MIT CSAIL

6.869 Advances in Computer Vision

Fall 2019

Problem Set 4

Posted: Thursday, October 3, 2019 **Due:** Thursday 23:59, October 10, 2019

6.869 and 6.819 students are expected to finish all problems unless there is an additional instruction.

We provide a python notebook with the code to be completed. You can run it locally or in Colab (upload it to Google Drive and select 'open in colab') to avoid setting up your own environment. Once you finish, run the cells and download the notebook to be submitted.

Submission Instructions: Please submit a .zip file named ⟨your_kerberos⟩.zip containing 1) report named report.pdf including your answers to all required questions with images and/or plots showing your results, and 2) the python notebook provided, with the cells run and the relevant source code. If you include other source code files for a given exercise, please indicate it in the report.

Late Submission Policy: If your pset is submitted within 7 days (rounding up) of the original deadline, you will receive partial credit. Such submissions will be penalized by a multiplicative coefficient that linearly decreases from 1 to 0.5.

Problem 1 *Texture Synthesis*

In this problem you will implement the Efros and Leung algorithm [1] for texture synthesis discussed in Section 10.3 of Forsyth and Ponce [2]. In addition to reading the textbook you may also find it helpful to visit Efros' texture synthesis website:
http://people.eecs.berkeley.edu/~efros/research/EfrosLeung.html

in which many of the implementation details described below can be found. The Efros and Leung algorithm synthesizes a new texture by performing an exhaustive search of a source texture for each synthesized pixel in the target image, in which sum-of-squared differences (SSD) is used to associate similar image patches in the source image with that of the target. The algorithm is initialized by randomly selecting a 3×3 patch from the source texture and placing it in the center of the target texture. The boundaries of this patch are then recursively filled until all pixels in the target image have been considered. Implement the Efros and Leung algorithm as the following Python function:

```
synthIm = SynthTexture(sample, w, s)
```

where `sample` is the source texture image, `w` is the width of the search window, and `s=[ht,` wt] specifies the height and width of the target image `synthIm`. As described above, this algorithm will create a new target texture image, initialized with a 3x3 patch from the source image. It will then grow this patch to fill the entire image. As discussed in the textbook, when growing the image un-filled pixels along the boundary of the block of synthesized values are considered at each iteration of the algorithm. A useful technique for recovering the location of these pixels is using *dilation*, a morphological operation that expands image regions.

Use `scipy.ndimage.binary_dilation`, `numpy.nonzero` and `numpy.ix_` routines to recover the un-filled pixel locations along the boundary of the synthesized block in the target image.

In addition to the above function we ask you to write a subroutine that for a given pixel in the target image, returns a list of possible candidate matches in the source texture along with their corresponding SSD errors. This function should have the following syntax:

```
bestMatches, errors = FindMatches(template, sample, G)
```

where `bestMatches` is the list of possible candidate matches with corresponding SSD errors specified by `errors`. `template` is the $w \times w$ image template associated with a pixel of the target image, `sample` is the source texture image, and `G` is a 2D Gaussian mask discussed below. This routine is called by `SynthTexture` and a pixel value is randomly selected from `bestMatches` to synthesize a pixel of the target image. To form `bestMatches` accept all pixel locations whose SSD error values are less than the minimum SSD value times $(1 + \epsilon)$. To avoid randomly selecting a match with unusually large error, also check that the error of the randomly selected match is below a threshold δ. Efros and Leung use threshold values of $\epsilon = 0.1$ and $\delta = 0.3$.

Note that `template` can have values that have not yet been filled in by the image growing routine. Mask the template image such that these values are not considered when computing SSD. Efros and Leung suggest using the following image mask:

```
Mask = G .* validMask
```

where `validMask` is a square mask of width w that is 1 where the template is filled, 0 otherwise and `G` is a 2D zero-mean Gaussian with variance $\sigma = w/6.4$ sampled on a $w \times w$ grid centered about its mean. `G` can be pre-computed using `scipy.ndimage.gaussian_filter` routine. The purpose of the Gaussian is to down-weight pixels that are farther from the center of the template. Also, make sure to normalize the mask such that its elements sum to 1.

Test and run your implementation using the grayscale source texture image `rings.jpg` provided in the folder, with window widths of $w = 5, 7, 13$, $s = [100, 100]$ and an initial starting seed of $(x, y) = (4, 32)$. Explain the algorithm's performance with respect to window size. For a given window size, if you re-run the algorithm with the same starting seed do you get the same result? Why or why not? Is this true for all window sizes? Also run your implementation using real color image `texture.jpg` and see how window size affects the performance.

Include in your report the synthesized textures that correspond to each window size along with answers to the above questions.

References

[1] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *ICCV*, 1999.

[2] David A Forsyth and Jean Ponce. Computer vision: a modern approach. 2003.