Assignment 05

Please refer to MyCourses/Gradescope for assignment deadlines.

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

- Canny edge detection
- Hough transform for line detection
- Hough transform for circle detection

There is a lot here, so each part has a large portion done in the provided code. There are just a few key parts for you to implement.

Part 1 of 3: Canny edge detection

As we saw in class, the main steps of the Canny algorithm are:

1. Conversion to a single-channel image (usually just grayscale)
2. Gaussian blurring
3. Gradient calculation with the Sobel operator
4. Conversion to magnitude and direction
5. Direction-dependent non-maximal suppression
6. Double thresholding to find strong, weak, and non-edges
7. Hysteresis: promoting weak edges to strong edges if they are connected to strong edges

In practice, you would generally use `cv.Canny()` to do this. If you look at the OpenCV documentation, you'll see that there are two overloaded signatures for `cv.Canny()`: one that takes in a single `uint8` image (the result of blurring in step 2), and one that takes in two `int16` images `dx` and `dy` (the result of the Sobel operators in step 3). The latter gives you a bit more control over how you compute the gradient.

In this part of the assignment, you will implement steps (2) and (3) above using a single derivative-of-Gaussians filter. You will then pass the results to the `dx` and `dy` arguments of `cv.Canny()` and confirm that it gives you the same result as if you had used `cv.Canny()` with the blurred single-channel image. (In practice, there will be a few minor differences in the outputs due again to our old friend, floating-point precision and rounding. Arguably, using the DoG filter is more accurate because it avoids the intermediate step of converting to `uint8` in between the blur operation and the gradient calculation.)

Instructions: Implement the missing code in `edge_detection.py`. Verify that you get the same (or nearly the same, considering possible rounding errors) whether using `my_canny_1` or `my_canny_2`.

The `bricks.jpg` file has been provided as a test image for this part of the assignment. Here's a visualization of what Part 1 of this assignment is asking you to do:

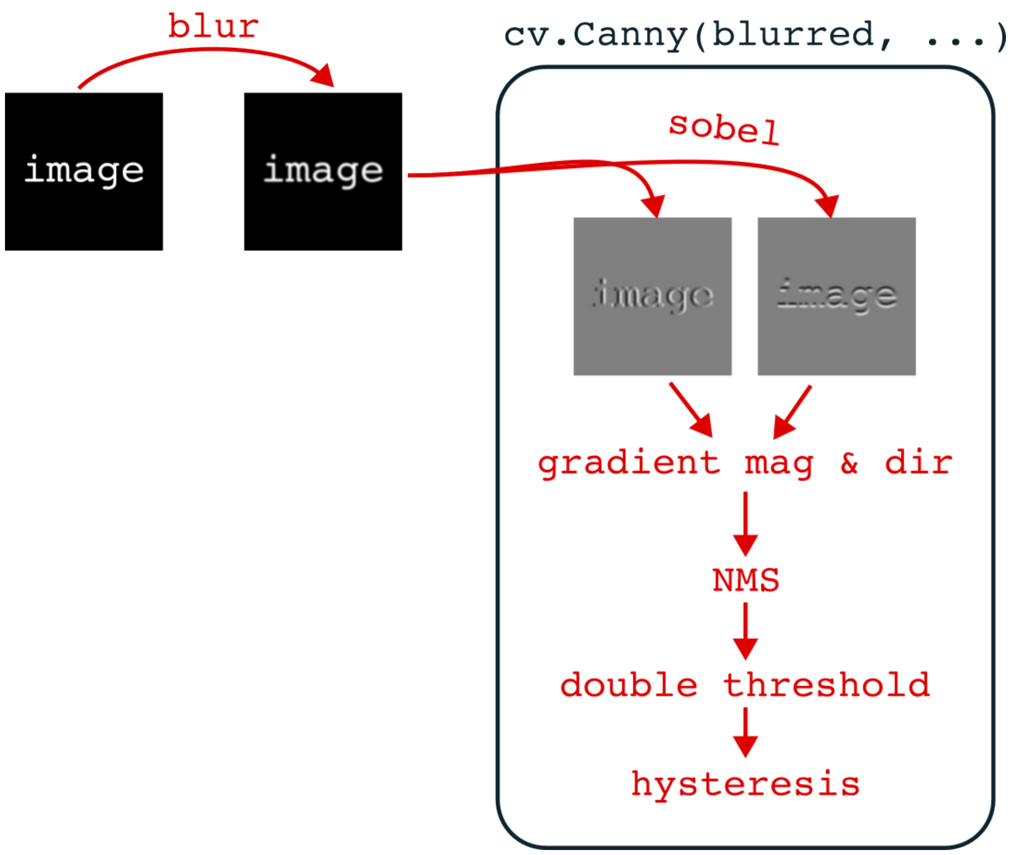

Figure 1: Visualization of the process for `my_edge_detect_1`

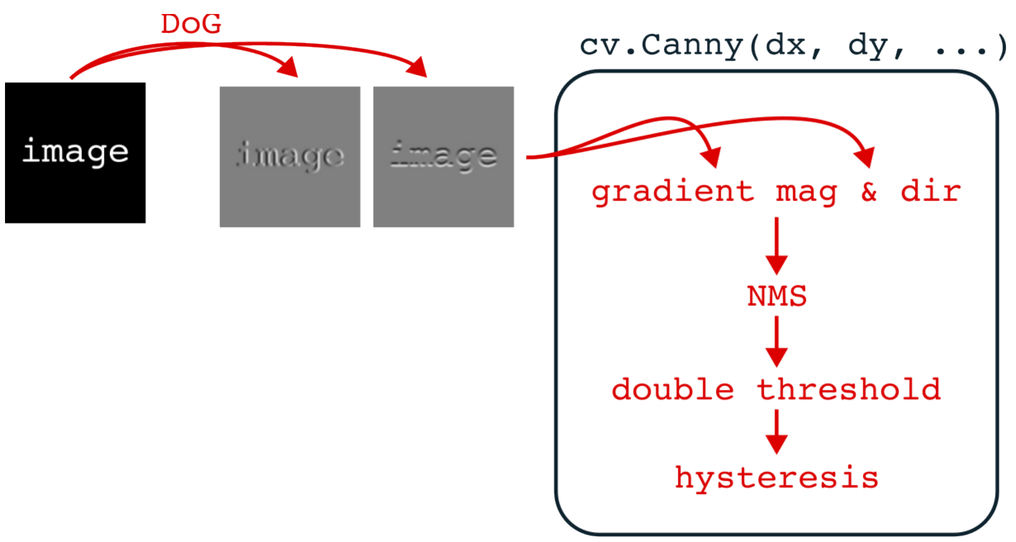

Figure 2: Visualization of the process for `my_edge_detect_2`

The key idea is that you must construct the DoG kernels such that you get the same behavior whether you call `cv.Canny(dx, dy)` or `cv.Canny(blurred)`. An important detail visualized here is that `cv.Canny(blurred)` is using Sobel filters internally. How can you construct a single DoG filter that replicates Gaussian blur followed by Sobel?

Part 2 of 3: Hough transform for line detection

The `hough_lines.py` file contains a large python class that shows the inner-workings of the Hough Line-Detection algorithm using primarily numpy operations. In practice, you would normally be using `cv.HoughLines()` or `cv.HoughLinesP()` to do this. But, this assignment is about understanding the underlying algorithm.

Instructions:

1. Implement the missing code in `hough_lines.py` according to the docstrings and comments.
2. Tune the parameters of the `main()` function such that your code detects as many of the **horizontal lines** in the `house.jpg` image as possible, without detecting any of the other lines. You will need to look carefully at the code and use your debugger to determine what the effect of changing each parameter is. Using the `config.py` file, record the parameters that you used.

