機器視覺作業報告

Homework #1

姓名: 林麗萍

學號: 01157157

日期: 2025 年 4 月 5 日

連接: https://youtu.be/ZO-kh2Y9uDU

1. Gaussian by Pytorch

方法

- 想法，演算步驟。

The main idea is to apply Gaussian Blur to an entire video using PyTorch instead of OpenCV or other conventional methods. This gives you more control and allows you to integrate PyTorch into your video processing pipeline.

Steps:

1. Load the input video.
2. Convert each frame to grayscale.
3. Normalize the pixel values and convert them into a 4D PyTorch tensor.
4. Define a 5x5 Gaussian kernel and normalize it.
5. Use torch.nn.functional.conv2d to apply the kernel to each frame.
6. Convert the output back to a numpy array and then to BGR color format.
7. Rebuild the video from the processed frames using ImageSequenceClip.
8. Add optional subtitle text and corner labels using moviepy.

- 重要程式片段說明。

1. Gaussian using PyTorch

```
def gaussian_blur_torch(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    gray = gray.astype(np.float32) / 255.0
```

2. Convert to a 4D tensor for convolution

```
tensor = torch.tensor(gray[None, None, :, :], dtype=torch.float32)

kernel = torch.tensor([[1, 4, 6, 4, 1],
                       [4, 16, 24, 16, 4],
                       [6, 24, 36, 24, 6],
                       [4, 16, 24, 16, 4],
                       [1, 4, 6, 4, 1]], dtype=torch.float32)
kernel = kernel / kernel.sum()  # Normalize the kernel
kernel = kernel[None, None, :, :]
```

3. Perform 2D convolution

```
blurred = F.conv2d(tensor, kernel, padding=2)
```

4. Process video frames

```
def process_video(input_path, operation):
    clip = VideoFileClip(input_path).without_audio()
    frames = []

    for frame in clip.iter_frames(fps=24, dtype='uint8'):
        frame_bgr = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
        processed = operation(frame_bgr)
        processed_rgb = cv2.cvtColor(processed, cv2.COLOR_BGR2RGB)
        frames.append(processed_rgb)

    return ImageSequenceClip(frames, fps=24)
```

2. HSV color space by PyTorch

方法

- 想法，演算步驟。

The main idea is to apply a color space transformation to a video by converting each frame from BGR (used by OpenCV) to HSV format. HSV (Hue, Saturation, Value) is often more intuitive for color-based operations and segmentation in computer vision.

Steps:

1. Load the original video using MoviePy without audio.
2. Extract each frame and convert it from RGB to BGR (since OpenCV uses BGR by default).
3. Use OpenCV's cvtColor function to convert the BGR frame to HSV.
4. Convert the HSV frame back to RGB so it can be used by MoviePy.
5. Store all processed frames into a list.
6. Reconstruct the video from the list of frames using ImageSequenceClip.
7. Add a text label ("HSV color space by PyTorch") at the top-left corner using TextClip.
8. Merge the text and processed video into a final composite video.
9. Export the final video using MoviePy with the filename 01157157-HSV.mp4.

- 重要程式片段說明。

1. Convert to HSV color space using PyTorch

```python
def convert_to_hsv_torch(frame):
    # Convert the frame from BGR to HSV using OpenCV
    hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    return hsv_frame
```

2. Process video frame

```python
def process_video(input_path, operation):
    clip = VideoFileClip(input_path).without_audio()
    frames = []

    for frame in clip.iter_frames(fps=24, dtype='uint8'):
        frame_bgr = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
        processed = operation(frame_bgr)
        processed_rgb = cv2.cvtColor(processed, cv2.COLOR_BGR2RGB)
        frames.append(processed_rgb)

    return ImageSequenceClip(frames, fps=24)
```

3. Histogram Equalization to the V

方法

- 想法，演算步驟。

The goal is to enhance the brightness of a video by applying histogram equalization to the V (Value) channel in HSV color space. This improves contrast without affecting hue or saturation, making the image appear clearer and more balanced in lighting.

Steps:

1. Load the original video using MoviePy and remove audio.
2. Convert each frame from RGB to BGR for compatibility with OpenCV.
3. Convert the BGR frame to HSV color space.
4. Split the HSV image into **H**, **S**, and **V** channels.
5. Apply cv2.equalizeHist() to the **V** channel to perform histogram equalization.
6. Merge the equalized **V** channel with the original **H** and **S** channels.

7. Convert the modified HSV image back to BGR, then to RGB.
8. Store each processed frame and recreate a new video using ImageSequenceClip.
9. Add a label “Histogram Equalization of V by PyTorch” at the top-left using TextClip.
10. Export the final video as 01157157-Histogram.mp4.

