

#PSNR (Peak Signal-to-Noise Ratio): Measures the quality of the enhanced image compared to the original image in terms of noise.

#SSIM (Structural Similarity Index): Measures the similarity between the enhanced image and the original image in terms of structural information.

import cv2

import numpy as np

from skimage.metrics import peak\_signal\_noise\_ratio as psnr

from skimage.metrics import structural\_similarity as ssim

from google.colab.patches import cv2\_imshow

# Read the original image

original\_image = cv2.imread('/content/Octopus1/img/10.jpg')

# Check if the original image is None

if original\_image is None:

print("Error: Unable to read the original image.")

else:

# Define a function to evaluate image enhancement techniques

def evaluate\_performance(enhanced\_image):

# Calculate PSNR

psnr\_score = psnr(original\_image, enhanced\_image)

# Calculate SSIM

ssim\_score = ssim(original\_image, enhanced\_image, multichannel=True)

return psnr\_score, ssim\_score

# Define a function to display images and evaluation scores

def display\_results(original\_image, enhanced\_image, psnr\_score, ssim\_score):

# Display original and enhanced images

cv2\_imshow(original\_image)

cv2\_imshow(enhanced\_image)

# Print evaluation scores

print('PSNR:', psnr\_score)

print('SSIM:', ssim\_score)

# Read the enhanced image obtained from the specific technique

enhanced\_image = cv2.imread('enhanceDsd\_image.jpg')

# Check if the enhanced image is None

if enhanced\_image is None:

print("Error: Unable to read the enhanced image.")

else:

# Evaluate performance using objective metrics

psnr\_score, ssim\_score = evaluate\_performance(enhanced\_image)

# Display results

display\_results(original\_image, enhanced\_image, psnr\_score, ssim\_score)



import cv2

def get\_video\_duration(video\_path):

# Open the video file

video = cv2.VideoCapture(video\_path)

# Get the total number of frames and frames per second (fps)

total\_frames = int(video.get(cv2.CAP\_PROP\_FRAME\_COUNT))

fps = int(video.get(cv2.CAP\_PROP\_FPS))

# Calculate the duration in seconds

duration\_sec = total\_frames / fps

# Convert duration to hours, minutes, and seconds

duration\_hours = int(duration\_sec // 3600)

duration\_min = int((duration\_sec % 3600) // 60)

duration\_sec = int(duration\_sec % 60)

# Close the video file

video.release()

return duration\_hours, duration\_min, duration\_sec

# Path to the video file

video\_path = "/content/Octopus1/Octopus1.mp4"

# Get the duration of the video

duration\_hours, duration\_min, duration\_sec = get\_video\_duration(video\_path)

# Print the duration

print(f"Video Duration: {duration\_hours} hours, {duration\_min} minutes, {duration\_sec} seconds")



import cv2

from google.colab.patches import cv2\_imshow

# Open the underwater video file

input\_video = cv2.VideoCapture('/content/Octopus1/Octopus1.mp4')

# Get the video properties

frame\_width = int(input\_video.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

frame\_height = int(input\_video.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

fps = int(input\_video.get(cv2.CAP\_PROP\_FPS))

total\_frames = int(input\_video.get(cv2.CAP\_PROP\_FRAME\_COUNT))

# Create a VideoWriter object to save the enhanced video

output\_video = cv2.VideoWriter('enhanced\_video.mp4',

cv2.VideoWriter\_fourcc(\*'mp4v'),

fps,

(frame\_width, frame\_height))

# Loop through each frame of the video

while input\_video.isOpened():

ret, frame = input\_video.read()

if not ret:

break

# Convert the frame to grayscale

gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Apply histogram equalization to enhance the frame

equalized\_frame = cv2.equalizeHist(gray\_frame)

# Convert the equalized frame back to BGR (if you want to display it)

equalized\_frame\_bgr = cv2.cvtColor(equalized\_frame, cv2.COLOR\_GRAY2BGR)

# Write the enhanced frame to the output video

output\_video.write(equalized\_frame\_bgr)

# Display the enhanced frame (optional)

cv2\_imshow(equalized\_frame\_bgr)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

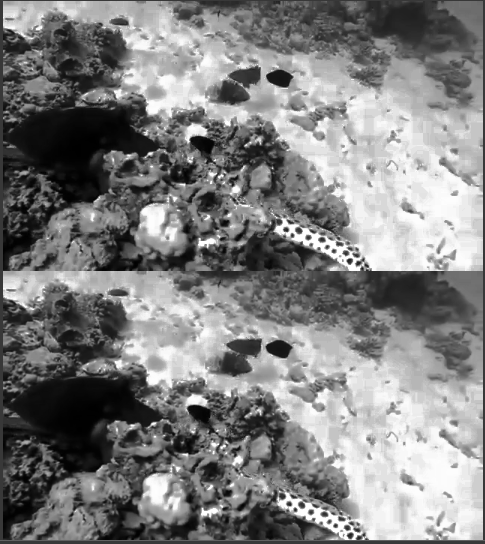
# Release the input and output video objects

input\_video.release()

output\_video.release()

# Close all OpenCV windows

cv2.destroyAllWindows()



import os

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D

from tensorflow.keras.models import Model

from sklearn.model\_selection import train\_test\_split

# Load and preprocess image data

def load\_images(directory):

images = []

valid\_extensions = ['.jpg', '.jpeg', '.png', '.bmp'] # Add more extensions if needed

for filename in os.listdir(directory):

filepath = os.path.join(directory, filename)

if os.path.isfile(filepath) and os.path.splitext(filename)[1].lower() in valid\_extensions:

try:

img = Image.open(filepath)

img = img.resize((64, 64)) # Resize image to (64, 64)

img = img.convert('RGB') # Convert image to RGB (in case it's grayscale)

img = np.array(img)

images.append(img)

except Exception as e:

print(f"Error loading image file: {filepath}. Exception: {e}")

if not images:

print("No images found in the directory.")

return np.array(images)

# Load images from directory

image\_directory = "/content/Octopus1/img"

images = load\_images(image\_directory)

# Normalize images

images = images.astype('float32') / 255.0

# Split data into training and validation sets

X\_train, X\_val = train\_test\_split(images, test\_size=0.2, random\_state=42)

# Define autoencoder model architecture

input\_img = Input(shape=(64, 64, 3))

x = Conv2D(32, (3, 3), activation='relu', padding='same')(input\_img)

x = MaxPooling2D((2, 2), padding='same')(x)

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

encoded = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(encoded)

x = UpSampling2D((2, 2))(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)

x = UpSampling2D((2, 2))(x)

decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input\_img, decoded)

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

# Train the autoencoder

history = autoencoder.fit(X\_train, X\_train,

epochs=10,

batch\_size=32,

shuffle=True,

validation\_data=(X\_val, X\_val))

# Plot loss curves

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

# Visualize original and reconstructed images

decoded\_images = autoencoder.predict(X\_val)

n = 10 # Number of images to display

plt.figure(figsize=(20, 4))

for i in range(n):

# Original images

ax = plt.subplot(2, n, i + 1)

plt.imshow(X\_val[i])

plt.title('Original')

plt.axis('off')

# Reconstructed images

ax = plt.subplot(2, n, i + 1 + n)

plt.imshow(decoded\_images[i])

plt.title('Reconstructed')

plt.axis('off')

plt.show()

