```
import os
import cv2
import random
import numpy as np
from glob import glob
from PIL import Image, ImageOps
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import layers, Model
from keras import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, UpSampling2D


from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
```

```
import os

# Define the path to the folder containing your ZIP file
folder_path = '/content/drive/MyDrive/ARB'

# Change the current working directory to the folder
os.chdir(folder_path)

os.listdir()

!unzip lol_dataset.zip
```

```
random.seed(5)

IMAGE_SIZE = 128
BATCH_SIZE = 4
MAX_TRAIN_IMAGES = 300


def read_image(image_path):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_png(image, channels=3)
    image.set_shape([None, None, 3])
```

```
    image = tf.cast(image, dtype=tf.float32) / 255.0
    return image


def random_crop(low_image, enhanced_image):
    low_image_shape = tf.shape(low_image)[:2]
    low_w = tf.random.uniform(
        shape=(), maxval=low_image_shape[1] - IMAGE_SIZE + 1, dtype=tf.int32
    )
    low_h = tf.random.uniform(
        shape=(), maxval=low_image_shape[0] - IMAGE_SIZE + 1, dtype=tf.int32
    )
    enhanced_w = low_w
    enhanced_h = low_h
    low_image_cropped = low_image[
        low_h : low_h + IMAGE_SIZE, low_w : low_w + IMAGE_SIZE
    ]
    enhanced_image_cropped = enhanced_image[
        enhanced_h : enhanced_h + IMAGE_SIZE, enhanced_w : enhanced_w + IMAGE_SI
    ]
    return low_image_cropped, enhanced_image_cropped


def load_data(low_light_image_path, enhanced_image_path):
    low_light_image = read_image(low_light_image_path)
    enhanced_image = read_image(enhanced_image_path)
    low_light_image, enhanced_image = random_crop(low_light_image, enhanced_imag
    return low_light_image, enhanced_image


def get_dataset(low_light_images, enhanced_images):
    dataset = tf.data.Dataset.from_tensor_slices((low_light_images, enhanced_ima
    dataset = dataset.map(load_data, num_parallel_calls=tf.data.AUTOTUNE)
    dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)
    return dataset


train_low_light_images = sorted(glob("./lol_dataset/our485/low/*"))[:MAX_TRAIN_I
train_enhanced_images = sorted(glob("./lol_dataset/our485/high/*"))[:MAX_TRAIN_I

val_low_light_images = sorted(glob("./lol_dataset/our485/low/*"))[MAX_TRAIN_IMAG
val_enhanced_images = sorted(glob("./lol_dataset/our485/high/*"))[MAX_TRAIN_IMAG

test_low_light_images = sorted(glob("./lol_dataset/eval15/low/*"))
test_enhanced_images = sorted(glob("./lol_dataset/eval15/high/*"))


train_dataset = get_dataset(train_low_light_images, train_enhanced_images)
```

```
val_dataset = get_dataset(val_low_light_images, val_enhanced_images)


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

```
Train Dataset: <_BatchDataset element_spec=(TensorSpec(shape=(4, None, None
Val Dataset: <_BatchDataset element_spec=(TensorSpec(shape=(4, None, None,
```

```
# Define the residual block
def residual_block(x, filters, kernel_size):
    y = layers.Conv2D(filters, kernel_size, activation='relu', padding='same')(x
    y = layers.BatchNormalization()(y)
    y = layers.Conv2D(filters, kernel_size, activation='relu', padding='same')(y
    y = layers.BatchNormalization()(y)
    y = layers.Add()([x, y])
    return y
def spatial_attention_block(input_tensor):
    average_pooling = tf.reduce_max(input_tensor, axis=-1)
    average_pooling = tf.expand_dims(average_pooling, axis=-1)
    max_pooling = tf.reduce_mean(input_tensor, axis=-1)
    max_pooling = tf.expand_dims(max_pooling, axis=-1)
    concatenated = layers.Concatenate(axis=-1)([average_pooling, max_pooling])
    feature_map = layers.Conv2D(1, kernel_size=(1, 1))(concatenated)
    feature_map = tf.nn.sigmoid(feature_map)
    return input_tensor * feature_map


def channel_attention_block(input_tensor):
    channels = list(input_tensor.shape)[-1]
    average_pooling = layers.GlobalAveragePooling2D()(input_tensor)
    feature_descriptor = tf.reshape(average_pooling, shape=(-1, 1, 1, channels))
    feature_activations = layers.Conv2D(
        filters=channels // 8, kernel_size=(1, 1), activation="relu"
    )(feature_descriptor)
    feature_activations = layers.Conv2D(
        filters=channels, kernel_size=(1, 1), activation="sigmoid"
    )(feature_activations)
    return input_tensor * feature_activations


def dual_attention_unit_block(input_tensor):
    channels = list(input_tensor.shape)[-1]
    feature_map = layers.Conv2D(
```

