MIT CSAIL

6.819/6.869 Advances in Computer Vision

Spring 2022

Problem Set 7: Texture Synthesis and Style Transfer

Posted: Tuesday, April 12, 2022 **Due:** Midnight, Tuesday, April 19, 2022

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: **Read Carefully! We will deduct points for not following instructions.** Please submit a `.zip` file named `<your kerberos>.zip`. At the root of the zip file should be three separate files: 1) a report named `<your kerberos>.pdf`, including your responses to all required questions with images and/or plots showing your results, 2) a file named `<your kerberos>-texture-synthesis.ipynb` of the python notebook you filled in with the cells run 3) a file named `<your kerberos>-style-transfer.ipynb` of the python notebook you filled in with the cells run. *There should be no other files and no nested directories (only the root directory) in your zip.* For any questions involving code, copy the relevant lines into your PDF writeup as succinctly as possible. Also include any relevant image outputs directly in the PDF. Please draw a box around your solution (`\fbox`).

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 (5pts)*

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: `http://graphics.cs.cmu.edu/people/efros/research/synthesis.html`, 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 = 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 *dilation*, a morphological operation that expands image regions. Specifically, one could use `scipy.ndimage.binary dilation`, `numpy.nonzero`, `numpy.ix_` routines to recover the un-filled 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] = 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 standard deviation $\sigma = w/6.4$ sampled on a $w \times w$ grid centered about its mean. `G` can be pre-computed using `scipy.ndimage.gaussian filter` routine or one can computed theselves following 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 follow three tasks in `texture-synthesis.ipynb`: (i) compute the Gaussian kernel, (ii) complete the `Find_matches` function, and (iii) sample from the output of `Find_matches` function and update the image. **Provide short code snippets for these in your PDF writeup**.

Test and run your implementation using the grayscale source texture image `rings.jpg` available in the pset, with window widths of $w = 5, 7, 13$, $s = [100, 100]$ and an initial starting seed

Figure 1: Screaming Taipei 101.

of $(x, y) = (3, 31)$. 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? **Answer these questions in your PDF writeup.**

In your report, include the synthesized textures that correspond to each window size along with answers to the above questions. **Copy these figures in your PDF writeup.**

Problem 2 *Neural Style Transfer (3pts)*

In the second part of the assignment, we will walk you through Neural Style Transfer, an algorithm presented by Gatys *et al.* [3]. Neural Style Transfer allows one to turn an input image into a new artistic style. For instance, Fig. 1 shows the magnifacent `Taipei-101`[1] with the art style from `The Scream`[2]. The algorithm takes as inputs the content-image and the style-image, and generates an output that looks similar to the content of the content-image and the artistic style of the style-image.

Please follow the step-by-step instructions within `style-transfer.ipynb` and answer the following questions:

(1) Implement the code for computing the gram matrix. **Copy the relevant lines into your PDF writeup.**

(2) Different from the content-loss which computes the mean square error of the feature maps of the optimized image and the content image, we computed the `gram_matrix` for the style-loss because _____. (Select all that apply.)

   (a) style-loss should encode the spatial correspondence between the optimized image and the style-image.

   (b) style-loss shouldn't encode the spatial correspondence between the optimized image and the style-image.

   (c) style-loss should match the distribution of features found in the optimized image and the style-image.

[1] https://en.wikipedia.org/wiki/Taipei_101
[2] https://en.wikipedia.org/wiki/The_Scream

(d) style-loss shouldn't match the distribution of features found in the optimized image and the style-image.

(3) Using `conv_5` or `conv_1` as the content-loss layer produces different results because _____. (Select all that apply.) **Attach the results in your report.**

(a) `conv_5` features care more about the exact pixel values reconstruction of the content-image.

(b) `conv_1` features care more about the exact pixel values reconstruction of the content-image.

(c) `conv_5` features care more about the semantic content of the content-image.

(d) `conv_1` features care more about the semantic content of the content-image.

(4) The style transfer algorithm presented here is cool, but we can do better! Notably, the iterative optimization process is slow, which limits its real-world application. Below, we attached several papers on speeding up the style transfer process. Briefly summarize one of the papers in your report:

(a) Perceptual Losses for Real-Time Style Transfer and Super-Resolution, Johnson *et al.* [5]

(b) Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization, Huang and Belongie [4].

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.

[3] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *CVPR*, 2016.

[4] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, 2017.

[5] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016.