- 重要程式片段說明。

1. Convert the frame from BGR to HSV using OpenCV

```
hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

2. Split the HSV image into H, S, V channels

```
h, s, v = cv2.split(hsv_frame)
```

3. Apply histogram equalization to the V channel

```
v_equalized = cv2.equalizeHist(v)
```

4. Merge the equalized V channel back with the original H and S channel

```
hsv_equalized = cv2.merge([h, s, v_equalized])
```

5. Convert the result back to BGR color space for display

```
bgr_equalized = cv2.cvtColor(hsv_equalized, cv2.COLOR_HSV2BGR)
```

6. Process video frame

```
def process_video(input_path, operation):
    clip = VideoFileClip(input_path).without_audio()
    frames = []

    for frame in clip.iter_frames(fps=24, dtype='uint8'):
        frame_bgr = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
        processed = operation(frame_bgr)
        processed_rgb = cv2.cvtColor(processed, cv2.COLOR_BGR2RGB)
        frames.append(processed_rgb)

    return ImageSequenceClip(frames, fps=24)
```

4. Gray Level Mapping of V

方法

- 想法，演算步驟。

The purpose of this video processing task is to enhance the brightness

details in dark regions by applying a logarithmic transformation to the V (Value) channel of each video frame in HSV color space. This nonlinear operation increases the visibility of darker pixels without overexposing brighter areas, simulating a kind of dynamic range compression.
Steps:

1. Load the input video using MoviePy and strip away the audio.
2. For every frame:
    a. Convert the RGB frame to BGR for OpenCV compatibility.
    b. Convert the frame from BGR to HSV color space.
    c. Split into **H**, **S**, and **V** channels.
    d. Apply the transformation V = c * log(1 + V) to the **V** channel.
    e. Clip values to ensure they remain in the valid range (0–255) and convert to uint8.
    f. Merge the transformed **V** channel with the original **H** and **S** channels.
    g. Convert the result back to BGR, then to RGB format.
3. Reassemble the video from the processed frames using ImageSequenceClip.
4. Add a label at the top-left corner: “Log Gray Level Mapping of V”.
5. Save the final video as 01157157-LogGray.mp4.

- 重要程式片段說明。
    1. Convert the frame from BGR to HSV using OpenCV

```
hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

    2. Split the HSV image into H, S, V channels

```
h, s, v = cv2.split(hsv_frame)
```

    3. Apply log transformation to the V channel

```
v_log = c * np.log(1 + v)
v_log = np.clip(v_log, 0, 255).astype(np.uint8)
```

    4. Merge the transformed V channel back with the original H and S

```
hsv_log = cv2.merge([h, s, v_log])
```

5. Convert the result back to BGR color space for display

```
bgr_log = cv2.cvtColor(hsv_log, cv2.COLOR_HSV2BGR)
```

6. Process video frames

```
def process_video(input_path, operation):
    clip = VideoFileClip(input_path).without_audio()
    frames = []

    for frame in clip.iter_frames(fps=24, dtype='uint8'):
        frame_bgr = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
        processed = operation(frame_bgr)
        processed_rgb = cv2.cvtColor(processed, cv2.COLOR_BGR2RGB)
        frames.append(processed_rgb)
```

結果結論

Throughout these four video processing tasks, I successfully applied different image enhancement techniques using OpenCV, PyTorch, and MoviePy. In the Gaussian Blur task, I implemented a custom convolutional kernel using PyTorch to soften the entire frame, which helped reduce noise and detail, a fundamental concept in preprocessing for computer vision.

For the HSV Conversion, I transformed each video frame from BGR to HSV color space. This exercise deepened my understanding of how different color models can reveal or isolate visual features more effectively, especially the separation of luminance (Value) from color information (Hue, Saturation).

In the Histogram Equalization task, I focused on enhancing contrast by modifying the V channel in the HSV space. This made the overall video more vibrant and visually clearer, especially in scenes with poor lighting. I also learned how splitting and merging color channels gives precise control over specific aspects of an image.

Lastly, in the Log Gray Level Mapping, I applied a logarithmic transformation to brighten darker pixels without overly enhancing brighter regions. This taught me how to use non-linear mappings for dynamic range compression, which is often used in medical imaging and shadow enhancement.

Overall, these exercises not only improved my skills in OpenCV and PyTorch but also gave me a practical understanding of fundamental image processing techniques from basic filtering to color space transformations and non-linear adjustments. I also practiced working with video processing pipelines using MoviePy and learned to overlay descriptive text for documentation purposes.

參考文獻

https://pytorch.org/tutorials/beginner/pytorch_with_examples.html
https://pythongeeks.org/learn-opencv-tutorial/
http://222.178.203.72:19005/whst/63/=vvvzfddjrenqfddjrznqf//image-processing-in-python/
https://www.sciencedirect.com/science/article/abs/pii/S0045790615002827
https://medium.com/@cindylin_1410/%E6%B7%BA%E8%AB%87-opencv-%E7%9B%B4%E6%96%B9%E5%9C%96%E5%9D%87%E8%A1%A1%E5%8C%96-ahe%E5%9D%87%E8%A1%A1-clahe%E5%9D%87%E8%A1%A1-ebc9c14a8f96
https://www.tutorialspoint.com/dip/gray_level_transformations.htm