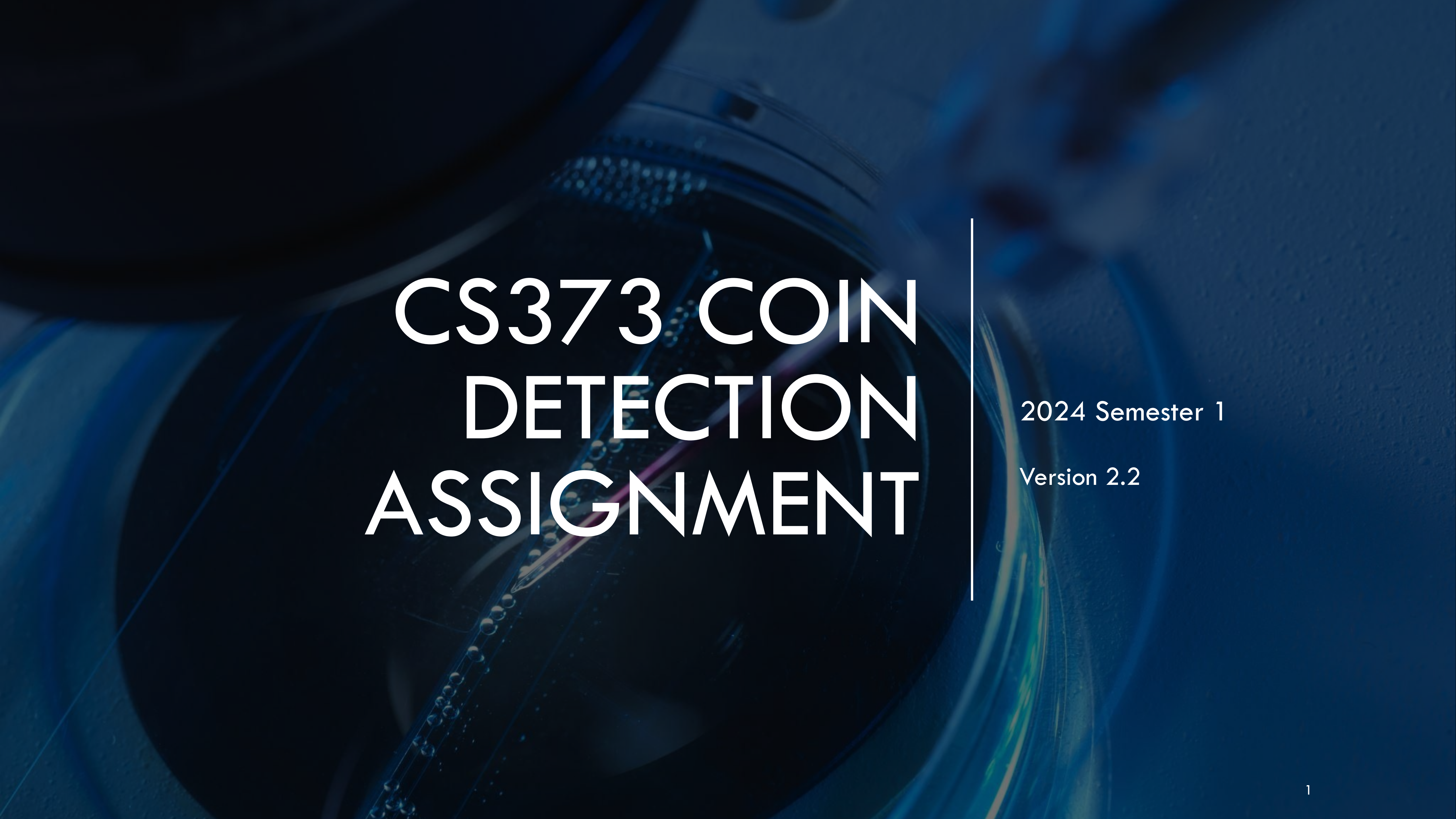


CS373 COIN DETECTION ASSIGNMENT

2024 Semester 1

Version 2.2

Deadline: 3rd June 2024, 23:59pm

- In this assignment, you will write a Python code pipeline to automatically detect all the coins in the given images. This is an individual assignment, so every student has to submit this assignment! This assignment is worth 15 marks.

- We have provided you with 6 images for testing your pipeline (you can find the images in the 'Images/easy' folder).
  - Your pipeline should be able to detect all the coins in the image labelled with easy-level. This will reward you with up to 10 marks.
  - For extension (up to 5 marks), try images labelled as hard-level images in the "Images/hard" folder.
  - Write a short reflective report about your extension. (Using Latex/Word)

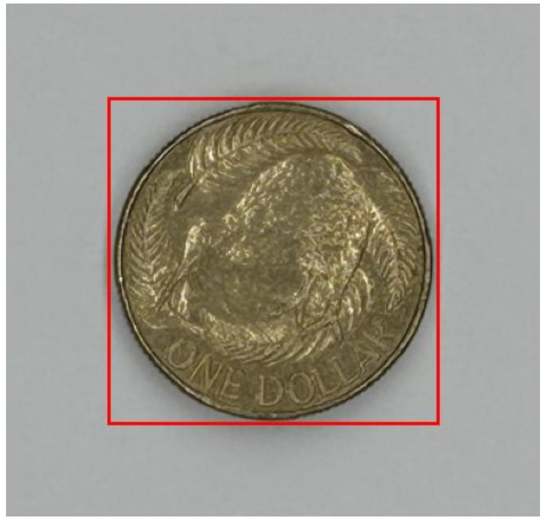
- To output the images shown on the slides for checking, you may use the following code:

