CS 4476: Computer Vision, Fall 2020

PS2

Instructor: Devi Parikh

Due: Thursday, September 24th, 11:59 pm

Instructions

1. The deliverables for this assignment must be submitted on Gradescope. Please follow the submission instructions carefully.
   - Save your written answers to a pdf file. LaTeX is strongly encouraged.
   - Answers to each problem must be on a separate page (or pages). On Gradescope uniquely assign each page to a single problem. Multiple sub-problems can be included on the same page.
   - Do not include your name on the answer sheet.
   - Upload your pdf to the `PS2` assignment on Gradescope.
   - Save your code and output images in a zip file.
   - Be sure to include all the files listed in the submission checklist.
   - Upload your zip file to the `PS2 Code` assignment on Gradescope.

1 Short answer problems [20 points]

1. Suppose we form a texture description using textons built from a filter bank of multiple anisotropic derivative of Gaussian filters at two scales and six orientations (as displayed below in Figure 1). Is the resulting representation sensitive to orientation or is it invariant to orientation? Explain why.

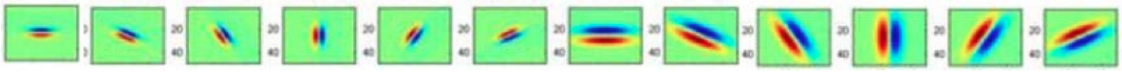

Figure 1: Filter bank

2. Consider Figure 2 below. Each small square denotes an edge point extracted from an image. Say we are going to use k-means to cluster these points' positions into k=2 groups. That is, we will run k-means where the feature inputs are the (x,y) coordinates of all the small square points. What is a likely clustering assignment that would result? Briefly explain your answer.

3. When using the Hough Transform, we often discretize the parameter space to collect votes in an accumulator array. Alternatively, suppose we maintain a continuous vote space. Which grouping algorithm (among k-means, mean-shift, or graph-cuts) would be appropriate to recover the model parameter hypotheses from the continuous vote space? Briefly describe and explain.

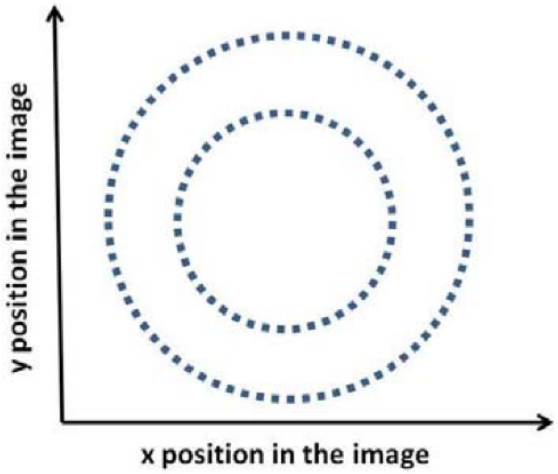

Figure 2: Edge points

4. Suppose we have run the connected components algorithm on a binary image, and now have access to the multiple foreground 'blobs' within it. Write pseudocode showing how to group the blobs according to the similarity of their outer boundary shape, into some specified number of groups. Define clearly any variables you introduce.

2 Programming [80 points]

2.1 Color quantization with k-means [40 points]

For this problem you will write code to quantize a color space by applying k-means clustering to the pixels in a given input image, and experiment with two different color spaces—RGB and HSV.

Part A: Implementation [20 points]

Follow the instructions in `KMeans.ipynb` to implement the functions below. `KMeans.ipynb` is included in this zip file. After you are done use the `notebook2script.py` script to export a file called `submissionKMeans.py`. Call the functions in this file to complete Part B. Include this file in your code submission.

- `quantized_img = quantize_rgb(img, k)` - Given an RGB image `img` and number of clusters `k`, quantize the 3-dimensional RGB space and map each pixel in the input image to its nearest k-means cluster. Use the k-means class from `sklearn.cluster`.
- `quantized_img = quantize_hsv(img, k)` - Given an RGB image `img` and number of clusters `k`, convert the image 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 k-means class from `sklearn.cluster`.
- `error = compute_quantization_error(img, quantized_img)` - Given an RGB image `img` and a quantized RGB image `quantized_img`, compute the SSD error (sum of squared error) between the two images.

- `hist_equal, hist_clustered = get_hue_histograms(img, k)` - Given an RGB image `img` and a number of clusters `k`, convert the image to HSV and compute two histograms of the Hue values. Let the first histogram `hist_equal` use `k` equally-spaced bins (uniformly dividing up the hue values). Let the second histogram `hist_clustered` use bins defined by the `k` clusters center memberships (i.e., all pixels belonging to hue cluster i go into the i-th bin, for i=1,...,k).

Part B: Experimentation [20 points]

1. [5 points] Use the functions defined above to quantize the `fish.jpg` image included in the zip file from part A. On you answer sheet, display the results of quantizing the image in (a) rgb space and (b) hsv space (as described above), print the SSD error between the original RGB image and the results of the (c-d) two quantization methods, and display the (e-f) two different Hue space histograms. Illustrate all of the results with two values of `k`, a lower value and higher value. Label all plots clearly with titles.
2. [15 points] On your answer sheet, explain all of the results. How do the two forms of histogram differ? How and why do results vary depending on the color space? The value of k? Across different runs?

2.2 Circle detection with the Hough Transform [40 points]

Implement a Hough transform circle detector that takes an input image and a fixed radius, and returns the centers of any detected circles of about that size. Include a function with the following form:

- `centers = detect_circles(img, radius, use_gradient)` - Given an RGB image `img`, a target `radius` that specifies the size of circle we are looking for, and a flag `use_gradient` that allows the user to optionally exploit the gradient direction measured at the edge points. The output centers is an `N x 2` matrix in which each row lists the `(x, y)` position of a detected circles' center. Save this function in a file called `submissionDetectCircles.py` and submit it.

Then experiment with the basic framework, and in your writeup analyze the following:

1. [10 points] Explain your implementation in concise steps (English, not code/pseudocode).
2. [10 points] Demonstrate the function applied to the provided images `jupiter.jpg` and `egg.jpg`. Display the accumulator arrays obtained by setting `use_gradient` to `True` and `False`. In each case, display the images with detected circle(s), labeling the figure with the radius.
3. [10 points] For one of the images, display and briefly comment on the Hough space accumulator array.
4. [5 points] Experiment with ways to determine how many circles are present by post-processing the accumulator array.
5. [5 points] For one of the images, demonstrate the impact of the vote space quantization (bin size).

3 [OPTIONAL] Extra credit [up to 10 points]

Extend your Hough circle detector implementation to detect circles of any radius. Demonstrate the method applied to the test images.

Python hints:

1. Useful Modules: `numpy`, `imageio`, `sklearn`, `skimage`, `matplotlib`.
2. Useful Classes/Functions: `sklearn.cluster.KMeans`, `skimage.color.rgb2hsv`, `skimage.color.rgb2gray`, `numpy.gradient`, `numpy.arctan`, `skimage.feature.canny`.

3. 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.

Submission Checklist

- ☐ Answer sheet
- ☐ `Zip` file containing:
  - ☐ `submissionKMeans.py`
  - ☐ `submissionDetectCircles.py`

This assignment is adapted from the PS2 assignment from Kristen Grauman's CS 376: Computer Vision at UT Austin.