Suggestions:

- Use your debugger and set breakpoints in `hough_lines.py`, but run the code using `run_hough.py` as the entry point. This will automatically apply the values in the `config.py` file to the house image and save out the result as a new image.

- Think carefully about what each parameter does. For example, if you're detecting too many edges in the tree region of the image, you might want to adjust the canny blur parameter. The output will also be sensitive to the size and resolution of the accumulator array and how this interacts with the chosen parameters for non-maximal suppression. Visualizing the accumulator array can be very helpful in understanding what's going on.

Debugging tip:

It's a good idea to try out the algorithm on a much simpler image and to visualize the accumulator array to make sure it's behaving as expected. For example, if we run line detection on this image:

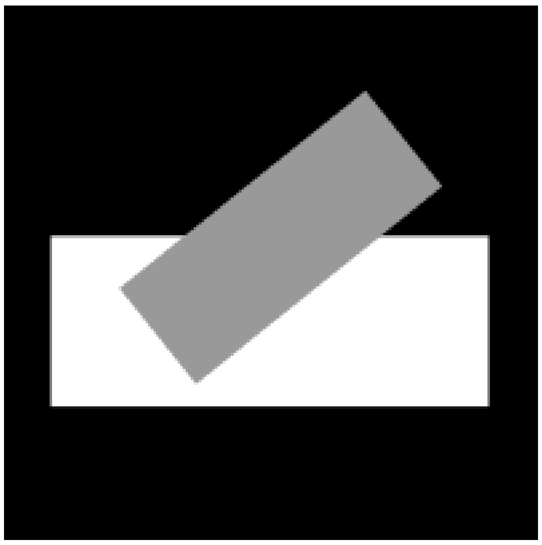

Figure 3: Some rectangles

then we expect the accumulator array to look something like this:

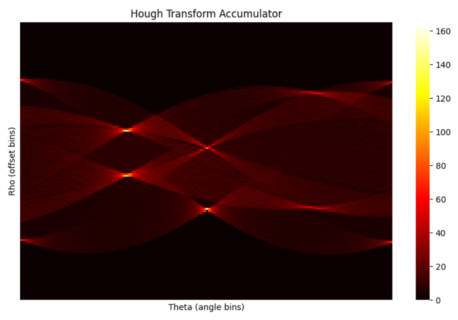

Figure 4: Accumulator array for rectangles

(Can you see how the peaks in the accumulator correspond to the lines in the image?)

Expected Output:

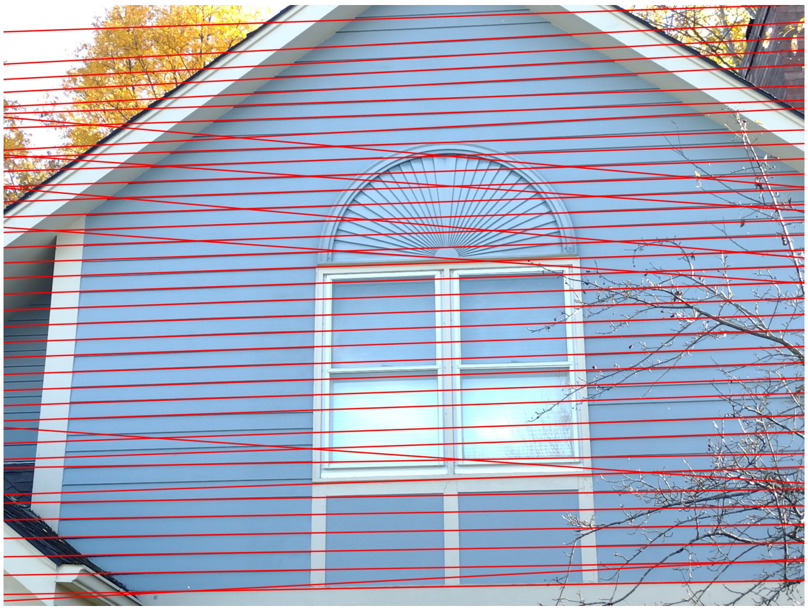

Figure 5: Example output having tuned the parameters for horizontal-line detection

Part 3 of 3: Hough transform for circle detection

Just like in Part 2, you will be doing a small amount of coding and a lot of parameter tuning.