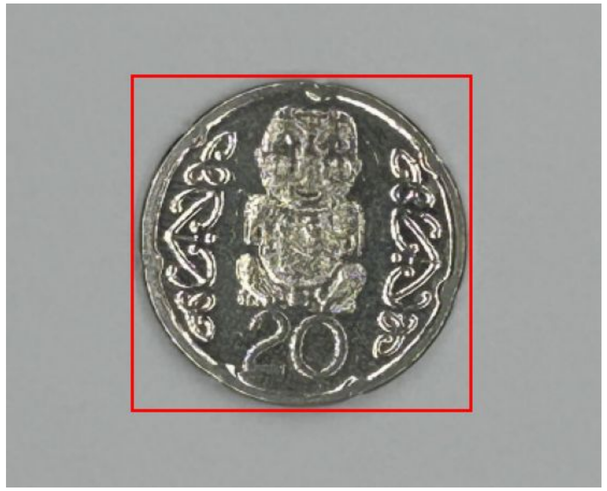
```
fig, axs = pyplot.subplots(1, 1)

# replace image with your image that you want to output

axs.imshow(image, cmap='gray')

pyplot.axis('off')

pyplot.tight_layout()

pyplot.show()
```

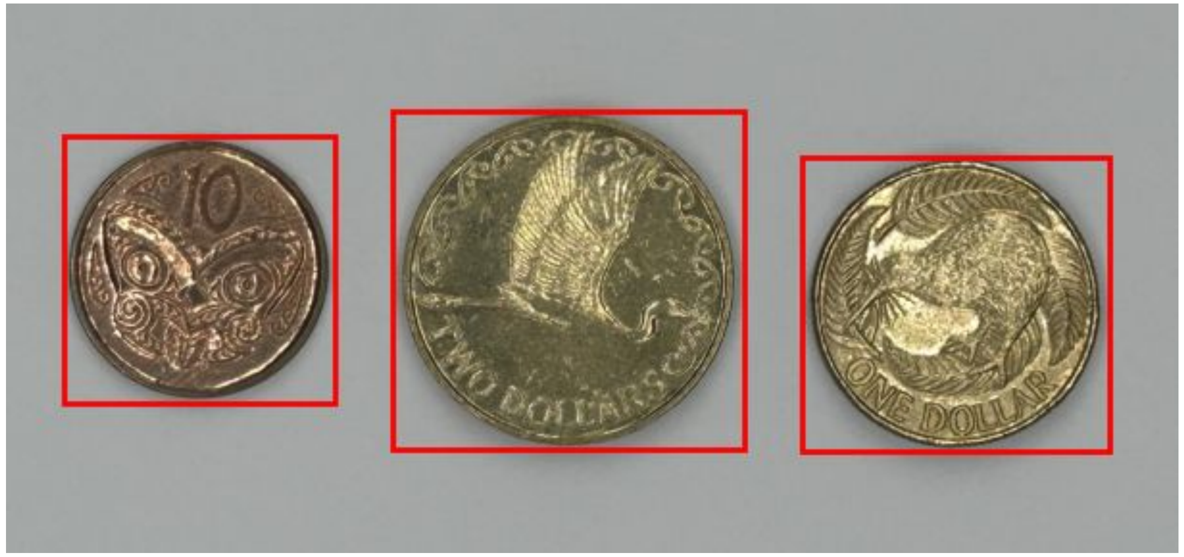
SUBMISSION

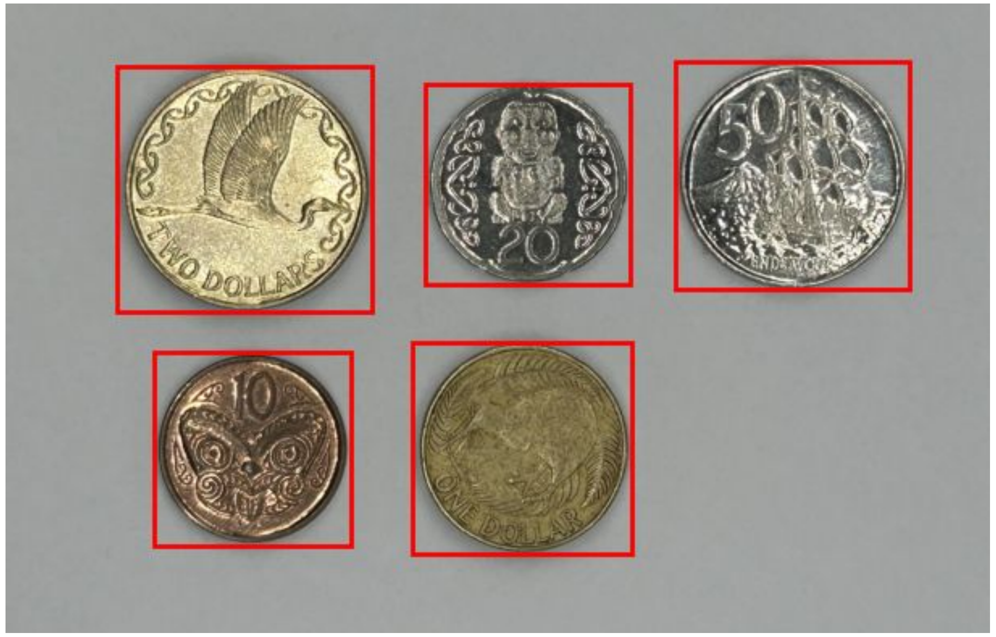
Please upload your submission as a zipped file of the assignment folder to the UoA Assignment Dropbox by following this link: https://canvas.auckland.ac.nz/courses/103807/assignments/383790

