Name: Seshasai.PB Reg_N0: 21MIA1005

Image and Video Analytics Assignment 2

Lab Task 1: Setup and Basic Extraction

Objective:

Install the necessary tools and libraries, and extract frame information from a video.

Steps:

1. **Install ffmpeg and ffmpeg-python**:
   - Install the ffmpeg tool and the ffmpeg-python library.
2. **Extract Frame Information**:
   - Extract frame information from a sample video.

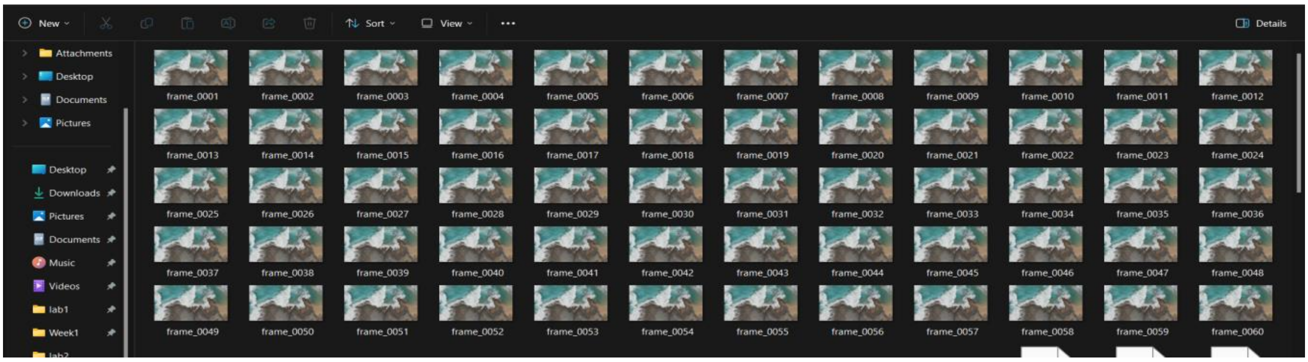
Code:

```
import sys

import ffmpeg

sys.path.append(r'C:\ffmpeg')

input_file = 'in.mp4'
output_pattern = 'frames/frame_%04d.jpeg'

ffmpeg.input(input_file).output(output_pattern).run()
```

my spyder console is not recognizing **‘ffmpeg’** I’m running it in **‘command prompt’**

```
D:\Sem7\Image and video analytics\Lab\lab2>python lab.py
```

So after running this we get our frames



To get information about each frame we can run this prompt in cmd

```
D:\Sem7\Image and video analytics\Lab\lab2>ffprobe -show_frames in.mp4
```

This will display information about each frame

```
color_range=tv
color_space=bt709
color_primaries=bt709
color_transfer=bt709
chroma_location=left
[/FRAME]
[FRAME]
media_type=video
stream_index=0
key_frame=0
pts=526
pts_time=21.040000
pkt_dts=N/A
pkt_dts_time=N/A
best_effort_timestamp=526
best_effort_timestamp_time=21.040000
duration=1
duration_time=0.040000
pkt_pos=56529996
pkt_size=197937
width=3840
height=2160
crop_top=0
crop_bottom=0
crop_left=0
crop_right=0
pix_fmt=yuv420p
sample_aspect_ratio=N/A
pict_type=P
interlaced_frame=0
top_field_first=0
repeat_pict=0
color_range=tv
color_space=bt709
color_primaries=bt709
color_transfer=bt709
chroma_location=left
[/FRAME]
[FRAME]
media_type=video
```

Lab Task 2: Frame Type Analysis

Objective:

Analyze the extracted frame information to understand the distribution of I, P, and B frames in a video.

Steps:

1. **Modify the Script**:
   - Count the number of I, P, and B frames.
   - Calculate the percentage of each frame type in the video.
2. **Analyze Frame Distribution**:
   - Plot the distribution of frame types using a library like matplotlib.
   - Plot a pie chart or bar graph showing the distribution of frame types using matplotlib.