```
        channels, kernel_size=(3, 3), padding="same", activation="relu"
    )(input_tensor)
    feature_map = layers.Conv2D(channels, kernel_size=(3, 3), padding="same")(
        feature_map
    )
    channel_attention = channel_attention_block(feature_map)
    spatial_attention = spatial_attention_block(feature_map)
    concatenation = layers.Concatenate(axis=-1)([channel_attention, spatial_atte
    concatenation = layers.Conv2D(channels, kernel_size=(1, 1))(concatenation)
    return layers.Add()([input_tensor, concatenation])

    # Add a de-blurring layer at the beginning
def deblur_layer(input_layer):
    # You can customize the parameters, such as kernel size and activation
    deblur = Conv2D(filters=64, kernel_size=3, activation='relu', padding='same'
    return deblur

def unet_model_with_multi_scale_residual_and_deblur():
    # Input Layer
    input_layer = tf.keras.layers.Input(shape=(128, 128, 3))

    # De-blurring Layer (you can adjust kernel size, stride, and filters)
    deblur = layers.Conv2D(filters=64, kernel_size=3, activation='relu', padding

    # Contracting Path (Encoder) with Multi-Scale Residual Block
    conv1 = residual_block(layers.Conv2D(64, 3, activation='relu', padding='same
    pool1 = layers.MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = residual_block(layers.Conv2D(128, 3, activation='relu', padding='sam
    pool2 = layers.MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = residual_block(layers.Conv2D(256, 3, activation='relu', padding='sam
    pool3 = layers.MaxPooling2D(pool_size=(2, 2))(conv3)

    # Bottleneck
    conv4 = residual_block(layers.Conv2D(512, 3, activation='relu', padding='sam

    # Expansive Path (Decoder) with Multi-Scale Residual Block
    up5 = layers.UpSampling2D(size=(2, 2))(conv4)
    up5 = layers.concatenate([up5, conv3], axis=-1)
    conv5 = residual_block(layers.Conv2D(256, 3, activation='relu', padding='sam

    up6 = layers.UpSampling2D(size=(2, 2))(conv5)
    up6 = layers.concatenate([up6, conv2], axis=-1)
    conv6 = residual_block(layers.Conv2D(128, 3, activation='relu', padding='sam

    up7 = layers.UpSampling2D(size=(2, 2))(conv6)
    up7 = layers.concatenate([up7, conv1], axis=-1)
```

```
    conv7 = residual_block(layers.Conv2D(64, 3, activation='relu', padding='same

    # Output Layer
    output_layer = layers.Conv2D(3, 1, activation='sigmoid')(conv7)

    model = Model(inputs=input_layer, outputs=output_layer)

    return model

# Create U-Net model with the multi-scale residual block
model = unet_model_with_multi_scale_residual_and_deblur()

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Print model summary
model.summary()



def charbonnier_loss(y_true, y_pred):
    return tf.reduce_mean(tf.sqrt(tf.square(y_true - y_pred) + tf.square(1e-3)))


def peak_signal_noise_ratio(y_true, y_pred):
    return tf.image.psnr(y_pred, y_true, max_val=255.0)


optimizer = keras.optimizers.Adam(learning_rate=0.00005)
model.compile(
    optimizer=optimizer, loss=charbonnier_loss, metrics=[peak_signal_noise_ratio
)

history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=50,
    callbacks=[
        keras.callbacks.ReduceLROnPlateau(
            monitor="val_peak_signal_noise_ratio",
            factor=0.5,
            patience=5,
            verbose=1,
            min_delta=1e-7,
            mode="max",
        )
    ],
)
```

```
plt.plot(history.history["loss"], label="train_loss")
plt.plot(history.history["val_loss"], label="val_loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Train and Validation Losses Over Epochs", fontsize=14)
plt.legend()
plt.grid()
plt.show()
```

```
plt.plot(history.history["peak_signal_noise_ratio"], label="train_psnr")
plt.plot(history.history["val_peak_signal_noise_ratio"], label="val_psnr")
plt.xlabel("Epochs")
plt.ylabel("PSNR")
plt.title("Train and Validation PSNR Over Epochs", fontsize=14)
plt.legend()
plt.grid()
plt.show()
```