- Don't put any virtual environment (venv) folders into this zip file, it just adds to the size, and we will have our own testing environment.
- Your code for executing the main coin detection algorithm has to be located in the provided "CS373_coin_detection.py" file!
- You can either put all of your code into that file, or use a modular structure with additional files (that, of course, have to be submitted in the zip file). However, we will only execute the "CS373_coin_detection.py" file to see if your code works for the main component!
- The main component of the assignment ("CS373_coin_detection.py") must not use any non-built-in Python packages (e.g., PIL, OpenCV, NumPy, etc.) except for Matplotlib. Ensure your IDE hasn't added any of these packages to your imports.
- For the extensions, please create a new Python source file called '*CS373_coin_detection_extension.py*'; this will ensure your extension part doesn't mix up with the main component of the assignment. Remember, your algorithm has to pass the main component first!
- Including a short PDF report about your extension.
- Important: Use a lab computer to test if your code works on Windows on a different machine (There are over 300 students, we cannot debug code for you if it doesn't work!)


easy_case_1 final output


easy_case_2 final output


easy_case_4 final output


easy_case_6 final output

ASSIGNMENT STEPS

1. **Convert to greyscale and normalize**
   I. Convert to grey scale image: read input image using the '*readRGBImageToSeparatePixelArrays()*' helper function. Convert the RGB image to greyscale (use RGB channel ratio 0.3 x red, 0.6 x green, 0.1 x blue), and round the pixel values to the nearest integer value.
   II. Contrast Stretching: stretch the values between 0 to 255 (using the 5-95 percentile strategy) as described on lecture slides *ImagesAndHistograms*, p20-68). Do not round your 5% and 95% cumulative histogram values. Your output for this step should be the same as the image shown on Fig. 2.

Hint 1: see lecture slides *ImagesAndHistograms* and Coderunner Programming quiz in Week 10.
Hint 2: for our example image (Fig. 1), the 5_percentile (f_min) = 86 and the 95_percentile (f_max) = 173.

We will use this image ('easy_case_1') as an example on this slides

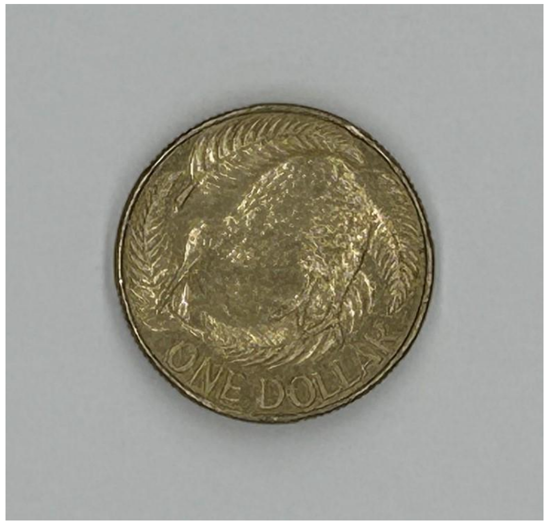


Fig. 1: input

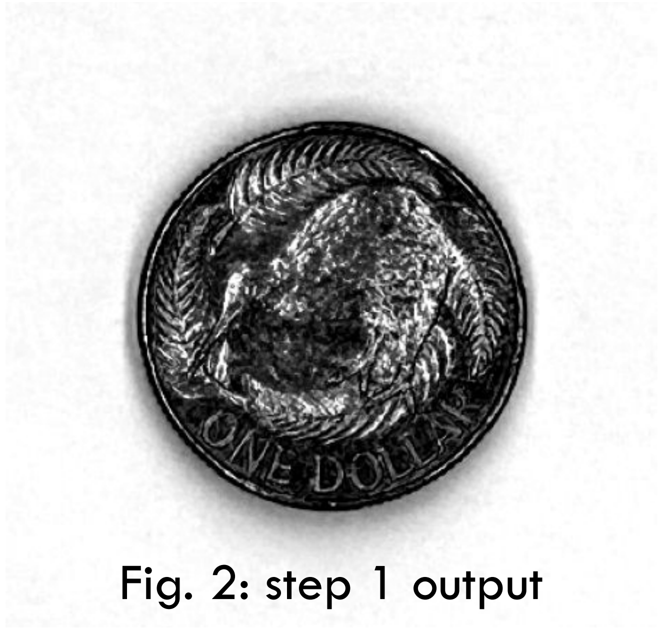


Fig. 2: step 1 output

2. Edge Detection

I. Apply a 3x3 Scharr filter in horizontal (x) and vertical (y) directions independently to get the edge maps (see Fig. 3 and Fig. 4), you should store the computed value for each individual pixel as Python *float*.

II. Take the absolute value of the sum between horizontal (x) and vertical (y) direction edge maps (see Hint 4). You do not need to round the numbers. The output for this step should be the same as the image shown on Fig. 5.

Hint 1: see lecture slides on edge detection and Coderunner Programming quiz in Week 11.

Hint 2: please use the 3x3 Scharr filter shown below for this assignment:

$$\frac{1}{32}\begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix} \qquad \frac{1}{32}\begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$$

Horizontal Edge (g_x) Vertical Edge (g_y)

Hint 4: you should use the *BorderIgnore* option and set border pixels to zero in output, as stated on the slide *Filtering*, p13.

