Image Segmentation

What is Image Segmentation?

- We can divide or partition the image into various parts called segments
- It’s not a great idea to process the entire image at the same time as there will be regions in the image which do not contain any information
- By dividing the image into segments, we can make use of the important segments for processing the image
- Image segmentation creates a pixel-wise mask for each object in the image
  - This technique gives us a granular understanding of the object(s) in the image

Different types of image segmentation

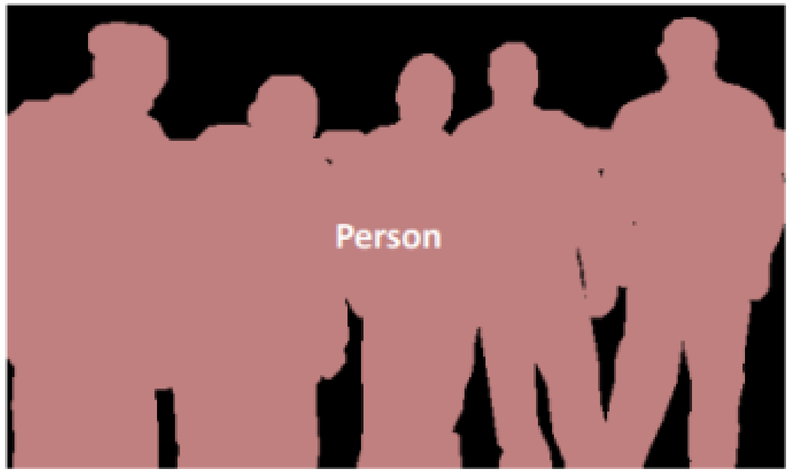


Image 1

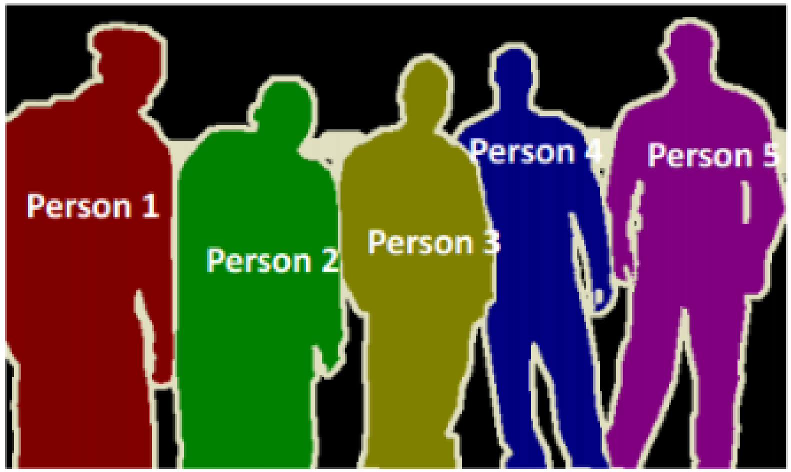


Image 2

- Image 1: Semantic segmentation
  - Every pixel belongs to a particular class (either background or person)
- Image 2: Instance segmentation
  - Different objects of the same class have different colors (Person 1 as red, Person 2 as green, background as black, etc.)

Threshold segmentation

- One simple way to segment different objects could be to use their pixel values
  - The pixel values will be different for the objects and the image’s background if there’s a sharp contrast between them.
- **Threshold segmentation**: the pixel values falling below or above the threshold can be classified accordingly (as an object or the background)

Threshold segmentation

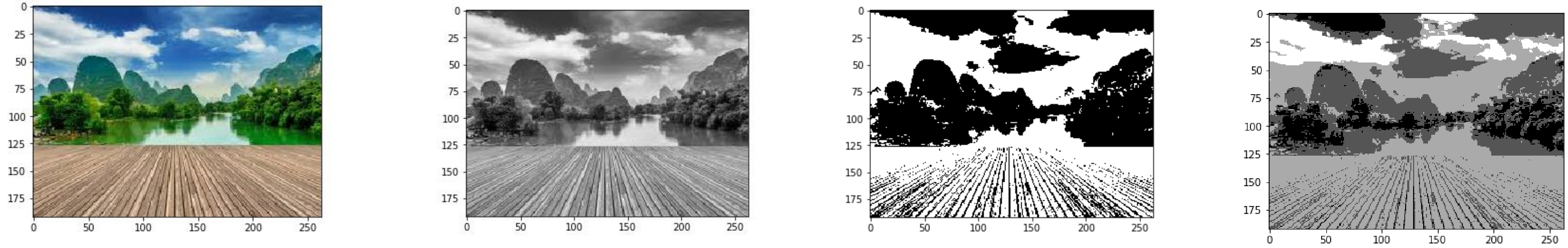
- Global threshold: If we want to divide the image into two regions (object and background), we define a single threshold value

- Local threshold: If we have multiple objects along with the background, we must define multiple thresholds

There are four different segments in the above image.
We can set different threshold values and check how the segments are made.

