Zero_DCE

February 28, 2023

[21]:
```
import os
import random
import numpy as np
from glob import glob
from PIL import Image, ImageOps
import matplotlib.pyplot as plt
import wandb
from mpl_toolkits.axes_grid1 import ImageGrid
from wandb.keras import WandbCallback
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, losses, optimizers, callbacks
```

[46]:
```
SEED = 10
random.seed(SEED)
tf.random.set_seed(SEED)

IMAGE_SIZE = 256
MAX_TRAIN_IMAGES = 400
BATCH_SIZE = 16
TRAIN_VAL_IMAGE_DIR = "LOL/train/low"
TEST_IMAGE_DIR = "LOL/test/low"
LEARNING_RATE = 1e-4
LOG_INTERVALS = 10
EPOCHS = 1
```

[47]:
```
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:]

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: 15
```

[48]:
```
AUTOTUNE = tf.data.AUTOTUNE
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)
```

[49]:
```
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)
```

[50]:
```
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:]

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: 15
```

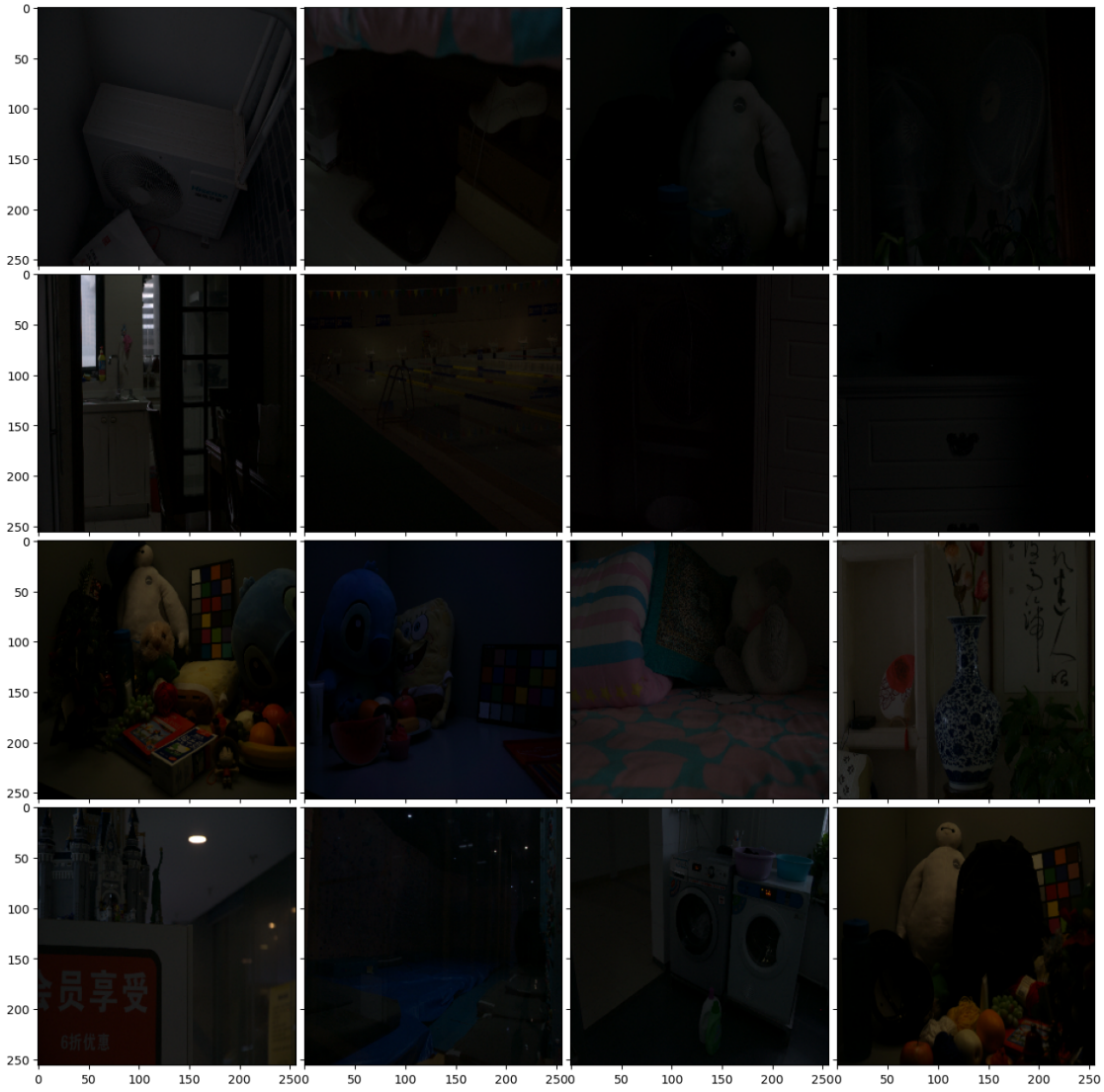
[51]:
```
def load_data(image_path):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_png(image, channels=3)
    image = tf.image.resize(images=image, size=[IMAGE_SIZE, IMAGE_SIZE])
    image = image / 255.0
    return image
```

```
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)
```

