2025

DIGITAL IMAGE PROCESSING

ASSIGNMENT 0

2023102070

INDIRA C REDDY
IIIT HYDERABAD

TABLE OF CONTENTS

INTRODUCTION

This is a simple, first-pass tour of digital image processing using just **NumPy for math** and **Pillow/ImageIO for input/output**. No fancy libraries, just pixels and a few classic tricks.
The report is split into nine tiny tasks (Q1–Q9): read/write images, tweak brightness and contrast, convert RGB→grayscale (three ways), add a basic pseudocolor look, do a green-screen swap, flip between frames and videos, and build a clean fade transition. Everything stays in array form (H×W or H×W×3) so it's easy to reason about what each line of code does. Goal was keep it clear, reproducible, and beginner-friendly. Each question shows the approach in one or two lines of math, the exact assumption (like value ranges or modes), and a quick observation of how the image changed.

GitHub Repo link

Q1

READING AN IMAGE INTO ARRAY

OBJECTIVE:

Read an image file and convert it into a NumPy array for processing.

APPROACH:

Use PIL.Image.open() to load file and convert to a NumPy array (uint8).
Supports grayscale (H×W) and RGB (H×W×3).
Print shape, type, and sample pixel values.

CODE SNIPPETS:

q1.py

Ubuntu-22.04 > home > indira > sem5_DIP_assignments > assign0 > questions > q1.py > ...

```python
from utils.image_utils import read_image, save_pixel_array_to_txt

def run_q1():
    path = "input_images/cat.jpeg"
    img_array = read_image(path)

    print("Image shape:", img_array.shape)
    save_pixel_array_to_txt(img_array, "output_images/q1_output_pixels.txt")
```

```python
#to read an image and return it as a numpy array.
#this supports grayscale and rgb too.
#q1
def read_image(path):
    img = Image.open(path)
    img_array = np.array(img)
    return img_array
```

```python
#to save the pixel values of an image array in a .txt fi
def save_pixel_array_to_txt(array, output_path):
    with open(output_path, 'w') as f:
        if len(array.shape) == 2:
            #grayscale
            for row in array:
                f.write(' '.join(str(val) for val in row) + '\n')
        elif len(array.shape) == 3:
            #color
            for row in array:
                row_str = ' '.join(' '.join(str(channel) for channel in pixel) for pixel in row)
                f.write(row_str + '\n')
        else:
            raise ValueError("Unsupported array shape")
```

OUTPUT:

```
indira@IndiraCReddy:~/sem5_DIP_assignments/assign0$ python3 main.py
Image shape: (261, 193, 3)
```

Input and output images in next page.

OBSERVATIONS:

Image is stored as an array with dimensions (height, width, channels).
Image data is now matrix-form, easy to index and process.
The output array shown below is for a color image.
For a grayscale image each pixel is denoted with only one value while for a color iamge each pixel is shown with 3 nos which representing R,G and B