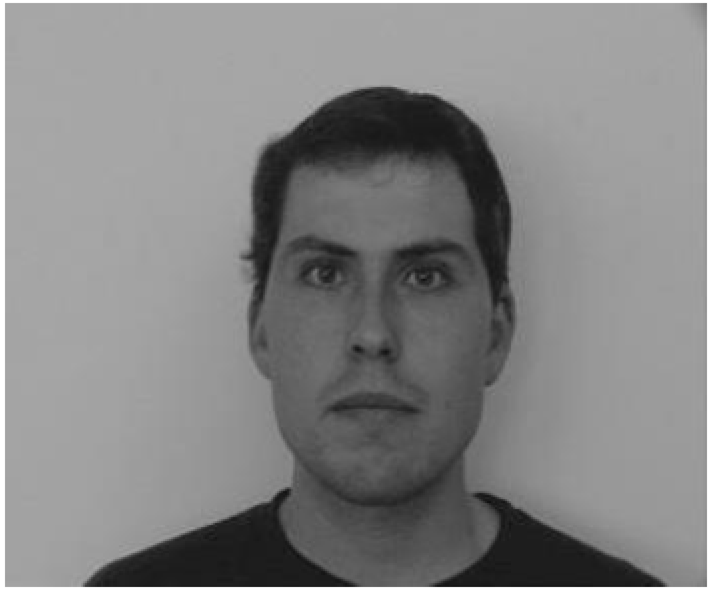
```
(192, 263, 3)

<matplotlib.image.AxesImage at 0x7fc845de2048>
```



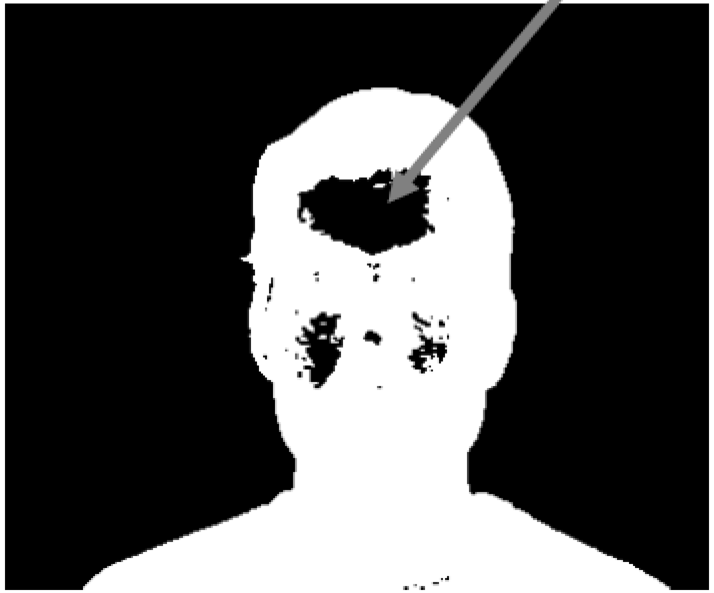
Gray-level thresholding

How can holes be filled?



