MIT CSAIL

6.8300/6.8301 Advances in Computer Vision

Spring 2023

Problem Set 7 - Part II

Posted: Tuesday, April 25, 2023 **Due:** Tuesday 11:59 pm, May 2, 2023

The relevant material for this Pset was covered in Lecture 19 and the talk on Stable Diffusion.

6.8300 students are expected to finish Problem 1.
6.8301 students are expected to finish Problem 1.

We provide two Python notebooks (for Problems 1 and 2, respectively) with the code to be completed. Only the notebook corresponding to Problem 1 will be graded. You can run each notebook locally or on Google Colab. To use Colab, upload the notebook to Google Drive and double-click the notebook (or right-click and select Open with Google Colaboratory), which will allow you to complete the problems without setting up your own environment.

Submission Instructions: To submit your Pset, navigate to the Pset 7 - Part II assignment submission window on Canvas and you will find a link that takes you to **Gradescope**. All you will need to submit is the **Python notebook (.ipynb)** corresponding to **Problem 1**. Please make sure to run all the cells before submitting your notebook, so that we can inspect and grade your output in addition to your code. If a problem requires you to write text or you would like to write comments, the easiest way to do so is by adding a new text cell and writing into it.

Attention: Failure to follow the submission instructions will result in point deductions.

Late Submission Policy: If your problem set 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* (10 points)

In this problem you will implement the Efros and Leung algorithm for texture synthesis [1] discussed in Section 9.3 of Forsyth and Ponce [2]. In addition to reading the textbook, you may also find it helpful to visit Efros' texture synthesis website in which many of the implementation details described below can be found.

As discussed in class, 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 = synth_texture(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 3×3 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, unfilled 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 *dilation*, a morphological operation that expands image regions. Specifically, one could use `scipy.ndimage.binary_dilation`, `numpy.nonzero`, `numpy.ix_` routines to recover the unfilled pixel locations along the boundary of the synthesized block in the target image. We have provided the sample code for this part, you are encouraged to take a look and play with it.

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] = find_matches(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 `synth_texture` 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 standard deviation $\sigma = w/6.4$ sampled on a $w \times w$ grid centered about its mean. `G` can be pre-computed using the `scipy.ndimage.gaussian_filter` routine or you can compute it yourself using the Gaussian equation. 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.

To summarize, you will need to complete the following three tasks in the notebook corresponding to Problem 1: (i) compute the Gaussian kernel, (ii) complete the `find_matches` function, and (iii) sample from the output of the `find_matches` function and update the image.

Test and run your implementation using the grayscale source texture image `rings.jpg` available in the notebook, with window widths of $w = 5, 7, 13$, $s = [100, 100]$ and an initial starting seed of $(x, y) = (3, 31)$ (the initial pixel). Make sure your notebook displays the synthesized textures that correspond to each of the three window sizes. Explain the algorithm's performance with respect to window size. Last, for a given window size, if you re-run the algorithm with the same starting seed (i.e. the same initial pixel), do you get the same result? Why or why not? Is this true for all window sizes? Include your answers in your notebook.

Problem 2 *Stable Diffusion* (0 points)

See the Python notebook corresponding to Problem 2.

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.* Pearson,, 2012.