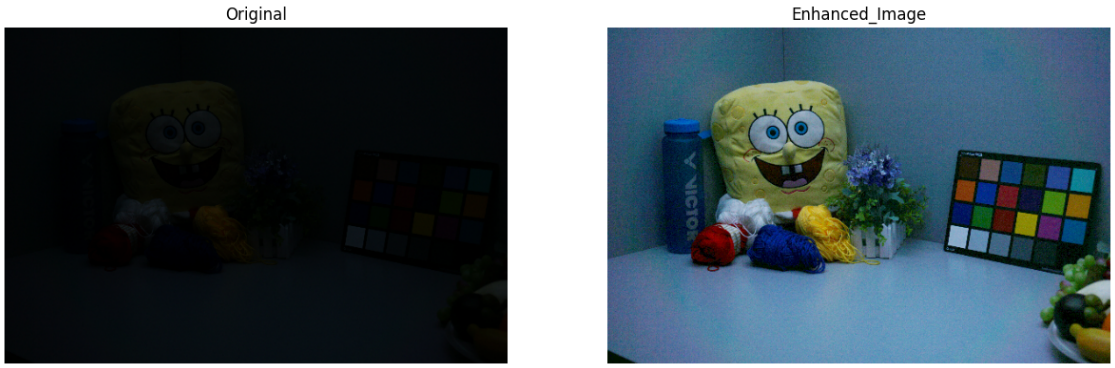
```
25/25            5s 192ms/step -
color_constancy_loss: 0.0027 - exposure_loss: 2.9237 -
illumination_smoothness_loss: 1.0883 - spatial_constancy_loss: 3.2339e-05 -
total_loss: 4.0148 - val_color_constancy_loss: 8.2698e-04 - val_exposure_loss:
2.9004 - val_illumination_smoothness_loss: 0.7038 - val_spatial_constancy_loss:
4.1918e-05 - val_total_loss: 3.6051
Epoch 3/60
25/25            5s 194ms/step -
color_constancy_loss: 0.0028 - exposure_loss: 2.9099 -
illumination_smoothness_loss: 0.7480 - spatial_constancy_loss: 6.3217e-05 -
total_loss: 3.6607 - val_color_constancy_loss: 8.3499e-04 - val_exposure_loss:
2.8876 - val_illumination_smoothness_loss: 0.4913 - val_spatial_constancy_loss:
6.8371e-05 - val_total_loss: 3.3798
Epoch 4/60
25/25            5s 194ms/step -
color_constancy_loss: 0.0028 - exposure_loss: 2.8976 -
illumination_smoothness_loss: 0.5387 - spatial_constancy_loss: 9.1071e-05 -
total_loss: 3.4393 - val_color_constancy_loss: 8.6481e-04 - val_exposure_loss:
2.8744 - val_illumination_smoothness_loss: 0.3592 - val_spatial_constancy_loss:
1.0130e-04 - val_total_loss: 3.2345
Epoch 5/60
25/25            5s 195ms/step -
color_constancy_loss: 0.0029 - exposure_loss: 2.8835 -
illumination_smoothness_loss: 0.4070 - spatial_constancy_loss: 1.3537e-04 -
total_loss: 3.2935 - val_color_constancy_loss: 9.0593e-04 - val_exposure_loss:
2.8581 - val_illumination_smoothness_loss: 0.2759 - val_spatial_constancy_loss:
1.5505e-04 - val_total_loss: 3.1351
Epoch 6/60
25/25            5s 197ms/step -
color_constancy_loss: 0.0030 - exposure_loss: 2.8664 -
illumination_smoothness_loss: 0.3202 - spatial_constancy_loss: 2.0411e-04 -
total_loss: 3.1898 - val_color_constancy_loss: 9.6219e-04 - val_exposure_loss:
2.8393 - val_illumination_smoothness_loss: 0.2189 - val_spatial_constancy_loss:
2.3468e-04 - val_total_loss: 3.0594
Epoch 7/60
25/25            5s 198ms/step -
color_constancy_loss: 0.0031 - exposure_loss: 2.8474 -
illumination_smoothness_loss: 0.2591 - spatial_constancy_loss: 2.9878e-04 -
total_loss: 3.1099 - val_color_constancy_loss: 0.0010 - val_exposure_loss:
2.8182 - val_illumination_smoothness_loss: 0.1780 - val_spatial_constancy_loss:
3.4350e-04 - val_total_loss: 2.9976
Epoch 8/60
25/25            5s 198ms/step -
color_constancy_loss: 0.0033 - exposure_loss: 2.8262 -
illumination_smoothness_loss: 0.2167 - spatial_constancy_loss: 4.2373e-04 -
total_loss: 3.0465 - val_color_constancy_loss: 0.0011 - val_exposure_loss:
2.7954 - val_illumination_smoothness_loss: 0.1511 - val_spatial_constancy_loss:
4.8445e-04 - val_total_loss: 2.9480
```