Instructions:

1. Implement the missing code in `hough_circles.py` according to the docstrings and comments.
2. Tune the parameters of the `main()` function separately for each of the 4 types of coins. As before, use your debugger to better understand the inner-workings of the circle detector and use the `config.py` file to record the parameters that you used.

Implementation Note: Unlike the line detector, the `hough_circles.py` file uses a 'soft voting' scheme which allows edges to cast a fractional vote for circles which are almost but not quite matching. Circle detection (with a fixed radius) is in some ways more intuitive than line detection, since both the edge coordinates and the parameter space are simple xy-coordinates. Because of these two factors, it's surprisingly straightforward to implement the accumulator update using the *impulse response* of a blurry circular kernel respinding to the binary edge image. This may not be a standard approach to implementing the Hough Transform, but it is fast and effective for this assignment, and builds a nice bridge to the previous assignment on correlation and convolution.

Suggestions: While it's unavoidable that you'll need to adjust the 'radius' parameter for each coin in `config.py`, see if you can find values for all the other parameters that work for all coins. This isn't strictly necessary, but it's nice for your sanity (and ours) if the logic is as repeatable as possible.

Debugging tip:

It's a good idea to try out the algorithm on a much simpler image and to visualize the accumulator array to make sure it's behaving as expected. For example, if we run circle detection on this image:

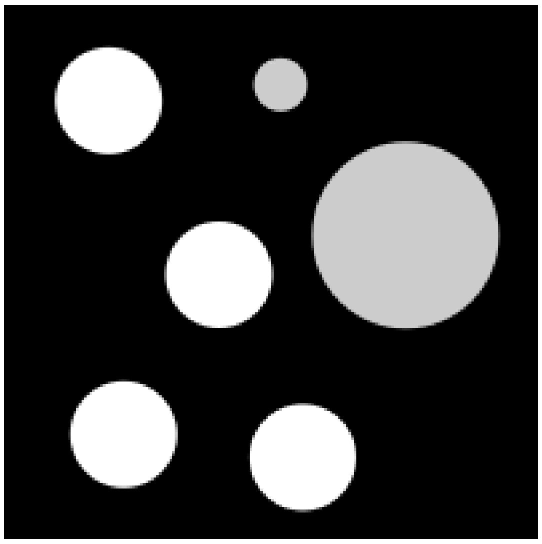

Figure 6: Some rectangles

with a radius of 20px, then we expect the accumulator array to look something like this:

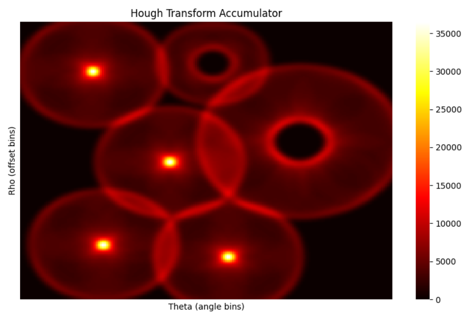

Figure 7: Accumulator array for rectangles

(Can you see how the peaks in the accumulator correspond to the mid-sized circles in the image?)

