Denoising Image Project Report

1. Introduction

1.1 Project Overview

The objective of this project is to enhance low-light images using a Convolutional Neural Network (CNN). Images captured under poor lighting conditions often suffer from issues such as noise, low visibility, and reduced detail, which can significantly hinder various computer vision tasks. The focus of this project is to develop a model capable of effectively denoising and enhancing these images, thereby improving their quality.

1.2 Objectives

- Develop a CNN Architecture: Create a CNN architecture specifically adapted for the task of image denoising.
- Training the model: Utilize a dataset of paired low and high light images to train the model.
- Evaluate Model Performance: Use appropriate metrics such as Peak Signal-to-Noise Ratio (PSNR) to assess the model's performance.
- Demonstration of Model: Demonstrate its ability to enhance low-light images.

2. Architecture

2.1 Data File

- Dataset is provided by the **VLG**. Dataset contains paired images taken under low and high light conditions. These paired images are essential for training the model because it we enhance the low light images and compare the enhanced images with high light images.

2.2 Model Architecture

- Libraries used
  - Numpy
  - cv2
  - os
  - imageio
  - glob
  - matplotlib
  - tensorflow
  - zipfile

➔ Model is designed to construct and train a convolutional neural network (CNN) for image denoising using the TensorFlow and Keras frameworks.

➔ The model architecture starts with an input layer designed to handle images of unspecified dimensions with three color channels (RGB).

➔ **The core of the network consists of multiple convolutional layers, each followed by batch normalization to improve generalization and prevent overfitting.**

➔ **Specifically, the network employs six convolutional layers, all using a 3x3 kernel size and 'relu' activation function.**

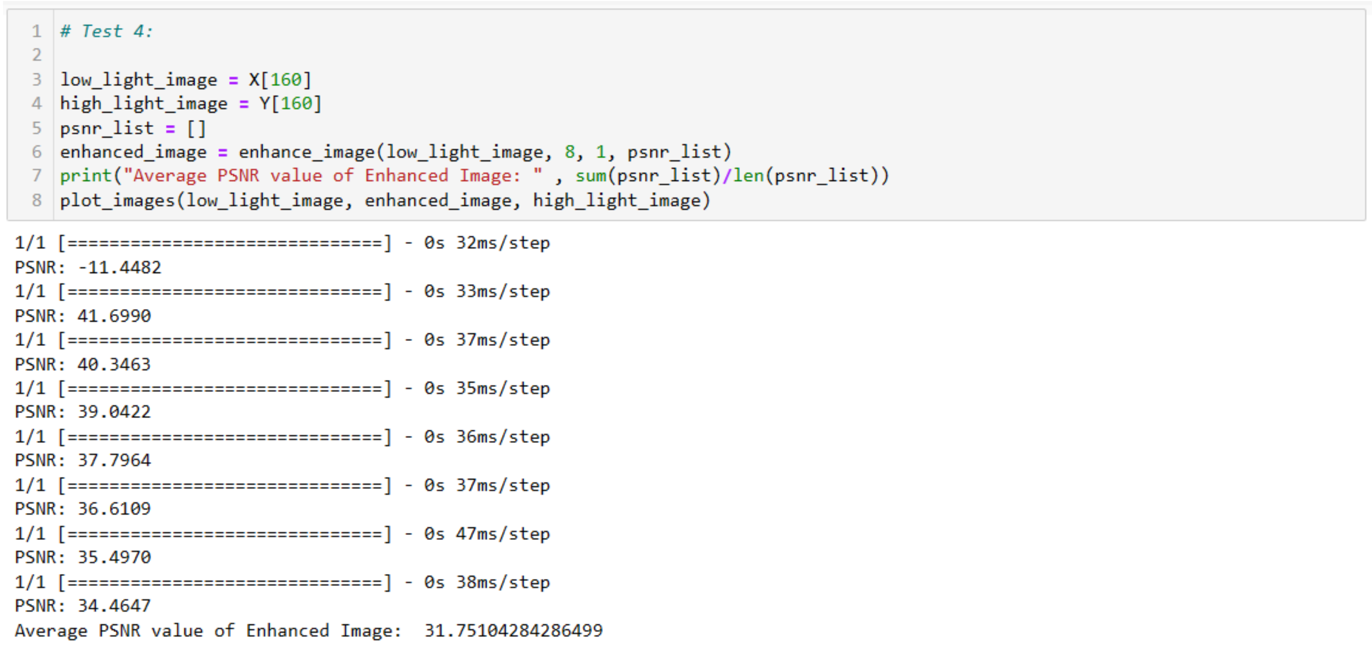
- ➔ Additionally, the model incorporates skip connections after the third and fifth convolutional layers, which help to mitigate the vanishing gradient problem and enhance feature propagation through the network.
- ➔ The final output layer uses a **sigmoid activation function** to generate a single-channel output, representing the denoised image.
- ➔ **The model is compiled with the Adam optimizer, using a learning rate of 0.01, and the mean squared error (MSE) loss function.**
- ➔ The peak signal-to-noise ratio (PSNR) is used as a metric to evaluate the quality of the denoised images. The PSNR metric is defined using TensorFlow's built-in function to ensure accurate assessment during training.