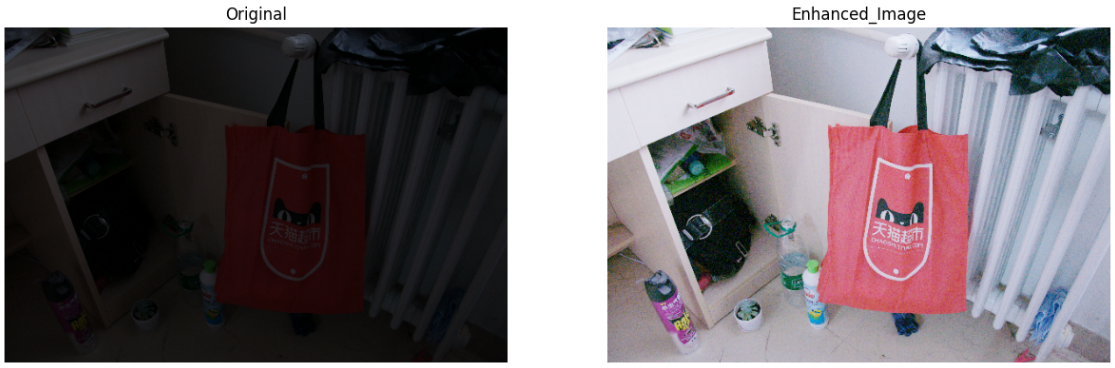
```
Epoch 9/60
25/25             5s 199ms/step -
color_constancy_loss: 0.0035 - exposure_loss: 2.8034 -
illumination_smoothness_loss: 0.1868 - spatial_constancy_loss: 5.7941e-04 -
total_loss: 2.9942 - val_color_constancy_loss: 0.0012 - val_exposure_loss:
2.7711 - val_illumination_smoothness_loss: 0.1322 - val_spatial_constancy_loss:
6.5746e-04 - val_total_loss: 2.9051
Epoch 10/60
25/25             5s 199ms/step -
color_constancy_loss: 0.0037 - exposure_loss: 2.7788 -
illumination_smoothness_loss: 0.1652 - spatial_constancy_loss: 7.7574e-04 -
total_loss: 2.9486 - val_color_constancy_loss: 0.0013 - val_exposure_loss:
2.7449 - val_illumination_smoothness_loss: 0.1192 - val_spatial_constancy_loss:
8.7707e-04 - val_total_loss: 2.8662
Epoch 11/60
25/25             0s 189ms/step -
color_constancy_loss: 0.0041 - exposure_loss: 2.7573 -
illumination_smoothness_loss: 0.1516 - spatial_constancy_loss: 0.0010 -
total_loss: 2.9139
```


```
25/25            7s 298ms/step -
color_constancy_loss: 0.0040 - exposure_loss: 2.7502 -
illumination_smoothness_loss: 0.1509 - spatial_constancy_loss: 0.0010 -
total_loss: 2.9061 - val_color_constancy_loss: 0.0014 - val_exposure_loss:
2.7114 - val_illumination_smoothness_loss: 0.1124 - val_spatial_constancy_loss:
0.0012 - val_total_loss: 2.8263
Epoch 12/60
25/25            5s 201ms/step -
color_constancy_loss: 0.0045 - exposure_loss: 2.7108 -
illumination_smoothness_loss: 0.1435 - spatial_constancy_loss: 0.0015 -
total_loss: 2.8603 - val_color_constancy_loss: 0.0016 - val_exposure_loss:
2.6610 - val_illumination_smoothness_loss: 0.1122 - val_spatial_constancy_loss:
0.0018 - val_total_loss: 2.7767
Epoch 13/60
25/25            5s 202ms/step -
color_constancy_loss: 0.0053 - exposure_loss: 2.6439 -
illumination_smoothness_loss: 0.1450 - spatial_constancy_loss: 0.0025 -
total_loss: 2.7967 - val_color_constancy_loss: 0.0022 - val_exposure_loss:
2.5627 - val_illumination_smoothness_loss: 0.1225 - val_spatial_constancy_loss:
0.0034 - val_total_loss: 2.6908
Epoch 14/60
```

```
25/25            5s 202ms/step -
color_constancy_loss: 0.0078 - exposure_loss: 2.4772 -
illumination_smoothness_loss: 0.1649 - spatial_constancy_loss: 0.0062 -
total_loss: 2.6561 - val_color_constancy_loss: 0.0045 - val_exposure_loss:
2.2676 - val_illumination_smoothness_loss: 0.1538 - val_spatial_constancy_loss:
0.0114 - val_total_loss: 2.4373
Epoch 15/60
25/25            5s 204ms/step -
color_constancy_loss: 0.0228 - exposure_loss: 1.8738 -
illumination_smoothness_loss: 0.2303 - spatial_constancy_loss: 0.0414 -
total_loss: 2.1684 - val_color_constancy_loss: 0.0225 - val_exposure_loss:
1.2226 - val_illumination_smoothness_loss: 0.2247 - val_spatial_constancy_loss:
0.1041 - val_total_loss: 1.5739
Epoch 16/60
25/25            5s 205ms/step -
color_constancy_loss: 0.0702 - exposure_loss: 0.8558 -
illumination_smoothness_loss: 0.2673 - spatial_constancy_loss: 0.2270 -
total_loss: 1.4203 - val_color_constancy_loss: 0.0495 - val_exposure_loss:
0.6330 - val_illumination_smoothness_loss: 0.2454 - val_spatial_constancy_loss:
0.3004 - val_total_loss: 1.2283
Epoch 17/60
25/25            5s 206ms/step -
color_constancy_loss: 0.0746 - exposure_loss: 0.7328 -
illumination_smoothness_loss: 0.2169 - spatial_constancy_loss: 0.2726 -
total_loss: 1.2969 - val_color_constancy_loss: 0.0461 - val_exposure_loss:
0.6733 - val_illumination_smoothness_loss: 0.1810 - val_spatial_constancy_loss:
0.2731 - val_total_loss: 1.1735
Epoch 18/60
25/25            5s 206ms/step -
color_constancy_loss: 0.0745 - exposure_loss: 0.7342 -
illumination_smoothness_loss: 0.1866 - spatial_constancy_loss: 0.2693 -
total_loss: 1.2646 - val_color_constancy_loss: 0.0475 - val_exposure_loss:
0.6551 - val_illumination_smoothness_loss: 0.1665 - val_spatial_constancy_loss:
0.2843 - val_total_loss: 1.1533
Epoch 19/60
25/25            5s 211ms/step -
color_constancy_loss: 0.0774 - exposure_loss: 0.7062 -
illumination_smoothness_loss: 0.0671 - spatial_constancy_loss: 0.2885 -
total_loss: 1.1393 - val_color_constancy_loss: 0.0495 - val_exposure_loss:
0.6286 - val_illumination_smoothness_loss: 0.0554 - val_spatial_constancy_loss:
0.3032 - val_total_loss: 1.0367
Epoch 38/60
25/25            5s 214ms/step -
color_constancy_loss: 0.0774 - exposure_loss: 0.7059 -
illumination_smoothness_loss: 0.0648 - spatial_constancy_loss: 0.2888 -
total_loss: 1.1370 - val_color_constancy_loss: 0.0497 - val_exposure_loss:
0.6269 - val_illumination_smoothness_loss: 0.0526 - val_spatial_constancy_loss:
0.3046 - val_total_loss: 1.0338
```

```
Epoch 39/60
25/25             5s 215ms/step -
color_constancy_loss: 0.0774 - exposure_loss: 0.7058 -