Original image

Peter $f[x,y]$

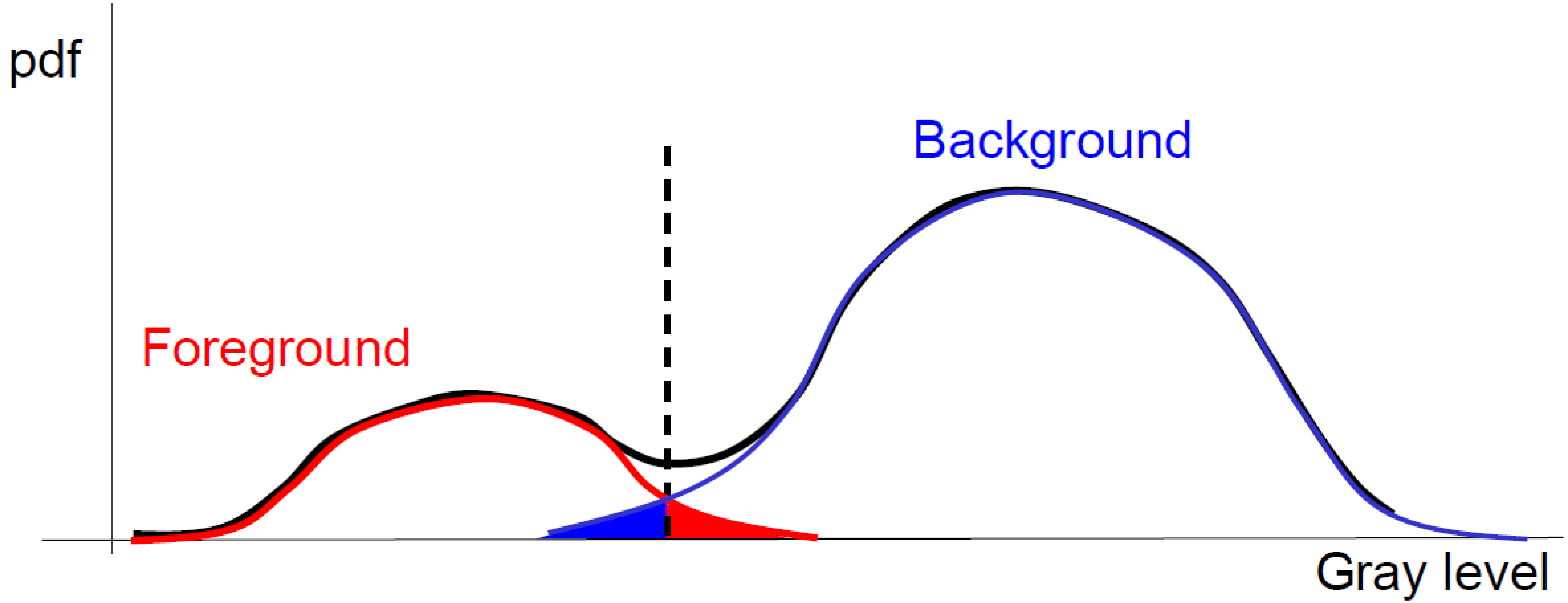


Thresholded

Peter $m[x,y]$

$f[x,y] \cdot m[x,y]$

How to choose the threshold?



Otsu’s Thresholding Method

- Based on a very simple idea: Find the threshold that minimizes the weighted within-class variance
- This turns out to be the same as maximizing the between-class variance
- Operates directly on the gray level histogram [e.g. 256 numbers, $P(i)$], so it’s fast (once the histogram is computed)

Otsu: Assumptions

- Histogram (and the image) are bimodal.
- No use of spatial coherence, nor any other notion of object structure.
- Assumes stationary statistics, but can be modified to be locally adaptive
- Assumes uniform illumination (implicitly), so the bimodal brightness behavior arises from object appearance differences only

The *weighted within-class variance* is:

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

Where the class probabilities are estimated as:

$$q_1(t) = \sum_{i=1}^{t} P(i) \qquad q_2(t) = \sum_{i=t+1}^{I} P(i)$$

And the class means are given by:

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \qquad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)}$$

Finally, the individual class variances are:

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)}$$

$$\sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}$$

Now, we could actually stop here. All we need to do is just run through the full range of t values [1,256] and pick the value that minimizes $\sigma_w^2(t)$.

But the relationship between the within-class and between-class variances can be exploited to generate a recursion relation that permits a much faster calculation.

Between/Within/Total Variance

- The basic idea is that the total variance does not depend on threshold (obviously)
- For any given threshold, the total variance is the sum of the within-class variances (weighted) and the *between class variance*, which is the sum of weighted squared distances between the class means and the grand mean

After some algebra, we can express the total variance as...

$$\sigma^2 = \underbrace{\sigma_w^2(t)}_{\text{Within-class, from before}} + \underbrace{q_1(t)[1 - q_1(t)][\mu_1(t) - \mu_2(t)]^2}_{\text{Between-class, } \sigma_B^2(t)}$$

Since the total is constant and independent of *t*, the effect of changing the threshold is merely to move the contributions of the two terms back and forth.