INPUT IMAGE USED

SAMPLE DATA RECIVED

186 178 167 186 178 167 186 178 167 187 179 168 187 179 168 188 180 169 188 180 169 188 180 169 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168
187 178 169 187 178 169 187 178 169 187 178 169 187 178 169 187 178 169 187 178 169 188 179 170 188 179 170 188 179 170 188 179 170 188 179 170 188 179 170 188 179 170 188 179 170
187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168
186 178 167 186 178 167 186 178 167 187 179 168 187 179 168 187 179 168 188 180 169 188 180 169 188 180 169 188 180 169 187 179 168 187 179 168 187 179 168 186 178 167 186 178 167
183 175 164 183 175 164 182 174 163 182 174 163 181 173 162 180 172 161 179 171 160 179 171 160 180 172 161 180 172 161 180 172 161 180 172 161 180 172 161 180 172 161 180 172 161
181 171 161 181 171 161 181 171 161 181 171 161 181 171 161 181 171 161 181 171 161 181 171 161 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162
183 173 163 183 173 163 183 173 163 183 173 163 183 173 163 183 173 163 183 173 163 183 173 163 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162
182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 183 173 163 183 173 163 183 173 163 183 173 163 183 173 163 183 173 163 183 173 163
184 176 165 184 176 165 184 176 165 184 176 165 185 177 166 185 177 166 185 177 166 186 178 167 187 179 168 187 179 168 187 179 168 188 180 169 188 180 169 189 181 170 189 181 170
189 180 171 189 180 171 190 181 172 191 182 173 192 183 174 192 183 174 192 185 175 192 185 175 192 188 177 192 188 177 193 189 178 193 189 178 193 191 179 194 192 180 195 193 181
199 194 188 199 194 188 199 194 188 200 195 189 200 195 189 200 195 189 201 196 190 200 195 189 200 195 189 201 196 190 201 196 190 202 197 191 203 198 192 204 199 193 203 200 193
205 201 198 203 202 198 204 203 199 205 204 200 208 205 200 209 206 201 210 207 202 211 208 203 212 207 201 212 207 201 213 209 200 213 209 200 215 208 200 216 209 201 216 209 201
216 207 198
185 177 166 186 178 167 186 178 167 186 178 167 187 179 168 187 179 168 187 179 168 188 180 169 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168
187 178 169 187 178 169 187 178 169 187 178 169 187 178 169 187 178 169 187 178 169 188 179 170 188 179 170 188 179 170 188 179 170 188 179 170 188 179 170 188 179 170 188 179 170
187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168
186 178 167 186 178 167 186 178 167 186 178 167 187 179 168 187 179 168 188 180 169 188 180 169 188 180 169 188 180 169 187 179 168 187 179 168 186 178 167 186 178 167 186 178 167
183 175 164 183 175 164 182 174 163 182 174 163 181 173 162 180 172 161 179 171 160 179 171 160 180 172 161 180 172 161 180 172 161 180 172 161 180 172 161 180 172 161 180 172 161
181 171 161 181 171 161 181 171 161 181 171 161 181 171 161 181 171 161 181 171 161 181 171 161 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162
183 173 163 183 173 163 183 173 163 183 173 163 183 173 163 183 173 163 183 173 163 183 173 163 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162
182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 183 173 163 183 173 163 183 173 163 183 173 163 183 173 163 183 173 163 183 173 163
183 175 164 184 176 165 184 176 165 184 176 165 185 177 166 185 177 166 185 177 166 187 179 168 187 179 168 187 179 168 188 180 169 188 180 169 188 180 169 189 181 170 189 181 170
189 180 171 189 180 171 190 181 172 191 182 173 192 183 174 192 183 174 192 185 175 192 185 175 192 188 177 192 188 177 193 189 178 193 189 178 193 191 179 194 192 180 195 193 181
198 193 187 199 194 188 199 194 188 200 195 189 200 195 189 200 195 189 200 195 189 200 195 189 200 195 189 200 195 189 201 196 190 202 197 191 203 198 192 203 198 192 203 200 193
204 200 197 203 202 198 204 203 199 205 204 200 208 205 200 209 206 201 210 207 202 210 207 202 212 207 201 212 207 201 212 208 199 213 209 200 215 208 200 215 208 200 216 209 201 216 209 201
216 207 198
184 176 165 185 177 166 185 177 166 185 177 166 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167
186 177 168 186 177 168 186 177 168 186 177 168 186 177 168 186 177 168 186 177 168 187 178 169 187 178 169 187 178 169 187 178 169 187 178 169 187 178 169 187 178 169 187 178 169
186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167 186 178 167
185 177 166 185 177 166 186 178 167 186 178 167 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168 187 179 168
183 175 164 183 175 164 182 174 163 182 174 163 181 173 162 180 172 161 179 171 160 179 171 160 180 172 161 180 172 161 180 172 161 180 172 161 180 172 161 180 172 161 180 172 161
181 171 161 181 171 161 181 171 161 181 171 161 181 171 161 181 171 161 181 171 161 181 171 161 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162 182 172 162

Q2

READING AN IMAGE INTO ARRAY

OBJECTIVE:
Convert a NumPy array back into an image and saving it

APPROACH:
Determine if array is grayscale or RGB based on its shape.
Use Pillow to create an image object from numpy array.
Save the output image to a specified output path.