To get type of frame information, Enter

```
D:\Sem7\Image and video analytics\Lab\lab2>ffprobe -show_frames in.mp4 | findstr "pict_type"
```

This will display type of each frame

```
pict_type=I
pict_type=B
pict_type=B
pict_type=B
pict_type=P
pict_type=B
pict_type=B
pict_type=B
pict_type=P
pict_type=B
pict_type=B
pict_type=B
pict_type=P
pict_type=B
pict_type=B
pict_type=B
pict_type=P
pict_type=B
pict_type=B
pict_type=B
pict_type=P
pict_type=B
pict_type=B
pict_type=B
pict_type=P
pict_type=B
pict_type=B
pict_type=B
pict_type=P
pict_type=B
pict_type=B
pict_type=B
pict_type=P
pict_type=B
pict_type=B
pict_type=B
pict_type=P
```

We can extract this and information to analyse and visualize the frames to get information

Code:

```
frame_types =["I"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
```

```
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"I"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
```

```
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
```

```
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"I"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
```

```
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"P"
,"B"
,"B"
,"B"
,"P"
,"I"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
```

```
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
```

```
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"I"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
```

```
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
```

```
,"P"
,"I"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
```

```
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"I"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
```

```
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
,"P"
,"B"
,"B"
,"B"
```