Hint 5: for computing the edge strength, you may use the following equation:

$$g_m(x, y) = |g_x(x, y)| + |g_y(x, y)|$$

Edge map on horizontal and vertical

Absolute grey level gradient on the horizontal direction

Absolute grey level gradient on the vertical direction

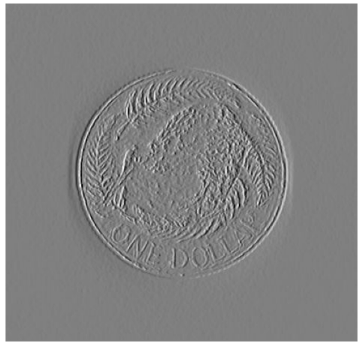


Fig. 3: Edge map (g_x) on horizontal direction

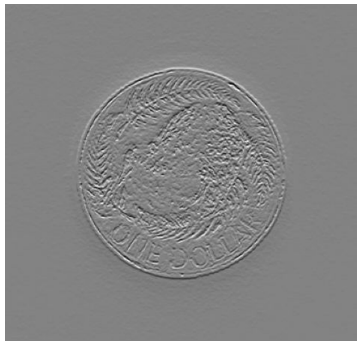


Fig. 4: Edge map (g_y) on vertical direction

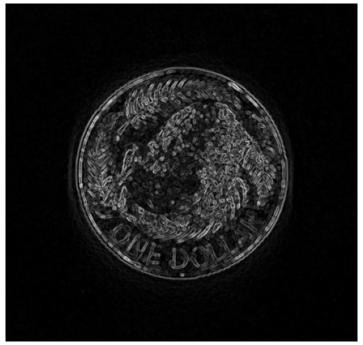


Fig. 5: Step 2 output (g_m)

3. Image Blurring

Apply 5x5 mean filter(s) to image. Your output for this step should be the same as the image shown on Fig. 7.

Hint 1: do not round your output values.
Hint 2: after computing the mean filter for one 5x5 window, you should take the absolute value of your result before moving to the next window.
Hint 3: you should use the *BorderIgnore* option and set border pixels to zero in output, as stated on the slide *Filtering*, p13.
Hint 3: try applying the filter three times to the image sequentially.
Hint 4: see lecture slides on image filtering and Coderunner Programming quiz in Week 11.

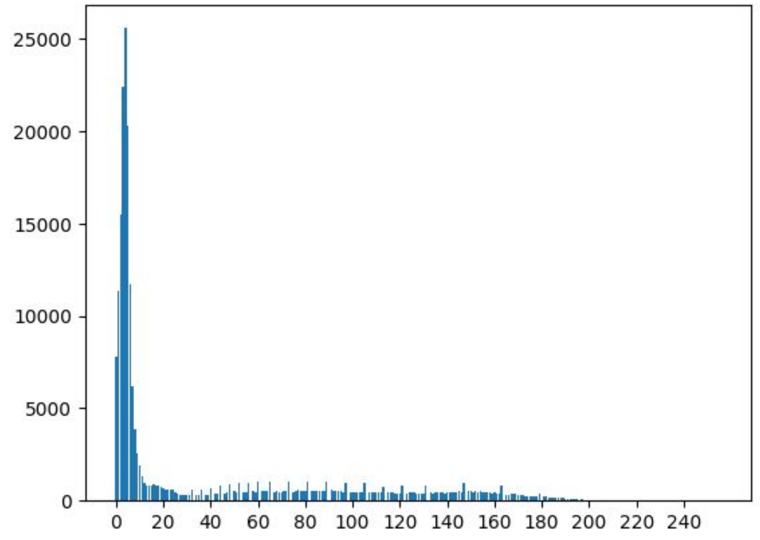


Fig. 6: Grayscale histogram for output from step 3

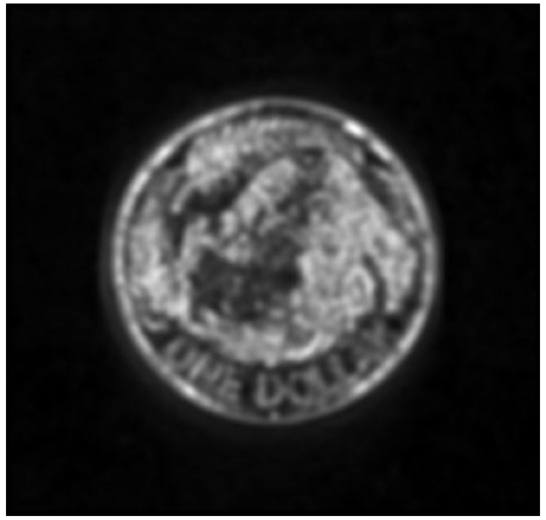


Fig. 7: Step 3 output

4. Threshold the Image

Perform a simple thresholding operation to segment the coin(s) from the black background. After performing this step, you should have a binary image (see Fig. 10).

Hint 1: 22 would be a reasonable value for thresholding for our example image, set any pixel value smaller than 22 to 0; this represents your background (region 1) in the image, and set any pixel value bigger or equal to 22 to 255; which represents your foreground (region 2) – the coin.
Hint 2: see lecture slides on image segmentation (p7) and see Programming quiz on Coderunner on Week 10.

