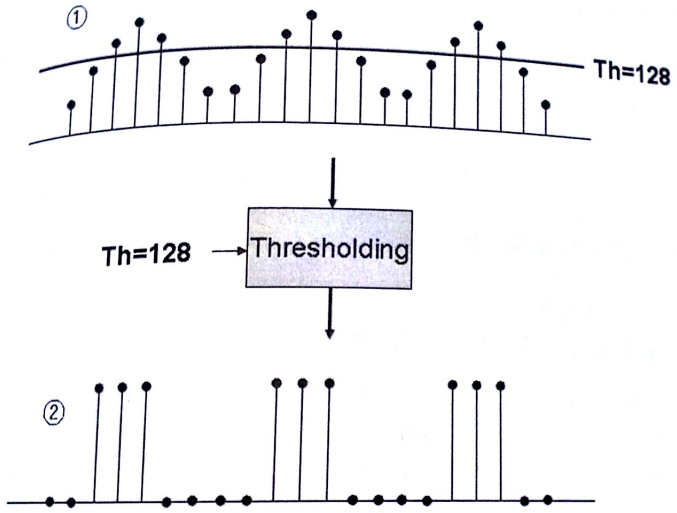
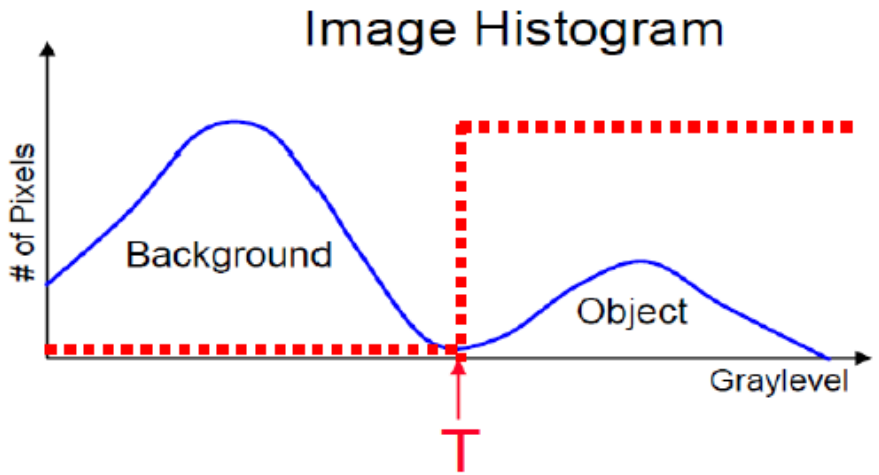
Thresholding and Otsu
Assignment 4

Gwenaelle Cunha Sergio
ABR Lab – BEP
KNU 2014.1

Threshold

- Signal quantization for when N=2
- Binarization
- Uses a threshold point (T)





Threshold

- Used to preprocess images
- Background vs object

- Static: T is constant in every image
- Dynamic: T can change according to the image or methods used

Static Binarization

$$g(x,y) = \begin{cases} 1 & f(x,y) > T \\ 0 & otherwise \end{cases}$$

<원본 영상>

T=20

T=50

T=100

T=170

T=220

Static Binarization - Code

```
int T[3] = {20, 100, 200};

for (int t = 0; t < 3; t++){
	for (int i = 0; i < mat.rows; i++){
		for (int j = 0; j < mat.cols; j++){
			if (mat.at<uchar>(i,j) > T[t]) {
				staticBin[t].at<uchar>(i,j) = 255;
			} else {
				staticBin[t].at<uchar>(i,j) = 0;
			}
		}
	}
}
```