Example Outputs:

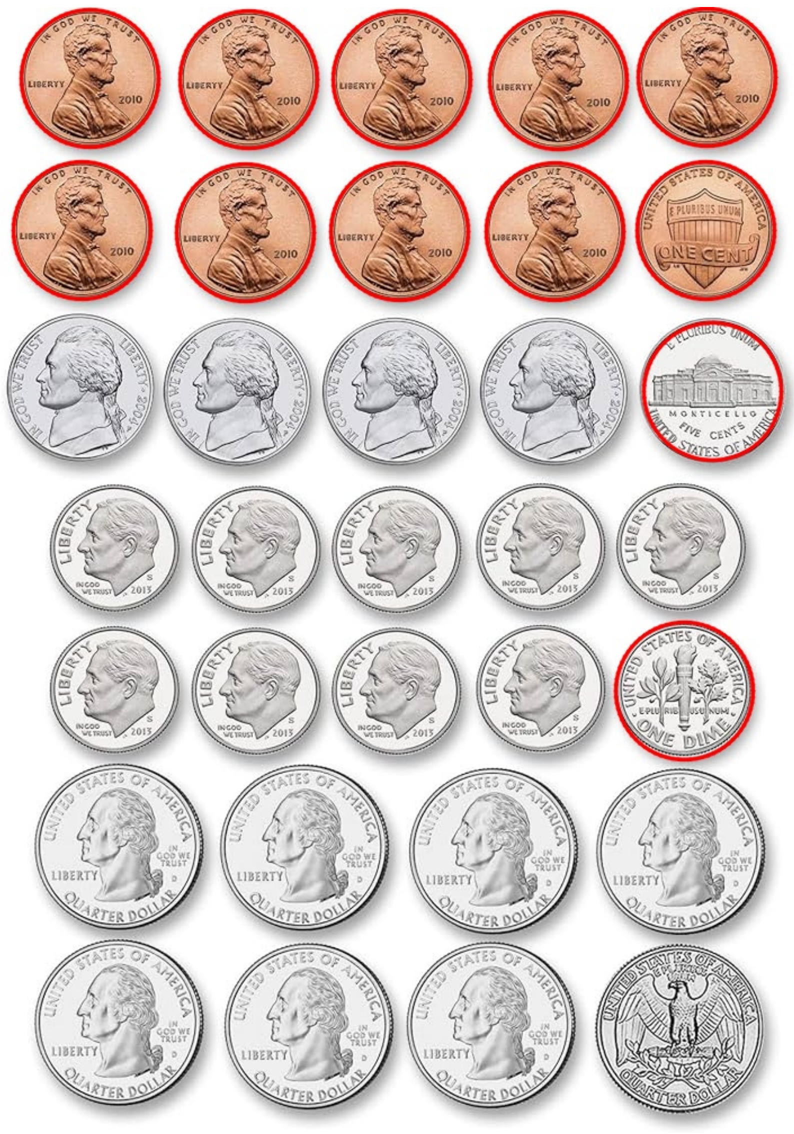

Figure 8: Example output having tuned the parameters for pennies

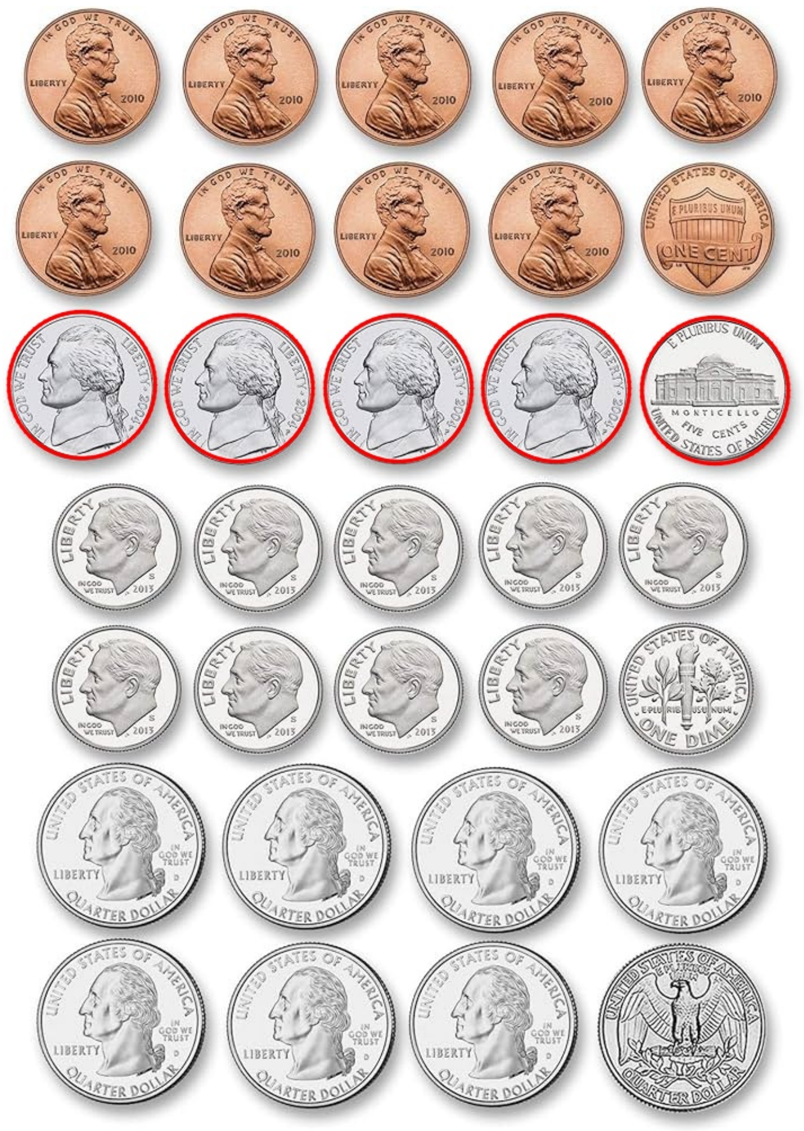

