CS 4476: Computer Vision, Spring 2021
PS4

Instructor: Judy Hoffman

Due: Tuesday, March 23rd, 11:58 pm ET

Instructions

1. The assignment must be done in Python3. No other programming languages are allowed.
2. Fill your answers in the answer sheet PPT provided and submit the file under the name: First-Name_LastName_PS4.pdf on Gradescope. Please do not modify the layout of the boxes provided in the answer sheet and fit your answers within the space provided.
3. Please enter your code in the designated areas of the template Python files. Please do not add additional functions/imports to the files. Points will be deducted for any changes to code/file names, use of static paths and anything else that needs manual intervention to fix.
4. Please submit your code and output files in a zipped format, using the helper script `zip_submission.py` with your GT username as a command line argument (using `--gt_username`), to Gradescope. Please do not create subdirectories within the main directory. The `.zip_dir_list.yml` file contains the required deliverable files, and `zip_submission.py` will fail if all the deliverables are not present in the root directory. Feel free to comment and uncomment them as you complete your solutions.
5. For the implementation questions, make sure your code is bug-free and works out of the box. Please be sure to submit all main and helper functions. Be sure to not include absolute paths. Points will be deducted if your code does not run out of the box.
6. If plots are required, you must include them in your Gradescope report and your code must display them when run. Points will be deducted for not following this protocol.
7. Ensure that you follow the instructions very carefully.

Setup

Note that we will be using a new conda environment for this project!

1. Install Miniconda. It doesn't matter whether you use Python 2 or 3 because we will create our own environment that uses 3 anyways.
2. Open the terminal
   (a) On Windows: open the installed **Conda prompt** to run the command.
   (b) On MacOS: open a terminal window to run the command
   (c) On Linux: open a terminal window to run the command
3. Navigate to the folder where you have the project

4. Create the conda environment for this project
   (a) On Windows: `conda env create -f proj4_env_win.yml`
   (b) On MacOS: `conda env create -f proj4_env_mac.yml`
   (c) On Linux: `conda env create -f proj4_env_linux.yml`
5. Activate the newly created environment
   (a) On Windows: use the command `conda activate proj4`
   (b) On MacOS: use the command `source activate proj4`
   (c) On Linux: use the command `source activate proj4`

Run the notebook using `jupyter notebook ./proj4_code/proj4.ipynb`.
At this point, you should see the jupyter notebook in your web browser. Follow all the instructions in the notebook for both the code + report portions of this project.

1 Color Quantization with K-Means [25 Points]

For this problem you will write code to quantize an image using k-means clustering and experiment with two different color spaces - RGB and HSV. Complete the functions defined below in `quantization_student.py`.

1. (7 points) Given an RGB image, quantize the 3-dimensional RGB space, and map each pixel in the input image to its nearest k-means center. That is, replace the RGB value at each pixel with its nearest cluster's average RGB value. Use the following form:

   ```
   [quantizedImg, clusterCenterColors] = quantizeRGB(origImg, k)
   ```

   where `origImg` and `quantizedImg` are `MxNx3` matrices of type `uint8`, `k` specifies the number of colors to quantize to, and `clusterCenterColors` is a `k x 3` array of the `k` centers. Use `proj4.ipynb` to run your implemented function and generate quantized images (for the file `fish.jpg`) with 3, 5 and 10 clusters using both RGB and HSV color spaces. Put these images in your report. (You can use the Scikit-Learn implementation of the k-means algorithm for this question.)

2. (7 points) Given an RGB image, convert to HSV, and quantize the 1-dimensional **Hue** space. Map each pixel in the input image to its nearest quantized Hue value, while keeping its Saturation and Value channels the same as the input. Convert the quantized output back to RGB color space. Use the following form:

   ```
   [quantizedImg, clusterCenterHues] = quantizeHSV(origImg, k)
   ```

   where `origImg` and `quantizedImg` are `MxNx3` matrices of type `uint8`, `k` specifies the number of clusters, and `clusterCenterHues` is a `k x 1` vector of the hue centers. Use `proj4.ipynb` to run your implemented function and generate quantized images (for the file `fish.jpg`) with 3, 5 and 10 clusters using both RGB and HSV color spaces. Put these images in your report. (You can use the Scikit-Learn implementation of the k-means algorithm for this question.)

3. (5 points) Write a function to compute the SSD error (sum of squared error) between the original RGB pixel values and the quantized values, with the following form:

   ```
   [error] = computeQuantizationError(origImg, quantizedImg)
   ```

where `origImg` and `quantizedImg` are both RGB images, and `error` is a scalar giving the total SSD error across the image. Write down the logarithmic error (base e) for all the generated RGB and HSV images (3, 5 and 10 clusters) in your report.
NOTE: The function should not return logarithmic error. To pass all the tests in the notebook and on Gradescope, please use `sklearn.cluster.KMeans` for clustering and set `random_state` to 101.

4. (6 points) Answer the following questions in your report. Please try to restrict your answers to 1-2 sentences of fairly high-level explanations.

   (a) (2 points) How do the results vary with the number of quantization bins?

   (b) (2 points) How are the **qualitative** results between RGB and HSV color spaces different? How would you explain this difference?

   (c) (2 points) Name and explain 2 metrics (apart from SSD) used for comparing the generated/modified image (here the modified image is the quantized image) with the original one.