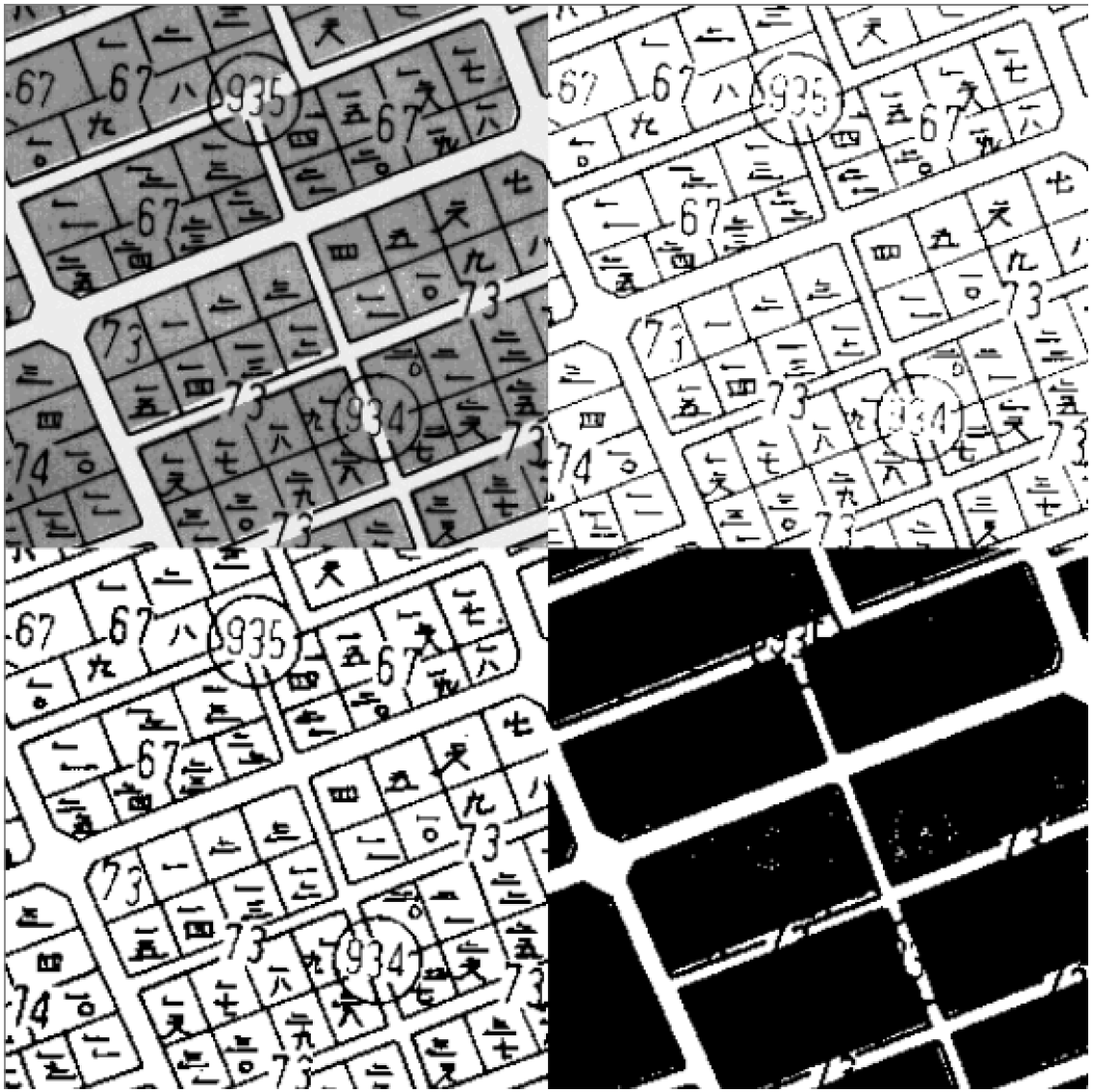
Static Binarization - Result

original

T = 20

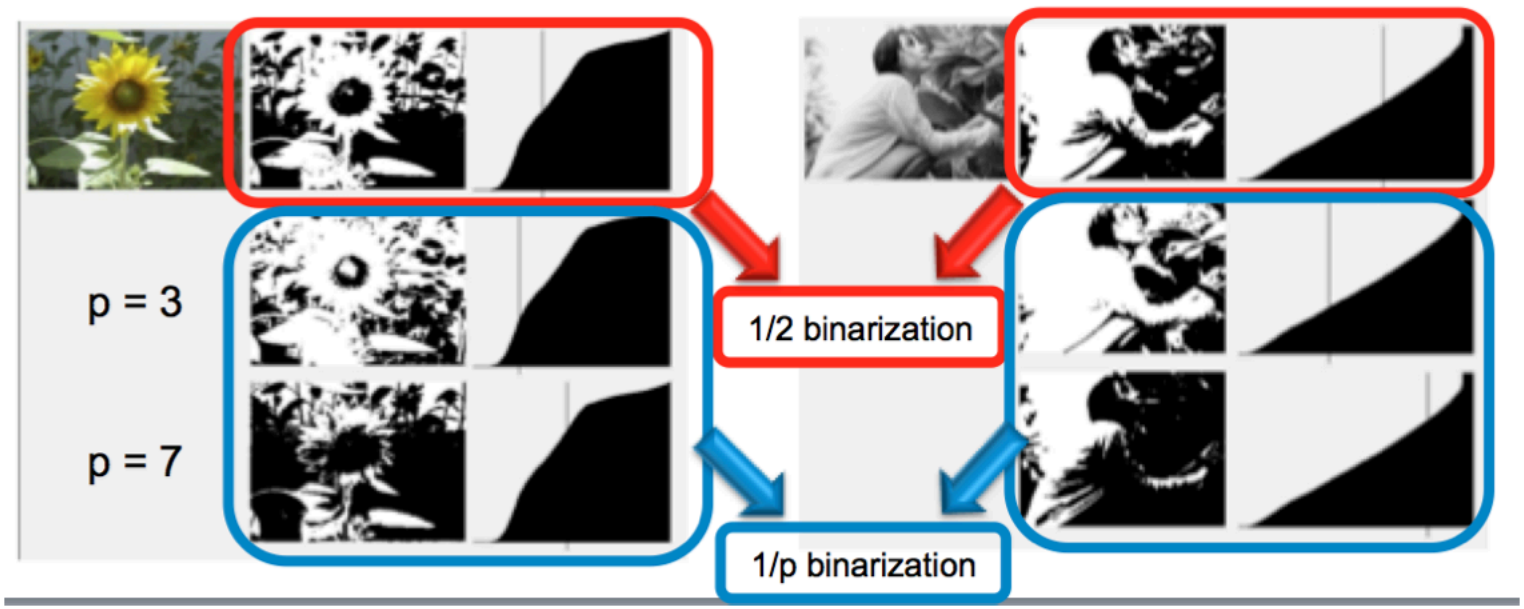
T = 100

T = 200



P-tile and ½ Binarization

- Get total number of pixels
- Divide by variable p to get the threshold
- When p = 2, T = pixels/2



