EEE F435 DIP

Assignment 1

Question 1:

import cv2

*def* resize\_frame(*frame*, *scale\_factor*, *interpolation\_method*):

    # Resize the frame using the specified interpolation method

    width = int(*frame*.shape[1]\**scale\_factor*)

    height = int(*frame*.shape[0]\**scale\_factor*)

    resized\_frame = cv2.resize(*frame*, (width, height), *interpolation*=*interpolation\_method*)

    return resized\_frame

*def* process\_video(*input\_path*, *output\_path*, *scale\_factor*, *interpolation\_method*):

    # Opens the input video file

    input\_video = cv2.VideoCapture(*input\_path*)

    # Get video properties

    frame\_count = int(input\_video.get(cv2.CAP\_PROP\_FRAME\_COUNT))

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

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

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

    # Create VideoWriter object to save the output video

    output\_video = cv2.VideoWriter(*output\_path*, cv2.VideoWriter\_fourcc(\*'XVID'), fps, (frame\_width, frame\_height))

  # Process each frame

    for i in range(frame\_count):

        ret, frame = input\_video.read()

        if ret == 0:

            break

        # Convert frame to grayscale

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

        # Resize the frame using the specified interpolation method

        resized\_frame = resize\_frame(gray\_frame, *scale\_factor*, *interpolation\_method*)

        # Write the processed frame to the output video

        output\_video.write(resized\_frame)

    # Releasing the objects

    input\_video.release()

    output\_video.release()

    # Getting size of the output video

    output\_video = cv2.VideoCapture(*output\_path*)

    output\_frame\_count = int(output\_video.get(cv2.CAP\_PROP\_FRAME\_COUNT))

    output\_frame\_width = int(output\_video.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

    output\_frame\_height = int(output\_video.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

    print(*f*"Output Video Size (Frames): {output\_frame\_width}x{output\_frame\_height} (Total Frames: {output\_frame\_count})")

    # Release VideoCapture object

    output\_video.release()

# Main function

if \_\_name\_\_ == "\_\_main\_\_":

    input\_video\_path = *r*'D:\4-2\EEE F435 DIP\assignment1\_video.mp4'        # Location of video

    output\_video\_nearest\_path = "output\_video\_nearest\_Q1.avi"

    output\_video\_bilinear\_path = "output\_video\_bilinear\_Q1.avi"

    output\_video\_bicubic\_path = "output\_video\_bicubic\_Q1.avi"

    scale\_factor = 2  # Increase by a scale factor of 2

    interpolation\_methods = [cv2.INTER\_NEAREST, cv2.INTER\_LINEAR, cv2.INTER\_CUBIC]

for method in interpolation\_methods:

        print(*f*"Processing video with interpolation method: {method}")

        if method == cv2.INTER\_NEAREST:

            output\_video\_path = output\_video\_nearest\_path

        elif method == cv2.INTER\_LINEAR:

            output\_video\_path = output\_video\_bilinear\_path

        else:

            output\_video\_path = output\_video\_bicubic\_path

        process\_video(input\_video\_path, output\_video\_path, scale\_factor, method)

Question 2:

import cv2

import numpy as np

# Defining functions for each of the operations

*def* power\_law\_transform(*img*, *gamma*):

    return np.power(*img* / 255.0, *gamma*) \* 255.0

*def* piecewise\_linear\_transform(*img*, *alpha*, *beta*):

    return np.clip(*alpha* \* *img* + *beta*, 0, 255).astype(np.uint8)

*def* gray\_level\_slice(*img*, *min\_intensity*, *max\_intensity*):

    return np.where(np.logical\_and(*img* >= *min\_intensity*, *img* <= *max\_intensity*), *img*, 0)

*def* bit\_plane\_slice(*img*, *bit\_plane*):

    return (*img* >> *bit\_plane*) & 1

*def* histogram\_equalization(*img*):

    hist, bins = np.histogram(*img*.flatten(), 256, [0, 256])

    cdf = hist.cumsum()

    cdf\_normalized = cdf \* hist.max() / cdf.max()

    cdf\_m = np.ma.masked\_equal(cdf, 0)

    cdf\_m = (cdf\_m - cdf\_m.min()) \* 255 / (cdf\_m.max() - cdf\_m.min())

    cdf = np.ma.filled(cdf\_m, 0).astype('uint8')

    return cdf[*img*]

*def* process\_video(*input\_video\_path*, *output\_video\_path*, *function*, \**args*):

    capt = cv2.VideoCapture(*input\_video\_path*)

    if not capt.isOpened():

        print("Error: Could not open video.")

        return

    # Getting video properties

    fps = capt.get(cv2.CAP\_PROP\_FPS)

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

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

    fourcc = cv2.VideoWriter\_fourcc(\*'XVID')        # .avi video format

    # VideoWriter object for saving the video

    out = cv2.VideoWriter(*output\_video\_path*, fourcc, fps, (frame\_width, frame\_height), *isColor*=False)

    while capt.isOpened():

        ret, frame = capt.read()

        if not ret:

            break

        processed\_frame = *function*(frame, \**args*)

        out.write(processed\_frame)

    capt.release()

    out.release()

    cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

    input\_video\_path = *r*'D:\4-2\EEE F435 DIP\assignment1\_video.mp4'      # Use r for escaping the string

    # Power Law Transformation

    output\_video\_path = "output\_video\_power\_law\_Q2.avi"

    gamma = 0.5         # Setting value of gamma

    process\_video(input\_video\_path, output\_video\_path, power\_law\_transform, gamma)

    # Piecewise Linear Transformation

    output\_video\_path = "output\_video\_linear\_transformed\_Q2.avi"

    alpha = 2.0         # Setting values of alpha and beta

    beta = 50.0

    process\_video(input\_video\_path, output\_video\_path, piecewise\_linear\_transform, alpha, beta)

    # Gray-Level Slicing

    output\_video\_path = "output\_video\_gray\_sliced\_Q2.avi"

    min\_intensity = 50      # Set the minimum intensity to 50

    max\_intensity = 150     # Set the maximum intensity to 150

    process\_video(input\_video\_path, output\_video\_path, gray\_level\_slice, min\_intensity, max\_intensity)

    # Bit-Plane Slicing

    output\_video\_path = "output\_video\_bitplane\_sliced\_Q2.avi"

    bit\_plane = 7

    process\_video(input\_video\_path, output\_video\_path, bit\_plane\_slice, bit\_plane)

    # Histogram Equalization

    output\_video\_path = "output\_video\_histogram\_Q2.avi"

    process\_video(input\_video\_path, output\_video\_path, histogram\_equalization)

Question 3:

import cv2

import numpy as np

*def* smoothing\_5x5(*img*):

    kernel = np.ones((5, 5), np.float32) / 25

    smoothed\_img = cv2.filter2D(*img*, -1, kernel)

    return smoothed\_img

*def* gaussian\_smoothing\_9x9(*img*):

    gaussian\_img = cv2.GaussianBlur(*img*, (9, 9), 0)

    return gaussian\_img

*def* process\_video(*input\_video\_path*, *output\_video\_path*, *smoothing\_function*):

    capt = cv2.VideoCapture(*input\_video\_path*)

    if not capt.isOpened():

        print("Error: Could not open video.")

        return

    # Getting video properties

    fps = capt.get(cv2.CAP\_PROP\_FPS)

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

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

    fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

    out = cv2.VideoWriter(*output\_video\_path*, fourcc, fps, (frame\_width, frame\_height), *isColor*=False)

    while capt.isOpened():

        ret, frame = capt.read()

        if not ret:

            break

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

        smoothed\_frame = *smoothing\_function*(gray\_frame)

        out.write(smoothed\_frame)

    capt.release()

    out.release()

    cv2.destroyAllWindows()

# Main function

if \_\_name\_\_ == "\_\_main\_\_":

    input\_video\_path = *r*'D:\4-2\EEE F435 DIP\assignment1\_video.mp4'

    # Apply 5x5 Smoothing Mask

    output\_video\_path = "output\_video\_5x5\_smoothing\_Q3.avi"

    process\_video(input\_video\_path, output\_video\_path, smoothing\_5x5)

    # Apply 9x9 Gaussian Mask

    output\_video\_path = "output\_video\_9x9\_gaussian\_smoothing\_Q3.avi"

    process\_video(input\_video\_path, output\_video\_path, gaussian\_smoothing\_9x9)

Question 4:

import cv2

import numpy as np

# defining Sobel and Laplacian mask functions

*def* sobel\_mask(*img*):

    sobelx = cv2.Sobel(*img*, cv2.CV\_64F, 1, 0, *ksize*=7)

    sobely = cv2.Sobel(*img*, cv2.CV\_64F, 0, 1, *ksize*=7)

    sobel\_combined = cv2.addWeighted(sobelx, 0.5, sobely, 0.5, 0)

    return cv2.convertScaleAbs(sobel\_combined)

*def* laplacian\_mask(*img*):

    return cv2.Laplacian(*img*, cv2.CV\_64F, *ksize*=7)

# Function to process video

*def* process\_video(*input\_video\_path*, *output\_video\_path*, *sharpening\_function*):

    capt = cv2.VideoCapture(*input\_video\_path*)

    if not capt.isOpened():

        print("Error: Could not open video.")

        return

    # Getting video properties

    fps = capt.get(cv2.CAP\_PROP\_FPS)

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

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

    # VideoWriter object to save video

    fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

    out = cv2.VideoWriter(*output\_video\_path*, fourcc, fps, (frame\_width, frame\_height), *isColor*=False)

    while capt.isOpened():

        ret, frame = capt.read()

        if not ret:

            break

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

        sharpened\_frame = *sharpening\_function*(gray\_frame)

        out.write(sharpened\_frame)

    capt.release()

    out.release()

    cv2.destroyAllWindows()

# main function

if \_\_name\_\_ == "\_\_main\_\_":

    input\_video\_path = *r*'D:\4-2\EEE F435 DIP\assignment1\_video.mp4'

    # Apply 7x7 Sobel Mask

    output\_video\_path = "output\_video\_sobel\_mask\_Q4.avi"

    process\_video(input\_video\_path, output\_video\_path, sobel\_mask)

    # Apply 7x7 Laplacian Mask

    output\_video\_path = "output\_video\_laplacian\_mask\_Q4.avi"

    process\_video(input\_video\_path, output\_video\_path, laplacian\_mask)

Question 5:

import cv2

import numpy as np

# Function defined for Fourier transform

*def* apply\_dft(*img*):

    f\_img = np.fft.fft2(*img*)

    fshift = np.fft.fftshift(f\_img)

    magnitude\_spectrum = 20 \* np.log(np.abs(fshift))

    return magnitude\_spectrum

# Function for video prcessing

*def* process\_video(*input\_video\_path*, *output\_video\_path*):

    capt = cv2.VideoCapture(*input\_video\_path*)

    if not capt.isOpened():

        print("Error: Could not open video.")

        return

    # Video properties

    fps = capt.get(cv2.CAP\_PROP\_FPS)

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

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

    # VideoWriter object

    fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

    out = cv2.VideoWriter(*output\_video\_path*, fourcc, fps, (frame\_width, frame\_height), *isColor*=False)

    while capt.isOpened():

        ret, frame = capt.read()

        if not ret:

            break

        # Converting to gray

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

        dft\_frame = apply\_dft(gray\_frame)

        # Normalize the DFT frame for visualization

        dft\_frame = cv2.normalize(dft\_frame, None, 0, 255, cv2.NORM\_MINMAX)

        dft\_frame = dft\_frame.astype(np.uint8)

        out.write(dft\_frame)

    capt.release()

    out.release()

    cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

    input\_video\_path = *r*'D:\4-2\EEE F435 DIP\assignment1\_video.mp4'

    output\_video\_path = "output\_video\_fourier\_Q5.avi"

    process\_video(input\_video\_path, output\_video\_path)