CODE SNIPPETS:

```python
# to write a numpy array into an image file
#again, supports grayscale(2D) and RGB color(3D) images too.
#q2
def write_image(array, output_path):
    if len(array.shape) == 2:
        mode = 'L'  #grayscale
    elif len(array.shape) == 3 and array.shape[2] == 3:
        mode = 'RGB'
    else:
        raise ValueError("Unsupported image array shape")

    img = Image.fromarray(np.uint8(a
    img.save(output_path)
```

main.py | image_utils.py 3 | q2.py 1

Ubuntu-22.04 > home > indira > sem5_DIP_assignments > assign0 > questions > q2.py > ...

```python
from utils.image_utils import read_image, write_image

def run_q2():
    input_path = "input_images/cat.jpeg"
    output_path = "output_images/q2_written_image.jpg"

    img_array = read_image(input_path)
    write_image(img_array, output_path)

    print("Image successfully written to", output_path)
```

OUTPUT:

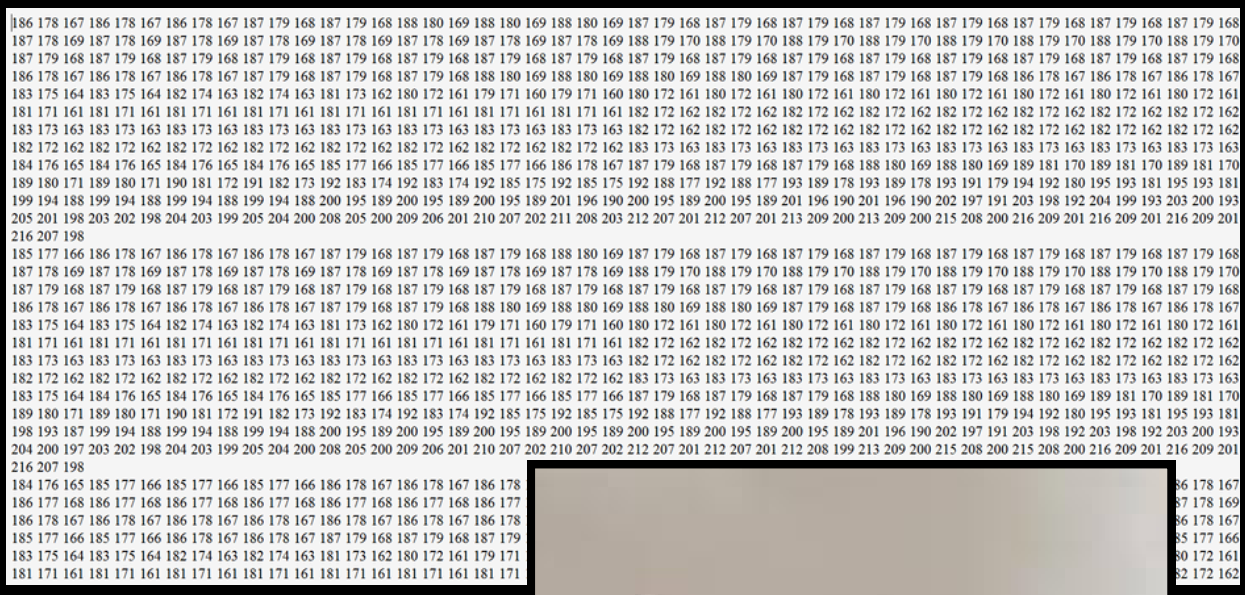
```
indira@IndiraCReddy:~/sem5_DIP_assignments/assign0$ python3 main.py
Image successfully written to output_images/q2_written_image.jpg
```

input and output images in next page.

OBSERVATIONS:
Values outside [0,255] would clip after casting. Output matches input layout (grayscale vs color).

INPUT IMAGE USED

CONVERTED TO ARRAY



OUTPUT - WRITTEN IMAGE

Q3

BRIGHTNESS ADJUSTMENT

OBJECTIVE:
Increase or decrease image brightness

APPROACH:
Load image into NumPy array.
For brightness: add a constant to all the pixel values
Clip values to [0, 255] before saving.
If ‘value’ added is positive we get a brighter image, if negative then we get a darker image.
Used int16 to avoid overflow and finalized as unit8

CODE SNIPPETS:

```python
from utils.image_utils import read_image, write_image, change_brightness

def run_q3():
    input_path = "input_images/cat.jpeg"
    output_brighter = "output_images/q3_brighter.jpg"
    output_darker = "output_images/q3_darker.jpg"

    img = read_image(input_path)

    brighter = change_brightness(img, 50)  #increase brightness
    darker = change_brightness(img, -50)   #decrease brightness

    write_image(brighter, output_brighter)
    write_image(darker, output_darker)

    print("Saved bright and dark versions!")