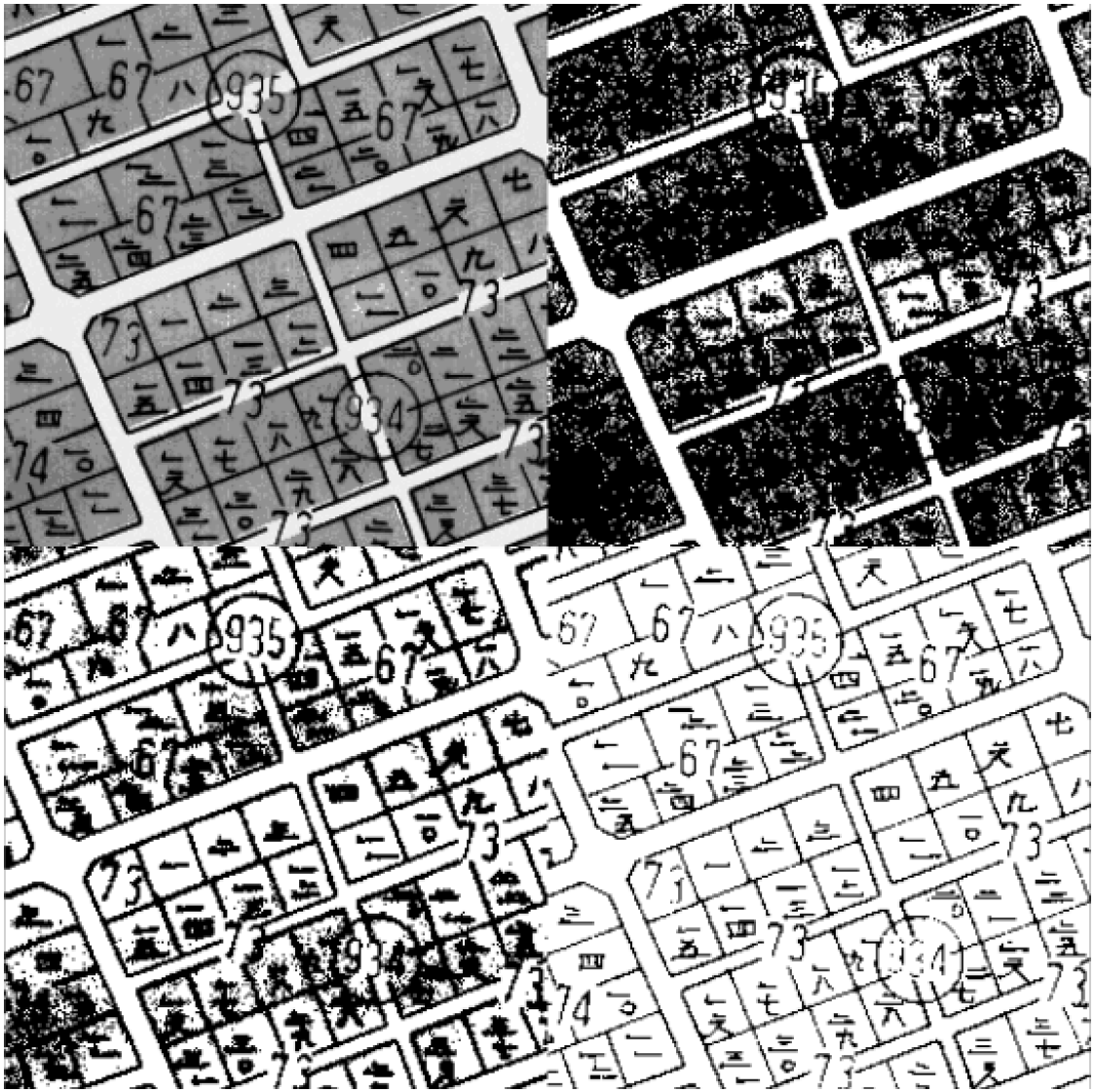
P-tile and ½ Binarization - Code

```
int p[3] = {2, 3, 7};
int pixels = mat.total();

int *histogram = new int[256]; histogram = getHistogramArray(mat);
int T[3] = {0, 0, 0};

for (int i = 0; i < 3; i++){
        int count = 0, threshold = pixels/p[i];
        for (int j = 0; j < 256; j++){
                count += histogram[j];
                if (count >= threshold) {
                        T[i] = j;
                        break;
                }
        }
}

//Same code as static: for every pixel, do
if (mat.at<uchar>(i,j) > T[t]) {
        ptileBin[t].at<uchar>(i,j) = 255;
} else {
        ptileBin[t].at<uchar>(i,j) = 0;
}
```