So, *minimizing the within-class variance is the same as maximizing the between-class variance.*

The nice thing about this is that we can compute the quantities in $\sigma_B^2(t)$ *recursively* as we run through the range of *t* values.

Finally...

Initialization... $q_1(1) = P(1)\,;\ \mu_1(0) = 0$

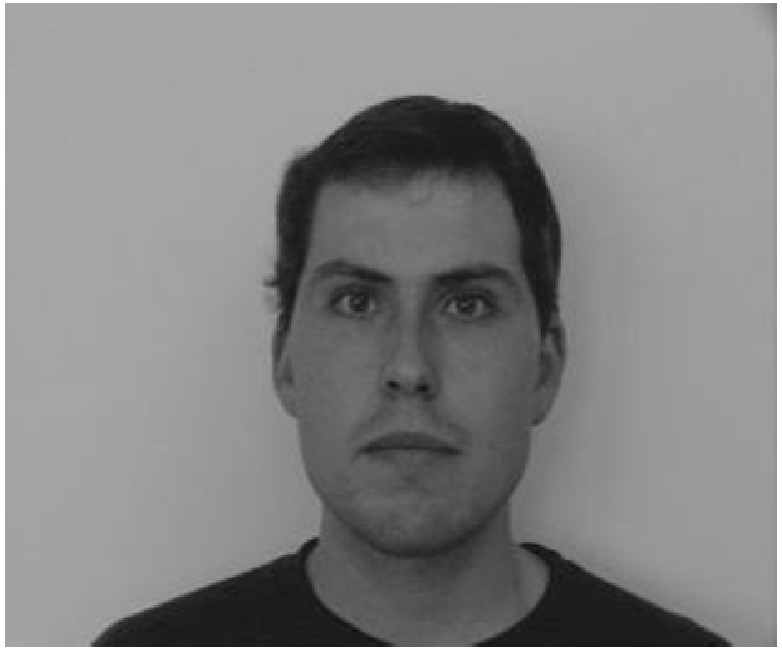
Recursion...