illumination_smoothness_loss: 0.0619 - spatial_constancy_loss: 0.2892 -
total_loss: 1.1343 - val_color_constancy_loss: 0.0502 - val_exposure_loss:
0.6215 - val_illumination_smoothness_loss: 0.0517 - val_spatial_constancy_loss:
0.3087 - val_total_loss: 1.0321
Epoch 40/60
25/25             5s 216ms/step -
color_constancy_loss: 0.0776 - exposure_loss: 0.7044 -
illumination_smoothness_loss: 0.0578 - spatial_constancy_loss: 0.2900 -
total_loss: 1.1299 - val_color_constancy_loss: 0.0500 - val_exposure_loss:
0.6236 - val_illumination_smoothness_loss: 0.0487 - val_spatial_constancy_loss:
0.3068 - val_total_loss: 1.0291
Epoch 41/60
25/25             0s 203ms/step -
color_constancy_loss: 0.0788 - exposure_loss: 0.7048 -
illumination_smoothness_loss: 0.0555 - spatial_constancy_loss: 0.2878 -
total_loss: 1.1269
```






```
25/25              7s 302ms/step -
color_constancy_loss: 0.0776 - exposure_loss: 0.7036 -
illumination_smoothness_loss: 0.0556 - spatial_constancy_loss: 0.2904 -
total_loss: 1.1272 - val_color_constancy_loss: 0.0502 - val_exposure_loss:
0.6215 - val_illumination_smoothness_loss: 0.0464 - val_spatial_constancy_loss:
0.3085 - val_total_loss: 1.0266
Epoch 42/60
25/25              5s 214ms/step -
color_constancy_loss: 0.0776 - exposure_loss: 0.7035 -
illumination_smoothness_loss: 0.0526 - spatial_constancy_loss: 0.2905 -
total_loss: 1.1242 - val_color_constancy_loss: 0.0504 - val_exposure_loss:
0.6193 - val_illumination_smoothness_loss: 0.0450 - val_spatial_constancy_loss:
0.3097 - val_total_loss: 1.0245
Epoch 43/60
25/25              5s 213ms/step -
color_constancy_loss: 0.0778 - exposure_loss: 0.7020 -
illumination_smoothness_loss: 0.0499 - spatial_constancy_loss: 0.2914 -
total_loss: 1.1212 - val_color_constancy_loss: 0.0504 - val_exposure_loss:
0.6196 - val_illumination_smoothness_loss: 0.0424 - val_spatial_constancy_loss:
0.3097 - val_total_loss: 1.0221
Epoch 44/60
```

```
25/25            5s 214ms/step -
color_constancy_loss: 0.0779 - exposure_loss: 0.7015 -
illumination_smoothness_loss: 0.0475 - spatial_constancy_loss: 0.2917 -
total_loss: 1.1185 - val_color_constancy_loss: 0.0505 - val_exposure_loss:
0.6180 - val_illumination_smoothness_loss: 0.0406 - val_spatial_constancy_loss:
0.3107 - val_total_loss: 1.0200
Epoch 45/60
25/25            5s 214ms/step -
color_constancy_loss: 0.0780 - exposure_loss: 0.7004 -
illumination_smoothness_loss: 0.0453 - spatial_constancy_loss: 0.2923 -
total_loss: 1.1160 - val_color_constancy_loss: 0.0505 - val_exposure_loss:
0.6181 - val_illumination_smoothness_loss: 0.0384 - val_spatial_constancy_loss:
0.3105 - val_total_loss: 1.0175
Epoch 46/60
25/25            5s 214ms/step -
color_constancy_loss: 0.0781 - exposure_loss: 0.6997 -
illumination_smoothness_loss: 0.0435 - spatial_constancy_loss: 0.2926 -
total_loss: 1.1139 - val_color_constancy_loss: 0.0505 - val_exposure_loss:
0.6182 - val_illumination_smoothness_loss: 0.0366 - val_spatial_constancy_loss:
0.3103 - val_total_loss: 1.0156
Epoch 47/60
25/25            5s 213ms/step -
color_constancy_loss: 0.0781 - exposure_loss: 0.6992 -
illumination_smoothness_loss: 0.0421 - spatial_constancy_loss: 0.2928 -
total_loss: 1.1123 - val_color_constancy_loss: 0.0508 - val_exposure_loss:
0.6151 - val_illumination_smoothness_loss: 0.0359 - val_spatial_constancy_loss:
0.3125 - val_total_loss: 1.0143
Epoch 48/60
25/25            5s 212ms/step -
color_constancy_loss: 0.0782 - exposure_loss: 0.6980 -
illumination_smoothness_loss: 0.0402 - spatial_constancy_loss: 0.2935 -
total_loss: 1.1099 - val_color_constancy_loss: 0.0505 - val_exposure_loss:
0.6181 - val_illumination_smoothness_loss: 0.0334 - val_spatial_constancy_loss:
0.3097 - val_total_loss: 1.0116
Epoch 49/60
25/25            5s 214ms/step -
color_constancy_loss: 0.0782 - exposure_loss: 0.6979 -
illumination_smoothness_loss: 0.0396 - spatial_constancy_loss: 0.2933 -
total_loss: 1.1091 - val_color_constancy_loss: 0.0509 - val_exposure_loss:
0.6134 - val_illumination_smoothness_loss: 0.0334 - val_spatial_constancy_loss:
0.3133 - val_total_loss: 1.0110
Epoch 50/60
25/25            10s 210ms/step -
color_constancy_loss: 0.0784 - exposure_loss: 0.6966 -
illumination_smoothness_loss: 0.0373 - spatial_constancy_loss: 0.2940 -
total_loss: 1.1063 - val_color_constancy_loss: 0.0506 - val_exposure_loss:
0.6163 - val_illumination_smoothness_loss: 0.0310 - val_spatial_constancy_loss:
0.3105 - val_total_loss: 1.0084
```

```
Epoch 51/60
25/25             0s 199ms/step -
color_constancy_loss: 0.0796 - exposure_loss: 0.6973 -
illumination_smoothness_loss: 0.0368 - spatial_constancy_loss: 0.2914 -
total_loss: 1.1050
```