[52]:
```
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)
```

[53]:
```
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()
```



[54]:
```
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)
```

[55]:
```
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))
```

[56]:
```
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))
```

[57]:
```
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
```

[58]:
```
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
```

[59]:
```
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,
        )
```

[60]:
```
zero_dce_model = ZeroDCE()
zero_dce_model.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, None, None,  0
                              3)]

 conv2d_14 (Conv2D)          (None, None, None,    896
['input_3[0][0]']
```

```
                                32)

 conv2d_15 (Conv2D)             (None, None, None,   9248
['conv2d_14[0][0]']
                                32)

 conv2d_16 (Conv2D)             (None, None, None,   9248
['conv2d_15[0][0]']
                                32)

 conv2d_17 (Conv2D)             (None, None, None,   9248
['conv2d_16[0][0]']
                                32)

 concatenate_6 (Concatenate)    (None, None, None,   0
['conv2d_17[0][0]',
                                64)
'conv2d_16[0][0]']

 conv2d_18 (Conv2D)             (None, None, None,   18464
['concatenate_6[0][0]']
                                32)

 concatenate_7 (Concatenate)    (None, None, None,   0
['conv2d_18[0][0]',
                                64)
'conv2d_15[0][0]']

 conv2d_19 (Conv2D)             (None, None, None,   18464
['concatenate_7[0][0]']
                                32)

 concatenate_8 (Concatenate)    (None, None, None,   0
['conv2d_19[0][0]',
                                64)
'conv2d_14[0][0]']

 conv2d_20 (Conv2D)             (None, None, None,   13848
['concatenate_8[0][0]']
                                24)

==================================================================================
==================
Total params: 79,416
Trainable params: 79,416
Non-trainable params: 0
----------------------------------------------------------------------------------
------------------
```

[61]:
```
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
```

[62]:
```
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), )
```

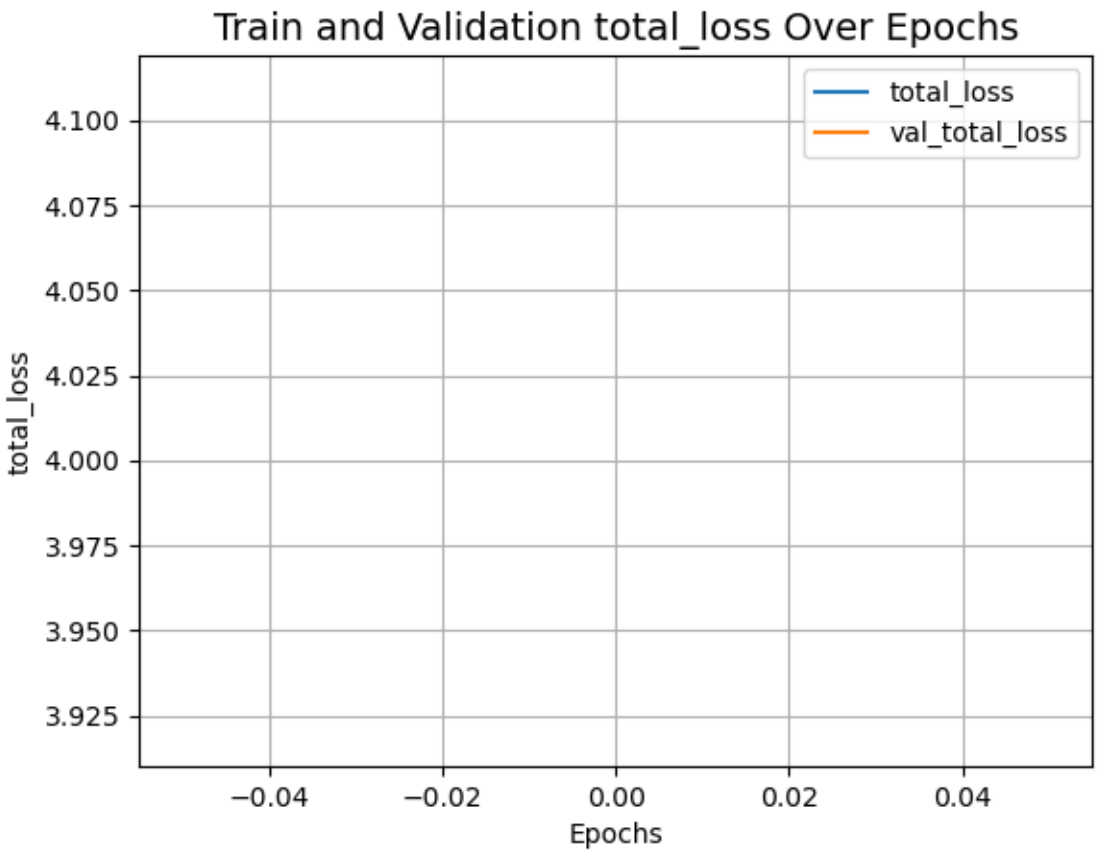
[63]:
```
zero_dce_model.compile(learning_rate=LEARNING_RATE)
history = zero_dce_model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=EPOCHS,
    callbacks=[
        LogPredictionCallback(
            image_files=random.sample(val_image_files, 4),
            log_interval=LOG_INTERVALS
        )
    ]
)
```