```

```python
#to adjust brightness of image
#'value' if positive, we get a brighter imager.
#if the 'value' is negative we get darker image
#q3
def change_brightness(img_array, value):
    #casting to int16 to avoid overflow/underflow cases
    img_int = img_array.astype(np.int16)
    #adding value, then clip to valid range
    brightened = np.clip(img_int + value, 0, 255)
    return brightened.astype(np.uint8)
```

OUTPUT:

```
indira@IndiraCReddy:~/sem5_DIP_assignments/assign0$ python3 main.py
Saved bright and dark versions!
```

Input and output images in next page.

OBSERVATIONS:
Adding a value or substracting a value shifts intensity uniformly.
Large positive/negative shifts can cause highlight/shadow clipping.

INPUT IMAGE USED

BRIGHTER IMAGE

DARKED IMAGE

Q4
CONTRAST ADJUSTMENT

OBJECTIVE:
Modify the contrast of an image.

APPROACH:
Load image into NumPy array.
Apply scaling relative to the mean pixel value (i.e; multiplying around a midpoint
Clip values to [0, 255] before saving.
Used float32 to avoid overflow and finalized as unit8

CODE SNIPPETS:

```python
from utils.image_utils import read_image, write_image, change_contrast

def run_q4():
    input_path = "input_images/cat.jpeg"
    output_high = "output_images/q4_high_contrast.jpg"
    output_low = "output_images/q4_low_contrast.jpg"

    img = read_image(input_path)

    high_contrast = change_contrast(img, 1.5)  # more punch
    low_contrast = change_contrast(img, 0.5)   # flatter

    write_image(high_contrast, output_high)
    write_image(low_contrast, output_low)

    print("Saved high and low contrast images!")
```

```python
#adjusting the contrast of image
#multiplying around a midpoint
#if faxtor>1 then more contrast, if 0<factor<1 then less contrast
#q4
def change_contrast(img_array, factor):
    img_float = img_array.astype(np.float32)
    contrasted = (img_float - 128) * factor + 128
    return np.clip(contrasted, 0, 255).astype(np.uint8)
```

OUTPUT:

```
indira@IndiraCReddy:~/sem5_DIP_assignments/assign0$ python3 main.py
Saved high and low contrast images!
```

Input and output images in next page.

OBSERVATIONS:
Increasing factor >1 boosts contrast, <1 reduces it.
Higher contrast expands differences but can clip extremes; lower contrast compresses tones for a flatter look.

INPUT IMAGE USED

HIGH CONTRAST IMAGE

LOW CONTRAST IMAGE

Q5

GRAYSCALE CONVERSIONS

OBJECTIVE:
Convert a color image to grayscale using different methods

APPROACH:
Average method: (R + G + B) / 3
Luminosity method: weighted sum (0.299R + 0.587G + 0.114B)
Lightness method: (max(R, G, B) + min(R, G, B)) / 2

CODE SNIPPETS:

```python
#q5
def rgb_to_grayscale(img_array, method='luminosity'):
    if len(img_array.shape) != 3 or img_array.shape[2] != 3:
        raise ValueError("Input must be a color (RGB) image")

    img = img_array.astype(np.float32)
    R, G, B = img[:,:,0], img[:,:,1], img[:,:,2]

    if method == 'average':
        gray = (R + G + B) / 3
    elif method == 'luminosity':
        gray = 0.299 * R + 0.587 * G + 0.114 * B
    elif method == 'lightness':
        gray = (np.max(img, axis=2) + np.min(img, axis=2)) / 2
    else:
        raise ValueError("Unknown grayscale method")

    return gray.astype(np.uint8)
```

```python
from utils.image_utils import read_image, write_image, rgb_to_grayscale

def run_q5():
    input_path = "input_images/cat.jpeg"

    avg = rgb_to_grayscale(read_image(input_path), method='average')
    lum = rgb_to_grayscale(read_image(input_path), method='luminosity')
    light = rgb_to_grayscale(read_image(input_path), method='lightness')

    write_image(avg, "output_images/q5_grayscale_average.jpg")
    write_image(lum, "output_images/q5_grayscale_luminosity.jpg")
    write_image(light, "output_images/q5_grayscale_lightness.jpg")

    print("Saved all grayscale versions!")
```

