```
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, Activation, Add, UpSampling2D, concatenate,Dropout,Multiply, Dense, Flatten, Lambda
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.regularizers import l2
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.losses import MeanSquaredError
model_checkpoint = ModelCheckpoint('best_model.weights.h5', save_best_only=True, save_weights_only=True)
```

```
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split

l = '/content/drive/MyDrive/Colab Notebooks/vlg/Train/low'
h = '/content/drive/MyDrive/Colab Notebooks/vlg/Train/high'

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

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

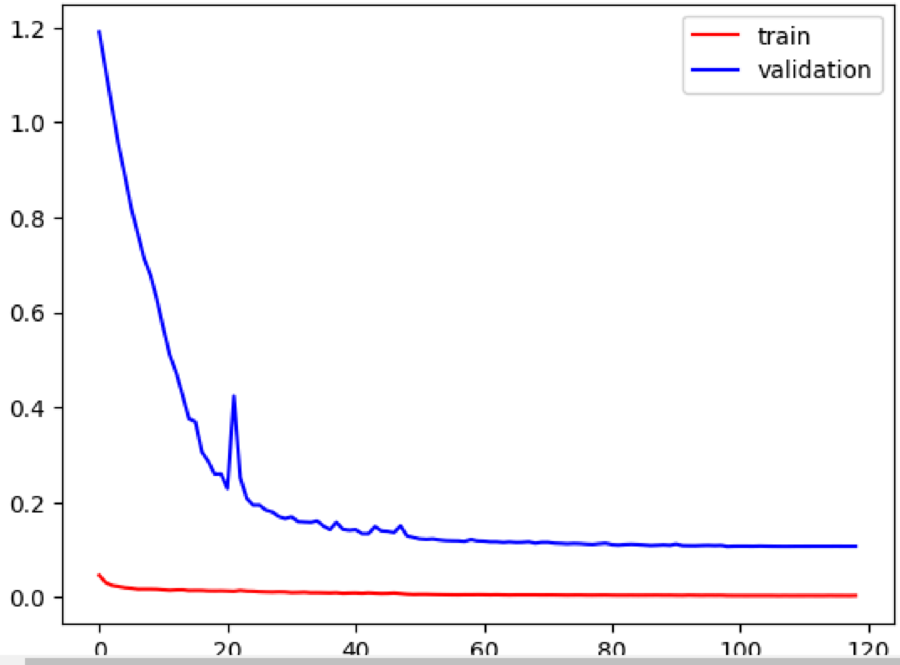
```
import tensorflow as tf
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.regularizers import l2
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, Activation, Add, UpSampling2D, concatenate,Dropout,Multiply,Flatten
from tensorflow.keras.models import Model
# Structural Similarity Index Measure (SSIM) loss
def ssim_loss(y_true, y_pred):
    return 1 - tf.reduce_mean(tf.image.ssim(y_true, y_pred, max_val=1.0))

# Combined loss:MSE handles pixel-wise accuracy, and SSIM ensures structural similarity
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