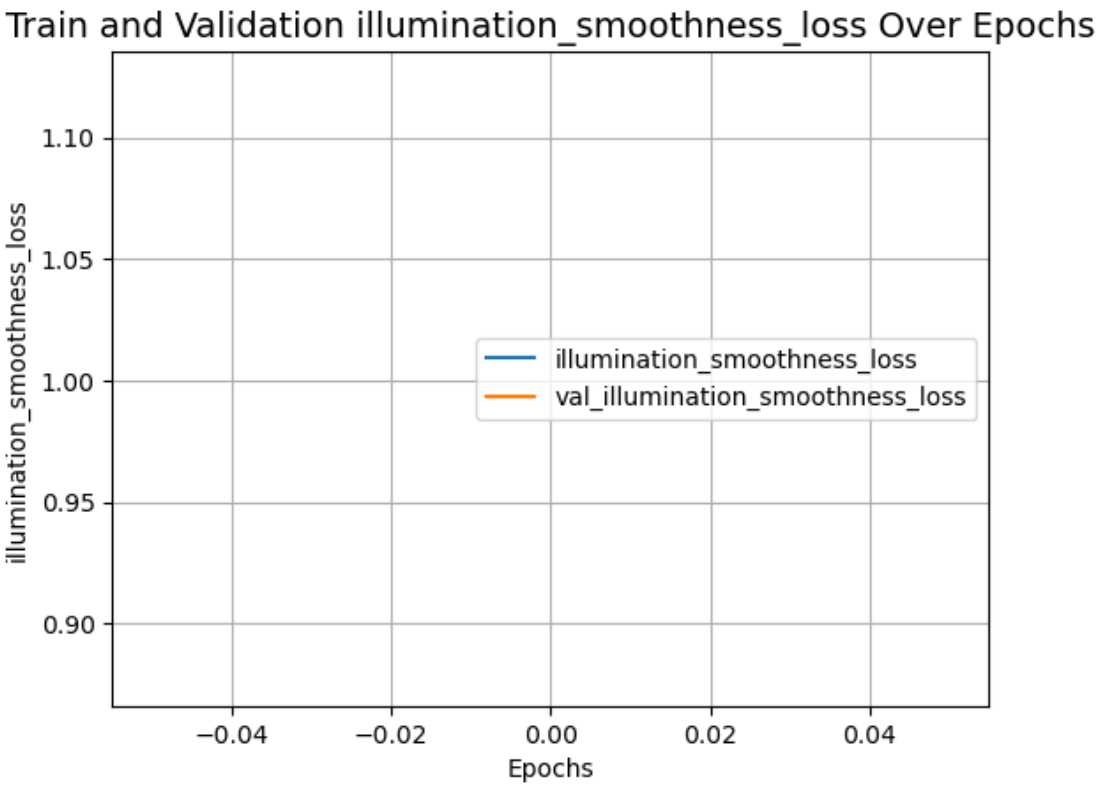
```
25/25 [==============================] - ETA: 0s - total_loss: 4.8402 -
```