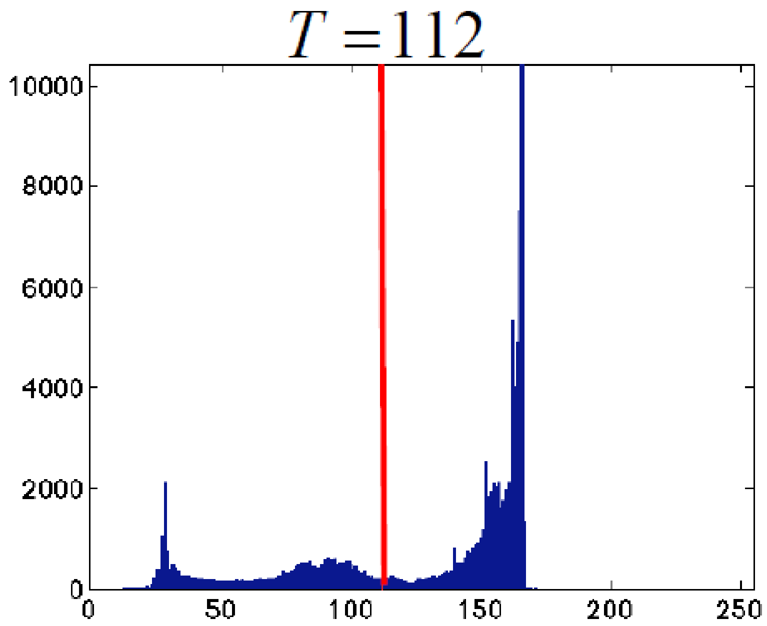
$$q_1(t+1) = q_1(t) + P(t+1)$$

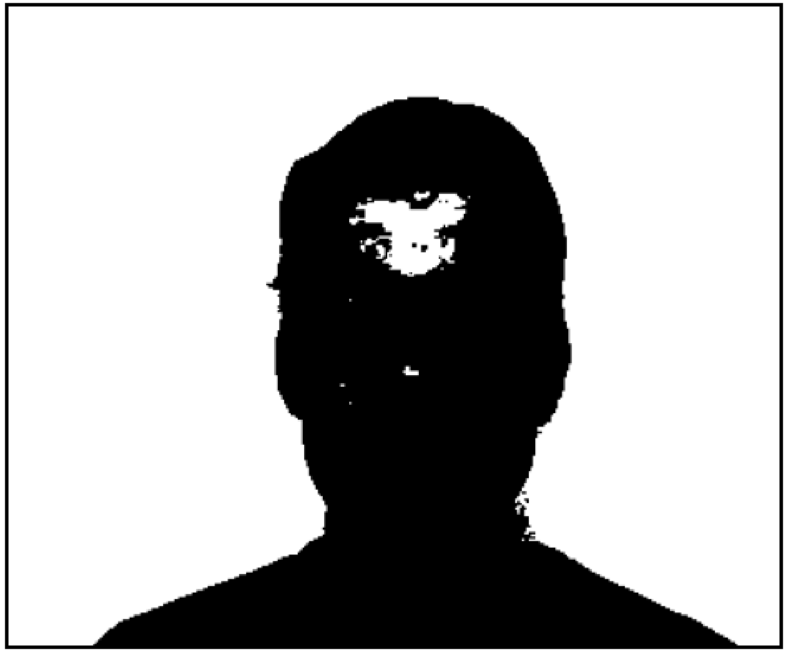
$$\mu_1(t+1) = \frac{q_1(t)\mu_1(t) + (t+1)P(t+1)}{q_1(t+1)}$$

$$\mu_2(t+1) = \frac{\mu - q_1(t+1)\mu_1(t+1)}{1 - q_1(t+1)}$$

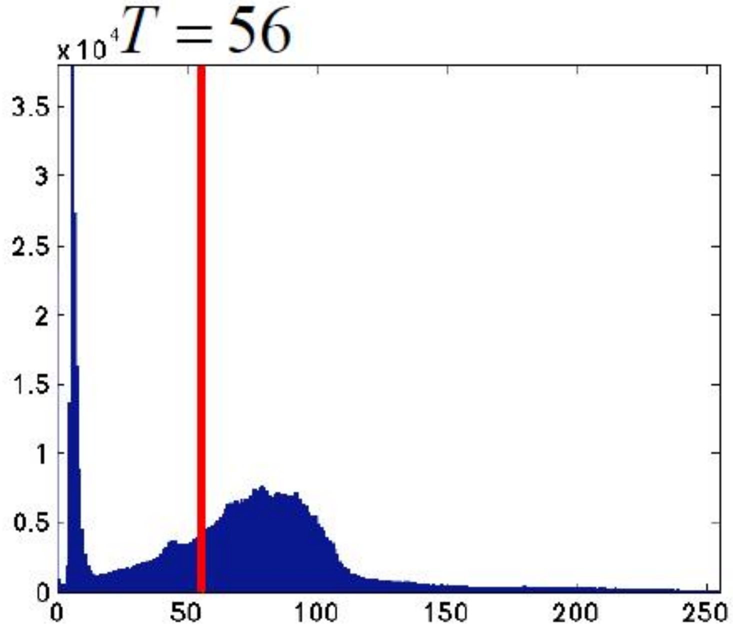
Unsupervised thresholding (cont.)