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]) #Adds the input back to the output for residual learning, improving gradient flow and performance.
    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)
```

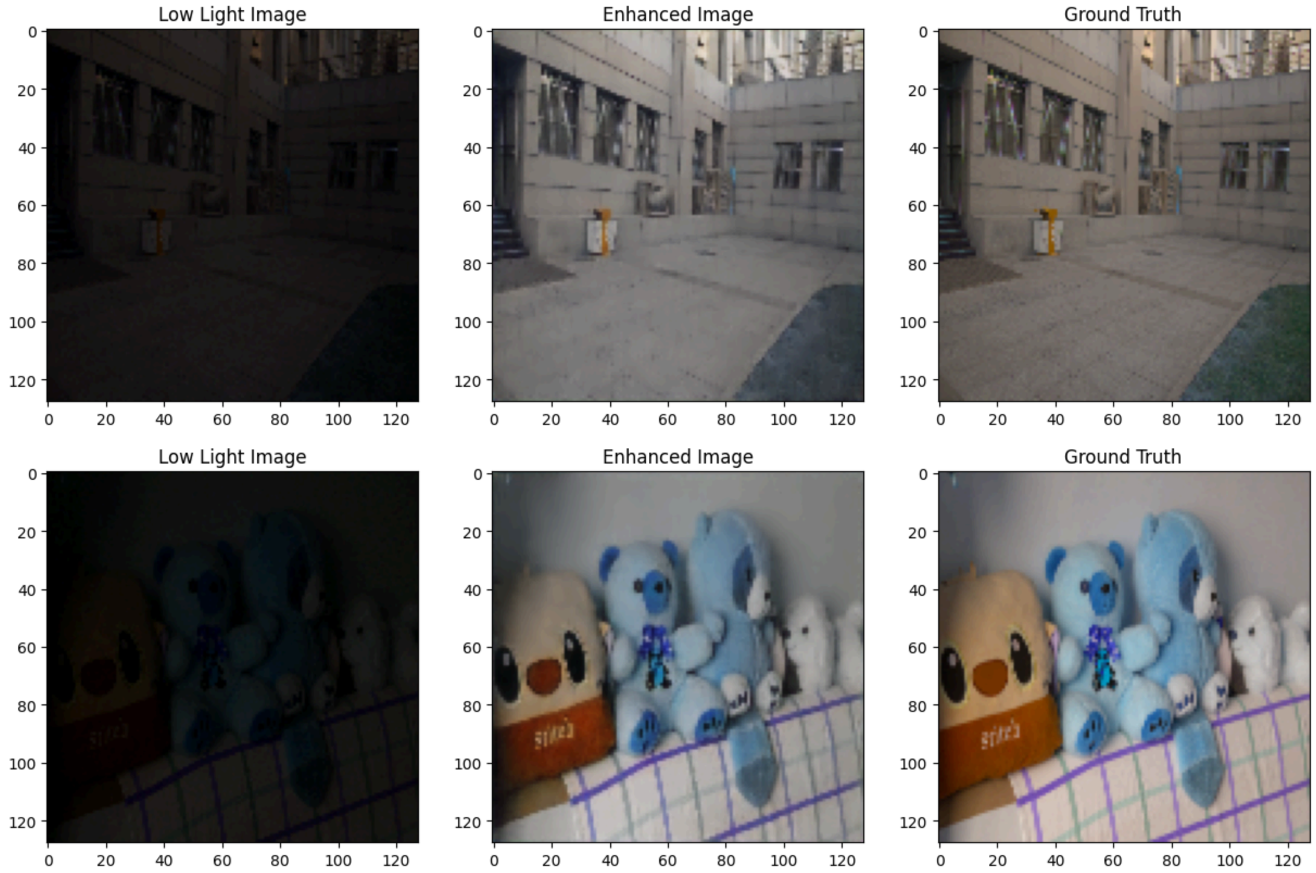
```
from tensorflow.keras.optimizers import Adam #Adaptive Moment Estimation
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import matplotlib.pyplot as plt
from tensorflow.keras.applications import VGG16
from sklearn.metrics import mean_squared_error

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,metrics=['mse'])

# Callbacks
early_stopping = EarlyStopping(patience=20, restore_best_weights=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=120,batch_size=10,validation_data=(X_test, y_test),
                            callbacks=[early_stopping, reduce_lr])
predictions = unet_residual.predict(X_test)