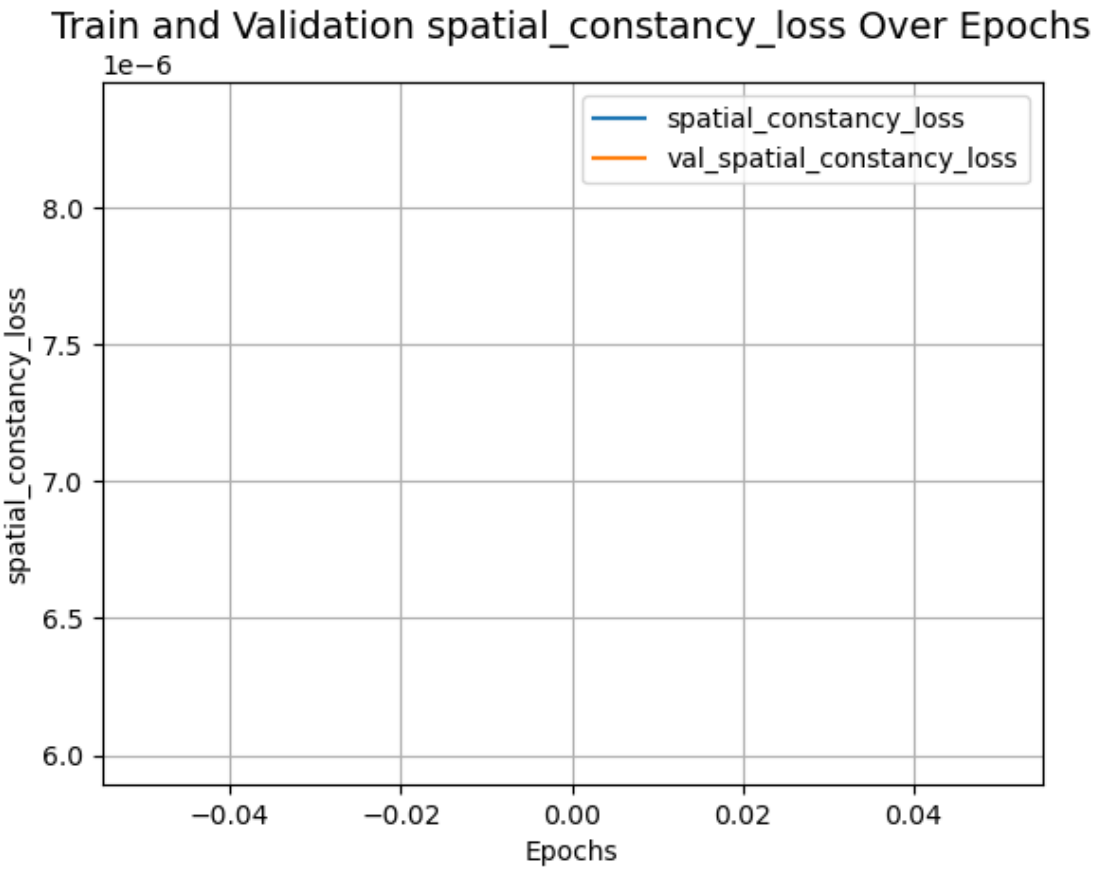
```
illumination_smoothness_loss: 1.8875 - spatial_constancy_loss: 9.8999e-06 -
color_constancy_loss: 0.0030 - exposure_loss: 2.9498
```

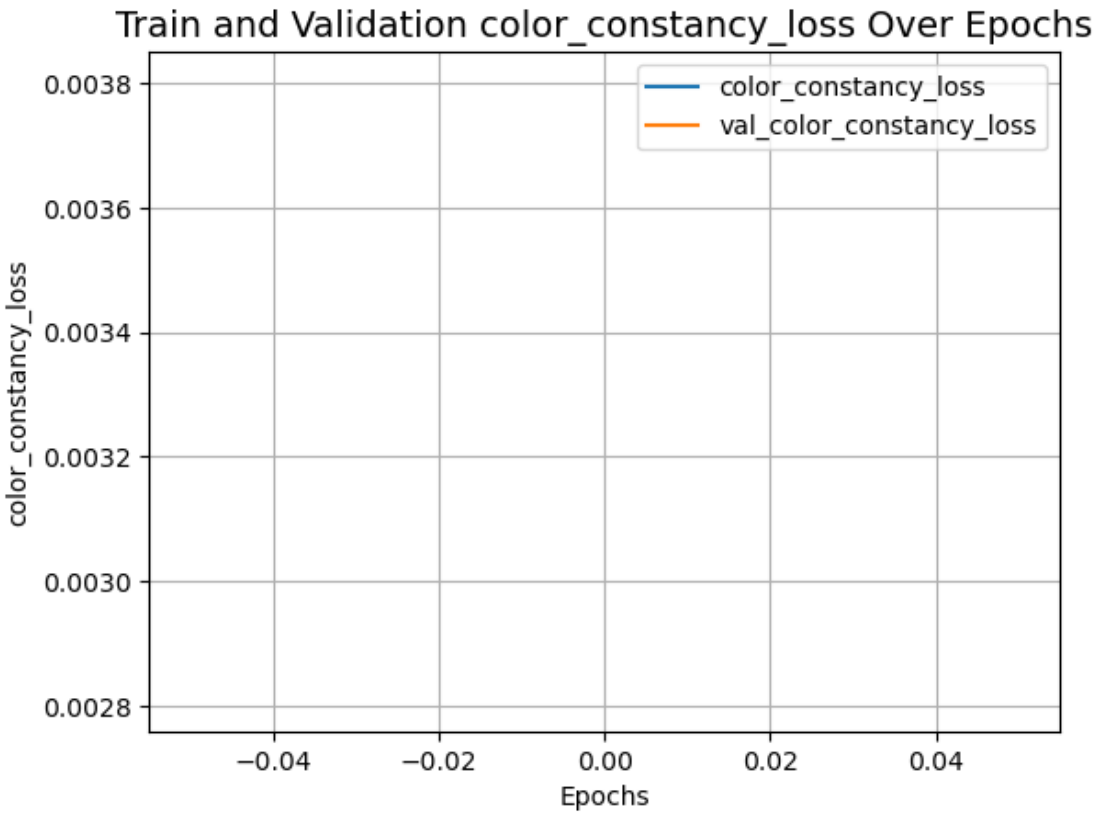

```
25/25 [==============================] - 351s 14s/step - total_loss: 4.8048 -
illumination_smoothness_loss: 1.8487 - spatial_constancy_loss: 9.7503e-06 -
color_constancy_loss: 0.0030 - exposure_loss: 2.9531 - val_total_loss: 4.1095 -
val_illumination_smoothness_loss: 1.1230 - val_spatial_constancy_loss:
8.3411e-06 - val_color_constancy_loss: 0.0028 - val_exposure_loss: 2.9836
```