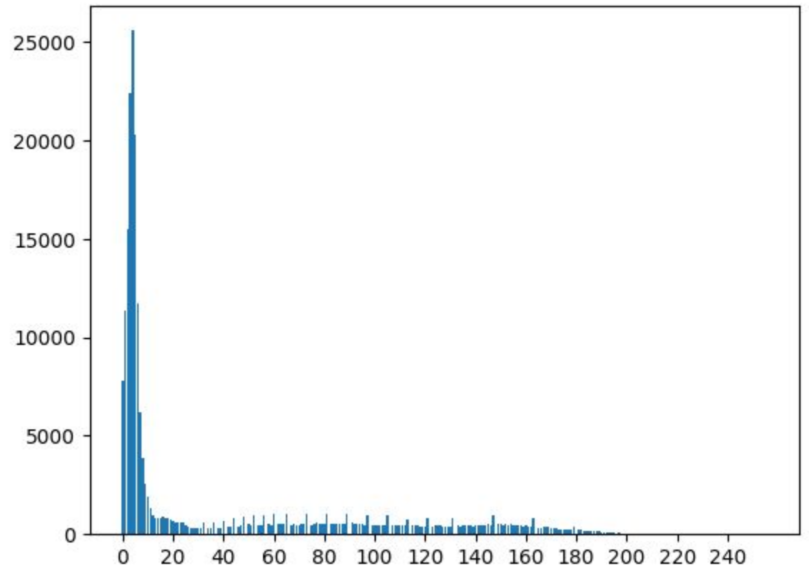


Fig. 8: Grayscale histogram for output from step 3

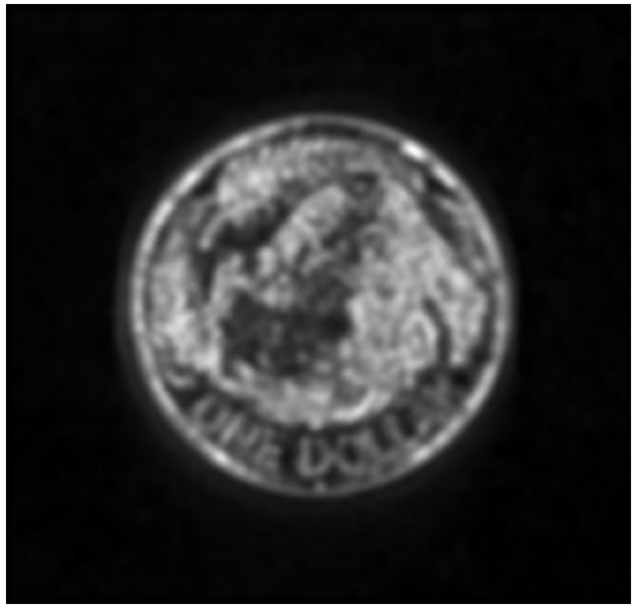


Fig. 9: Step 3 output

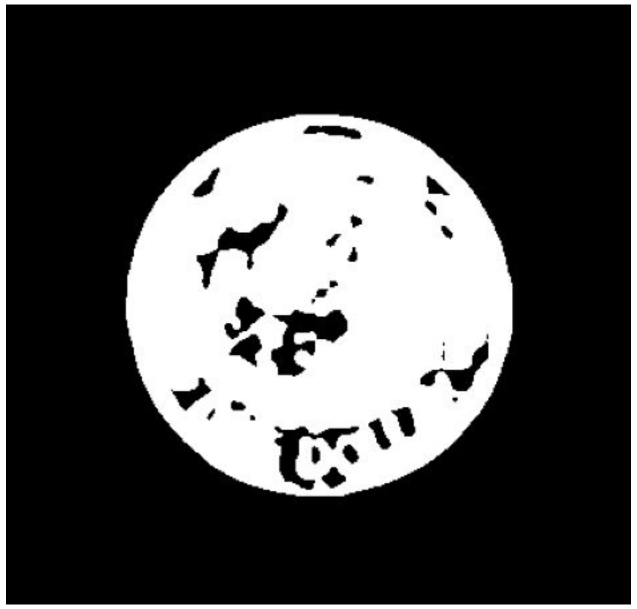


Fig. 10: Step 4 output

5. Erosion and Dilation

Perform several dilation steps followed by several erosion steps. You may need to repeat the dilation and erosion steps multiple times. Your output for this step should be the same as the image shown on Fig. 11.

Hint 1: use circular 5x5 kernel, see Fig. 12 for the kernel details.
Hint 2: the filtering process has to access pixels that are outside the input image. So, please use the *BoundaryZeroPadding* option, see lecture slides *Filtering*, p13.
Hint 2: try to perform dilation 3-4 times first, and then erosion 3-4 times. You may need to try a couple of times to get the desired output.
Hint 3: see lecture slides on image morphology and Coderunner Programming quiz in Week 12.

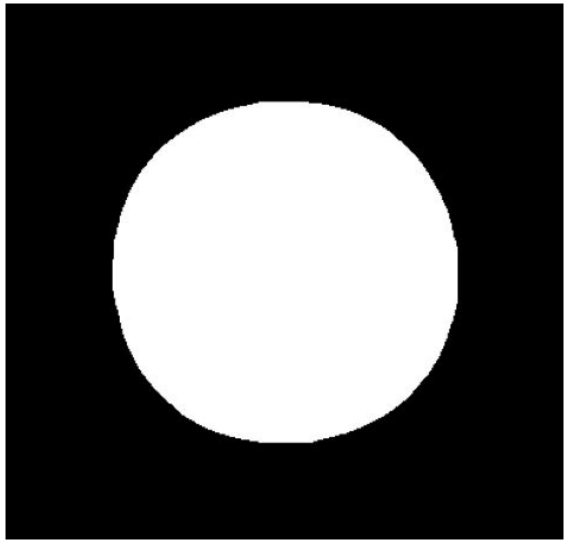
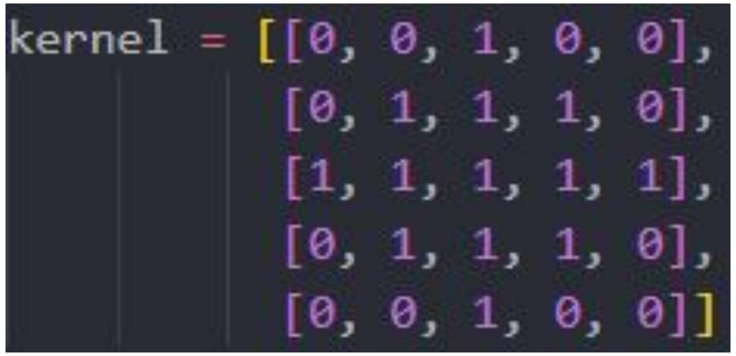


Fig. 11: Step 5 output



```
kernel = [[0, 0, 1, 0, 0],
          [0, 1, 1, 1, 0],
          [1, 1, 1, 1, 1],
          [0, 1, 1, 1, 0],
          [0, 0, 1, 0, 0]]
```

Fig. 12: Circular 5x5 kernel for dilation and erosion

6. Connected Component Analysis

Perform a connected component analysis to find **all** connected components. Your output for this step should be the same as the image shown on Fig. 13.