P-tile and ½ Binarization - Result

original

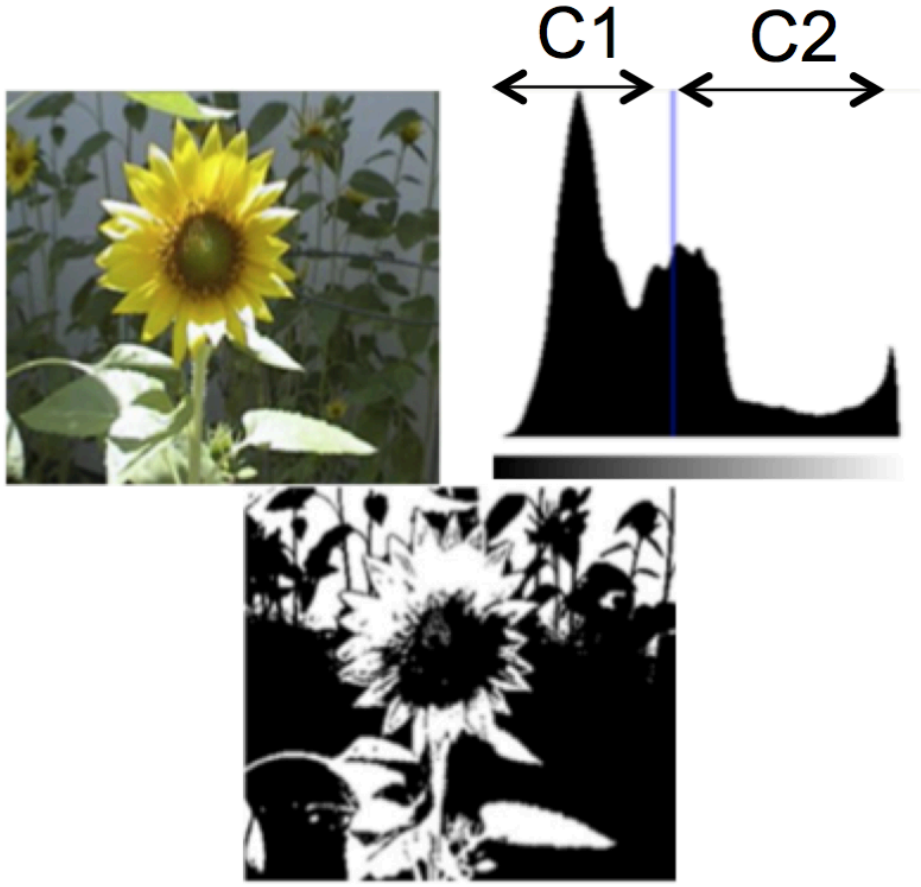
p = 2



p = 3

p = 7

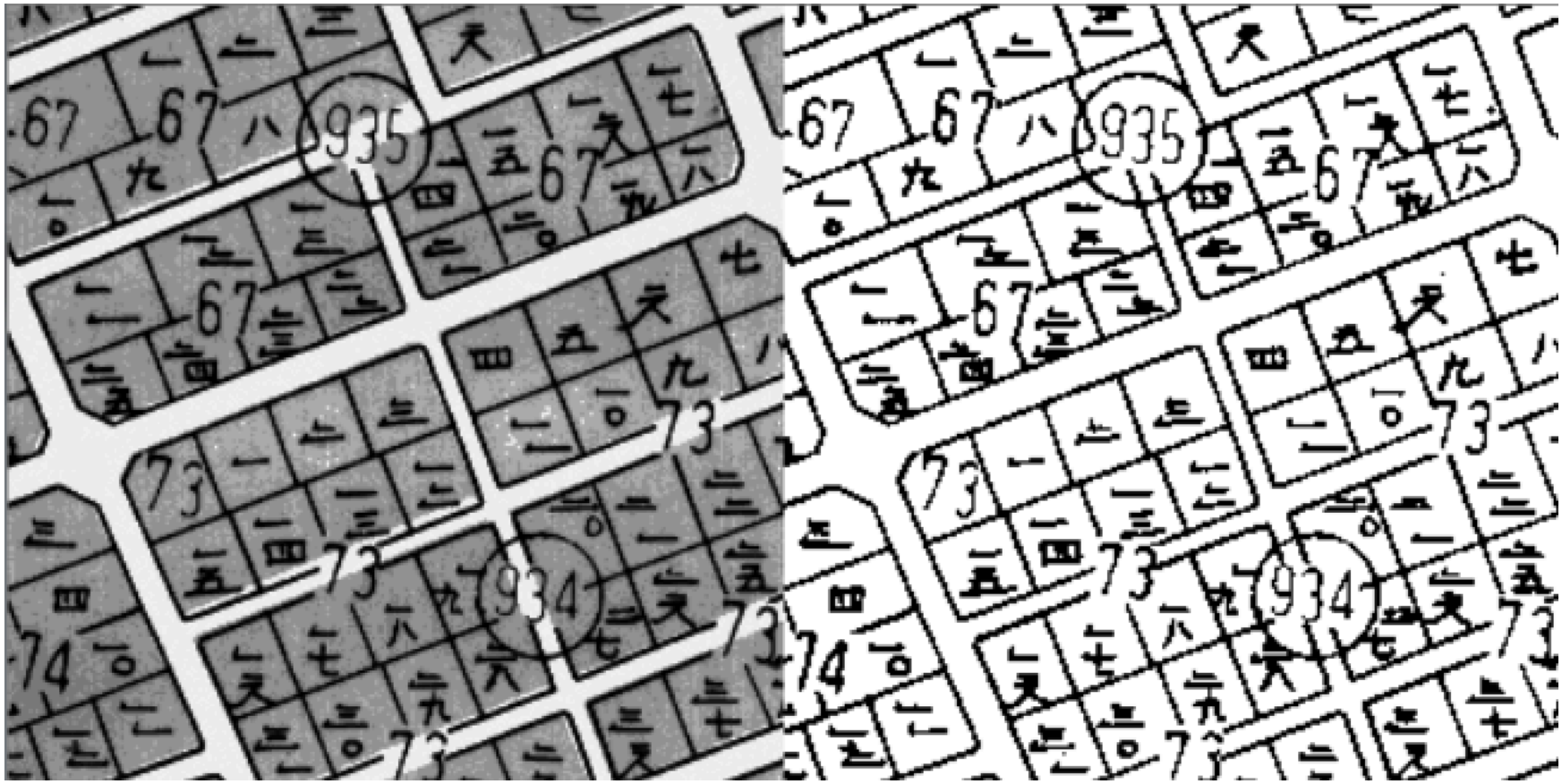
Iterative Binarization



Iterative Binarization - Code

```
int sum = 0, error = 0.1, T = 0, previousT = error+1;
int total = mat.rows*mat.cols;

for (int i = 0; i < mat.rows; i++){
        for (int j = 0; j < mat.cols; j++){
                sum += mat.at<uchar>(I,j);
        }
}
T = cvRound(sum/total);

while (abs(T-previousT) > error) {
        previousT = T;

        int sum2 = 0, mu1 = 0, mu2 = 0, cont1 = 0, cont2 = 0;
        for (int i = 0; i < mat.rows; i++){
                for (int j = 0; j < mat.cols; j++){
                        if (mat.at<uchar>(I,j) < T) {
                                mu1 += mat.at<uchar>(I,j);
                                cont1++;
                        } else {
                                mu2 += mat.at<uchar>(I,j);
                                cont2++;
                        }
                }
        }

        mu1 /= cont1;
        mu2 /= cont2;
        T = cvRound((mu1+mu2)/2);

        //Same code as static: for every pixel, check if it's bigger than T (255) or smaller (0)
}
```

Iterative Binarization - Result



Otsu

- Goal: obtain maximum between-class variance (clear valley, clear threshold)