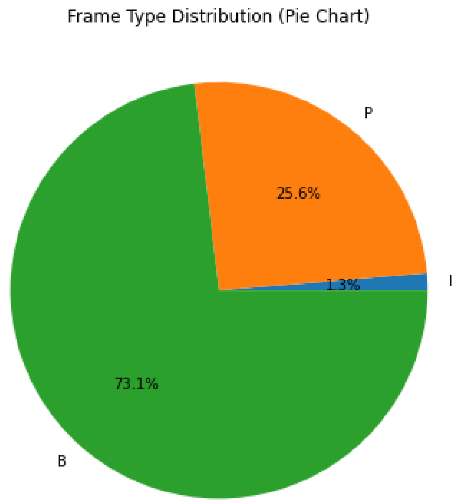
```
,"P"
,"B"
,"B"
,"P"
,"P"]






# Count the number of each frame type
frame_counts = {'I': 0, 'P': 0, 'B': 0}
for frame_type in frame_types:
    if frame_type in frame_counts:
        frame_counts[frame_type] += 1

# Calculate total frames
total_frames = len(frame_types)

# Calculate percentage of each frame type
frame_percentages = {ftype: (count / total_frames) * 100 for ftype,
count in frame_counts.items()}

# Print the results
print(f"Frame Counts: {frame_counts}")
print(f"Frame Percentages: {frame_percentages}")


import matplotlib.pyplot as plt

def plot_distribution(frame_counts, frame_percentages):
    # Plotting the Pie Chart
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.pie(frame_counts.values(), labels=frame_counts.keys(),
autopct='%1.1f%%')
    plt.title('Frame Type Distribution (Pie Chart)')

    # Plotting the Bar Graph
    plt.subplot(1, 2, 2)
    plt.bar(frame_counts.keys(), frame_counts.values(), color=['red',
'green', 'blue'])
    plt.xlabel('Frame Type')
    plt.ylabel('Count')
    plt.title('Frame Type Distribution (Bar Graph)')

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

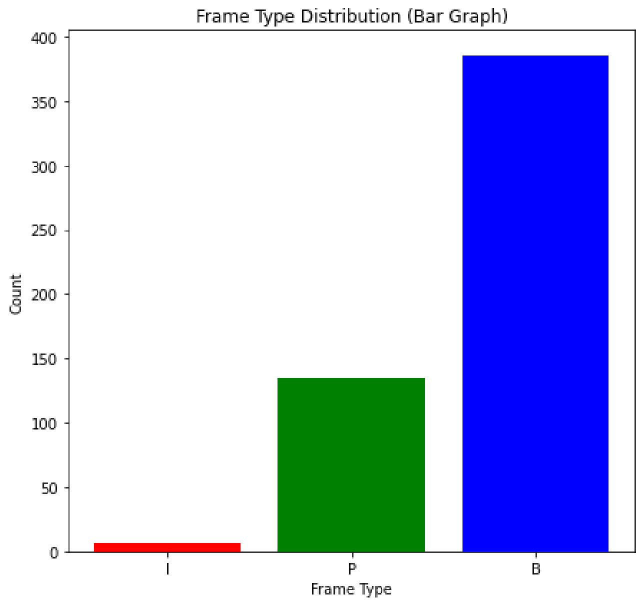
```
plot_distribution(frame_counts, frame_percentages)
```

Output:

Frame Counts: {'I': 7, 'P': 135, 'B': 386}

Frame Percentages: {'I': 1.3257575757575757, 'P': 25.568181818181817, 'B': 73.10606060606061}





Lab Task 3: Visualizing Frames

Objective:

Extract actual frames from the video and display them using Python.

Steps:

1. **Extract Frames**:
   - Use ffmpeg to extract individual I, P, and B frames from the video.
   - Save these frames as image files.
2. **Display Frames**:
   - Use a library like PIL (Pillow) or opencv-python to display the extracted frames.

Tasks:

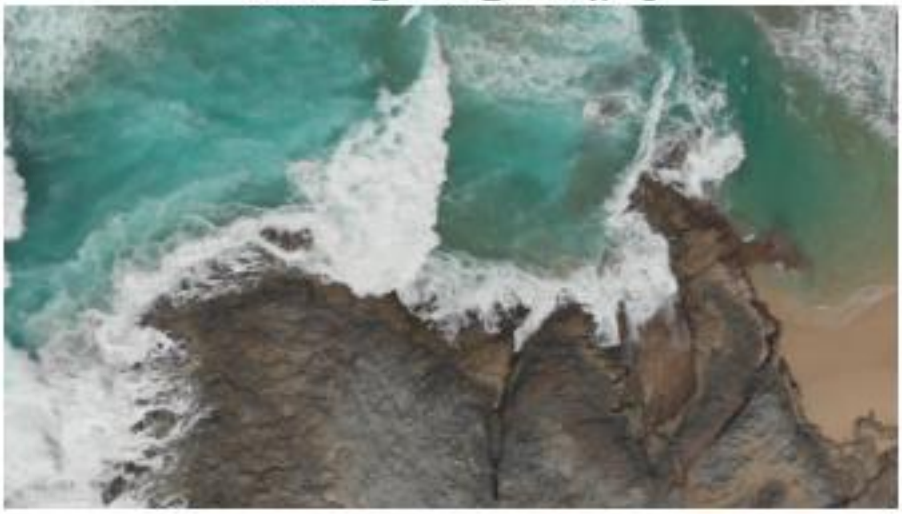
1. Save I, P, and B frames as separate image files using ffmpeg.
2. Use PIL or opencv-python to load and display these frames in a Python script.
3. Compare the visual quality of I, P, and B frames.

Code:

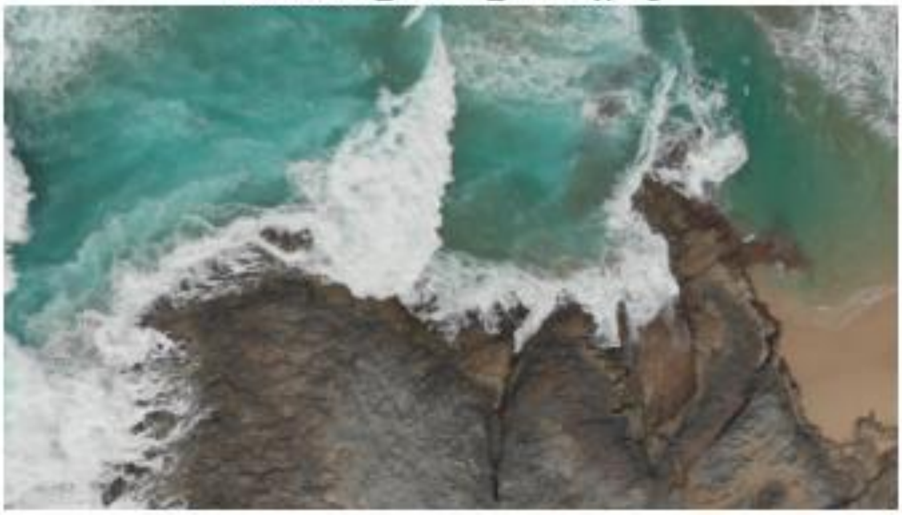
```
import os
# Path to frames directory
frames_dir = './frames/'  # Adjust this path as needed

