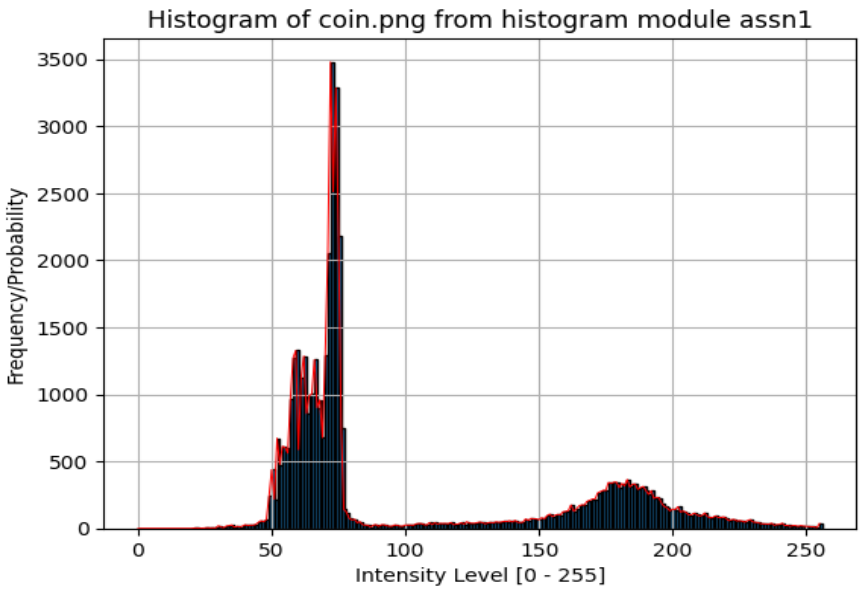
E9 241 Digtal Image Processing, Assignment 1

Kartikeyan Iyer (SR No. 25922)

1. Histogram



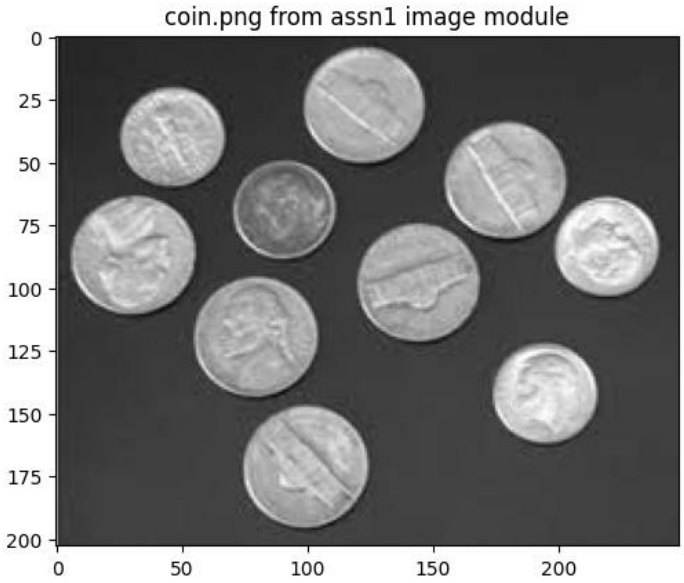
In this histogram of the image **coins.png** it is clear that there are dark and bright pixels, the dark pixels more than bright ones in number, which are present in clusters, hence Otsu's algorithm more than enough to find a decent enough threshold to binarize the image.

The average intensity of the image is **103.3050**
This was verified by both direct method by iterating thorugh pixels of the image and the histogram method or calculating the overall mean of normalized histogram.

The two values are within 10^{-6} neighbourhoods of each other, so we can consider it to be preceise.

2. Otsu's Binarization

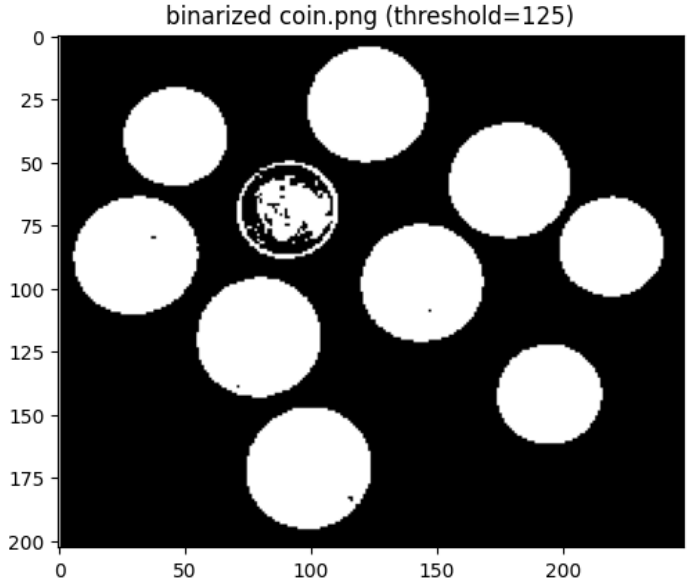
The original image of **coins.png** is:



Using Otsu's method for binarization,

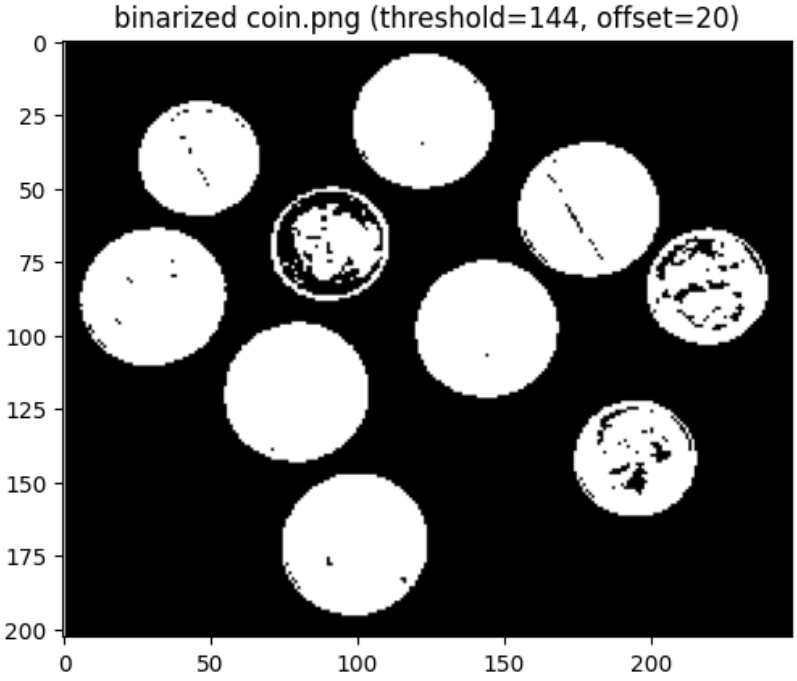
using within class variance $\sigma_w^2(t) = \omega_0 \cdot \sigma_0^2(t) + \omega_1 \cdot \sigma_1^2(t)$ and between class variance $\sigma_b^2(t) = \omega_0 \cdot \omega_1(\mu_0 - \mu_1)^2$ as the objective functions to optimize over the threshold t.

The threshold received for original image with minimizing within class variance was **125**.



After applying an offset of 20 and then using Otsu's algorithm with maximizing between class variance, that resulted in a threshold of **144** which is not the expected ***145*** (as increase in offset should directly relate to the increase in threshold, since it only depends on profile of histogram and is translation invariant), because some of the brighter pixels move towards and get clamped at 255 intensity level.