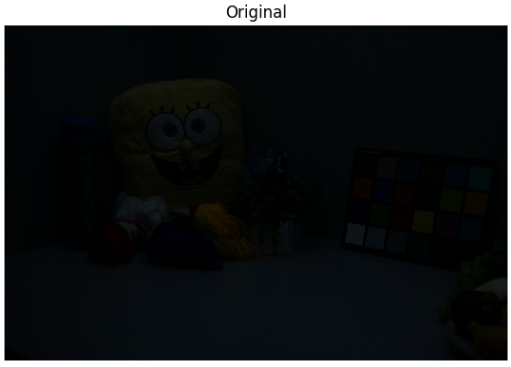
Original

Enhanced_Image

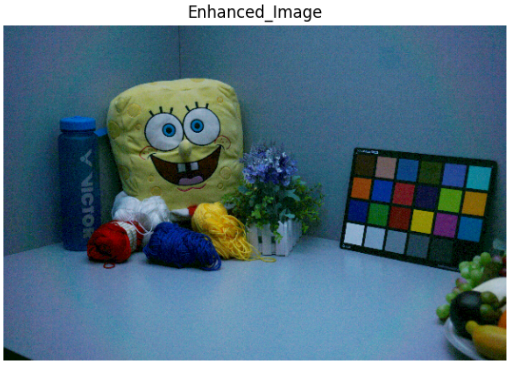
Original

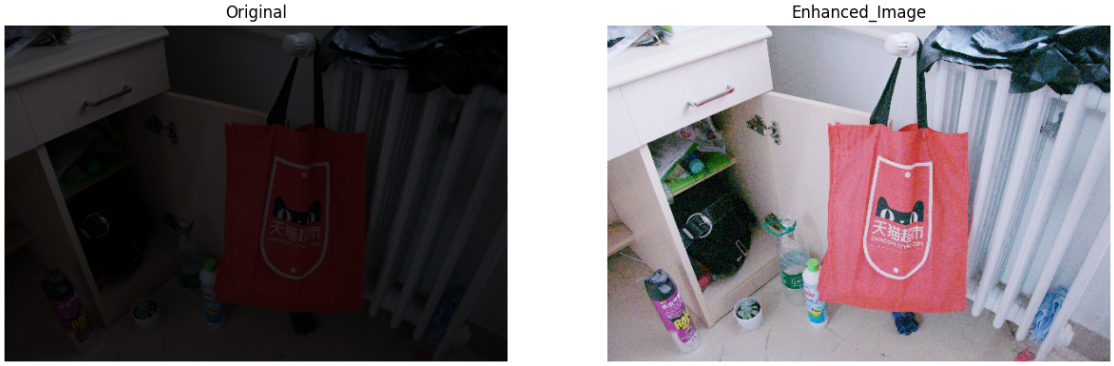
Enhanced_Image

Original



Enhanced_Image



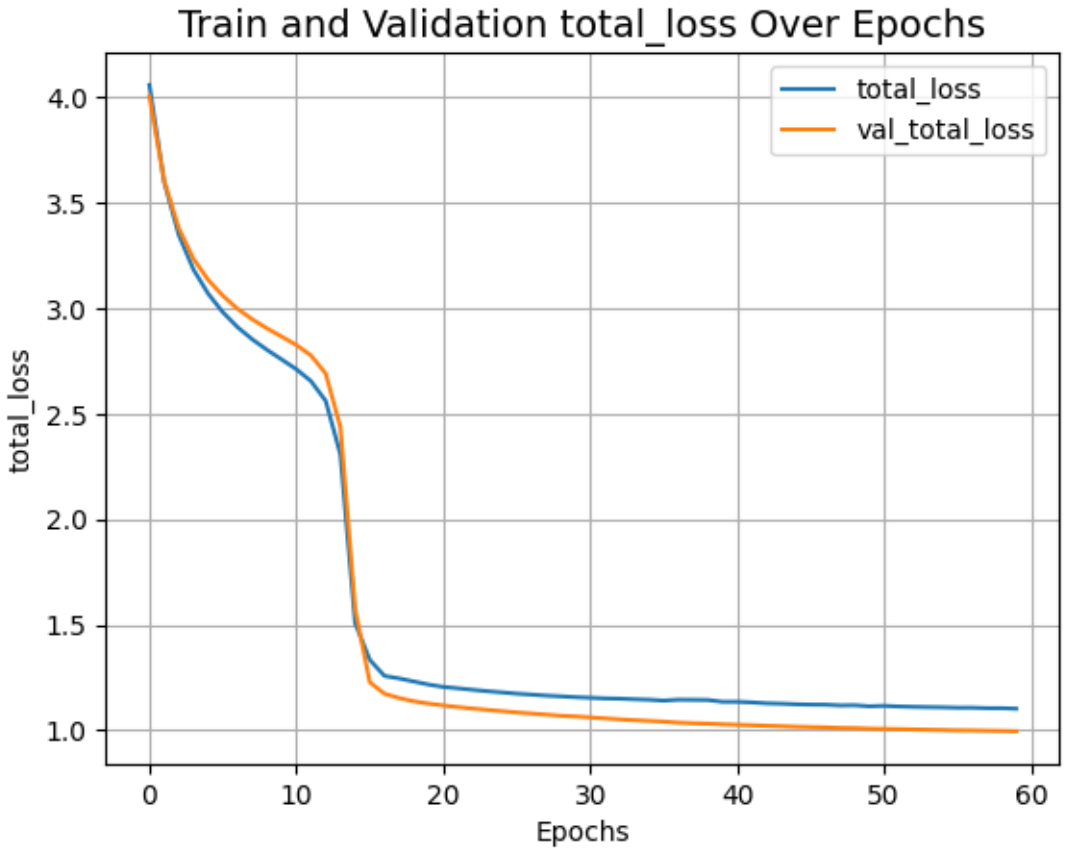


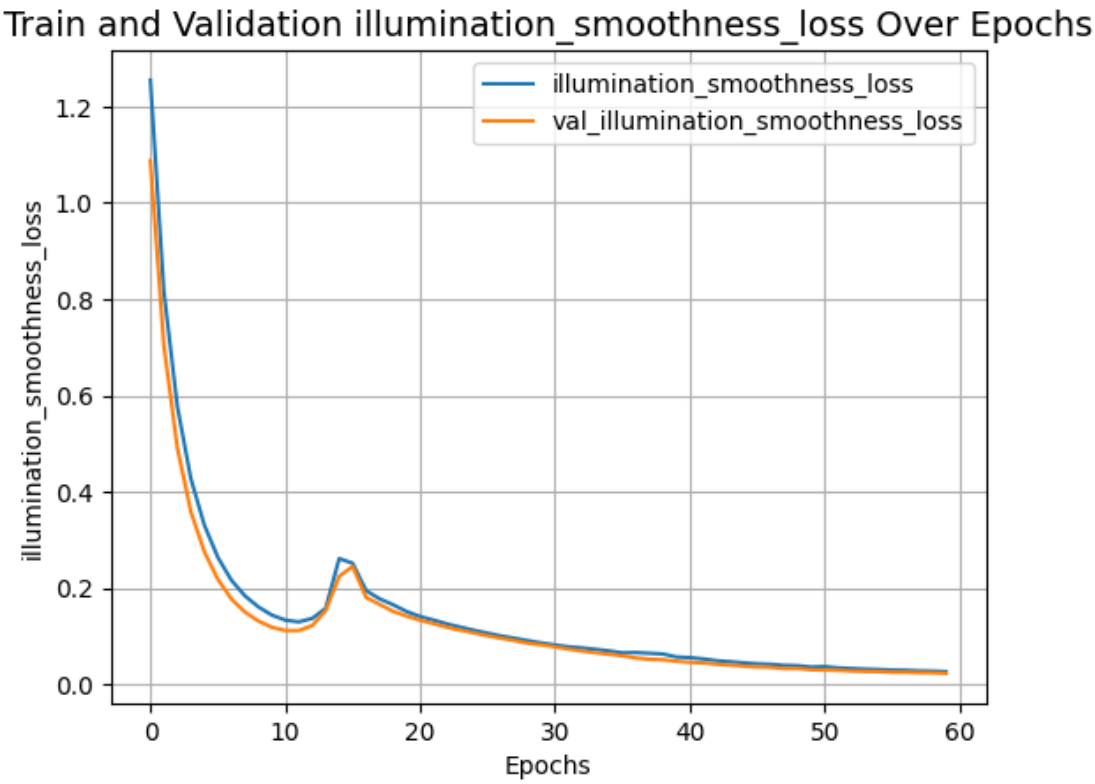

```
25/25            7s 298ms/step -
color_constancy_loss: 0.0784 - exposure_loss: 0.6962 -
illumination_smoothness_loss: 0.0368 - spatial_constancy_loss: 0.2940 -
total_loss: 1.1054 - val_color_constancy_loss: 0.0508 - val_exposure_loss:
0.6138 - val_illumination_smoothness_loss: 0.0303 - val_spatial_constancy_loss:
0.3123 - val_total_loss: 1.0072
Epoch 52/60
25/25            5s 211ms/step -
color_constancy_loss: 0.0784 - exposure_loss: 0.6958 -
illumination_smoothness_loss: 0.0353 - spatial_constancy_loss: 0.2941 -
total_loss: 1.1036 - val_color_constancy_loss: 0.0510 - val_exposure_loss:
0.6120 - val_illumination_smoothness_loss: 0.0297 - val_spatial_constancy_loss:
0.3135 - val_total_loss: 1.0062
Epoch 53/60
25/25            5s 212ms/step -
color_constancy_loss: 0.0786 - exposure_loss: 0.6945 -
illumination_smoothness_loss: 0.0337 - spatial_constancy_loss: 0.2949 -
total_loss: 1.1017 - val_color_constancy_loss: 0.0511 - val_exposure_loss:
0.6116 - val_illumination_smoothness_loss: 0.0284 - val_spatial_constancy_loss:
0.3135 - val_total_loss: 1.0045
Epoch 54/60
25/25            5s 215ms/step -
color_constancy_loss: 0.0787 - exposure_loss: 0.6938 -
illumination_smoothness_loss: 0.0327 - spatial_constancy_loss: 0.2950 -
total_loss: 1.1002 - val_color_constancy_loss: 0.0511 - val_exposure_loss:
0.6108 - val_illumination_smoothness_loss: 0.0276 - val_spatial_constancy_loss:
0.3135 - val_total_loss: 1.0031
Epoch 55/60
25/25            5s 216ms/step -
color_constancy_loss: 0.0787 - exposure_loss: 0.6932 -
illumination_smoothness_loss: 0.0319 - spatial_constancy_loss: 0.2951 -
total_loss: 1.0989 - val_color_constancy_loss: 0.0513 - val_exposure_loss:
0.6091 - val_illumination_smoothness_loss: 0.0273 - val_spatial_constancy_loss:
0.3145 - val_total_loss: 1.0021
```