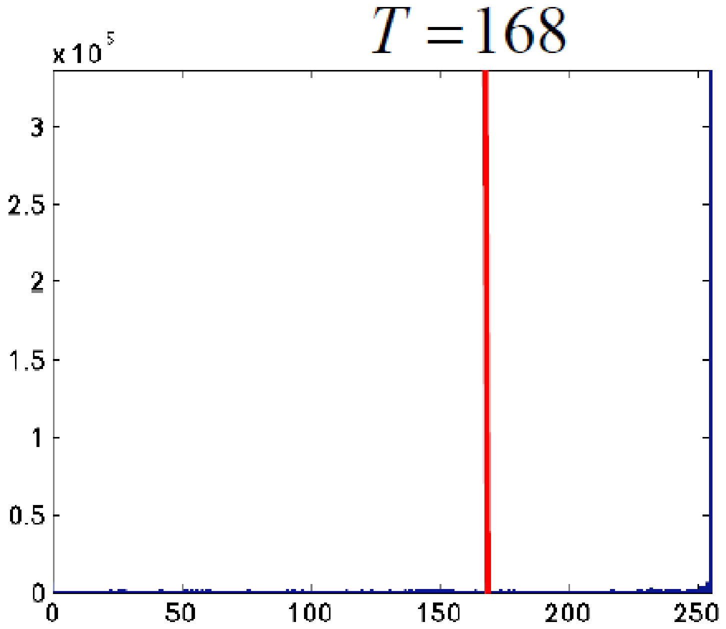




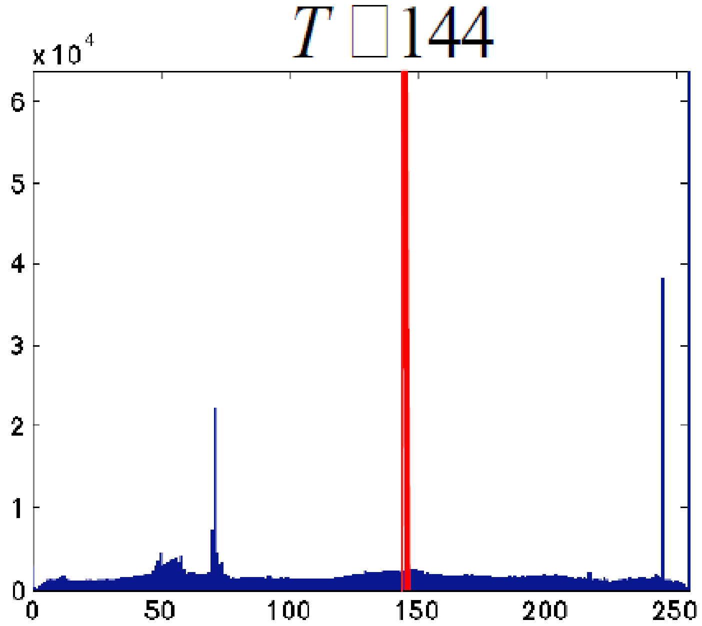
Unsupervised thresholding (cont.)



Unsupervised thresholding (cont.)

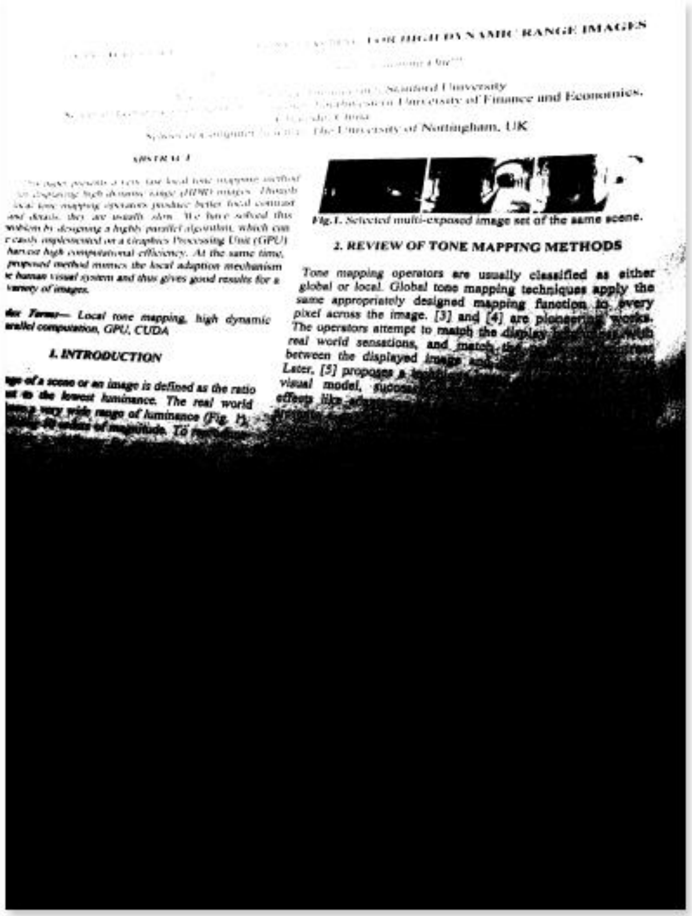


Unsupervised thresholding (cont.)