2.3 Training the model

- ➔ For training, the `GenerateInputs` function prepares the input data by reshaping and normalizing low-quality and high-quality image pairs. The high-quality images are converted from grayscale to RGB if necessary, and the inputs are reshaped to match the model's requirements. **The model is trained for 20 epochs with a specified number of steps per epoch.**

3 PNSR Value and enhanced output image

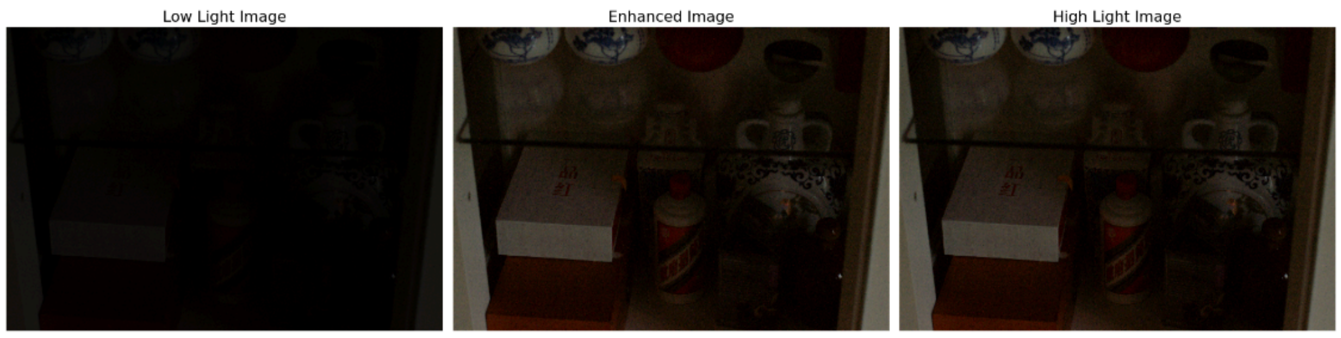
3.1 Example 1:



```python
# Test 4:

low_light_image = X[160]
high_light_image = Y[160]
psnr_list = []
enhanced_image = enhance_image(low_light_image, 8, 1, psnr_list)
print("Average PSNR value of Enhanced Image: " , sum(psnr_list)/len(psnr_list))
plot_images(low_light_image, enhanced_image, high_light_image)
```

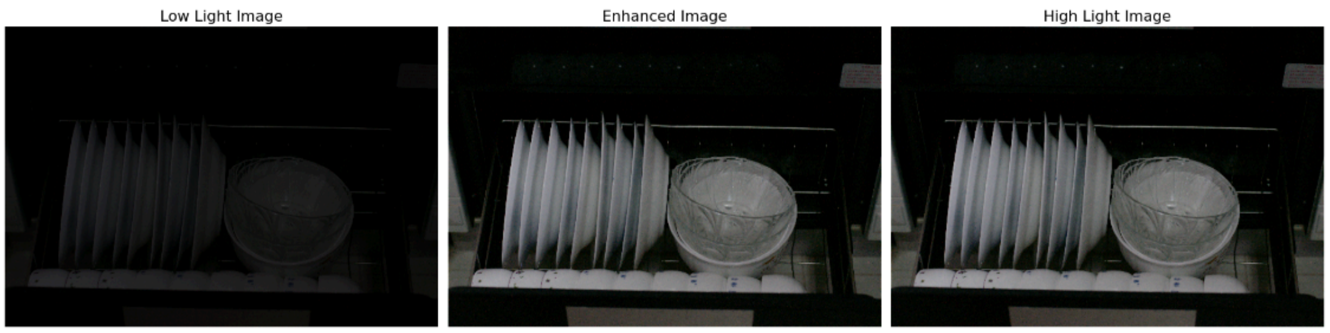
```
1/1 [==============================] - 0s 32ms/step
PSNR:  -11.4482
1/1 [==============================] - 0s 33ms/step
PSNR:  41.6990
1/1 [==============================] - 0s 37ms/step
PSNR:  40.3463
1/1 [==============================] - 0s 35ms/step
PSNR:  39.0422
1/1 [==============================] - 0s 36ms/step
PSNR:  37.7964
1/1 [==============================] - 0s 37ms/step
PSNR:  36.6109
1/1 [==============================] - 0s 47ms/step
PSNR:  35.4970
1/1 [==============================] - 0s 38ms/step
PSNR:  34.4647
Average PSNR value of Enhanced Image:  31.75104284286499
```



