Université de Lille

AUTOMATIC IMAGE SEGMENTATION BY HISTOGRAM ANALYSIS

RECONNAISSANCE DES FORMES

Iacob Sergiu

27th February 2020

Contents

Chapter 1

Introduction

1.1 Context

This report is the result of a university assignment. It aims to prove that the student understood the motivation, goals and means of studying pattern recognition. Therefore, this is a way of summarizing a series of observations and experiments done on images containing shapes.

1.2 Motivation

L'objectif du TP est de segmenter une image en niveaux de gris par le seuillage automatique des niveaux de gris par des macros R que vous allez concevoir. In order to be able to work with shapes, we must first acquire them. Most of the time, they are not given to us in their simplest form, but found in images with all kinds of backgrounds. They must be extracted as precisely as possible, making sure we differentiate them from the background as accurately as we can. Where there's a need for great precision and accuracy, there's also a need for research, so in this report we'll be looking over some methods we can use to identify objects (shapes).

1.3 Goals

Having multiple grayscale images, we want to be able to do a *binary segmentation* on them in order to differentiate between the objects and the image's background. Given the fact that we only have gray values at our disposal, we must look into ways of using them in order to state that a certain pixel belongs to either an object or to the background.

CHAPTER 2

BINARISATION USING BAYES' THEOREM

2.1 LOADING DATA

The image we'll try to apply binarisation to is the following:

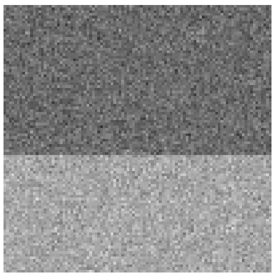

FIGURE 2.1
Image to experiment on

Which is composed of the following 2 images:

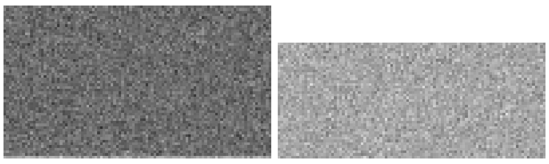

FIGURE 2.2
Separate images

The images will be be loaded as gray images:

```r
# Chargement d'une image en niveaux de gris
rdfReadGreyImage <- function (nom) {
    image <- readImage (nom)
    if (length (dim (image)) == 2) {
        image
    } else {
        channel (image, 'red')
    }
}
```

LISTING 2.1
Loading data in R

2.2 TECHNICAL ACKNOWLEDGEMENTS

Please note that for the following scripts, some data was pre-loaded to avoid loading it every time.

```r
# loading some data only once
image <- rdfReadGreyImage ("2classes_100_100_8bits_2016.png")
above <- rdfReadGreyImage ("2classes_100_100_8bits_omega1_2016.png")
below <- rdfReadGreyImage ("2classes_100_100_8bits_omega2_2016.png")

nbins <- 256
h <- hist (as.vector (image), freq=FALSE, breaks = seq (0, 1, 1 / nbins))
h1 <- hist (as.vector (above), freq=FALSE, breaks = seq (0, 1, 1 / nbins))
h2 <- hist (as.vector (below), freq=FALSE, breaks = seq (0, 1, 1 / nbins))
```

LISTING 2.2
Loading some data only once

2.3 FIXED THRESHOLD

Taking a look at the histogram, we can have a "manual" attempt at binarising the image: using a *fixed threshold*. We simply choose a value for the threshold and we assign, for each pixel, one of the predicted classes as it follows:

$$\hat{w_1} = \{P \in I | I(P) < \hat{X}\}$$
$$\hat{w_2} = \{P \in I | I(P) \geq \hat{X}\}$$

where $\hat{X}$ is our threshold.

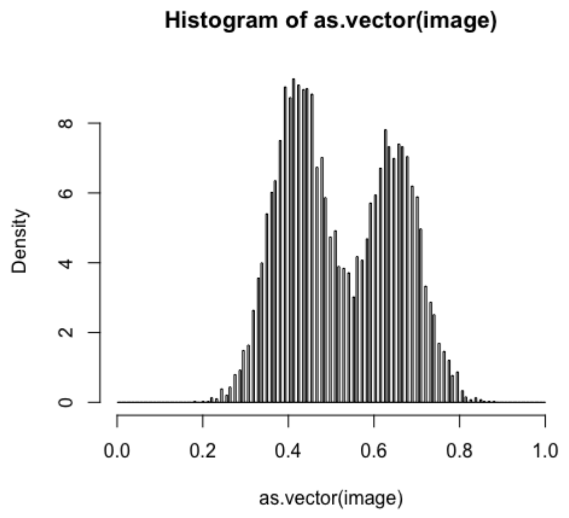

FIGURE 2.3
Histogram of grey values for

```r
# for each threshold, calculate the binary image
# image has values between [0, 1] so our threshold has to be in the same range
# for each pixel, the expression (pixel_value - threshold) >= 0 will assign it
to one of two classes: w1 or w2
binaire50 <- (image - 0.5) >= 0
display (binaire50, "image binaire 0.35", method="raster", all=TRUE)
binaire55 <- (image - 0.55) >= 0
display (binaire55, "image binaire 0.35", method="raster", all=TRUE)
binaire60 <- (image - 0.6) >= 0
display (binaire60, "image binaire 0.35", method="raster", all=TRUE)
```

LISTING 2.3
Using fixed threshold

Let's take a look at the results, using threshold values of 0.5, 0.55 and 0.6.

