Assignment 03

Please refer to MyCourses/Gradescope for assignment deadlines.

The goal of this assignment is to give you hands-on practice with the following concepts:

- correlation and convolution
- separable filters
- handling bounding boxes
- intersection-over-union (IoU)
- template matching

Part 1: DIY correlation function and separable filters

Correlation can be done with the OpenCV function `cv.filter2D`. Convolution can be done by first flipping the kernel then using `filter2D`, since convolution is just correlation with the sign of the horizontal (l)and vertical (k) indices flipped. Here, you're going to implement your own version of correlation in numpy to see how it works under the hood and to better appreciate the "separable filters" property.

Write the body of the `my_correlation` function in `correlation.py`. The `my_correlation` function should make use of `numpy` but may not call any `cv` functions. Note that you will have to implement your own padding in order to match the behavior or `cv.BORDER_REPLICATE`. (Hint: you don't actually need to create a new padded image, you just need to re-use pixels from the border any time the index would take you outside the `[0...h-1, 0...w-1]` range). Be careful not to modify the original image in this function!

Note: Your `my_correlation` function must match the behavior of `cv.filter2D` regardless of whether the input is a single-channel or multi-channel image, and regardless of the content and data type of the kernel. The `np.atleast_3d` function may be useful for writing code that is agnostic to the number of channels in the input image.

Once you have it working, see how fast you can make your correlation function. There will be a leaderboard for the fastest code on Gradescope. **Incorrect** submissions will not be considered for the leaderboard. First, get the functionality correct. Then make it fast. Note that the leaderboard will test your code both with random *separable* filters and with random *inseparable* filters. Since separable filters are much faster to apply (linear in the width and height of the filter rather than quadratic), you s hould be able to get a much faster time by

- detecting whether the filter is separable

- if so, splitting it into its u and v components, then applying the filter in two passes rather than one
- if not, applying the filter in one pass as before

Exactly how to tell whether a given 2D filter is separable is handled by an interesting bit of linear algebra – equation (3.21) in the CVAA book. I consider this hard and out of scope, so it's been given to you as a helper function in `utils.py`. Feel free to import and use those functions.

The fastest submissions will be ones that both take advantage of separability, and which offload as much computation as possible to numpy's C backend using vectorization. Loops in Python are slow!

Winners on the leaderboard will be awarded extra participation points, which counts as extra credit towards your final grade.

- 1st place: +5 EC
- 2nd place: +4 EC
- 3rd place: +3 EC
- 4th place: +2 EC
- All others who do better than the median: +1 EC

Part 2: Handling bounding boxes

In part 3 below, you'll use template matching to do some basic object detection. A detected object will be denoted by its **bounding box**. A bounding box is a rectangle that encloses the object. To prepare for part 3, then, you need to first fill in the missing code in `bounding_boxes.py`. See the functions' type signatures and docstrings for more information on what you'll be implementing.

Part 3: Template Matching using OpenCV

Template Matching uses the correlation operator to "search" for a small template image within a larger scene. We'll use this to build our first object detector in `template_match.py`. You're given a scene image (`mario_small.jpg`) and a template image (`coin_small.png`). In this first pair of images, the template is exactly the same size and orientation as the coin objects in the scene which you're trying to find, so no scaling or rotation is necessary. Your task is to

1. understand the overall logic of the `template_match.py` file
2. implement the missing code in `template_match.py`
3. set parameters in `config.py` which will be passed as keyword arguments to `find_objects_by_template` on gradescope

See if you can find reasonable parameter values that will maximize true-positive detections, minimize false-positive detections, and run in a reasonable amount of time. Is there a tradeoff? Were you able to find one set of parameters that work for both images? Do the same parameters work regardless of which "mode" argument you pass to `cv.matchTemplate()`? No need to respond, but you're encouraged to experiment and see what you can come up with.

Figure 1: Here's an example output using `visualize_matches` with the `mario_small.jpg` and `coin_small.png` images. A small red box has been drawn around each detected object, and a total count has been displayed in the bottom left corner.

If you look at the files carefully, you'll notice that the `coin_small.png` image has exactly the correct size and orientation to match the coins in the `mario_small.jpg` image. This is not a coincidence. Template matching is not by itself robust to scale or rotation changes. We need the image to "match" the template well enough for it to count as a match.

You're also given a second image, `mario_bigger.jpg` without a corresponding scaled-up template.

This is where **multi-scale template matching** comes in. The main logic of multi-scale matching is already implemented in `template_match.py`, and if you've already implemented the missing code blocks, you may have a functional multi-scale detector. However, there's still the question of which parameters to run it with; what values make sense for `scale_factor` and `levels` that will work **not just for `mario_bigger.jpg`, but for any other mario screenshot we might test your code with?**

The main thing still left for you to do is to choose values in `config.py` for `"other_images"` that

1. are not too different from the threshold values for `mario_small.jpg`
2. have sensible `scale_factor` and `levels` parameters that work for `mario_bigger.jpg`
3. …and which work for some other secret mario images that we might test your code with.

Collaboration and Generative AI disclosure and Assignment Reflection

Did you collaborate with anyone? Did you use any Generative AI tools? How did the assignment go? Briefly explain what you did in the `collaboration-disclosure.txt` file. **Completing the collaboration disclosure for each assignment is required.** Completing the reflection part of the file is optional, but it is a good way to give feedback to the instructor and grader about how the course is going.

Submitting

As described above, use `make submission.zip` to generate a zip file that you can upload to Gradescope. You can upload as many times as you like before the deadline. To account for late penalties, contact the instructor or grader directly if you plan on uploading any further revisions after the deadline.