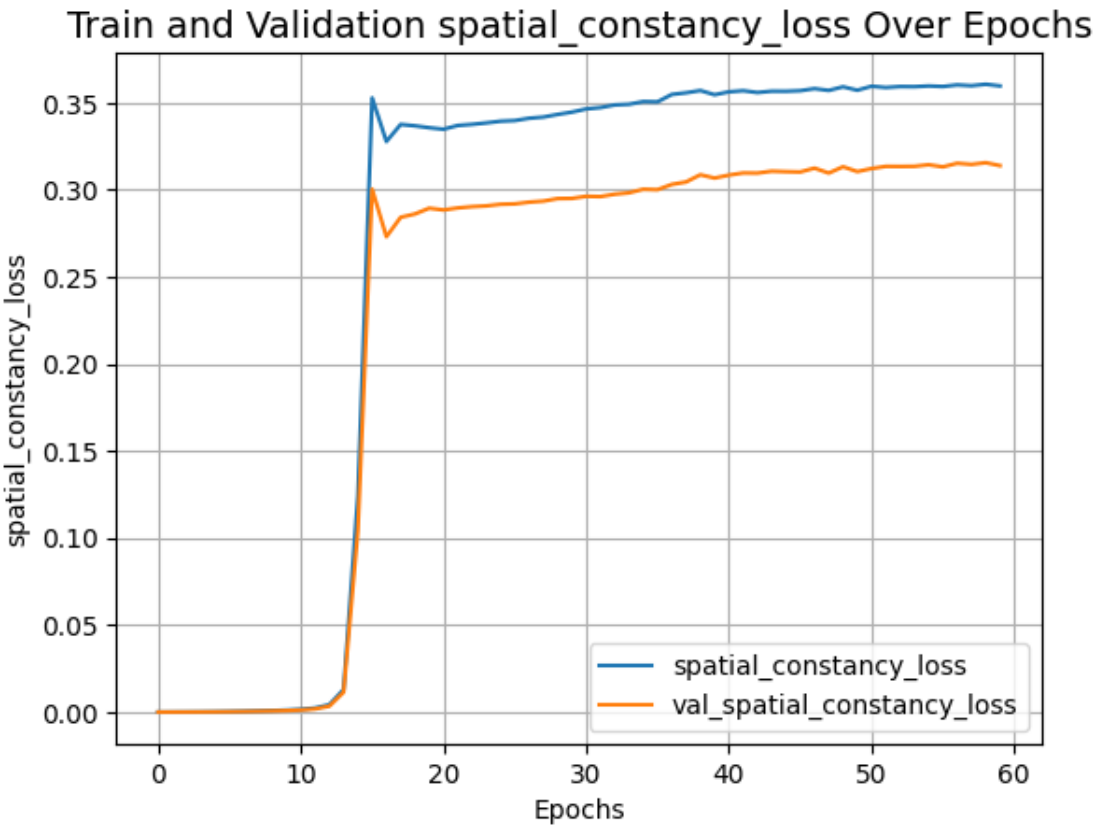
```
Epoch 56/60
25/25             5s 215ms/step -
color_constancy_loss: 0.0788 - exposure_loss: 0.6922 -
illumination_smoothness_loss: 0.0308 - spatial_constancy_loss: 0.2954 -
total_loss: 1.0973 - val_color_constancy_loss: 0.0512 - val_exposure_loss:
0.6101 - val_illumination_smoothness_loss: 0.0259 - val_spatial_constancy_loss:
0.3132 - val_total_loss: 1.0004
Epoch 57/60
25/25             5s 215ms/step -
color_constancy_loss: 0.0789 - exposure_loss: 0.6918 -
illumination_smoothness_loss: 0.0304 - spatial_constancy_loss: 0.2955 -
total_loss: 1.0965 - val_color_constancy_loss: 0.0514 - val_exposure_loss:
0.6072 - val_illumination_smoothness_loss: 0.0259 - val_spatial_constancy_loss:
0.3153 - val_total_loss: 0.9998
Epoch 58/60
25/25             5s 215ms/step -
color_constancy_loss: 0.0790 - exposure_loss: 0.6908 -
illumination_smoothness_loss: 0.0292 - spatial_constancy_loss: 0.2959 -
total_loss: 1.0948 - val_color_constancy_loss: 0.0514 - val_exposure_loss:
0.6075 - val_illumination_smoothness_loss: 0.0250 - val_spatial_constancy_loss:
0.3146 - val_total_loss: 0.9985
Epoch 59/60
25/25             5s 215ms/step -
color_constancy_loss: 0.0790 - exposure_loss: 0.6901 -
illumination_smoothness_loss: 0.0285 - spatial_constancy_loss: 0.2961 -
total_loss: 1.0938 - val_color_constancy_loss: 0.0516 - val_exposure_loss:
0.6057 - val_illumination_smoothness_loss: 0.0246 - val_spatial_constancy_loss:
0.3156 - val_total_loss: 0.9975
Epoch 60/60
25/25             5s 215ms/step -
color_constancy_loss: 0.0791 - exposure_loss: 0.6894 -
illumination_smoothness_loss: 0.0277 - spatial_constancy_loss: 0.2963 -
total_loss: 1.0925 - val_color_constancy_loss: 0.0514 - val_exposure_loss:
0.6073 - val_illumination_smoothness_loss: 0.0236 - val_spatial_constancy_loss:
0.3139 - val_total_loss: 0.9961
```

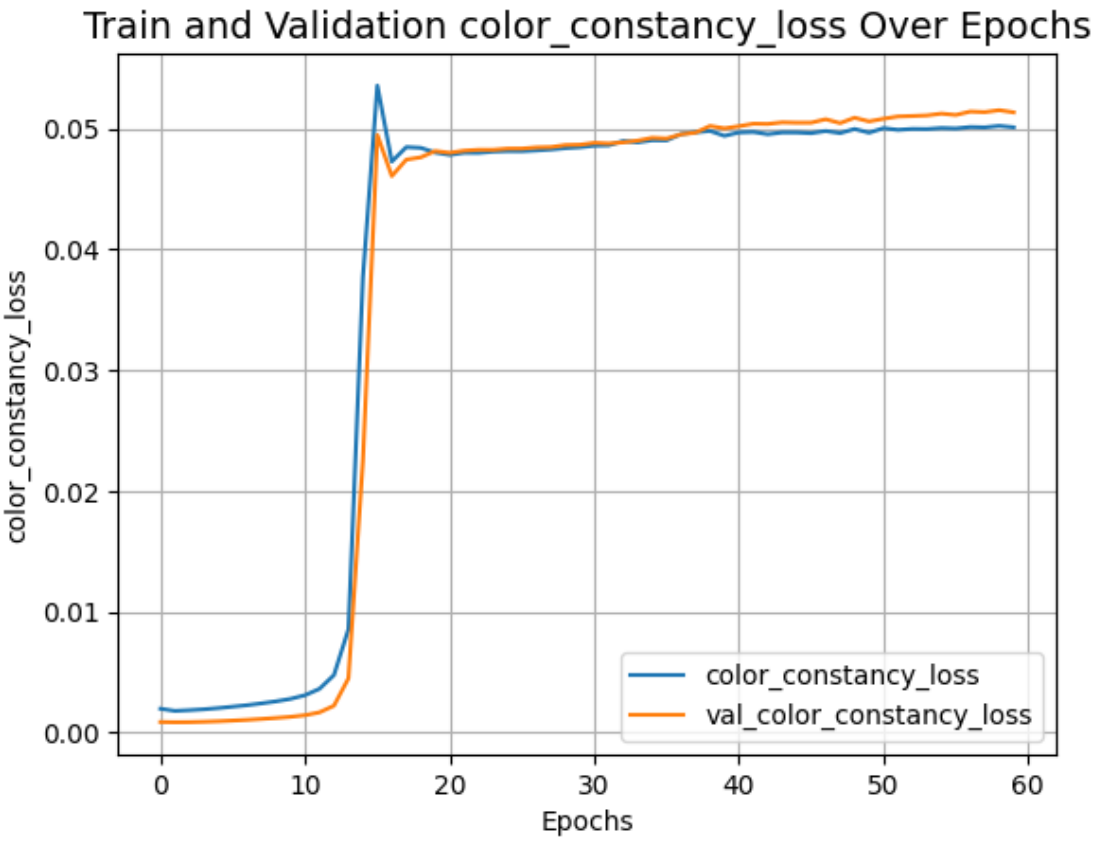
[20]:
```python
def plot_result(item):
    plt.plot(history.history[item], label=item)
    plt.plot(history.history["val_" + item], label="val_" + item)
    plt.xlabel("Epochs")
    plt.ylabel(item)
    plt.title("Train and Validation {} Over Epochs".format(item), fontsize=14)
    plt.legend()
    plt.grid()
    plt.show()
```

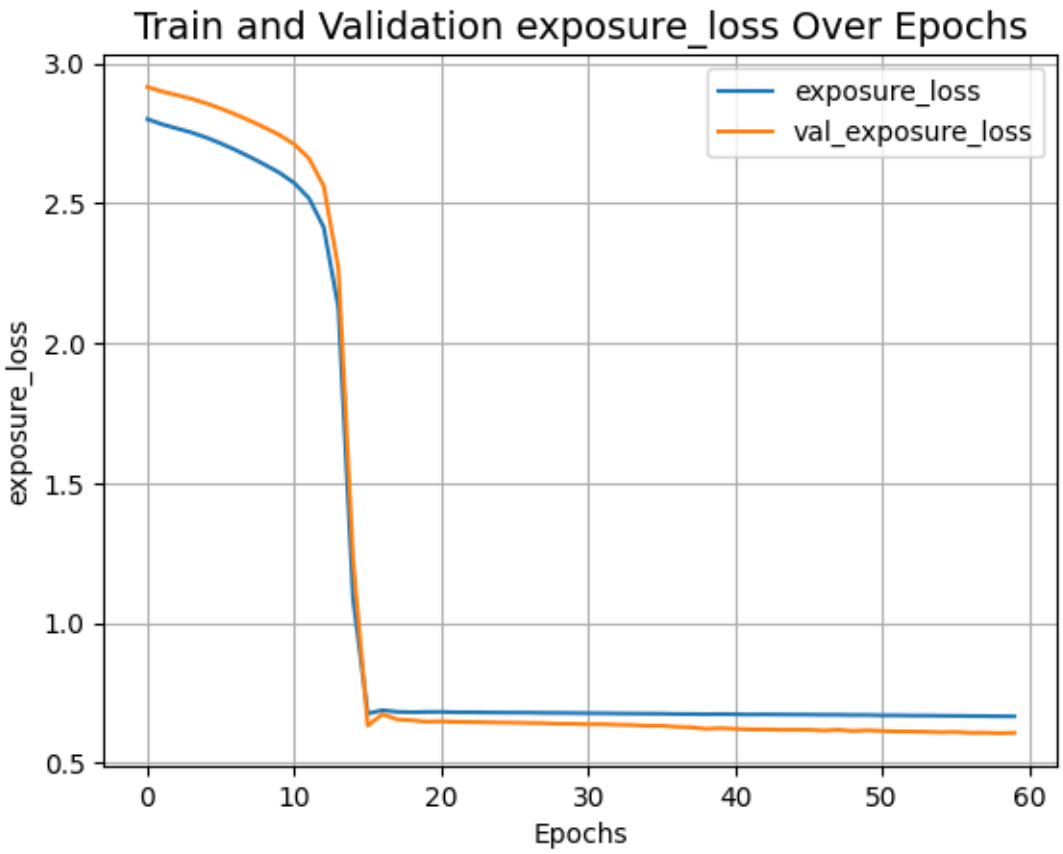
```
plot_result("total_loss")
plot_result("illumination_smoothness_loss")
plot_result("spatial_constancy_loss")
plot_result("color_constancy_loss")
plot_result("exposure_loss")
```




Train and Validation illumination_smoothness_loss Over Epochs
illumination_smoothness_loss
val_illumination_smoothness_loss
illumination_smoothness_loss
Epochs
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
10
20
30
40
50
60