# Calculate Mean Squared Error
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

# Calculate PSNR for the entire test set
psnr = calculate_psnr(y_test, predictions)
print(f'PSNR: {psnr}')
```

```
Epoch 93/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 21s 391ms/step - loss: 0.0964 - mse: 0.0039 - val_loss: 0.1082 - val_mse
Epoch 94/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 20s 382ms/step - loss: 0.0933 - mse: 0.0038 - val_loss: 0.1081 - val_mse
Epoch 95/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 20s 376ms/step - loss: 0.0924 - mse: 0.0037 - val_loss: 0.1087 - val_mse
Epoch 96/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 20s 374ms/step - loss: 0.0947 - mse: 0.0041 - val_loss: 0.1091 - val_mse
Epoch 97/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 20s 374ms/step - loss: 0.0924 - mse: 0.0038 - val_loss: 0.1086 - val_mse
Epoch 98/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 21s 382ms/step - loss: 0.0913 - mse: 0.0035 - val_loss: 0.1090 - val_mse
Epoch 99/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 21s 387ms/step - loss: 0.0902 - mse: 0.0035 - val_loss: 0.1066 - val_mse
Epoch 100/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 20s 375ms/step - loss: 0.0891 - mse: 0.0031 - val_loss: 0.1072 - val_mse
Epoch 101/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 15s 378ms/step - loss: 0.0878 - mse: 0.0030 - val_loss: 0.1073 - val_mse
Epoch 102/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 21s 379ms/step - loss: 0.0908 - mse: 0.0032 - val_loss: 0.1073 - val_mse
Epoch 103/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 20s 373ms/step - loss: 0.0869 - mse: 0.0031 - val_loss: 0.1071 - val_mse
Epoch 104/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 15s 389ms/step - loss: 0.0881 - mse: 0.0030 - val_loss: 0.1075 - val_mse
Epoch 105/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 20s 381ms/step - loss: 0.0885 - mse: 0.0032 - val_loss: 0.1073 - val_mse
Epoch 106/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 15s 376ms/step - loss: 0.0888 - mse: 0.0031 - val_loss: 0.1070 - val_mse
Epoch 107/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 21s 378ms/step - loss: 0.0875 - mse: 0.0029 - val_loss: 0.1069 - val_mse
Epoch 108/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 20s 375ms/step - loss: 0.0893 - mse: 0.0030 - val_loss: 0.1068 - val_mse
Epoch 109/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 15s 387ms/step - loss: 0.0880 - mse: 0.0030 - val_loss: 0.1069 - val_mse
Epoch 110/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 20s 372ms/step - loss: 0.0895 - mse: 0.0031 - val_loss: 0.1070 - val_mse
Epoch 111/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 20s 372ms/step - loss: 0.0884 - mse: 0.0031 - val_loss: 0.1070 - val_mse
Epoch 112/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 21s 383ms/step - loss: 0.0888 - mse: 0.0031 - val_loss: 0.1070 - val_mse
Epoch 113/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 20s 375ms/step - loss: 0.0868 - mse: 0.0029 - val_loss: 0.1071 - val_mse
Epoch 114/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 15s 379ms/step - loss: 0.0873 - mse: 0.0029 - val_loss: 0.1071 - val_mse
Epoch 115/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 15s 375ms/step - loss: 0.0867 - mse: 0.0030 - val_loss: 0.1071 - val_mse
Epoch 116/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 15s 374ms/step - loss: 0.0877 - mse: 0.0030 - val_loss: 0.1071 - val_mse
Epoch 117/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 20s 371ms/step - loss: 0.0868 - mse: 0.0028 - val_loss: 0.1071 - val_mse
Epoch 118/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 21s 377ms/step - loss: 0.0895 - mse: 0.0030 - val_loss: 0.1070 - val_mse
Epoch 119/120
39/39 ━━━━━━━━━━━━━━━━━━━━ 21s 386ms/step - loss: 0.0875 - mse: 0.0030 - val_loss: 0.1070 - val_mse
4/4 ━━━━━━━━━━━━━━━━━━━━ 25s 2s/step
Mean Squared Error: 0.012538667449033973
PSNR: 19.017486158403994
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['mse'],color='red',label='train')
plt.plot(history.history['val_loss'],color='blue',label='validation')
plt.legend()
plt.show()
```




```
# Display some results
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()
```


Low Light Image
Enhanced Image
Ground Truth
Low Light Image
Enhanced Image
Ground Truth