Sometimes, a global threshold does not work

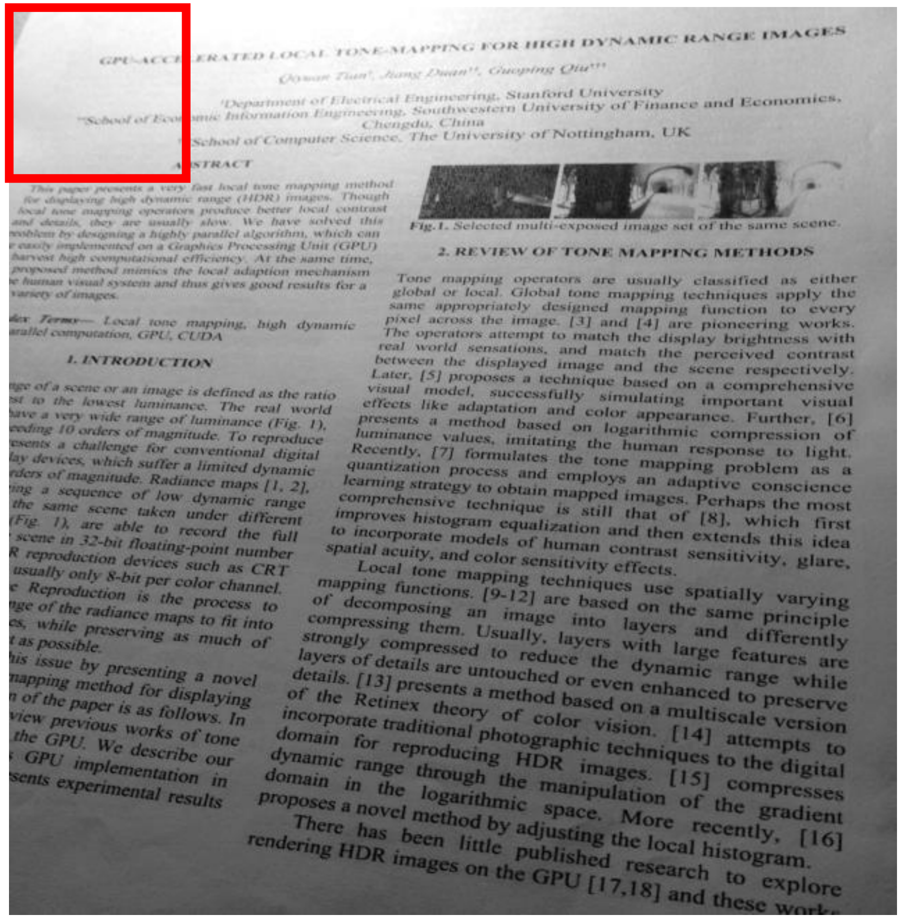
Original image



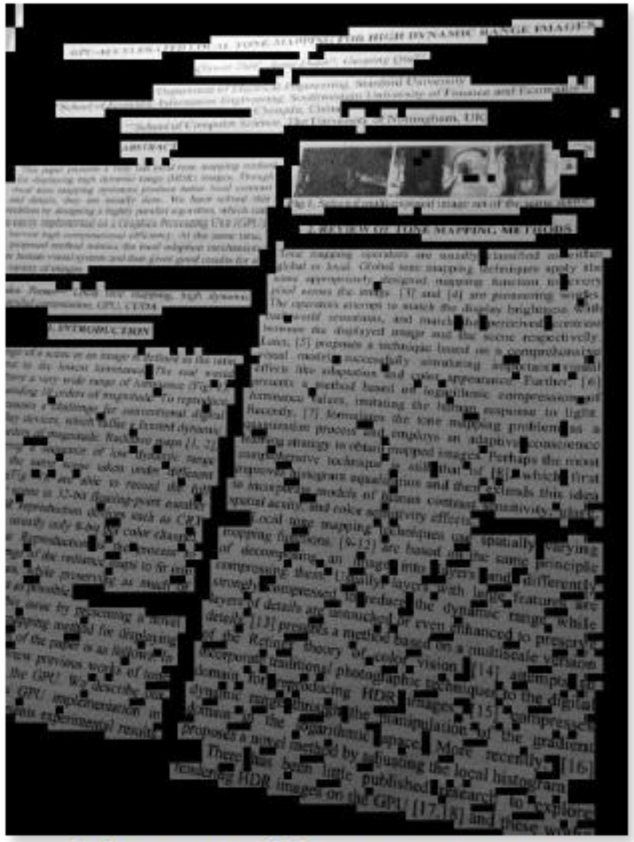
Thresholded with Otsu's Method

Locally adaptive thresholding

- Slide a window over the image
- For each window position, decide whether to perform thresholding
  - Thresholding should not be performed in uniform areas
  - Use variance or other suitable criterion
- Non-uniform areas: apply Otsu's method (based on local histogram)
- Uniform areas: classify the entire area as foreground or background based on mean value



Locally adaptive thresholding (example)



Non-uniform areas

Local threshold values

Locally thresholded result