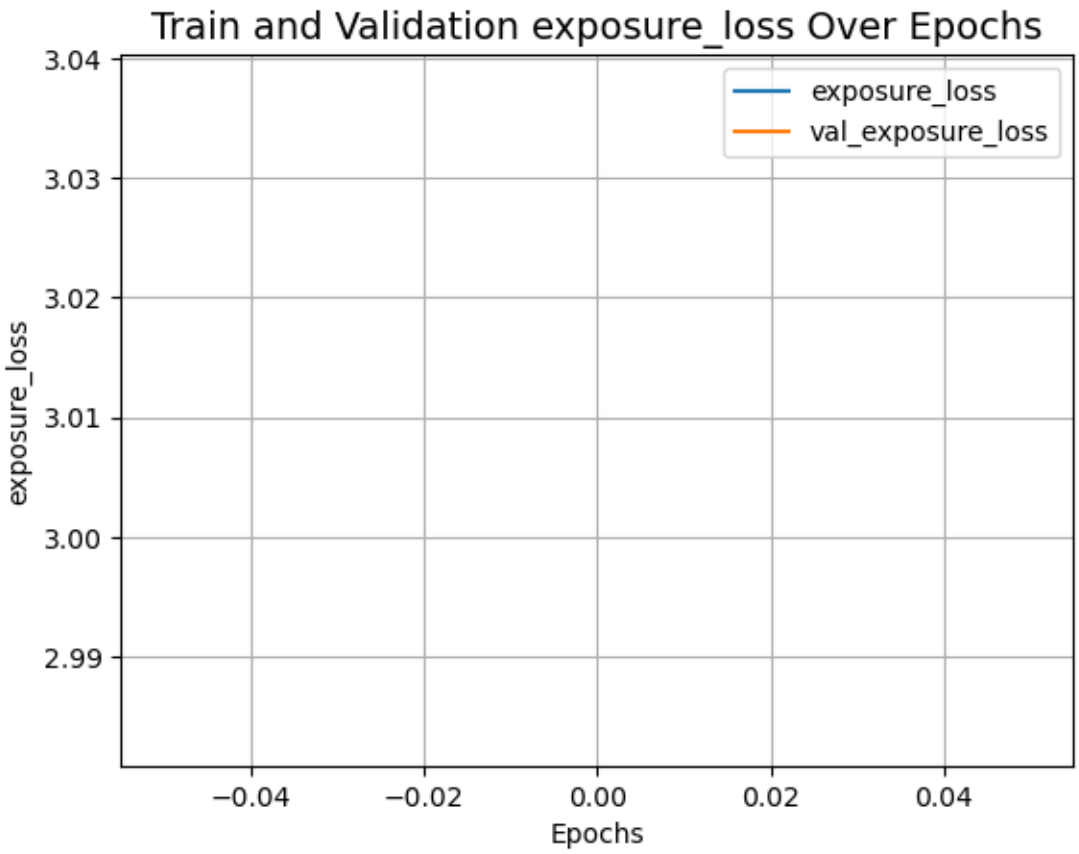
[65]:
```python
def plot_result(item):
    plt.plot(history.history[item], label=item)
    plt.plot(history.history["val_" + item], label="val_" + item)
    plt.xlabel("Epochs")
    plt.ylabel(item)
    plt.title("Train and Validation {} Over Epochs".format(item), fontsize=14)
    plt.legend()
    plt.grid()
    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 total_loss Over Epochs
total_loss
val_total_loss
4.100
4.075
4.050
4.025
4.000
3.975
3.950
3.925
total_loss
−0.04
−0.02
0.00
0.02
0.04
Epochs


Train and Validation illumination_smoothness_loss Over Epochs
illumination_smoothness_loss
1.10
1.05
1.00
0.95
0.90
−0.04
−0.02
0.00
0.02
0.04
Epochs