Figure 9: Example output having tuned the parameters for nickels

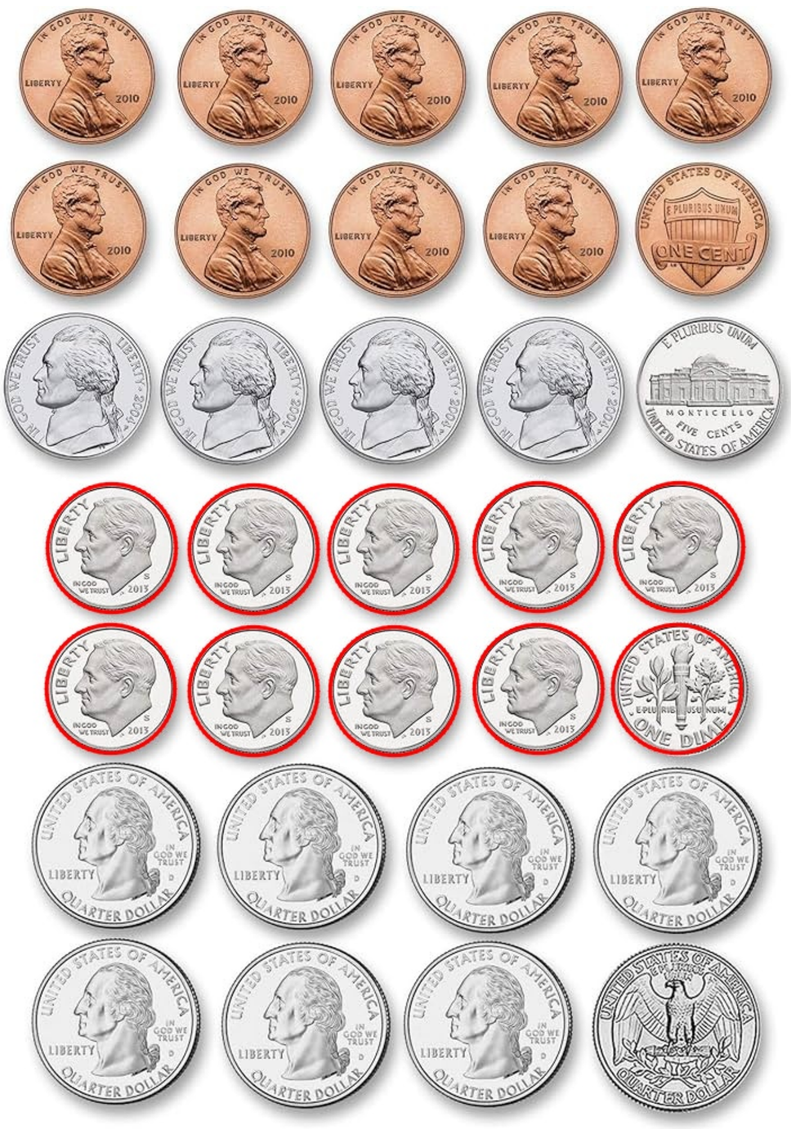

Figure 10: Example output having tuned the parameters for dimes