2 Circle Detection with Hough Transform [40 Points]

1. (16 + 4 = 20 points) Implement a Hough Transform circle detector that takes an input image and a fixed radius, and returns the centers and radii of any detected circles of about that size and the Hough space used for finding the centers. Include a function with the following form:

   ```
   [circles, houghAccumulator] = detectCircles(im, radius, threshold, useGradient)
   ```

   where `im` is the input image (a matrix of size `MxNx3` of type `uint8`), `radius` specifies the size of circle we are looking for, `threshold` corresponds to the voting threshold used to identify circles from the accumulator array and `useGradient` is a boolean flag that allows the user to optionally exploit the gradient direction measured at the edge points. Identify circle candidates by thresholding to find local maxima in the hough space.

   The output `circles` is an `Kx3` matrix in which each row lists the `(x,y)` position of a detected circle's center and the corresponding radii – *i.e.,* a row contains the elements `[x, y, radius]`. Similarly, `houghAccumulator` is an `MxN` matrix denoting the Hough Accumulator array. Complete the implementation of the function in the file `detect_circles_student.py` and briefly explain your implementation in concise steps in the report (English, not code).

2. (10 points) **Circle detection on synthetic images.** In this part, we will check the output of your implementation on the provided synthetically generated image, `im1.jpg`.

   (a) (6 points) Demonstrate the output of the function applied to the image `im1.jpg` to detect circles of radii 75, 90, 100 and 143 by setting `useGradient` to `True` and `False`. In each case, display the images with the detected circle(s), labeling the figure with the circumference and center. Include the outputs for each case in your report.

   (b) (4 points) Experiment with the threshold values to determine circle parameters from the accumulator array. In your report include images of the selected circles along with the corresponding Hough Space for low (≤ 0.4), mid-range (≈ 0.7) and high (≥ 0.95) thresholds. Explain how your results vary with increasing thresholds and why that may be the case. You can set `useGradient` to either `True` or `False`.

3. (10 points) **Circle detection on real images.** Demonstrate the output of the function applied to the real image `biliards.jpg`. Check if your implementation is capable of identifying the balls by varying the radii in the range 17-20. You might have to modulate the `threshold` parameter for this part –

try varying the same between low to high ranges (see 2.2(b)). In the report, include images with the detected circle(s), labeling the figure with the circumference and center. You can set `useGradient` to either `True` or `False`.

NOTE: You are not allowed to use any hough transform based circle detector from existing libraries – for instance, `cv2.HoughCircles` – and are expected to implement the algorithm on your own. As helper functions, you may use `skimage.color.rgb2gray` (for RGB to grayscale conversion), `skimage.feature.canny` (for edge detection) and any denoising functions if required. Additionally, you may use the `showCircles()` function defined in the same file to visualize the detected circles. You can use `im1.jpg` to debug / verify your implementation (see 2.2(a)). Note that your implementation will be evaluated on a different image. For the sake of simplicity, you can assume that the test image for 2.1 will have the same basic color scheme as `im1.jpg`. Any hyper-parameters you tune for `im1.jpg` should also be applicable for the test image.
Tips: For debugging, it might be helpful to visualize the intermediate edge detection outputs and vary the voting threshold passed as input to the function.

3 [Extra Credit] Circle Detection with Hough Transform on Real Images with Unknown Radii [10 points]

Extend your Hough circle detector implementation to detect circles of any radius. Optimize your code for the image `jupiter.jpg`. You would be awarded points proportional to the number of circles detected. Detecting all of the circles gives you the maximum credit. Be sure to include the generated results in your report.

Tips, Tricks, and Common Problems

- Make sure you're not swapping x and y coordinates at some point. If your interest points aren't showing up where you expect or if you're getting out of bound errors you might be swapping x and y coordinates. Remember, images expressed as NumPy arrays are accessed image[y,x].
- Useful Modules: `numpy, scipy, skimage, matplotlib`.
- Useful Functions: `scipy.cluster.vq.kmeans2, numpy.gradient, numpy.arctan skimage.color.rgb2hsv, skimage.color.rgb2gray, skimage.feature.canny sklearn.cluster.Kmeans, scipy.misc, scipy.ndimage`.
- If the variable img is a 3d matrix containing a color image, img.reshape(-1, 3) will yield a 2d matrix with the RGB features as its rows.

Rubric

Code: The score for each part is provided below. Please refer to the submission results on Gradescope for a detailed breakdown.

Part 1: Color Quantization	25
Part 2: Circle Detection	40
Extra Credit	10
Total	*65 (+10)*

Submission Instructions and Deliverables

Code zip: The following code deliverables will be uploaded as a zip file on Gradescope.

Deliverables

1. `proj4_code/quantization_student.py`
   (a) `quantizeRGB()`
   (b) `quantizeHSV()`
   (c) `computeQuantizationError()`
2. `proj3_code/detect_circles_student.py`
   (a) `detectCircles()`
3. `proj4_code/proj4.ipynb`

Do not create this zip manually! You are supposed to use the command `python zip_submission.py -gt_username <username>` for this.
Report: The final thing to upload is the PDF export of the report on gradescope.
To summarize, the deliverables are as follow:

- Submit the code as zip on Gradescope at PS4 - Code
- Submit the report as PDF on Gradescope at PS4 - Report. Please refer to the pptx template where we have detailed the points associated with each question.

There is no submission to be done on Canvas. Good luck!

This iteration of the assignment is developed by Prithvijit Chattopadhyay and Judy Hoffman. This assignment was originally developed by James Hays, Samarth Brahmbhatt, and John Lambert, and updated by Judy Hoffman, Mitch Donley, and Vijay Upadhya.