illumination_smoothness_loss
val_illumination_smoothness_loss

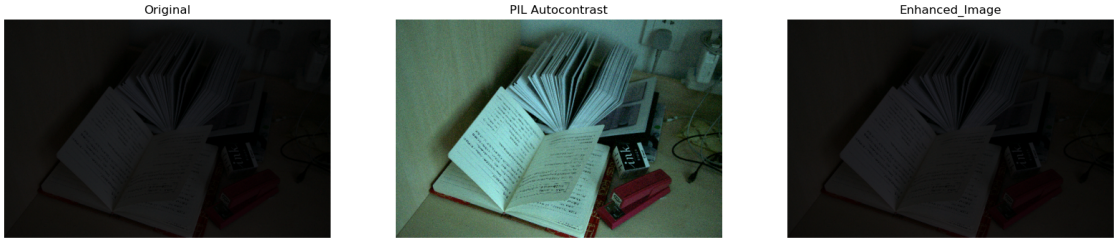

Train and Validation spatial_constancy_loss Over Epochs
1e−6
spatial_constancy_loss
val_spatial_constancy_loss
8.0
7.5
7.0
6.5
6.0
spatial_constancy_loss
−0.04
−0.02
0.00
0.02
0.04
Epochs


Train and Validation color_constancy_loss Over Epochs
color_constancy_loss
val_color_constancy_loss
0.0038
0.0036
0.0034
0.0032
0.0030
0.0028
color_constancy_loss
−0.04
−0.02
0.00
0.02
0.04
Epochs

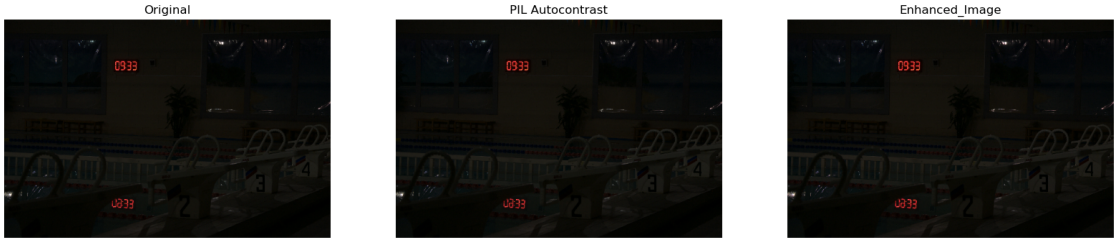


[66]:
```
for image_file in test_image_files:
    original_image = Image.open(image_file)
    enhanced_image = infer(original_image)
    plot_results(
