Low Light enchancement

Project Overview

Low-light conditions often pose significant challenges for image processing tasks, affecting both visual quality and the performance of subsequent computer vision algorithms. The aim of this project is to develop and compare methods for enhancing images captured in low-light environments. The project involves implementing and evaluating traditional enhancement techniques as well as a novel method based on the Low-light Image Enhancement (LIME) algorithm.

Motivation

Poor lighting conditions can severely impact the quality of captured images, making it difficult to extract useful information. Enhancing low-light images is essential for various applications, including surveillance, medical imaging, photography, and autonomous driving. By improving image visibility and contrast, we can ensure better performance for automated systems and provide clearer visual information for human inspection.

Objectives

1. **Implement Traditional Enhancement Techniques**: Apply traditional image enhancement methods, such as brightness adjustment, contrast enhancement, and gamma correction.
2. **Implement LIME Algorithm**: Implement the LIME algorithm to enhance low-light images by improving local contrast and brightness.
3. **Compare Methods**: Evaluate the performance of both traditional and LIME-based enhancement methods using standard metrics such as PSNR (Peak Signal-to-Noise Ratio), SSIM (Structural Similarity Index), and MSE (Mean Squared Error).
4. **Create Demonstrative Videos**: Generate output videos showcasing the enhanced images using both techniques for side-by-side comparison.

Methodology

1. **Data Collection**:
   - Capture or collect a set of low-light images and videos for testing the enhancement algorithms.
2. **Traditional Enhancement Techniques**:
   - **Brightness Adjustment**: Increase the brightness of the image by scaling pixel values.
   - **Contrast Enhancement**: Apply histogram equalization to improve image contrast.
   - **Gamma Correction**: Adjust the gamma values to correct the overall brightness of the image.
3. **LIME Algorithm**:

    - Implement the LIME algorithm which enhances the image by decomposing it into illumination and reflectance components. The algorithm focuses on improving the illumination component to achieve better visibility.
4. **Video Processing**:
    - Extract frames from the input video.
    - Apply enhancement techniques to each frame.
    - Compile the enhanced frames back into a video for visualization.
5. **Performance Evaluation**:
    - Compute PSNR, SSIM, and MSE between the original and enhanced images to quantify improvement.
    - Visualize results using side-by-side comparisons and metric values.