After erosion and dilation, you may find there are still some holes in the binary image. That is fine, as long as it is one connected component.

Hint 1: see lecture slides on *Segmentation_II*, p4-6, and Coderunner Programming quiz in Week 12.

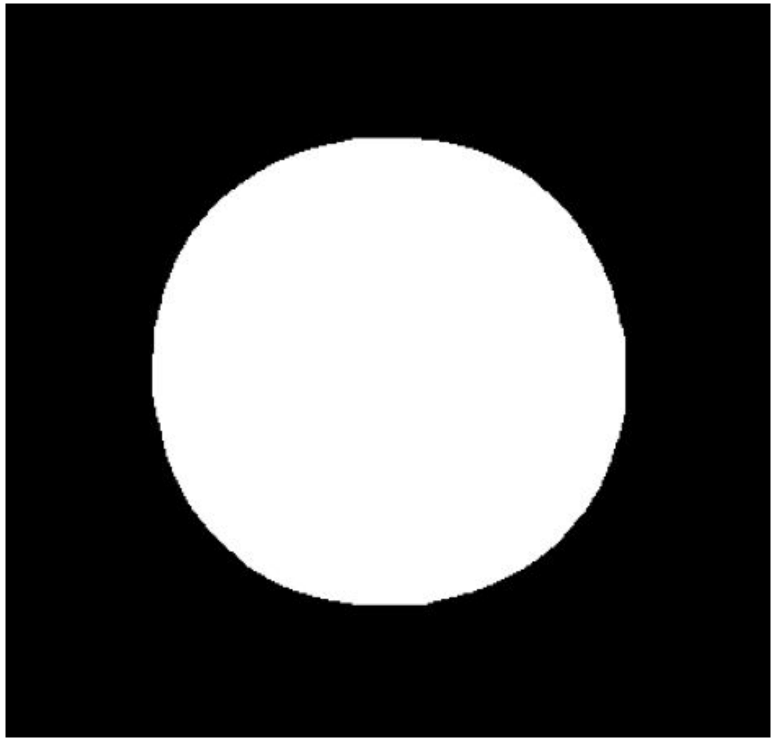


Fig. 13: Step 6 output

7. Draw Bounding Box

Extract the bounding box(es) around all regions that your pipeline has found by looping over the image and looking for the minimum and maximum x and y coordinates of the pixels in the previously determined connected components. Your output for this step should be the same as the image shown on Fig. 14.

Make sure you record the bounding box locations for each of the connected components your pipeline has found.

We will provide code for drawing the bounding box(es) in the image, so please store all the bounding box locations in a Python list called '*bounding_box_list*', so our program can loop through all the bounding boxes and draw them on the output image.

Below is an example of the '*bounding_box_list*' for our example image on the right.

```
Bounding_box_list=[[74, 68, 312, 303]]
```