# Rename frames
for i, frame_type in enumerate(frame_types):
    original_filename = f"frame_{i+1:04d}.jpeg"
    new_filename = f"{frame_type}_frame_{i+1:04d}.jpeg"
    original_path = os.path.join(frames_dir, original_filename)
    new_path = os.path.join(frames_dir, new_filename)
    if os.path.exists(original_path):
        os.rename(original_path, new_path)
    else:
        print(f"File {original_filename} does not exist in the 
directory.")

import cv2
import matplotlib.pyplot as plt

# Define a function to display images using OpenCV
def display_frames(frame_paths):
    for path in frame_paths:
        # Load the image
        image = cv2.imread(path)
        # Convert BGR to RGB for displaying using matplotlib
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        plt.imshow(image_rgb)
        plt.title(f"Frame: {os.path.basename(path)}")
        plt.axis('off')
        plt.show()

# Example paths (Replace with actual paths after renaming)
example_I_frame = os.path.join(frames_dir, 'I_frame_0001.jpeg')
example_P_frame = os.path.join(frames_dir, 'P_frame_0005.jpeg')
example_B_frame = os.path.join(frames_dir, 'B_frame_0002.jpeg')

# Displaying example frames
display_frames([example_I_frame, example_P_frame, example_B_frame])
```

Quality comparison of different image types:

Frame: I_frame_0001.jpeg



Frame: P_frame_0005.jpeg



Frame: B_frame_0002.jpeg

Lab Task 4: Frame Compression Analysis

Objective:

Analyze the compression efficiency of I, P, and B frames.

Steps:

1. **Calculate Frame Sizes**:
    - Calculate the file sizes of extracted I, P, and B frames.
    - Compare the average file sizes of each frame type.
2. **Compression Efficiency**:
    - Discuss the role of each frame type in video compression.
    - Analyze why P and B frames are generally smaller than I frames.

Code:

```
import os

# Directory containing the renamed frames
frames_dir = './frames/'

# Initialize dictionaries to store sizes and counts
frame_sizes = {'I': [], 'P': [], 'B': []}

# Calculate the file sizes
for filename in os.listdir(frames_dir):
    if filename.startswith('I_') or filename.startswith('P_') or 
filename.startswith('B_'):
        frame_type = filename.split('_')[0]
        file_path = os.path.join(frames_dir, filename)
        file_size = os.path.getsize(file_path)
        frame_sizes[frame_type].append(file_size)

# Calculate average sizes
average_sizes = {frame_type: sum(sizes) / len(sizes) if sizes else 0 
for frame_type, sizes in frame_sizes.items()}

# Print out the results
print("Average File Sizes (bytes):")
for frame_type, avg_size in average_sizes.items():
    print(f"{frame_type}: {avg_size:.2f} bytes")
```

Output:

```
Sem7/Image and video analytics/Lab/lab2')
Average File Sizes (bytes):
I: 225845.00 bytes
P: 198026.24 bytes
B: 196880.56 bytes
```

Analysis

These results align more closely with typical expectations, where I-frames are larger than P-frames and B-frames:

1. **I-Frames** are expected to be larger because they store a complete image without reference to other frames.
2. **P-Frames** are smaller than I-frames as they only store differences from previous frames, using predictive coding.
3. **B-Frames** are usually the smallest, leveraging both past and future frames to encode differences with high efficiency.

Lab Task 5: Advanced Frame Extraction

Objective:

Extract frames from a video and reconstruct a part of the video using only I frames.

Steps:

1. **Extract and Save I Frames**:
   - Extract I frames from the video and save them as separate image files.
2. **Reconstruct Video**:
   - Use the extracted I frames to reconstruct a portion of the video.
   - Create a new video using these I frames with a reduced frame rate.

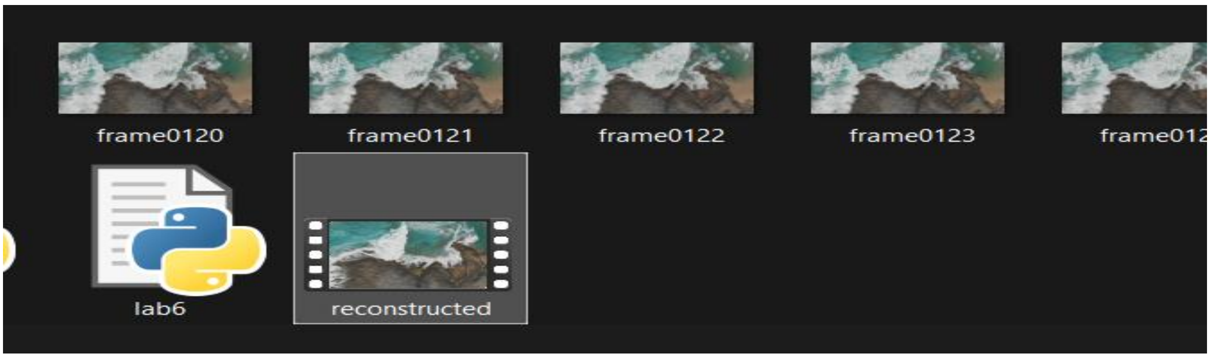
Code:

