Report: Image Enhancement Using a UNet-Residual Network with Attention Mechanism

1. Introduction

The architecture used in this project is a combination of a UNet model, residual blocks, and attention mechanisms. This model is designed to enhance low-light images by learning from a dataset of paired low-light and high-quality images.

Specifications

- **Input Size:** 128x128 pixels
- **Model Components:** UNet, Residual Blocks, Attention Mechanisms
- **Optimizer:** Adam with a learning rate of 0.0001
- **Loss Function:** Combined loss (Mean Squared Error + Structural Similarity Index Measure)
- **Epochs:** 100
- **Batch Size:** 32
- **Callbacks:** EarlyStopping, ModelCheckpoint, ReduceLROnPlateau

Performance Metrics

- **Mean Squared Error (MSE):** `0.011920978525241195`
- **Peak Signal-to-Noise Ratio (PSNR):** `19.236880943717413`

Paper Implemented

The implementation draws inspiration from various research papers on image enhancement and deep learning models like UNet, residual networks, and attention mechanisms. One such influential paper is:

- **Title:** "Image Super-Resolution Using Deep Convolutional Networks"
- **Link:** [Image Super-Resolution Using Deep Convolutional Networks]

2. Project Details

Loading and Preprocessing Data: Load images from directories and preprocess them by resizing and normalizing.

```
import cv2

import os

import numpy as np

def load_images_from_directory(a):

      images = []

      for i in sorted(os.listdir(a)):

      img = cv2.imread(os.path.join(a, i))

      img = cv2.resize(img, (128, 128))

        images.append(img)

      return np.array(images)

low_images = load_images_from_directory(l) / 255.0

high_images = load_images_from_directory(h) / 255.0
```

Splitting Data: Split the data into training and testing sets.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(low_images, high_images,
test_size=0.2, random_state=42)
```

Loss Functions: Define custom loss functions that combine MSE and SSIM for better image quality assessment.

```
import tensorflow as tf

from tensorflow.keras.losses import MeanSquaredError

def ssim_loss(y_true, y_pred):

      return 1 - tf.reduce_mean(tf.image.ssim(y_true, y_pred, max_val=1.0))

def combined_loss(y_true, y_pred):
```

```
	mse = MeanSquaredError()(y_true, y_pred)

	s_loss = ssim_loss(y_true, y_pred)

	return mse + 0.5 * s_loss
```

Model Architecture: Define the model architecture combining UNet, residual blocks, and attention mechanisms.

```
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization,
Activation, Add, UpSampling2D, concatenate, Dropout

from tensorflow.keras.models import Model

from tensorflow.keras.regularizers import l2

def residual_dense_block(x, filters, kernel_size=3, dropout_rate=0.3):

	res = Conv2D(filters,kernel_size,padding='same',kernel_regularizer=l2
(1e-4))(x)

	res = BatchNormalization()(res)

	res = Activation('relu')(res)

	res = Dropout(dropout_rate)(res)

	res = Conv2D(filters, kernel_size, padding='same',
kernel_regularizer=l2(1e-4))(res)

	res = BatchNormalization()(res)

	res = Activation('relu')(res)

	res = Dropout(dropout_rate)(res)

	res = Conv2D(filters, kernel_size, padding='same',
kernel_regularizer=l2(1e-4))(res)

	res = BatchNormalization()(res)

	res = Add()([res, x])

	return res

def attention_block(x, filters):

	f = Conv2D(filters // 8, (1, 1), padding='same')(x)

	f = BatchNormalization()(f)
```

```
    f = Activation('relu')(f)

    g = Conv2D(filters // 8, (1, 1), padding='same')(x)

    g = BatchNormalization()(g)

    g = Activation('relu')(g)

    h = Conv2D(filters, (1, 1), padding='same')(x)

    h = BatchNormalization()(h)

    h = Activation('relu')(h)

    s = Multiply()([f, g])

    beta = Activation('softmax')(s)

    beta = Conv2D(filters, (1, 1), padding='same')(beta)

    o = Multiply()([beta, h])

    return Add()([x, o])

def unet_residual_model(input_shape):

    inputs = Input(input_shape)