A list of list

Bounding_box_min_x

Bounding_box_min_y

Bounding_box_max_x

Bounding_box_max_y

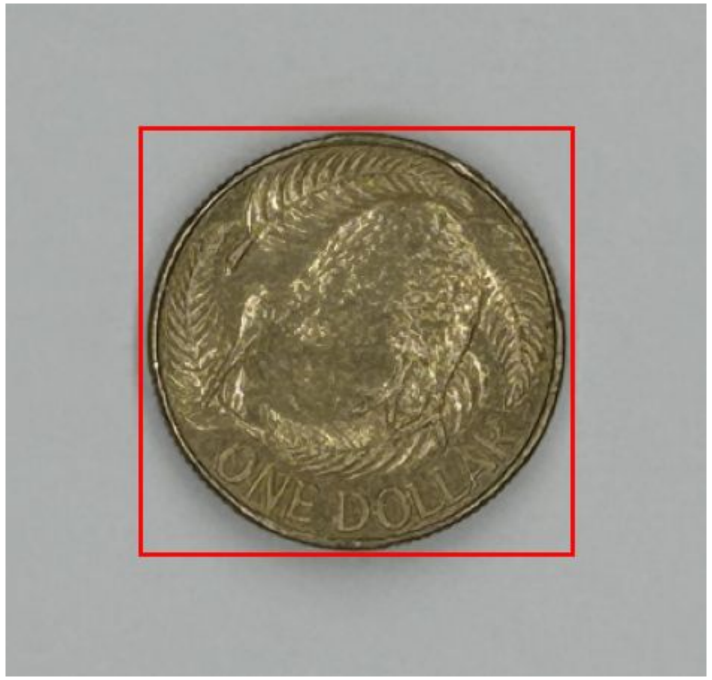
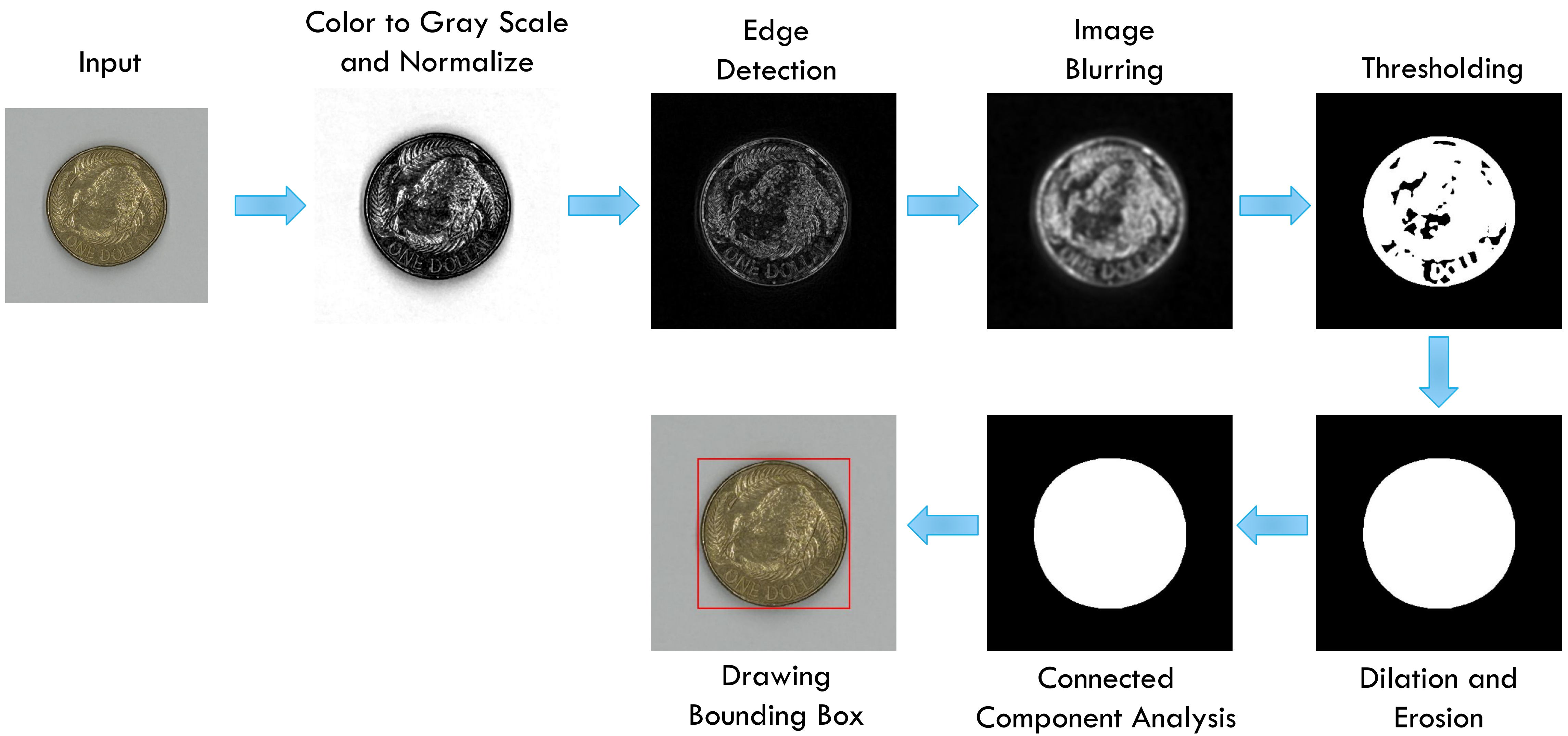
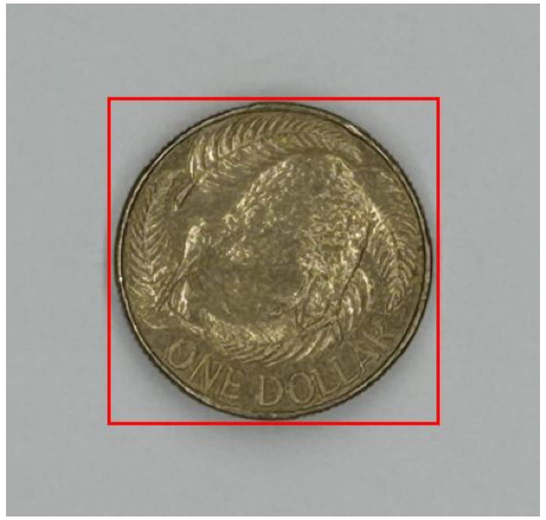