OUTPUT:

```
indira@IndiraCReddy:~/sem5_DIP_assignments/assign0$ python3 main.py
Saved all grayscale versions!
```

Input and output images in next page.

OBSERVATIONS:
Average: Equal weight to all channels. Simple but flat-looking.
Luminosity: Weighted for human perception (green-heavy). Looks more balanced and realistic.gives better perceptual accuracy than average.
Lightness: Based on max and min. Gives a contrasty, stylized feel.

INPUT IMAGE USED

AVERAGE GRAYSCALE

LUMINOSITY GRAYSCALE

LIGHTNESS GRAYSCALE

Q6

PSEUDOCOLOR MAPPING

OBJECTIVE:

Apply false coloring to grayscale images for better visual interpretation.

APPROACH:

Covert original image to grayscale images

Function maps grayscale intensities to colors using a two-part scheme

- Values 0-127 are transformed into a spectrum from blue to red.
- Values 128-255 are transformed into a spectrum from red to yellow.

CODE SNIPPETS:

```python
from utils.image_utils import read_image, write_image, rgb_to_grayscale, grayscale_to_pseudo

def run_q6():
    input_path = "input_images/cat.jpeg"

    gray = rgb_to_grayscale(read_image(input_path))  # default = luminosity
    pseudo = grayscale_to_pseudocolor(gray)

    write_image(pseudo, "output_images/q6_pseudocolor.j
    print("Pseudocolor image saved!")
```

```python
#converts a grayscale image to a pseudocolor using a basic colormap
#output is hopefully an RGB image
#q6
def grayscale_to_pseudocolor(gray_img):
    if len(gray_img.shape) != 2:
        raise ValueError("Input must be a grayscale image")

    height, width = gray_img.shape
    pseudocolor = np.zeros((height, width, 3), dtype=np.uint8)

    for i in range(height):
        for j in range(width):
            val = gray_img[i, j]
            if val < 128:
                #map 0 - 127: Blue to Red
                ratio = val / 128
                r = int(255 * ratio)
                g = 0
                b = int(255 * (1 - ratio))
            else:
                #map 128-255: Red to Yellow
                ratio = (val - 128) / 127
                r = 255
                g = int(255 * ratio)
                b = 0
            pseudocolor[i, j] = [r, g, b]

    return pseudocolor
```

OUTPUT:

```
indira@IndiraCReddy:~/sem5_DIP_assignments/assign0$ python3 main.py
Pseudocolor image saved!
```

Input and output images in next page.

OBSERVATIONS:

Got a heatmap kind of look as an output image.

Pseudocolor can highlight intensity variations better than plain grayscale.

Color mapping makes small intensity differences more visible than plain gray; shows smooth two-stage color ramp.

INPUT IMAGE USED

PSEUDOCOLOR IMAGE

Q7

GREEN SCREEN REPLACEMENT

OBJECTIVE:
Replace green background in a foreground image with another background

APPROACH:
Create a green mask with (g > r*threshold) & (g > b*threshold). Resize background to foreground size, copy BG wherever mask is true.
Ensure both images are same sized.

CODE SNIPPETS:

```
replacing green pixels in foreground with background pixels
q7
ef replace_green_screen(fg_img, bg_img, threshold=1.3):
    #removing alpha channel if present
    #alpha channel
    if fg_img.shape[2] == 4:
        fg_img = fg_img[:, :, :3]
    if bg_img.shape[2] == 4:
        bg_img = bg_img[:, :, :3]

    if fg_img.shape != bg_img.shape:
        bg_img = Image.fromarray(bg_img)
        bg_img = bg_img.resize((fg_img.shape[1], fg_img.shape[0]))  #resizing to match (W, H)
        bg_img = np.array(bg_img)

        #print(fg_img.shape)
        #print(bg_img.shape)
        #raise ValueError("Foreground and background must be same shape")

    result = fg_img.copy()

    #extracting RGB channels
    r, g, b = fg_img[:,:,0], fg_img[:,:,1], fg_img[:,:,2]

    #creating a green mask
    green_mask = (g > r * threshold) & (g > b * threshold)

    #appling mask - replace green pixels with background
    result[green_mask] = bg_img[green_mask]

    return result
```

```
from utils.image_utils import read_image, write_image, replace_green_screen

def run_q7():
    fg_path = "input_images/foreground.png"
    bg_path = "input_images/background.jpg"

    fg = read_image(fg_path)
    bg = read_image(bg_path)

    output = replace_green_screen(fg, bg)
    write_image(output, "output_images/q7_minions_in_the_sky.jpg")

    print("Green screen replaced and saved!")
```

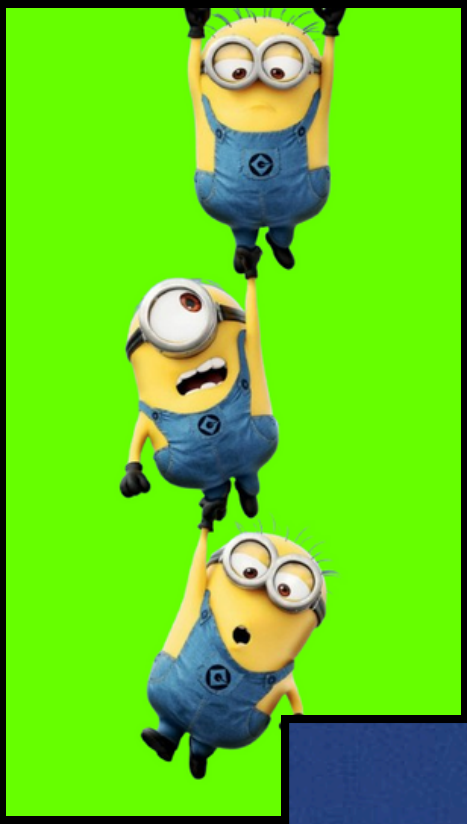
OUTPUT:

```
indira@IndiraCReddy:~/sem5_DIP_assignments/assign0$ python3 main.py
Green screen replaced and saved!
```

Input and output images in next page.

OBSERVATIONS:
Successfully replaced the green pixels with background pixels
Works well on clean green areas. May pick up green-ish clothes/objects; fine-tune threshold based on scene.

GREEN SCREEN IMAGE



BACKGROUND IMAGE

RESULT - MINIONS IN THE SKY

Q8

VIDEO I/O

OBJECTIVE:
Extract frames from video and reconstruct back.

APPROACH:
Read video using imageio
Save each image frame as .jpg
Recombine into .mp4 file using imageio.mimsave

CODE SNIPPETS:

```python
from utils.image_utils import extract_frames_from_video, frames_to_video

def run_q8():
    video_path = "videos/sample_video.mp4"
    frame_folder = "output_images/q8_frames"
    output_video = "videos/q8_output_video.mp4"

    num_frames = extract_frames_from_video(video_path, frame_folder)
    print(f"Extracted {num_frames} frames.")

    frames_to_video(frame_folder, output_video)
    print(f"Reconstructed video saved to {output_video}")
```

```python
#q8
#reading a video file and save each frame as an image into an output_fold
def extract_frames_from_video(video_path, output_folder):
    reader = imageio.get_reader(video_path)
    os.makedirs(output_folder, exist_ok=True)

    for i, frame in enumerate(reader):
        frame_path = os.path.join(output_folder, f"frame_{i:03d}.jpg")
        imageio.imwrite(frame_path, frame)

    return i + 1  #total number of frames
#reading all frames in input folder and then writing them into a video
#assuming all the frames are names in order like frame_000.jpg,etc.
def frames_to_video(input_folder, output_video_path, fps=24):
    writer = imageio.get_writer(output_video_path, fps=fps)

    frame_files = sorted(
        [f for f in os.listdir(input_folder) if f.endswith('.jpg')]
    )

    for filename in frame_files:
        frame = imageio.imread(os.path.join(input_folder, filename))
        writer.append_data(frame)

    writer.close()
```

OUTPUT:

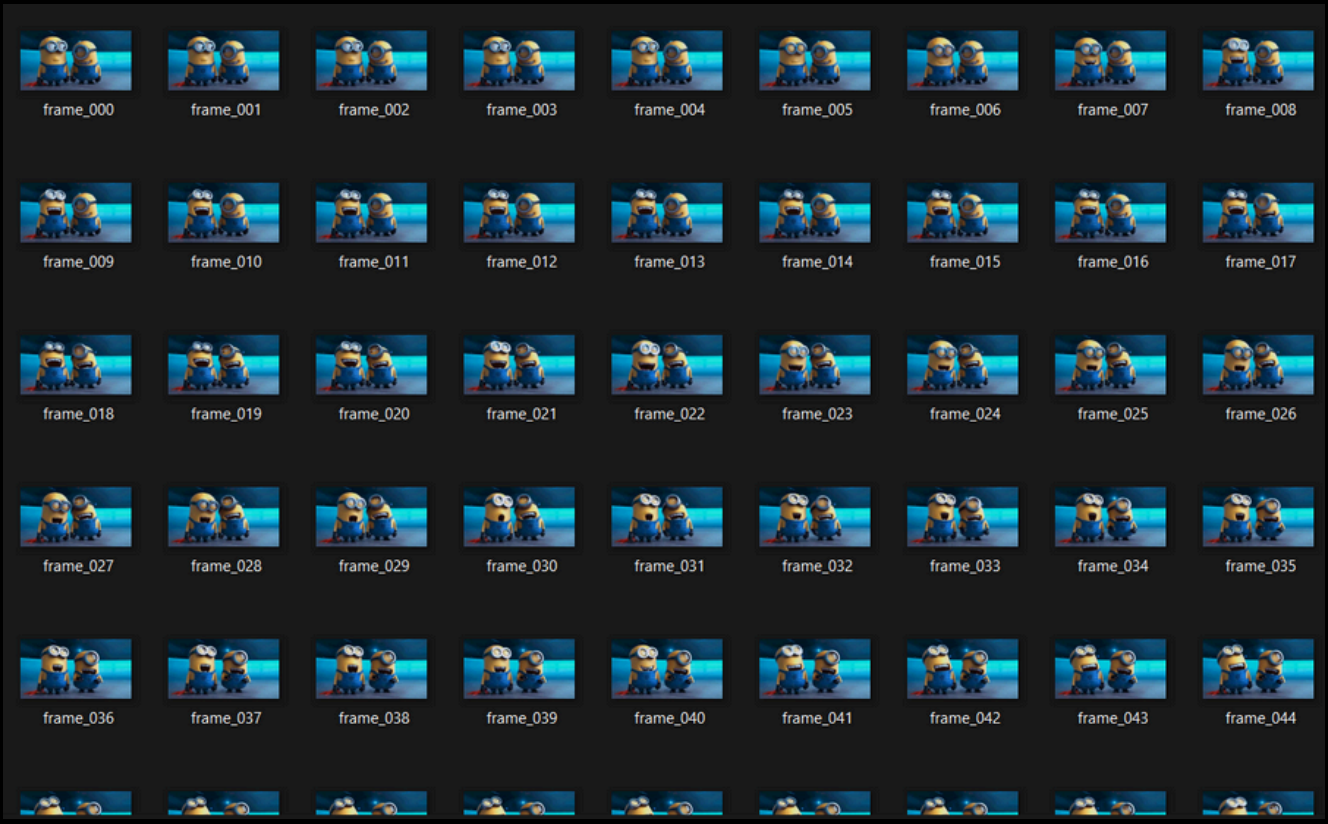
```
indira@IndiraCReddy:~/sem5_DIP_assignments/assign0$ python3 main.py
Extracted 68 frames.
```

Input video link and output image & video links in next page.

OBSERVATIONS:
Video was segmented into 68 frames and successful in reconstructing the video from said frames.
Frames are assumed sequentially named and same size; FPS controls output duration.
Must ensure frame dimensions are divisible by 16 for best codec compatibility.