        [original_image, ImageOps.autocontrast(original_image), enhanced_image],
        ["Original", "PIL Autocontrast", "Enhanced_Image"],
        (20, 12),
    )
```


Original
PIL Autocontrast
Enhanced_Image

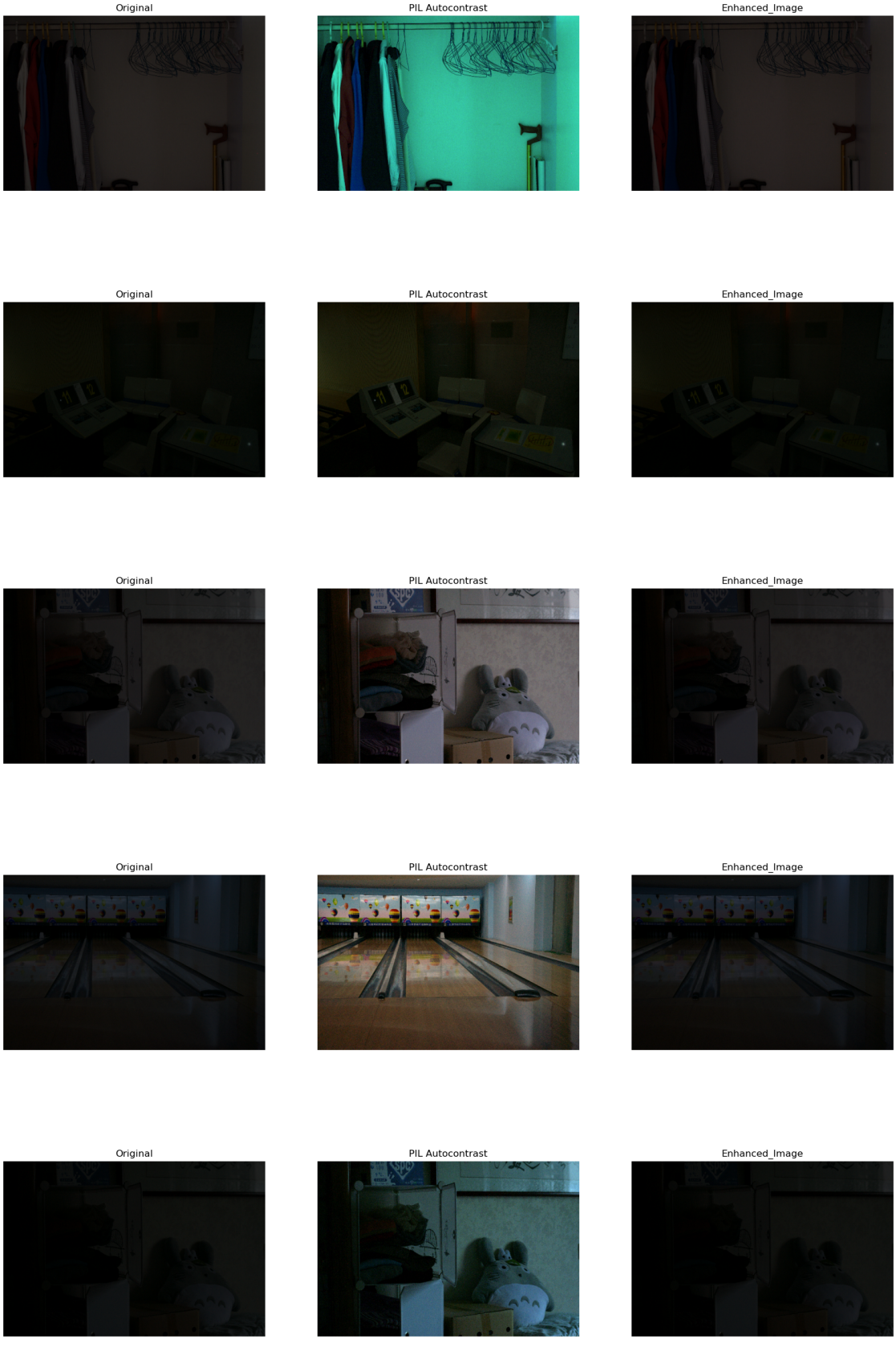

Original
PIL Autocontrast
Enhanced_Image


Original
PIL Autocontrast
Enhanced_Image

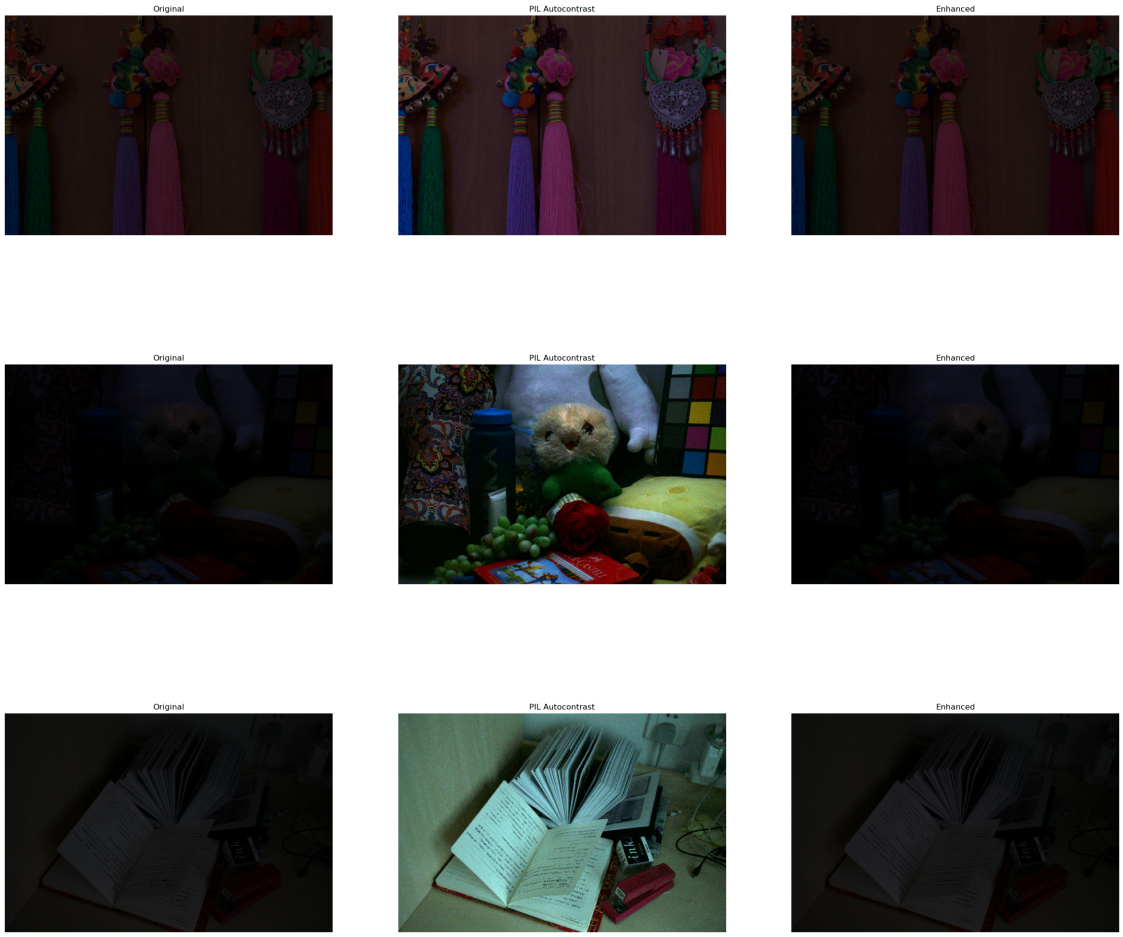

Original
PIL Autocontrast
Enhanced_Image


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
Original
PIL Autocontrast
Enhanced_Image

[67]:
```
game_of_thrones_images = glob("LOL/test/low/*.png")
for image_file in game_of_thrones_images:
    original_image = Image.open(image_file)
    enhanced_image = infer(original_image)