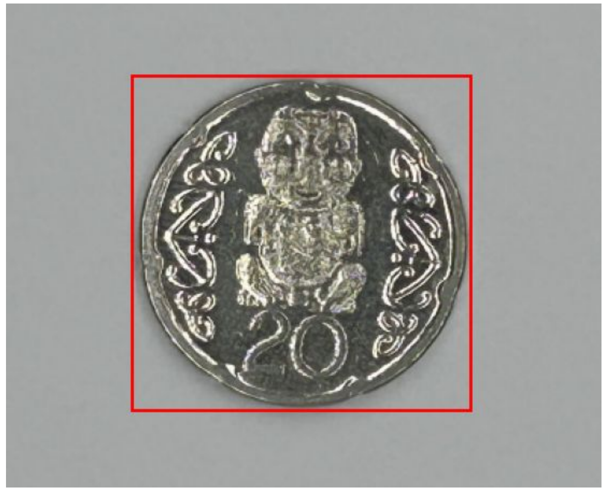
Fig. 14: Step 7 output

Coin Detection Full Pipeline

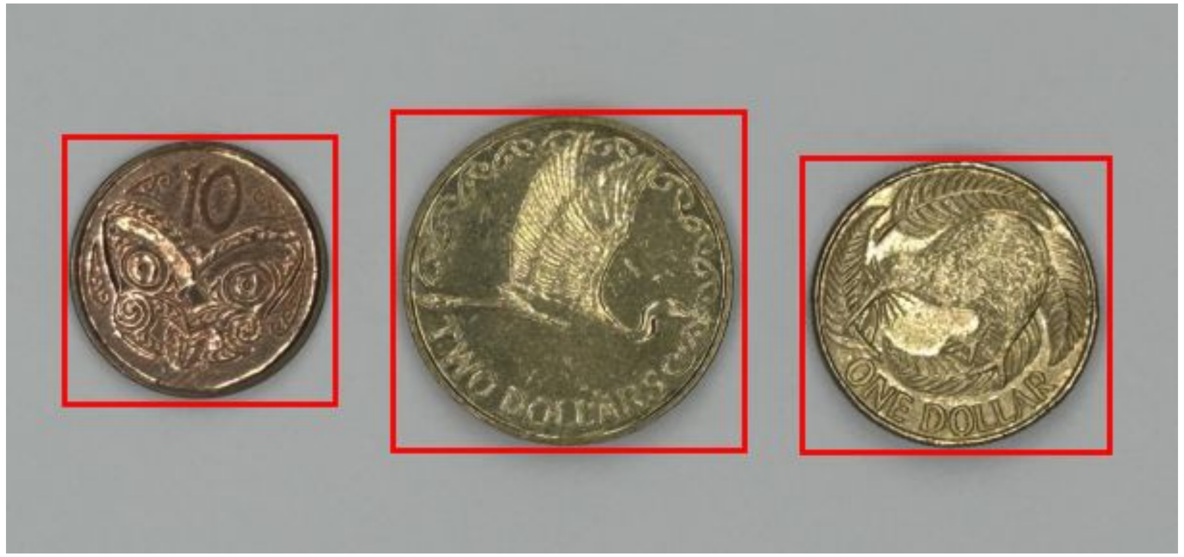




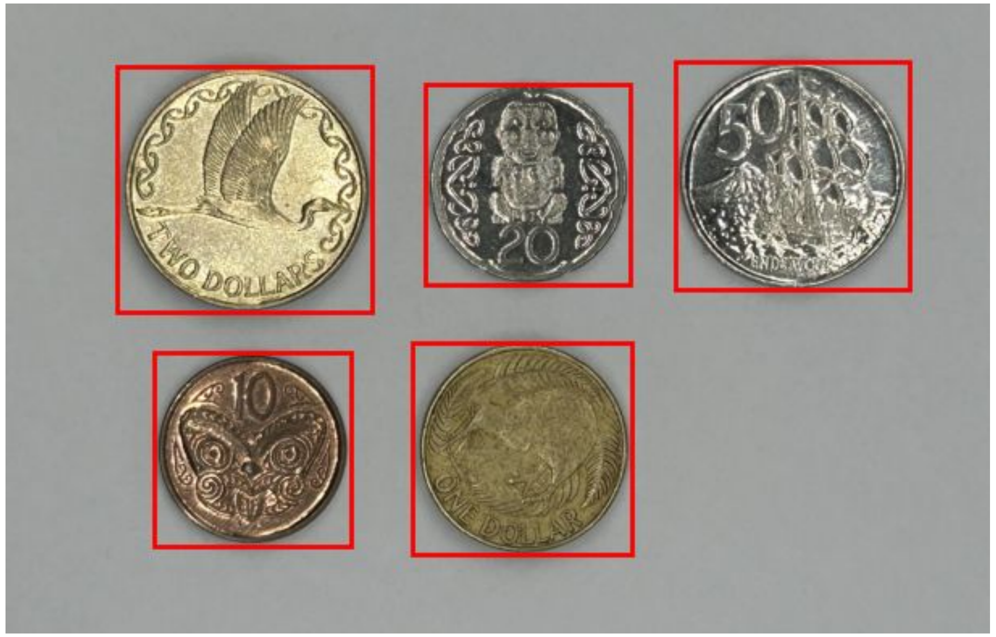
easy_case_1 final output



easy_case_2 final output



easy_case_4 final output

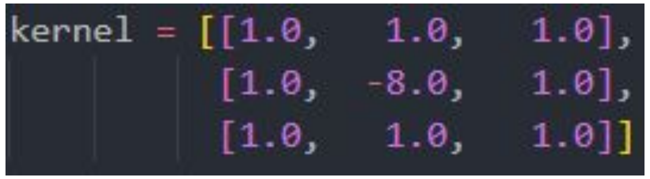


easy_case_6 final output

EXTENSION

For this extension (worth 5 marks), you are expected to alter some parts of the pipeline.

- Using Laplacian filter for image edge detection
  - Please use the Laplacian filter kernel on the right (see Fig. 15).
  - You may need to change subsequent steps as well, if you decide to use Laplacian filter.



```
kernel = [[1.0,  1.0,  1.0],
          [1.0, -8.0,  1.0],
          [1.0,  1.0,  1.0]]
```

Fig. 15: Laplacian filter kernel

- Output number of coins your pipeline has detected.

- Testing your pipeline on the hard-level images we provided.
  - For some hard-level images, you may need to look at the size of the connected components to decide which component is the coin.

- Identify the type of coins (whether it is a 1-dollar coin, 50-cent coin, etc.).
  - Since different type of coins have different sizes, you may want to compute the area of the bounding box in the image to identify them.

- etc.

Submissions that make the most impressive contributions will get full marks. Please create a new Python source file called ‘*CS373_coin_detection_extension.py*’ for your extension part, and include a short PDF report about your extension. Try to be creative!