```
from tensorflow.keras.models import load_model
# Save the model if needed
model.save('/content/drive/MyDrive/ARB/image_enhancement_dunet_v250e.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079:
  saving_api.save_model(
```

```
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.utils.img_to_array(original_image)
    image = image.astype("float32") / 255.0
    image = np.expand_dims(image, axis=0)
    output = model.predict(image)
    output_image = output[0] * 255.0
    output_image = output_image.clip(0, 255)
    output_image = output_image.reshape(
        (np.shape(output_image)[0], np.shape(output_image)[1], 3)
    )
    output_image = Image.fromarray(np.uint8(output_image))
    original_image = Image.fromarray(np.uint8(original_image))
    return output_image
```

```
for low_light_image in random.sample(test_low_light_images, 8):
    original_image = Image.open(low_light_image)
    #original_image = original_image.resize((128, 128))
    enhanced_image = infer(original_image)
    plot_results(
        [original_image, ImageOps.autocontrast(original_image), enhanced_image],
        ["Original", "PIL Autocontrast", "DUO-Net Enhanced"],
        (20, 12),
    )
```

```
1/1 [==============================] - 0s 40ms/step
```


```
1/1 [==============================] - 0s 151ms/step
```

Original PIL Autocontrast DUO-Net Enhanced

```
1/1 [==============================] - 0s 55ms/step
```

Original PIL Autocontrast DUO-Net Enhanced

```
1/1 [==============================] - 0s 36ms/step
```

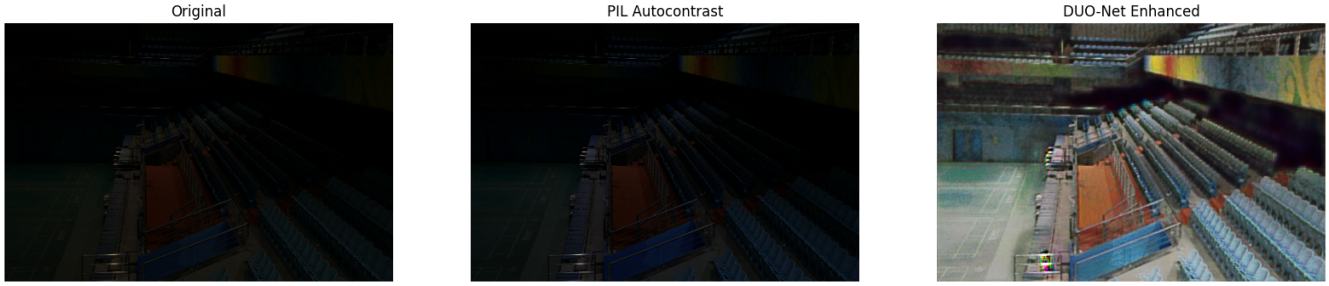
Original PIL Autocontrast DUO-Net Enhanced

```
1/1 [==============================] - 0s 44ms/step
```

Original PIL Autocontrast DUO-Net Enhanced

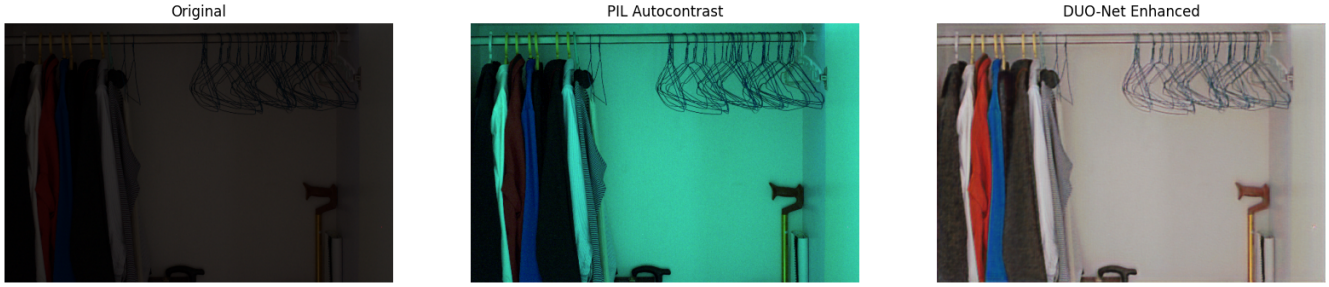
```
1/1 [==============================] - 0s 49ms/step
```

Original PIL Autocontrast DUO-Net Enhanced



```
1/1 [==============================] - 0s 60ms/step
```

Original PIL Autocontrast DUO-Net Enhanced



```
1/1 [==============================] - 0s 38ms/step
```