    plot_results(
        [original_image, ImageOps.autocontrast(Image.fromarray(np.
 ↪array(original_image)[:, :, :3])), enhanced_image],
        ["Original", "PIL Autocontrast", "Enhanced"],
        (30, 22),
    )
```




Original
PIL Autocontrast
Enhanced
Original
PIL Autocontrast
Enhanced
Original
PIL Autocontrast
Enhanced
Original
PIL Autocontrast
Enhanced
Original
PIL Autocontrast
Enhanced

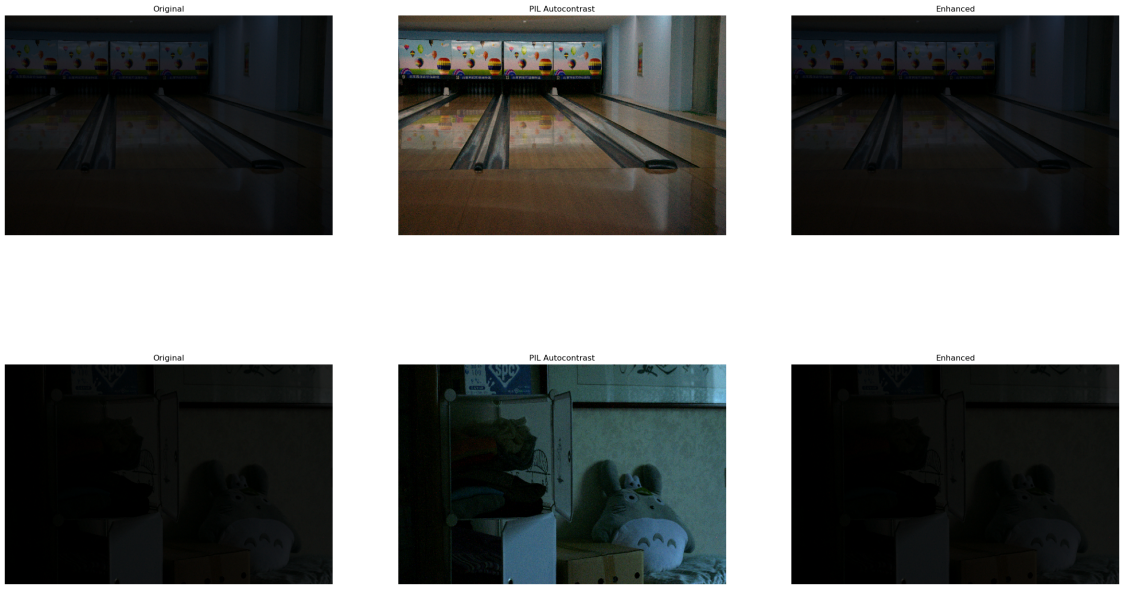

Original
PIL Autocontrast
Enhanced


Original
PIL Autocontrast
Enhanced


Original
PIL Autocontrast
Enhanced


Original
PIL Autocontrast
Enhanced


Original
PIL Autocontrast
Enhanced


Original
PIL Autocontrast
Enhanced
Original
PIL Autocontrast
Enhanced

[ ]:

[ ]: