ELEC/COMP 447/546
Assignment 3
Due: Feb 28, 2023, Midnight

1.0 Optical Flow

In this problem, you will implement both the Lucas-Kanade and Horn-Schunck algorithms. Your implementations should use a Gaussian pyramid to properly account for large displacements. You can use your pyramid code from Homework 2, or you may simply use Opencv's pyrDown function to perform the blur+downsampling. You may also use Opencv's Sobel filter to obtain spatial (x,y) gradients of an image.

1.1 Lucas-Kanade (5 points)

Implement the Lucas-Kanade algorithm, and demonstrate tracking points on this video.

1. Select corners from the first frame using the Harris corner detector. You can use this command: `corners = cv.cornerHarris(gray_img,2,3,0.04).`
2. Track the points through the entire video by applying Lucas-Kanade between each pair of successive frames. This will yield one 'trajectory' per point, with length equal to the number of video frames.
3. Create a gif showing the tracked points overlaid as circles on the original frames. You can draw a circle on an image using cv.circle. You can save a gif with this code:

   ```
   import imageio
   imageio.mimsave('tracking.gif', im_list, fps=10)
   ```

   where `im_list` is a list of your output images. You can open this gif in your web browser to play it as a video and visualize your results. Show a few frames of the gif in your report, and save the gif in your Google Drive, and place the link to it in your report. Make sure to allow view access to the file!
4. Answer the following questions:
   a. Do you notice any inaccuracies in the point tracking? Where and why?
   b. How does the tracking change when you change the local window size used in Lucas-Kanade?

1.2 Horn-Schunck (5 points)

Implement the Horn-Schunck algorithm. Display the flow fields for the 'Army,' 'Backyard,' and 'Mequon' test cases from the Middlebury dataset, located here. Consider 'frame10.png' as the first frame, and 'frame11.png' as the second frame for all cases.

Use this code to display each predicted flow field as a colored image:

```
hsv = np.zeros(im.shape, dtype=np.uint8)
hsv[..., 1] = 255
mag, ang = cv.cartToPolar(flow_x, flow_y)
hsv[..., 0] = ang * 180 / np.pi / 2
hsv[..., 2] = cv.normalize(mag, None, 0, 255, cv.NORM_MINMAX)
out = cv.cvtColor(hsv, cv.COLOR_HSV2RGB)
```

1.3 ELEC/COMP 546 Only: Improving Horn-Schunck with superpixels (5 points)
Recall superpixels discussed in lecture and described further in this paper. How might you use superpixels to improve the performance of Horn-Schunck? Can you incorporate your idea into the smoothness + brightness constancy objection function? Define any notation you wish to use in the equation. *You don't have to implement your idea in code for this question.*

2.0 Image Compression with PCA

In this problem, you will use PCA to compress images, by encoding small patches in low-dimensional subspaces. Download these two images:

Test Image 1
Test Image 2

Do the following steps for each image *separately*.

2.1 Use PCA to model patches (5 points)
Randomly sample at least 1,000 16 x 16 patches from the image. Flatten those patches into vectors (should be of size 16 x 16 x 3). Run PCA on these patches to obtain a set of principal components. Please write your own code to perform PCA. You may use `numpy.linalg.eigh`, or `numpy.linalg.svd` to obtain eigenvectors.

Display the first 36 principal components as 16 x 16 images, arranged in a 6 x 6 grid (Note: remember to sort your eigenvalues and eigenvectors by decreasing eigenvalue magnitude!). Also report the % of variance captured by all principal components (not just the first 36) in a plot, with the x-axis being the component number, and y-axis being the % of variance explained by that component.

2.2 Compress the image (5 points)

Show image reconstruction results using 1, 3, 10, 50, and 100 principal components. To do this, divide the image into non-overlapping 16 x 16 patches, and reconstruct each patch independently using the principal components. Answer the following questions:

1. Was one image easier to compress than another? If so, why do you think that is the case?
2. What are some similarities and differences between the principal components for the two images, and your interpretation for the reason behind them?

Submission Instructions

All code must be written using Google Colab (see course website). Every student must submit a zip file for this assignment in Canvas with 2 items:

1. An organized report submitted as a PDF document. The report should contain all image results (intermediate and final), and answer any questions asked in this document. It should also contain any issues (problems encountered, surprises) you may have found as you solved the problems. The heading of the PDF file should contain:
    a. Your name and Net ID.
    b. Names of anyone you collaborated with on this assignment.
    c. A link to your Colab notebook (remember to change permissions on your notebook to allow viewers).
2. A pdf copy of your Colab notebook.

Collaboration Policy

I encourage collaboration both inside and outside class. You may talk to other students for general ideas and concepts, but you should write your own code, answer questions independently, and submit your own work.

Plagiarism

Plagiarism of any form will not be tolerated. You are expected to credit all sources explicitly. If you have any doubts regarding what is and is not plagiarism, talk to me.