3.2 Example 2:

```
# Test 1:
low_light_image = X[1]
high_light_image = Y[1]
psnr_list = []
enhanced_image = enhance_image(low_light_image, 8, 1, psnr_list)
print("Average PSNR value of Enhanced Image: " , sum(psnr_list)/len(psnr_list))
plot_images(low_light_image, enhanced_image, high_light_image)
```

```
1/1 [==============================] - 0s 39ms/step
PSNR: -19.0615
1/1 [==============================] - 0s 32ms/step
PSNR: 36.0969
1/1 [==============================] - 0s 41ms/step
PSNR: 35.0303
1/1 [==============================] - 0s 37ms/step
PSNR: 34.0489
1/1 [==============================] - 0s 35ms/step
PSNR: 33.1616
1/1 [==============================] - 0s 33ms/step
PSNR: 32.3763
1/1 [==============================] - 0s 37ms/step
PSNR: 31.7007
1/1 [==============================] - 0s 37ms/step
PSNR: 31.1417
Average PSNR value of Enhanced Image:  26.811851501464844
```



I have tested 2 images and I got PSNR value for example 1 and example 2 are 31.75 and 26.81 respectively.

4 Code explanation

→ from PIL import Image

```
# Load high quality images
hq_path = r'extracted_dataset/Train/high'
hq_files = [os.path.join(hq_path, f) for f in os.listdir(hq_path) if f.endswith('.png')]
hq_images = []

hq_files.sort()
for fname in hq_files:
    img = Image.open(fname)
    img_array = np.array(img)
    hq_images.append(img_array)
hq_images = np.array(hq_images)
```

```
# Load low quality images
lq_path = r'extracted_dataset/Train/low'
lq_files = [os.path.join(lq_path, f) for f in os.listdir(lq_path) if f.endswith('.png')]
lq_images = []

lq_files.sort()
for fname in lq_files:
    img = Image.open(fname)
    img_array = np.array(img)
    lq_images.append(img_array)
lq_images = np.array(lq_images)

print(hq_images.shape)
print(lq_images.shape)
```

The provided code segment is designed to load high-quality and low-quality images from a specified dataset directory for use in training the image denoising model. **The code uses the Python Imaging Library (PIL) to open and process each image file.** After processing all images, these lists are converted to NumPy arrays, yielding a four-dimensional array where the dimensions represent the number of images, image height, image width, and the number of color channels (RGB). This step is crucial for ensuring that the images are correctly formatted and ready for input into the CNN model. **The use of PIL for image loading ensures compatibility with a wide range of image formats and maintains the integrity of image data.**

→ Firstly I have written the code to delete .DS_Store file but at the end of project I write this “if f.endswith('.png')” in code of loading the dataset. Then there is no use of this code. So, I commented it.

→Creating Model

```
from tensorflow.keras.layers import Conv2D, Add, Input, BatchNormalization, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

kernel_size = (3, 3)
activation_function = 'relu'
padding_type = 'same'
dropout_rate = 0.3

# Input layer
input_layer = Input(shape=(None, None, 3), name='img')

# Convolutional layers with batch normalization and dropout
conv1 = Conv2D(64, kernel_size, activation=activation_function,
padding=padding_type)(input_layer)
bn1 = BatchNormalization()(conv1)
conv1_d = Dropout(dropout_rate)(bn1)

conv2 = Conv2D(64, kernel_size, activation=activation_function,
padding=padding_type)(conv1_d)
```

```
bn2 = BatchNormalization()(conv2)
conv2_d = Dropout(dropout_rate)(bn2)

conv3 = Conv2D(64, kernel_size, activation=activation_function,
padding=padding_type)(conv2_d)
bn3 = BatchNormalization()(conv3)
conv3_d = Dropout(dropout_rate)(bn3)

conv4 = Conv2D(64, kernel_size, activation=activation_function,
padding=padding_type)(conv3_d)
bn4 = BatchNormalization()(conv4)
conv4_d = Dropout(dropout_rate)(bn4)

# Skip connections
skip1 = Add()([conv3_d, conv4_d])

# Additional convolutional layer
conv5 = Conv2D(64, kernel_size, activation=activation_function, padding=padding_type)(skip1)
bn5 = BatchNormalization()(conv5)
conv5_d = Dropout(dropout_rate)(bn5)

# Another skip connection
skip2 = Add()([conv2_d, conv5_d])

# Further convolutional layer
conv6 = Conv2D(64, kernel_size, activation=activation_function, padding=padding_type)(skip2)
bn6 = BatchNormalization()(conv6)
conv6_d = Dropout(dropout_rate)(bn6)

# Final skip connection
skip3 = Add()([conv1_d, conv6_d])

# Output layer with sigmoid activation
output_layer = Conv2D(1, kernel_size, activation='sigmoid', padding=padding_type)(skip3)

# Construct the model
denoising_model = Model(inputs=input_layer, outputs=output_layer)

# Compile the model
denoising_model.compile(optimizer=Adam(learning_rate=0.01), loss='mse',
metrics=[tf.image.psnr])

# Summary of the model
denoising_model.summary()
```