Source code:

```
 import argparse

from argparse import RawTextHelpFormatter

import glob

from os import makedirs

from os.path import join, exists, basename, splitext

import cv2

from tqdm import tqdm

from exposure_enhancement import enhance_image_exposure

import numpy as np

from scipy import fft

from skimage import io, exposure, img_as_ubyte, img_as_float

import matplotlib.pyplot as plt

import os

from PIL import Image

import shutil

from skimage.metrics import peak_signal_noise_ratio as psnr, structural_similarity as ssim

from sklearn.metrics import mean_squared_error


input_video_path = 'Input Videos/Amazing night vision - ColorVu Camera Demo.mp4'

input_frames_folder = 'input_frames/'

output_fps = 5  # frames per second
```

```
duration = 5  # seconds
output_frames_folder1 = 'output_frames_existing/'
output_video_path_existing = 'output_video_existing.avi'
output_frames_folder = 'output_frames/'
output_video_path_proposed = 'output_video_proposed.avi'
from IPython.display import HTML

# Create HTML code to embed the video
video_html = """
<video width="640" height="480" controls>
  <source src="{}" type="video/mp4">
  Your browser does not support the video tag.
</video>
""".format(input_video_path)

# Display the HTML code in the Jupyter Notebook
HTML(video_html)

def enhance_image_existing(input_path, output_path):
    # Load the image
    brightness_alpha = 1.2
    contrast_beta = 10
    gamma = 1.5
    img = cv2.imread(input_path)

    # Apply brightness adjustment
    enhanced_brightness = cv2.convertScaleAbs(img, alpha=brightness_alpha, beta=0)

    # Apply contrast adjustment
    enhanced_contrast = cv2.addWeighted(enhanced_brightness, 1, img, 0, contrast_beta)
    # Apply gamma correction for overall enhancement
    enhanced_img = np.power(enhanced_contrast / 255.0, gamma)
    enhanced_img = np.uint8(enhanced_img * 255)

    # Save the enhanced image
    cv2.imwrite(output_path, enhanced_img)
    return enhanced_img

# Open the video file
cap = cv2.VideoCapture(input_video_path)
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
num_frames = output_fps * duration

# Create the input frames folder if it doesn't exist
if not exists(input_frames_folder):
    makedirs(input_frames_folder)

# Create the output frames folder for existing method if it doesn't exist
if not exists(output_frames_folder1):
    makedirs(output_frames_folder1)

# Create a video writer object for the existing method
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter(output_video_path_existing, fourcc, 25.0, (frame_width,
frame_height))

# Read frames from the video
frame_count = 0
while True:
    ret, frame = cap.read()
    print(f"Frame count is {frame_count}")
    if not ret or frame_count >= num_frames:
        break

    # Save the input frame
    input_frame_path = os.path.join(input_frames_folder, f"input_frame_{frame_count +
1}.png")
    frame = cv2.resize(frame, (400, 400))
    cv2.imwrite(input_frame_path, frame)

    # Apply enhancement to the input frame and save the output frame
    output_frame_path = os.path.join(output_frames_folder1, f"output_frame_{frame_count +
1}.png")
    enhance_image_existing(input_frame_path, output_frame_path)

    for _ in range(3):
        out.write(cv2.imread(output_frame_path))

    frame_count += 1
    print(f"Output Frames Completed for Existing Method: {frame_count}")

# Release video capture and writer objects
cap.release()
out.release()
print(f"Processing completed. Existing Output video created and stored at
{output_video_path_existing}")

# Create a video from frames for the existing method
image_files = [f for f in os.listdir(output_frames_folder1) if f.endswith('.png')]
if image_files:
    image_files.sort()
    first_image = cv2.imread(os.path.join(output_frames_folder1, image_files[0]))
    height, width, _ = first_image.shape

    out = cv2.VideoWriter(output_video_path_existing, fourcc, 25.0, (width, height))
    for image_file in image_files:
        frame = cv2.imread(os.path.join(output_frames_folder1, image_file))
        out.write(frame)
    out.release()
    print(f"Video created: {output_video_path_existing}")

def enhance_image_with_lime(input_path, output_path):
    image = cv2.imread(input_path)
    enhanced_image = enhance_image_exposure(image, 0.6, 0.15, "lime")
    cv2.imwrite(output_path, enhanced_image)

# Open the video file again for the proposed method
cap = cv2.VideoCapture(input_video_path)

# Create the output frames folder for proposed method if it doesn't exist
if not exists(output_frames_folder):
    makedirs(output_frames_folder)

# Create a video writer object for the proposed method
out = cv2.VideoWriter(output_video_path_proposed, fourcc, 25.0, (frame_width,
frame_height))

frame_count = 0
while True:
    ret, frame = cap.read()
    print(f"Frame count is {frame_count}")
    if not ret or frame_count >= num_frames:
        break

    # Save the input frame
    input_frame_path = os.path.join(input_frames_folder, f"input_frame_{frame_count + 1}.png")
    frame = cv2.resize(frame, (400, 400))
    cv2.imwrite(input_frame_path, frame)

    # Apply LIME enhancement to the input frame and save the output frame
    output_frame_path = os.path.join(output_frames_folder, f"output_frame_{frame_count + 1}.png")
    enhance_image_with_lime(input_frame_path, output_frame_path)

    for _ in range(3):
        out.write(cv2.imread(output_frame_path))

    frame_count += 1
    print(f"Output Frames Completed for Proposed Method: {frame_count}")

cap.release()
out.release()
print(f"Processing completed. Proposed Output video created and stored at {output_video_path_proposed}")

# Create a video from frames for the proposed method
image_files = [f for f in os.listdir(output_frames_folder) if f.endswith('.png')]
if image_files:
    image_files.sort()
    first_image = cv2.imread(os.path.join(output_frames_folder, image_files[0]))
    height, width, _ = first_image.shape
    out = cv2.VideoWriter(output_video_path_proposed, fourcc, 25.0, (width, height))
    for image_file in image_files:
        frame = cv2.imread(os.path.join(output_frames_folder, image_file))
        out.write(frame)
    out.release()
    print(f"Video created: {output_video_path_proposed}")

# Visualization and Metric Calculation

def compute_metrics(original, enhanced):
    psnr_value = psnr(original, enhanced)
    ssim_value, _ = ssim(original, enhanced, full=True)
    mse_value = mean_squared_error(original.flatten(), enhanced.flatten())
    return psnr_value, ssim_value, mse_value

# Example comparison for a single frame or image

input_image_path = 'sample images/15.jpg'

# Existing method enhancement
img = cv2.imread(input_image_path)
enhanced_brightness = cv2.convertScaleAbs(img, alpha=brightness_alpha, beta=0)
enhanced_contrast = cv2.addWeighted(enhanced_brightness, 1, img, 0, contrast_beta)
enhanced_img1 = np.power(enhanced_contrast / 255.0, gamma)
enhanced_img1 = np.uint8(enhanced_img1 * 255)

# Proposed method enhancement
enhanced_img2 = enhance_image_exposure(img, 0.6, 0.15, "lime")
```

```
plt.figure(figsize=(10, 4))
plt.subplot(131)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')

plt.subplot(132)
plt.imshow(cv2.cvtColor(enhanced_img1, cv2.COLOR_BGR2RGB))
plt.title('Existing Output')

plt.subplot(133)
plt.imshow(cv2.cvtColor(enhanced_img2, cv2.COLOR_BGR2RGB))
plt.title('Proposed Output')

plt.show()

# Compute metrics for both methods
psnr_existing, ssim_existing, mse_existing = compute_metrics(img, enhanced_img1)
psnr_proposed, ssim_proposed, mse_proposed = compute_metrics(img, enhanced_img2)

print("Metrics for Existing Enhancement:")
print("PSNR:", psnr_existing)
print("SSIM:", ssim_existing)
print("MSE:", mse_existing)

print("\nMetrics for Proposed Enhancement:")
print("PSNR:", psnr_proposed)
print("SSIM:", ssim_proposed)
print("MSE:", mse_proposed)
```