```
import cv2
import os

# Path to the directory containing I frames
i_frame_dir = './I_frames/'  # Update this path as needed
output_video_path = 'reconstructed.mp4'  # Output path for the
reconstructed video

# Define frame rate (we'll use 3.5 fps for at least 2 seconds duration)
frame_rate = 3.5
num_frames = 7
```

```
# Get the list of frame file names
frame_files = [f for f in sorted(os.listdir(i_frame_dir)) if
f.startswith('I_frame_')]

# Check if the number of frames matches the expected count
if len(frame_files) != num_frames:
    print(f"Error: Expected {num_frames} frames, found
{len(frame_files)}.")
else:
    # Load the first frame to get the frame size
    first_frame = cv2.imread(os.path.join(i_frame_dir, frame_files[0]))
    height, width, layers = first_frame.shape

    # Define the codec and create VideoWriter object
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Codec for .mp4 files
    video_writer = cv2.VideoWriter(output_video_path, fourcc,
frame_rate, (width, height))

    # Write each frame to the video
    for frame_file in frame_files:
        frame_path = os.path.join(i_frame_dir, frame_file)
        frame = cv2.imread(frame_path)
        video_writer.write(frame)

    # Release the video writer
    video_writer.release()
    print(f"Video created successfully and saved as
{output_video_path}")
```

Output:

```
D:\Sem7\Image and video analytics\Lab\lab2>python lab6.py
Video created successfully and saved as reconstructed.mp4
```



Lab Task 6: Compare the quality of images using PSNR, SSIM, MSE

PSNR

Code:

```
from math import log10, sqrt
import cv2
import numpy as np

def PSNR(original, compressed):
  mse = np.mean((original - compressed) ** 2)
  if(mse == 0):
    return 100
  max_pixel = 255.0
  psnr = 20 * log10(max_pixel / sqrt(mse))
  return psnr

def main():
  original = cv2.imread("./frames/I_frame_0001.jpeg")
  compressed = cv2.imread("./frames/B_frame_0002.jpeg", 1)
  value = PSNR(original, compressed)
  print(f"PSNR value is {value} dB")

if __name__ == "__main__":
  main()
```

Output:

```
In [4]: runfile('D:/Sem7/Image and video analytics/Lab/lab2/task6_PSNR.py',
wdir='D:/Sem7/Image and video analytics/Lab/lab2')
PSNR value is 36.325366048378825 dB
```

SSIM

```
import torch
import torch.nn.functional as F
import numpy as np
import math
from PIL import Image
import cv2

def gaussian(window_size, sigma):
    """
    Generates a list of Tensor values drawn from a gaussian 
distribution with standard
    diviation = sigma and sum of all elements = 1.

    Length of list = window_size
    """
    gauss =  torch.Tensor([math.exp(-(x - 
window_size//2)**2/float(2*sigma**2)) for x in range(window_size)])
    return gauss/gauss.sum()


gauss_dis = gaussian(11, 1.5)
print("Distribution: ", gauss_dis)
print("Sum of Gauss Distribution:", torch.sum(gauss_dis))


def create_window(window_size, channel=1):

    # Generate an 1D tensor containing values sampled from a gaussian 
distribution
    _1d_window = gaussian(window_size=window_size, 
sigma=1.5).unsqueeze(1)