Here I have created CNN for image denoising using the tensorflow and keras libraries. I have use **6 convoultional Layers with all having ReLU (Rectified Linear Unit) activation function** in which input of each layer is output of previous layer. **And 3 skip layer. And sigmoid function for output layer.**

→ Training the model

```
def GenerateInputs(X, Y):

    for i in range(len(X)):

        X_input = X[i].reshape(1, 400, 600, 3)

        Y_temp = Y[i] / 255

        if Y_temp.shape[-1] == 1:

            Y_temp = np.repeat(Y_temp, 3, axis=-1)  # Converting grayscale to RGB by repeating the
channel

        Y_input = Y_temp.reshape(1, 400, 600, 3)

        yield (X_input, Y_input[:, :, :, 0:1])  # Ensuring that y_input has a single channel

denoising_model.fit(GenerateInputs(lq_images, hq_images), epochs=20, verbose=1,
steps_per_epoch=20)  # This epoch number after testing
```

Function Breakdown:

- **Input Parameters**:
  - `X`: A list or array of low-quality images.
  - `Y`: A list or array of high-quality images.
- **Iteration Over Images**:
  - The function iterates over each image in `X` and `Y` using a for-loop (`for i in range(len(X)):`).
- **Reshaping Low-Quality Image**:
  - `X_input = X[i].reshape(1, 400, 600, 3)`: The current low-quality image is reshaped to the dimensions `(1, 400, 600, 3)`, where `1` is the batch size, `400` is the height, `600` is the width, and `3` represents the RGB color channels. This reshape ensures the image is in the expected input format for the model.
- **Normalizing and Reshaping High-Quality Image**:
  - `Y_temp = Y[i] / 255`: The high-quality image is normalized by dividing by 255 to scale the pixel values to the range `[0, 1]`.
  - `if Y_temp.shape[-1] == 1: Y_temp = np.repeat(Y_temp, 3, axis=-1)`: This condition checks if the high-quality image is grayscale (having only one channel). If it is, the channel is repeated three times to convert it to an RGB image.
  - `Y_input = Y_temp.reshape(1, 400, 600, 3)`: The normalized image is reshaped to the dimensions `(1, 400, 600, 3)`.
- **Yielding Input-Output Pairs**:
  - `yield (X_input, Y_input[:, :, :, 0:1])`: The function yields a tuple containing the low-quality image (`X_input`) and the first channel of the high-quality image (`Y_input[:, :, :, 0:1]`). This effectively converts the

high-quality image to a single-channel format suitable for the output of the denoising model.

- **Epochs**:
  - `epochs=20`: The model is trained for 20 epochs. An epoch is one complete pass through the entire training dataset.
- **Verbose:**
  - `verbose=1`: This setting controls the verbosity level of the training process. A value of 1 means that the progress of the training will be displayed, including a progress bar and the training metrics for each epoch.
- **Steps per Epoch**:
  - `steps_per_epoch=20`: This defines the number of steps (batches of samples) to run in each epoch. Here, it specifies that each epoch consists of 20 batches. The total number of iterations per epoch is 20.

→ Testing the model

```
def load_images_from_folder(folder_path):

    all_files = sorted(glob.glob(folder_path + "/*"))

    images = [imageio.imread(file) for file in all_files]

    return np.array(images)

# Define the path to your images

path_low = r'extracted_dataset/Train/low'

path_high = r'extracted_dataset/Train/high'

# Load the low light images

X = load_images_from_folder(path_low)

#Load the high light images

Y = load_images_from_folder(path_high)
```

Function Breakdown:

Getting the List of Files:

- `all_files = sorted(glob.glob(folder_path + "/*"))`: This line uses the `glob` module to get a list of all files in the specified folder. The `*` wildcard ensures that all files are matched. The `sorted` function ensures that the files are processed in a consistent, alphabetical order.