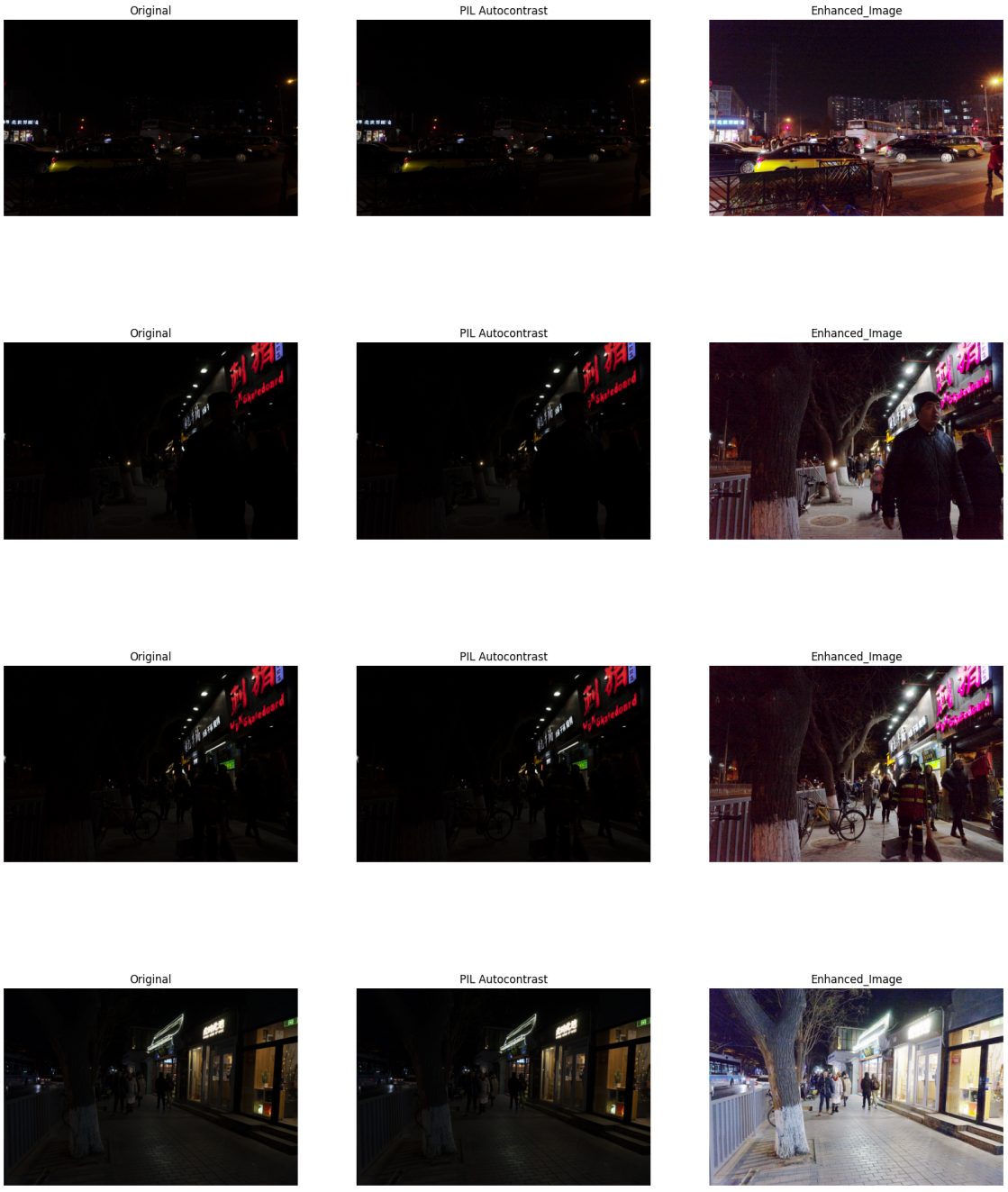

Train and Validation spatial_constancy_loss Over Epochs
spatial_constancy_loss
Epochs
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0
10
20
30
40
50
60
spatial_constancy_loss
val_spatial_constancy_loss


Train and Validation color_constancy_loss Over Epochs
color_constancy_loss
Epochs
0.00
0.01
0.02
0.03
0.04
0.05
0
10
20
30
40
50
60
color_constancy_loss
val_color_constancy_loss



[21]:
```
# for image_file in test_image_files:
#     original_image = Image.open(image_file)
#     enhanced_image = infer(original_image)
#     plot_results(
#         [original_image, ImageOps.autocontrast(original_image),␣
 ↪enhanced_image],
#         ["Original", "PIL Autocontrast", "Enhanced_Image"],
#         (20, 12),
#     )


from PIL import Image, ImageOps

# Counter to limit the number of processed images
max_images = 20
processed_count = 0

for image_file in test_image_files:
    if processed_count >= max_images:
        break
```

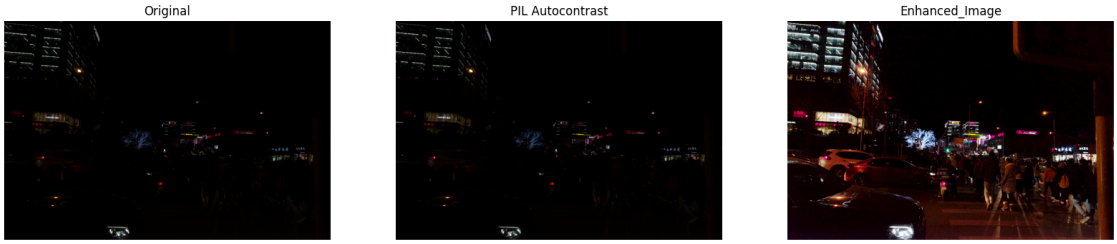
```
original_image = Image.open(image_file)
enhanced_image = infer(original_image)
plot_results(
    [original_image, ImageOps.autocontrast(original_image), enhanced_image],
    ["Original", "PIL Autocontrast", "Enhanced_Image"],
    (20, 12),
)

processed_count += 1
```

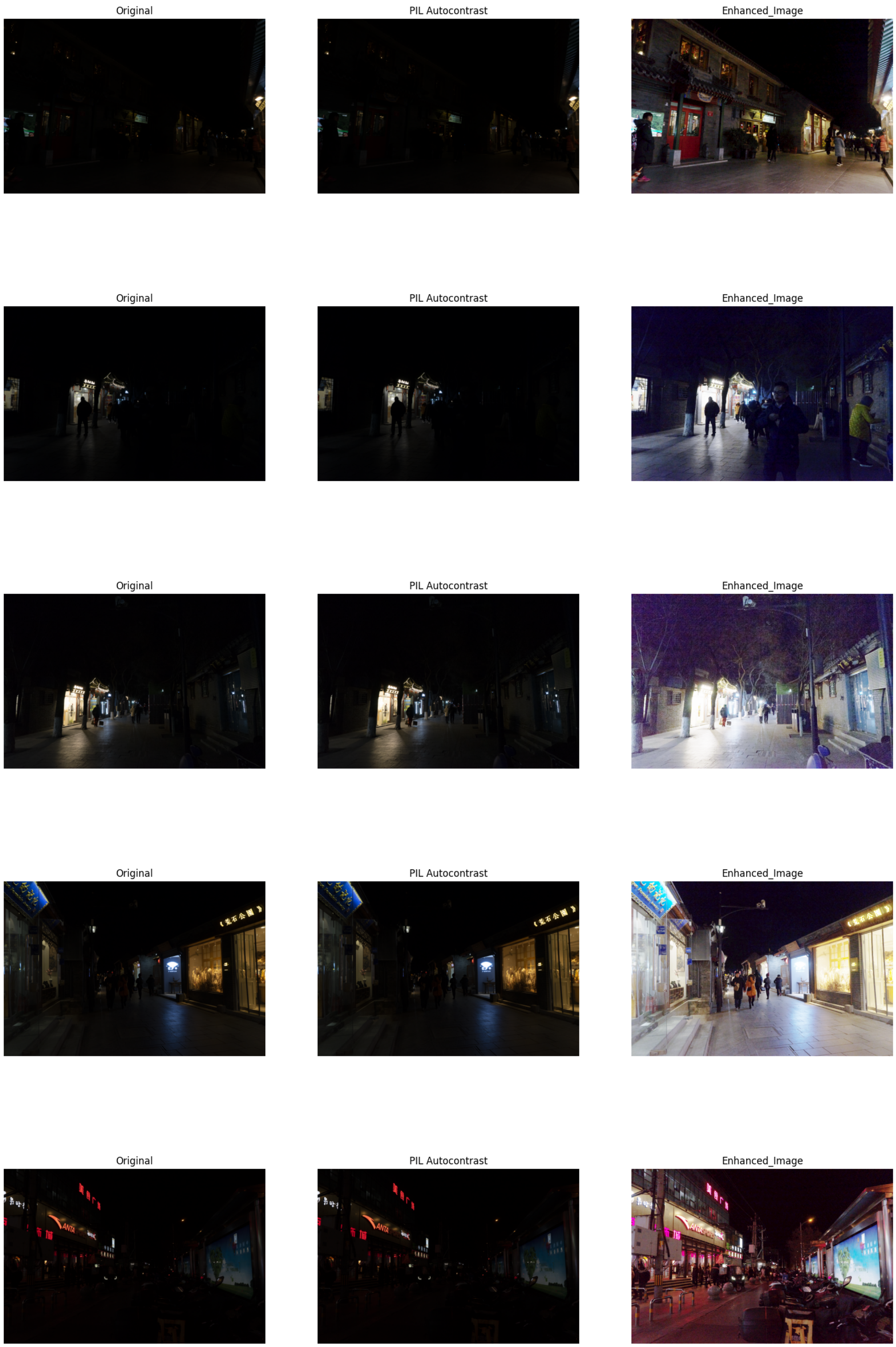



Original
PIL Autocontrast
Enhanced_Image


Original
PIL Autocontrast
Enhanced_Image


Original
PIL Autocontrast
Enhanced_Image

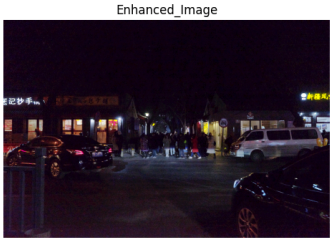

Original
PIL Autocontrast
Enhanced_Image


Original
PIL Autocontrast
Enhanced_Image
Original
PIL Autocontrast
Enhanced_Image
Original
PIL Autocontrast
Enhanced_Image
Original
PIL Autocontrast
Enhanced_Image
Original
PIL Autocontrast
Enhanced_Image