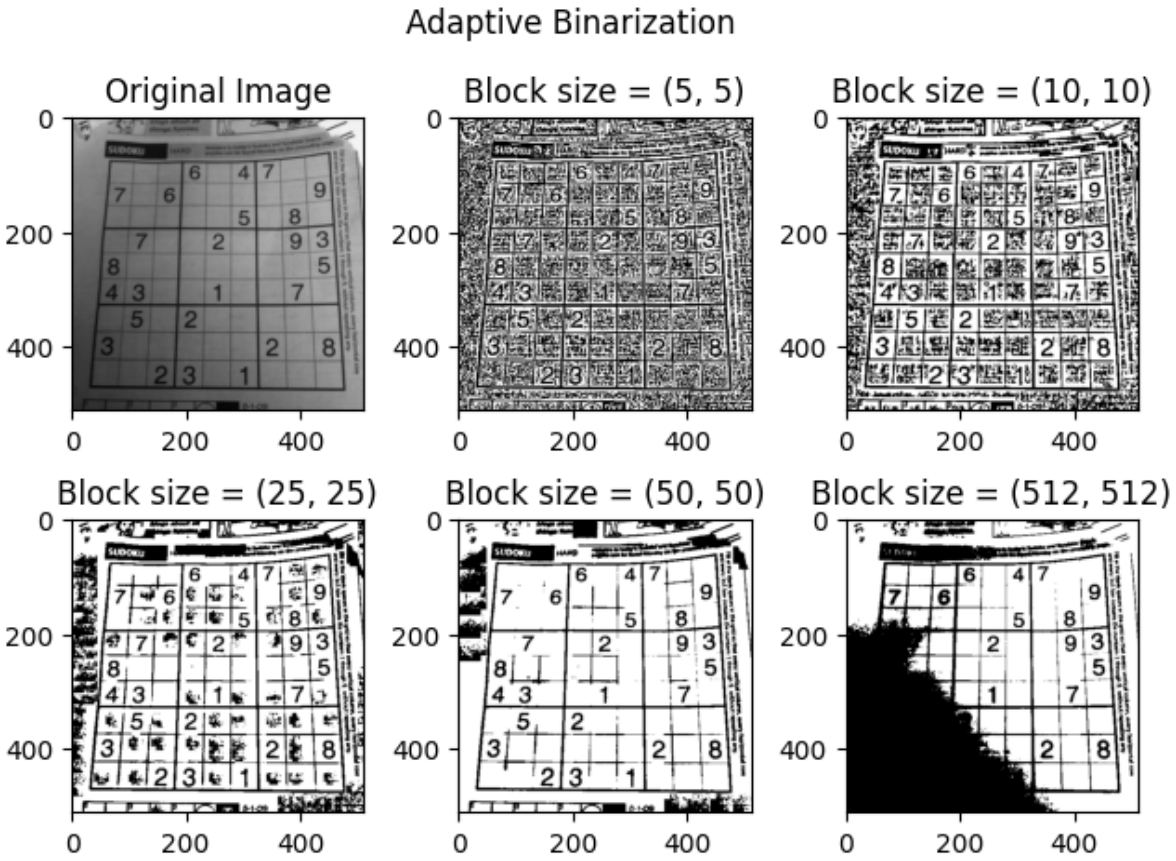
This causes a decrease in variance for the class 1 cluster since the pixels are now more squished together due to clamping, thereby giving more weight to the term of class 0 pixels in the within class variance, which causes the optimal threshold to be less than expected.



3. Adaptive Binarization

In our case here, block size which is small is more useful when we want to capture very small local variations, which larger block sizes average out or smoothen.

The comparison are as follows:



If bigger sizes are used, not all the details are captured due to overall averaging. Local details are missed. As block size decreases, things become more and more clearer until after getting too small, unnecessary noise is also captured, hence making smaller blocks sensitive to noise in the image.

The main point to highlight in the diagram is, that a full binarization (block size being the dimensions of image) will be very sensitive to shadows, so the binarization process blacks out half the image.

4. Connected Component Analysis (CCA)

In the 4 neighbour version, the algorithm is:

Algorithm 1.1:

- INPUT: Binary Image I is a binary image of size $M \times N$
- OUTPUT: A region matrix R of same size as I

1. Create a region matrix full of zeros R, and set region counter $k = 1$
2. Iterate through all the pixels of I
    - **a.** If $I(i,j) = 0$: **continue**
    - **b.** Else:
        - i. $top\_label = R(i-1, j)$
        - **ii.** $left\_label = R(i, j-1)$
        - **iii.** If both labels are 0: new region found, $R(i,j) = k; k++;$
        - **iv.** Else: find which of the non-zero labels to mark it with

Now in marking $R(i,j)$ if only one of the labels (top or left) is **non-zero**, then mark $R(i,j)$ as that. Otherwise, if both are non-zero and equal, then mark it as any one of those. But when they are not equal, then you have mark the two regions as equivalent.

Resolving Merges:

Here, we can simply maintain a dictionary. In the merging case, label $R(i,j)$ as the minimum of the two labels and merge the larger label region into minimum label region by doing $D[max] = min$.

In the second pass, we can basically search through the dictionary to find the root label corresponding to the label currently at $R(i,j)$ and update that.

This connected component labelling is exactly what a **Disjoint Set Union (DSU)** solves. So, we create a DSU to handle merges and in the first pass it just records all the merges and in the second it resolves them pixel by pixel.

The same idea is applied to 8-neighbour version where we look at the labels $R(i-1,j), R(i,j-1), R(i-1,j-1)\ and\ R(i,j+1)$. We basically find which of these are currently distinct regions, merge the regions corresponding to non-zero labels and resolve them in the second pass. Algo is pretty much the same except in case of a white pixel, we basically find the non-zero neighbours and merge them.

After applying this algo, the largest letter is the **N from "Cultivation"** of **546** white pixels.