Reading the Images:

- `images = [imageio.imread(file) for file in all_files]`: This line iterates over the list of file paths obtained from the previous step. For each file path, `imageio.imread` is used to read the image into memory. The images are stored in a list called `images`.

Loading the Images

- **Loading Low-Quality Images**:
  - `X = load_images_from_folder(path_low)`: This calls the `load_images_from_folder` function with the path to the low-quality images. The loaded images are stored in the variable `X`.
- **Loading High-Quality Images**:
  - `Y = load_images_from_folder(path_high)`: This calls the `load_images_from_folder` function with the path to the high-quality images. The loaded images are stored in the variable `Y`.

→ Loading our Model

```
from tensorflow.keras.models import load_model

# Define the custom PSNR metric function

def psnr_metric(y_true, y_pred):

    return tf.image.psnr(y_true, y_pred, max_val=1.0)

# Load the model with the custom PSNR metric

model = load_model('mymodel.h5', custom_objects={'psnr_metric': psnr_metric})

# Optional: Print the model summary to verify

model.summary()
```

Function Breakdown:

- **Parameters**:
  - `y_true`: The ground truth images.
  - `y_pred`: The predicted images produced by the model.
- **Function Body**:
  - `return tf.image.psnr(y_true, y_pred, max_val=1.0)`: This line uses TensorFlow's built-in function `tf.image.psnr` to calculate the PSNR value. The `max_val=1.0` parameter indicates that the pixel values are normalized between 0 and 1.
- **Parameters**:

- `'mymodel.h5'`: The path to the saved model file.
- `custom_objects={'psnr_metric': psnr_metric}`: This dictionary maps the name of the custom metric function to the actual function definition.

```
def psnr_metric(y_true, y_pred):

    return tf.image.psnr(y_true, y_pred, max_val=1.0)


def enhance_image(img, index, flag, psnr_list):

    if index == 0:

        return img


    elif flag == 1:

        h, w, c = img.shape

        test = model.predict(img.reshape(1, h, w, 3))

        temp = img / 255.0

        enhanced_image = temp + ((test[0,:,:,:] * temp) * (1 - temp))

        psnr_value = psnr_metric(tf.convert_to_tensor(img, dtype=tf.float32),
tf.convert_to_tensor(enhanced_image * 255, dtype=tf.float32)).numpy()

        print(f"PSNR: {psnr_value:.4f}")

        psnr_list.append(psnr_value)

        index -= 1

        flag = 0

        return enhance_image(enhanced_image, index, flag, psnr_list)


    else:

        h, w, c = img.shape

        temp = model.predict(img.reshape(1, h, w, 3))
        enhanced_image = img + ((temp[0,:,:,:] * img) * (1 - img))

        psnr_value = psnr_metric(tf.convert_to_tensor(img, dtype=tf.float32),
tf.convert_to_tensor(enhanced_image, dtype=tf.float32)).numpy()

        print(f"PSNR: {psnr_value:.4f}")

        psnr_list.append(psnr_value)

        index -= 1

        return enhance_image(enhanced_image, index, flag, psnr_list)
```

Function Breakdown:

- **Parameters**:
  - `y_true`: The ground truth image tensor.
  - `y_pred`: The predicted or enhanced image tensor.
- **Return**:
  - The PSNR value calculated using TensorFlow's built-in `tf.image.psnr` function with `max_val=1.0`, indicating that the pixel values are normalized between 0 and 1

```
def plot_images(low_light_image, enhanced_image, high_light_image):

    plt.figure(figsize=(20, 15))


    # Plot low light image

    plt.subplot(1, 3, 1)

    plt.title("Low Light Image", fontsize=16)

    plt.imshow(low_light_image)

    plt.axis('off')


    # Plot enhanced image

    plt.subplot(1, 3, 2)

    plt.title("Enhanced Image", fontsize=16)

    plt.imshow(enhanced_image)

    plt.axis('off')
```

```
# Plot High light image
plt.subplot(1, 3, 3)
plt.title("High Light Image", fontsize=16)
plt.imshow(enhanced_image)
plt.axis('off')

plt.tight_layout()
plt.show()
```

6. PSNR Values:

I have 4 test on this model and I have got following PSNR values:

1. **PSNR Value:- 26.811**
2. **PSNR value:- 25.53**
3. **PSNR value;- 25.55**
4. **PSNR values:- 31.75**