Original

PIL Autocontrast

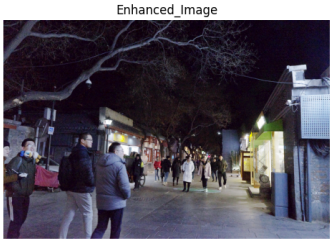
Enhanced_Image



Original

PIL Autocontrast

Enhanced_Image



Original

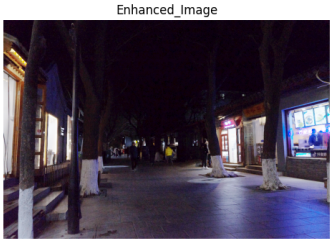
PIL Autocontrast

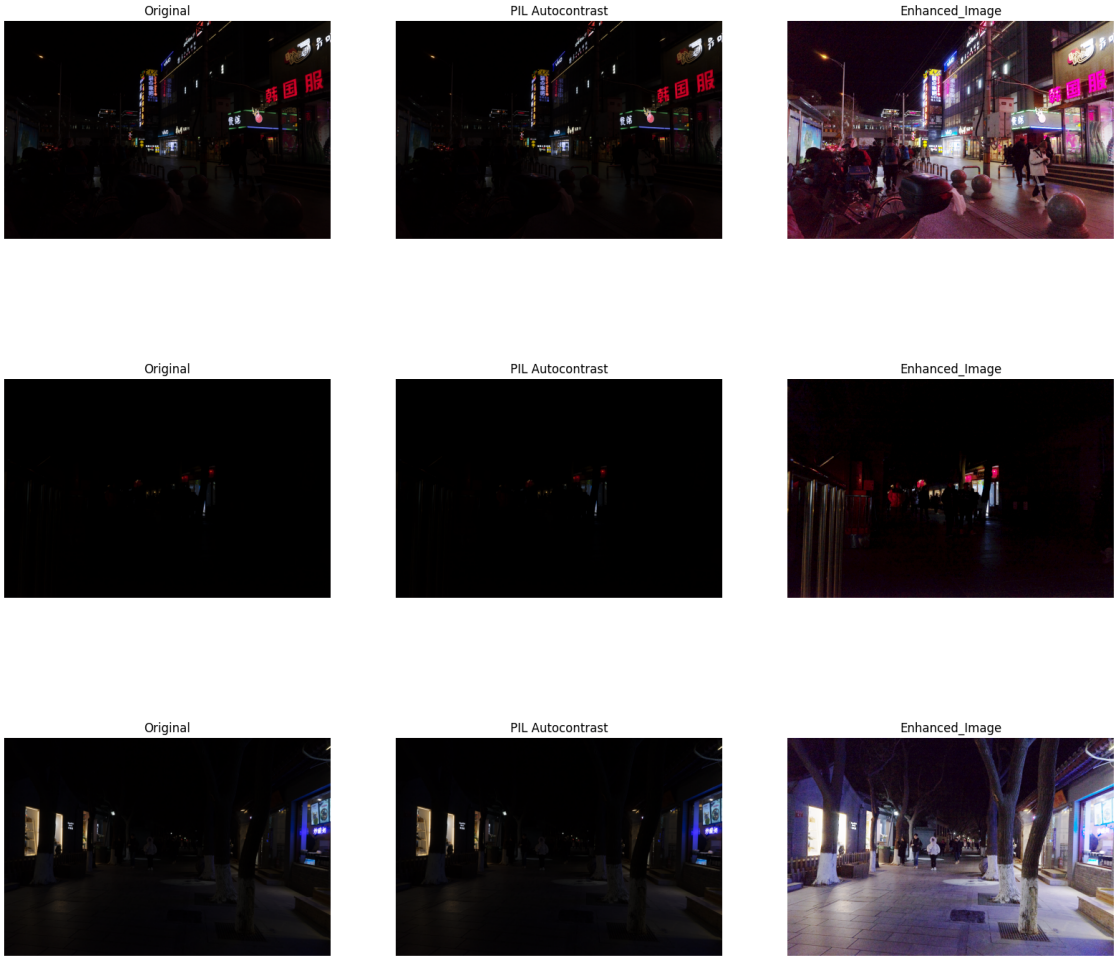
Enhanced_Image

Original

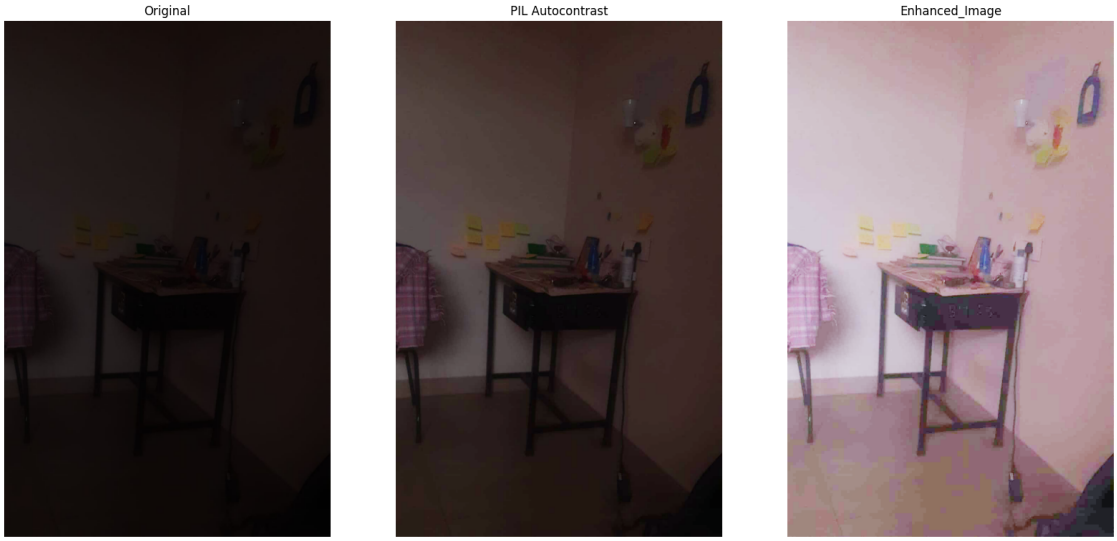
PIL Autocontrast

Enhanced_Image





[22]:
```
image_file='/kaggle/input/abcdefghijkl/rekha.jpeg'
original_image = Image.open(image_file)
enhanced_image = infer(original_image)
plot_results(
        [original_image, ImageOps.autocontrast(original_image), enhanced_image],
        ["Original", "PIL Autocontrast", "Enhanced_Image"],
        (20, 12),
    )
```



[26]:
```
image_file='/kaggle/input/dip-class2/class2.jpeg'

original_image = Image.open(image_file)
enhanced_image = infer(original_image)
plot_results(
        [original_image, ImageOps.autocontrast(original_image), enhanced_image],
        ["Original", "PIL Autocontrast", "Enhanced_Image"],
        (20, 12),
    )
```