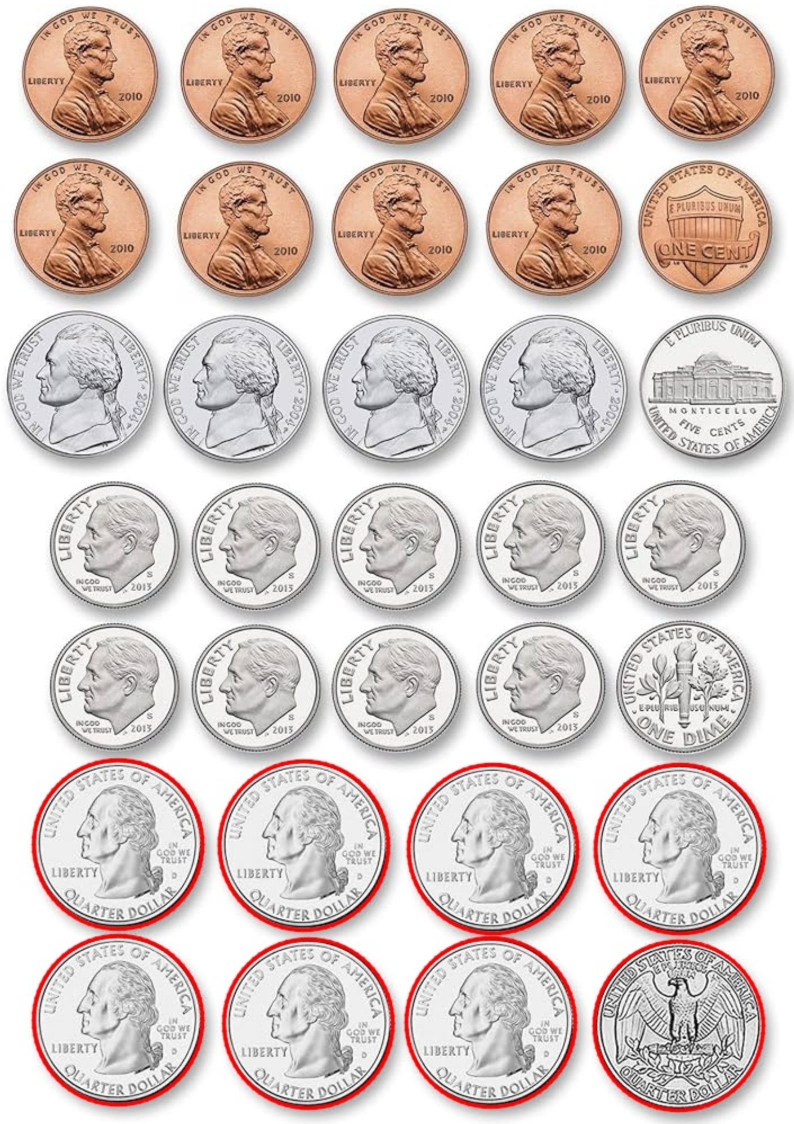
Figure 11: Example output having tuned the parameters for quarters

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.