INPUT VIDEO LINK

SAMPLE FRAMES OBTAINED



OUTPUT VIDEO LINK

Q9

IMAGE TRANSITION VIDEO

OBJECTIVE:

Create a fade transition between two images

APPROACH:

Load two images of same size (resize second image to first if needed)
Generate num_frames blends with alpha = i/(num_frames-1) and blend = (1-α)*img1 + α*img2.
Blend them progressively over those frames
Save as .mp4 file using imageio.

CODE SNIPPETS:

```python
#q9
#creates a list of frames fading from img1 to img2
def create_fade_transition(img1, img2, num_frames=24):
    if img1.shape != img2.shape:
        img2 = Image.fromarray(img2)
        img2 = img2.resize((img1.shape[1], img1.shape[0]))
        img2 = np.array(img2)
    #if img1.shape != img2.shape:
    #    raise ValueError("Images must be the same shape for transition")

    img1 = img1.astype(np.float32)
    img2 = img2.astype(np.float32)
    frames = []

    for i in range(num_frames):
        alpha = i / (num_frames - 1)
        blended = (1 - alpha) * img1 + alpha * img2
        frames.append(blended.astype(np.uint8))

    return frames
```

```python
from utils.image_utils import read_image, create_fade_transition
import imageio
import numpy as np
from PIL import Image

def run_q9():
    img1 = read_image("input_images/transition_start.jpg")
    img2 = read_image("input_images/transition_end.jpg")

    # Resize img2 to match img1 if needed
    if img1.shape != img2.shape:

        img2 = Image.fromarray(img2).resize((img1.shape[1], img1.shape[0]))
        img2 = np.array(img2)

    frames = create_fade_transition(img1, img2, num_frames=72) #3 sec at 24fps
    output_path = "videos/q9_transition_video.mp4"
    writer = imageio.get_writer(output_path, fps=24)

    for frame in frames:
        writer.append_data(frame)
    writer.close()

    print(f"Transition video saved to {output_path}")
```

OUTPUT:

```
indira@IndiraCReddy:~/sem5_DIP_assignments/assign0$ python3 main.py
IMAGEIO FFMPEG_WRITER WARNING: input image is not divisible by macro_block_size=16, resizing from (613, 407) t
o (624, 416) to ensure video compatibility with most codecs and players. To prevent resizing, make your input
image divisible by the macro_block_size or set the macro_block_size to 1 (risking incompatibility).
Transition video saved to videos/q9_transition_video.mp4
```

Input images and output video link in next page.

OBSERVATIONS:

Produces a smooth crossfade. More frames = smoother transition; mismatched sizes were handled by resizing.
Duration depends on frame count and FPS.

INPUT IMG1 - TRANSITION START

INPUT IMG2 - TRANSITION END

OUTPUT VIDEO LINK

CONCLUSION

Mission accomplished: treated images as plain NumPy arrays, did safe brightness/contrast tweaks, compared three grayscale styles (luminosity looked the most natural), added a simple two-step pseudocolor to reveal subtle differences, replaced green backgrounds with a clean RGB-ratio mask, and handled frames↔video plus a smooth crossfade.
It's intentionally basic, so a few tradeoffs are expected (looped pseudocolor, simple green key, fixed-size/FPS assumptions). For the next round, I'd keep the same structure and add tiny quality-of-life upgrades like before/after histograms and a one-liner CLI for reproducibility.

Key Learnings

- Arrays > magic: treating images as H×W / H×W×3 makes every effect feel obvious and debuggable.
- Dtypes matter: do math in a wider type (int16/float32), then clip and cast to uint8 - no overflow drama.
- Clipping is real: big brightness/contrast pushes can crush shadows/highlights; "how much is too much" shows up in histograms.
- Grayscale isn't one thing: Average is simple but flat; Luminosity matches human perception better; Lightness can look stylized.
- Color maps reveal detail: even a basic blue→red→yellow ramp makes subtle intensity changes pop more than plain gray.
- Masking is a heuristic: the green-screen ratio works on clean screens; threshold tuning is normal (and scene-dependent).
- Video is just time + order: consistent naming, consistent size, and a chosen FPS fully define the output feel.
- Small structure = big clarity: one folder per task, predictable I/O paths, and tiny functions make the pipeline easy to test and explain.