    # Converting to 2D
    _2d_window = 
_1d_window.mm(_1d_window.t()).float().unsqueeze(0).unsqueeze(0)

    window = torch.Tensor(_2d_window.expand(channel, 1, window_size, 
window_size).contiguous())

    return window


window = create_window(11, 3)
print("Shape of gaussian window:", window.shape)
```

```
def ssim(img1, img2, val_range, window_size=11, window=None,
size_average=True, full=False):

    L = val_range # L is the dynamic range of the pixel values (255 for
8-bit grayscale images),

    pad = window_size // 2

    try:
        _, channels, height, width = img1.size()
    except:
        channels, height, width = img1.size()

    # if window is not provided, init one
    if window is None:
        real_size = min(window_size, height, width) # window should be
atleast 11x11
        window = create_window(real_size,
channel=channels).to(img1.device)

    # calculating the mu parameter (locally) for both images using a
gaussian filter
    # calculates the luminosity params
    mu1 = F.conv2d(img1, window, padding=pad, groups=channels)
    mu2 = F.conv2d(img2, window, padding=pad, groups=channels)

    mu1_sq = mu1 ** 2
    mu2_sq = mu2 ** 2
    mu12 = mu1 * mu2

    # now we calculate the sigma square parameter
    # Sigma deals with the contrast component
    sigma1_sq = F.conv2d(img1 * img1, window, padding=pad,
groups=channels) - mu1_sq
    sigma2_sq = F.conv2d(img2 * img2, window, padding=pad,
groups=channels) - mu2_sq
    sigma12 =  F.conv2d(img1 * img2, window, padding=pad,
groups=channels) - mu12

    # Some constants for stability
    C1 = (0.01 ) ** 2  # NOTE: Removed L from here (ref PT
implementation)
    C2 = (0.03 ) ** 2

    contrast_metric = (2.0 * sigma12 + C2) / (sigma1_sq + sigma2_sq +
C2)
```

```
    contrast_metric = torch.mean(contrast_metric)

    numerator1 = 2 * mu12 + C1
    numerator2 = 2 * sigma12 + C2
    denominator1 = mu1_sq + mu2_sq + C1
    denominator2 = sigma1_sq + sigma2_sq + C2

    ssim_score = (numerator1 * numerator2) / (denominator1 *
denominator2)

    if size_average:
        ret = ssim_score.mean()
    else:
        ret = ssim_score.mean(1).mean(1).mean(1)

    if full:
        return ret, contrast_metric

    return ret



# helper function to load images
load_images = lambda x: np.asarray(Image.open(x).resize((480, 640)))

# Helper functions to convert to Tensors
tensorify = lambda x: torch.Tensor(x.transpose((2, 0,
1))).unsqueeze(0).float().div(255.0)

def display_imgs(x, transpose=True, resize=True):
    if resize:
        x = cv2.resize(x, (400, 400))
    if transpose:
        x = cv2.cvtColor(x, cv2.COLOR_BGR2RGB)
    cv2.imshow('Image', x)
    cv2.waitKey(0)
    cv2.destroyAllWindows()



# The true reference Image
img1 = load_images("./frames/I_frame_0001.jpeg")

# The False image
img2 = load_images("./frames/B_frame_0002.jpeg")

# The noised true image
noise = np.random.randint(0, 255, (640, 480, 3)).astype(np.float32)
```

```
noisy_img = img1 + noise

_img1 = tensorify(img1)
_img2 = tensorify(img2)
true_vs_false = ssim(_img1, _img2, val_range=255)
print("True vs False Image SSIM Score:", true_vs_false)

# Check SSIM score of True image vs Noised_true Image
_img1 = tensorify(img1)
_img2 = tensorify(noisy_img)
true_vs_false = ssim(_img1, _img2, val_range=255)
print("True vs Noisy True Image SSIM Score:", true_vs_false)


# Check SSIM score of True image vs True Image
_img1 = tensorify(img1)
true_vs_false = ssim(_img1, _img1, val_range=255)
print("True vs True Image SSIM Score:", true_vs_false)
```

Output:

```
D:\Sem7\Image and video analytics\Lab\lab2>python task7.py
Distribution:  tensor([0.0010, 0.0076, 0.0360, 0.1094, 0.2130, 0.2660, 0.2130, 0.1094, 0.0360,
        0.0076, 0.0010])
Sum of Gauss Distribution: tensor(1.)
Shape of gaussian window: torch.Size([3, 1, 11, 11])
D:\Sem7\Image and video analytics\Lab\lab2\task7.py:105: UserWarning: The given NumPy array is not writable, and PyTorch does not support non-writable tenso
rs. This means writing to this tensor will result in undefined behavior. You may want to copy the array to protect its data or make it writable before conve
rting it to a tensor. This type of warning will be suppressed for the rest of this program. (Triggered internally at ..\torch\csrc\utils\tensor_numpy.cpp:21
2.)
  tensorify = lambda x: torch.Tensor(x.transpose((2, 0, 1))).unsqueeze(0).float().div(255.0)
True vs False Image SSIM Score: tensor(0.9795)

D:\Sem7\Image and video analytics\Lab\lab2>python task7.py
Distribution:  tensor([0.0010, 0.0076, 0.0360, 0.1094, 0.2130, 0.2660, 0.2130, 0.1094, 0.0360,
        0.0076, 0.0010])
Sum of Gauss Distribution: tensor(1.)
Shape of gaussian window: torch.Size([3, 1, 11, 11])
D:\Sem7\Image and video analytics\Lab\lab2\task7.py:105: UserWarning: The given NumPy array is not writable, and PyTorch does not support non-writable tenso
rs. This means writing to this tensor will result in undefined behavior. You may want to copy the array to protect its data or make it writable before conve
rting it to a tensor. This type of warning will be suppressed for the rest of this program. (Triggered internally at ..\torch\csrc\utils\tensor_numpy.cpp:21
2.)
  tensorify = lambda x: torch.Tensor(x.transpose((2, 0, 1))).unsqueeze(0).float().div(255.0)
True vs False Image SSIM Score: tensor(0.9795)
True vs Noisy True Image SSIM Score: tensor(0.0573)
True vs True Image SSIM Score: tensor(1.)

D:\Sem7\Image and video analytics\Lab\lab2>python task6_SSIM.py
Distribution:  tensor([0.0010, 0.0076, 0.0360, 0.1094, 0.2130, 0.2660, 0.2130, 0.1094, 0.0360,
        0.0076, 0.0010])
Sum of Gauss Distribution: tensor(1.)
Shape of gaussian window: torch.Size([3, 1, 11, 11])
D:\Sem7\Image and video analytics\Lab\lab2\task6_SSIM.py:105: UserWarning: The given NumPy array is not writable, and PyTorch does not support non-writable
tensors. This means writing to this tensor will result in undefined behavior. You may want to copy the array to protect its data or make it writable before
converting it to a tensor. This type of warning will be suppressed for the rest of this program. (Triggered internally at ..\torch\csrc\utils\tensor_numpy.c
pp:212.)
  tensorify = lambda x: torch.Tensor(x.transpose((2, 0, 1))).unsqueeze(0).float().div(255.0)
True vs False Image SSIM Score: tensor(0.9795)
True vs Noisy True Image SSIM Score: tensor(0.0572)
True vs True Image SSIM Score: tensor(1.)
```

MSE:

```
import numpy as np
from PIL import Image
import cv2
def mse(imageA, imageB):
    # Ensure the images are numpy arrays
    imageA = np.array(imageA)
    imageB = np.array(imageB)

    # Check if the images have the same shape
    if imageA.shape != imageB.shape:
        raise ValueError("Input images must have the same dimensions.")

    # Compute the MSE
    err = np.sum((imageA.astype("float") - imageB.astype("float")) **
2)
    err /= float(imageA.shape[0] * imageA.shape[1] * imageA.shape[2])

    return err


# Load images using the provided helper functions or directly with cv2
or PIL
img1 = cv2.imread("./frames/I_frame_0381.jpeg")
img2 = cv2.imread("./frames/B_frame_0002.jpeg")

# Calculate MSE
mse_value = mse(img1, img2)
print(f"MSE between the images: {mse_value}")
```

Output:

```
In [5]: runfile('D:/Sem7/Image and video analytics/Lab/lab2/task6_MSE.py',
wdir='D:/Sem7/Image and video analytics/Lab/lab2')
MSE between the images: 2022.4889285542054
```