    # Encoder

    c1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)

    c1 = residual_dense_block(c1, 64)

    c1 = attention_block(c1, 64)

    p1 = Conv2D(64, (2, 2), strides=(2, 2), padding='same')(c1)

    c2 = Conv2D(128, (3, 3), activation='relu', padding='same')(p1)

    c2 = residual_dense_block(c2, 128)

    c2 = attention_block(c2, 128)

    p2 = Conv2D(128, (2, 2), strides=(2, 2), padding='same')(c2)

    c3 = Conv2D(256, (3, 3), activation='relu', padding='same')(p2)

    c3 = residual_dense_block(c3, 256)

    c3 = attention_block(c3, 256)
```

```
p3 = Conv2D(256, (2, 2), strides=(2, 2), padding='same')(c3)

c4 = Conv2D(512, (3, 3), activation='relu', padding='same')(p3)

c4 = residual_dense_block(c4, 512)

c4 = attention_block(c4, 512)

p4 = Conv2D(512, (2, 2), strides=(2, 2), padding='same')(c4)

c5 = Conv2D(1024, (3, 3), activation='relu', padding='same')(p4)

c5 = residual_dense_block(c5, 1024)

c5 = attention_block(c5, 1024)

# Decoder

u6 = UpSampling2D((2, 2))(c5)

u6 = concatenate([u6, c4])

c6 = Conv2D(512, (3, 3), activation='relu', padding='same')(u6)

c6 = residual_dense_block(c6, 512)

u7 = UpSampling2D((2, 2))(c6)

u7 = concatenate([u7, c3])

c7 = Conv2D(256, (3, 3), activation='relu', padding='same')(u7)

c7 = residual_dense_block(c7, 256)

u8 = UpSampling2D((2, 2))(c7)

u8 = concatenate([u8, c2])

c8 = Conv2D(128, (3, 3), activation='relu', padding='same')(u8)

c8 = residual_dense_block(c8, 128)

u9 = UpSampling2D((2, 2))(c8)

u9 = concatenate([u9, c1])

c9 = Conv2D(64, (3, 3), activation='relu', padding='same')(u9)

c9 = residual_dense_block(c9, 64)

outputs = Conv2D(3, (1, 1), activation='sigmoid')(c9)
```

```
    return Model(inputs, outputs)
```

Model Compilation and Training: Compile and train the model using the training set and validate on the test set.

```
from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
ReduceLROnPlateau

input_shape = X_train.shape[1:]  # height, width, channels

unet_residual = unet_residual_model(input_shape)

unet_residual.compile(optimizer=Adam(learning_rate=0.0001), loss=combined_loss)

early_stopping = EarlyStopping(patience=20, restore_best_weights=True)

model_checkpoint = ModelCheckpoint('best_model.weights.h5',
save_best_only=True, save_weights_only=True)

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5,
min_lr=1e-9)

history = unet_residual.fit(X_train, y_train, epochs=100, batch_size=32,
validation_data=(X_test, y_test),callbacks=[early_stopping,model_checkpoint
,reduce_lr])
```

Model Evaluation: Evaluate the model using MSE and PSNR metrics.

```
from sklearn.metrics import mean_squared_error

predictions = unet_residual.predict(X_test)

mse = mean_squared_error(y_test.flatten(), predictions.flatten())

print(f'Mean Squared Error: {mse}')

def calculate_psnr(y_true, y_pred):

    mse = np.mean((y_true - y_pred) ** 2)

    if mse == 0:

    return 100

    PIXEL_MAX = 1.0

    psnr = 20 * np.log10(PIXEL_MAX / np.sqrt(mse))
```

```
      return psnr

psnr = calculate_psnr(y_test, predictions)

print(f'PSNR: {psnr}')
```

Visualization: Visualize the results by displaying low-light images, enhanced images, and ground truth images.

```
import matplotlib.pyplot as plt

for i in range(3):

    plt.figure(figsize=(15, 5))

    plt.subplot(1, 3, 1)

    plt.title('Low Light Image')

    plt.imshow(X_test[i])

    plt.subplot(1, 3, 2)

    plt.title('Enhanced Image')

    plt.imshow(predictions[i])

    plt.subplot(1, 3, 3)

    plt.title('Ground Truth')

    plt.imshow(y_test[i])

      plt.show()
```

3. Summary

Findings

- The UNet-residual network with attention mechanisms effectively enhances low-light images.
- The combined loss function incorporating MSE and SSIM improves the perceptual quality of the enhanced images.
- The model achieved a PSNR of `19.2368`, indicating good reconstruction quality.

Methods to Improve

- **Data Augmentation:** Enhance the training dataset with more varied samples to improve generalization.
- **Hyperparameter Tuning:** Experiment with different learning rates, batch sizes, and dropout rates.
- **Advanced Architectures:** Incorporate more advanced attention mechanisms or multi-scale feature extraction techniques.
- **Ensemble Methods:** Combine multiple models to leverage their strengths and improve overall performance.