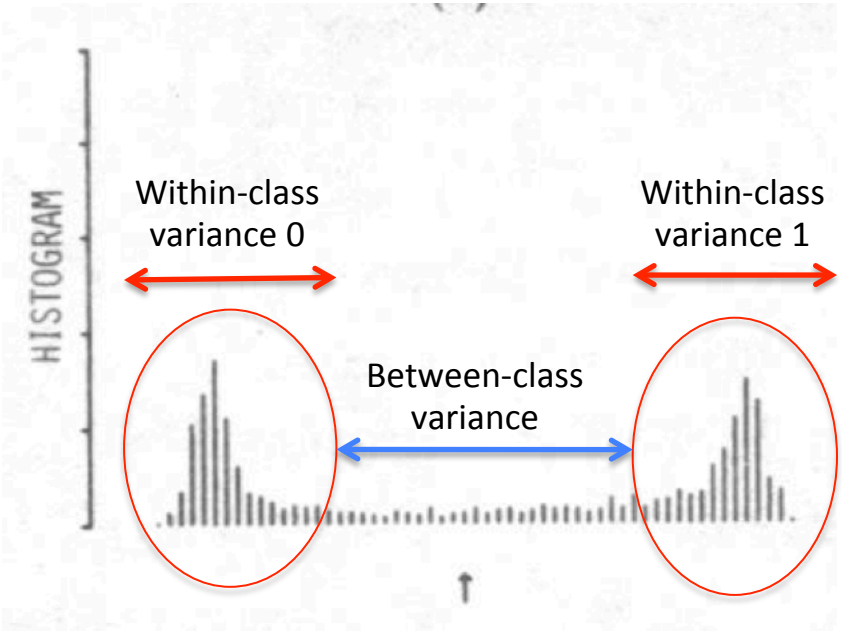
$$\sigma_B^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2$$
$$= \omega_0\omega_1(\mu_1 - \mu_0)^2$$

$$\omega_0 = \Pr(C_0) = \sum_{i=1}^{k} p_i = \omega(k)$$

$$\omega_1 = \Pr(C_1) = \sum_{i=k+1}^{L} p_i = 1 - \omega(k)$$

$$\mu_0 = \sum_{i=1}^{k} i \Pr(i|C_0) = \sum_{i=1}^{k} ip_i/\omega_0 = \mu(k)/\omega(k)$$

$$\mu_1 = \sum_{i=k+1}^{L} i \Pr(i|C_1) = \sum_{i=k+1}^{L} ip_i/\omega_1 = \frac{\mu_T - \mu(k)}{1 - \omega(k)}$$



Otsu- Code

```
//Compute histogram and probabilities of each intensity level
int *histogram = new int[256]; histogram = getHistogramArray(mat);
double probabilities[256];
for (int i = 0; i < 256; i++)
	probabilities[i] = (double) histogram[i]/mat.total();

double omega_0, mu_0, omega_1, mu_1, sigma_squared_current = 0, sigma_squared = 0; int threshold = 0;

for (int T = 0; T < 256; T++) {
	omega_0 = 0.0, mu_0 = 0.0, omega_1 = 0.0, mu_1 = 0.0; //initializing variables
	//omega
	for (int i = 0; i < T; i++)
		omega_0 += probabilities[i];
	omega_1 = 1- omega_0;

	//mu
	for (int i = 0; i < T; i++){
		if (omega_0 == 0) mu_0 += probabilities[i]*i;
		else mu_0 += probabilities[i]*i/omega_0;
	}
	for (int i = T; i < 256; i++){
		if (omega_1 == 0) mu_1 += probabilities[i]*i;
		else mu_1 += probabilities[i]*i/omega_1;
	}

	sigma_squared_current = omega_0*omega_1*(mu_1-mu_0)*(mu_1-mu_0);

	if (sigma_squared < sigma_squared_current) {
		sigma_squared = sigma_squared_current;
		threshold = T;
	}
}
```

Otsu - Result

original

OpenCv

custom