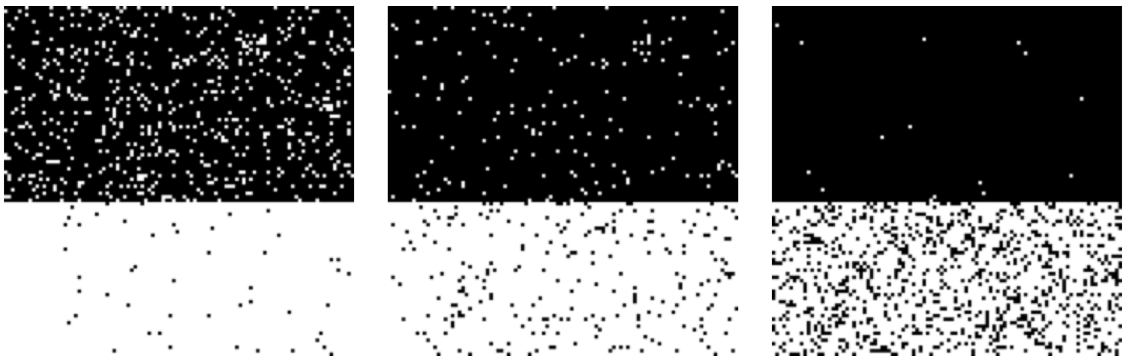

FIGURE 2.4
Fixed threshold results

It is not hard to see that there is a lot of noise in our predictions and they are far from perfect. The threshold value of 0.6 might seem attractive, but it only manages to classify well most of the pixels from the image on the top. Further more, it heavily missclassifies the pixels from the image on the bottom.

2.4 A PRIORI CLASS PROBABILITIES

Our image 2.1 is formed of 2 separate images 2.2. We can try to calculate the following probabilities:

$P(w_1)$, the probability of a pixel to be in w_1 class
$P(w_2)$, the probability of a pixel to be in w_1 class

Knowing that the dimension of the merged image is 100x100 and the 2 images are 100x43, 100x57, the probabilities above are easy to calculate.

```
calculateClassAPrioriProbabilities <- function(){
    # pw1 = (number_of_pixels_from_the_first_image) / (number_of_total_pixels)
    # pw1 = (number_of_pixels_from_the_second_image) / (number_of_total_pixels)
    noTotalPixels = dim(image)[1] * dim(image)[2]
    pw1 = (dim(above)[1] * dim(above)[2]) / noTotalPixels
    pw2 = (dim(below)[1] * dim(below)[2]) / noTotalPixels
    c(pw1, pw2)
}
```

LISTING 2.4
A priori class probabilities

In this case, the probabilities are $P(w_1) = 0.57$ and $P(w_2) = 0.43$.

2.5 CONDITIONAL PROBABILITIES

Having $P(w_1)$ and $P(w_2)$, we can take a step further and calculate the following conditional probabilities:

$P(X|w_1)$, the probability of a pixel having the gray value of X if it's in the w_1 class
$P(X|w_2)$, the probability of a pixel having the gray value of X if it's in the w_2 class

These are, according to Wikipedia 2020, equal to:

$$P(X|w) = \frac{P(w|X) * P(X)}{P(w)}$$

We already have $P(w)$ and $P(X)$ is simply the number of pixels with the gray value of X divided by the total number of pixels. $P(w|X)$ is the number of pixels with the value X in the image corresponding to w divided by the total number of pixels with a value of X.

```r
calculateConditionalProbability <- function (X, probs){
    # let's take a look at the histogram again
    # we can notice that h$counts[X + 1] gives us the number of pixels that have
 the gray value of X
    # the "+ 1" above is because in R arrays start from 1

    # so, P(X | I) = the probability of a pixel from the merged image to have
the gray value of X
    px <- h$counts[X + 1] / sum(h$counts)
    # P(X | w) = the probability of a pixel from the merged image to have the
gray value of X
    # IF that pixel is from the w class
    # Using bayes theorem, we get:
    # P(X|w) = (P(w | X) * P(X)) / P(w)
    # and P(w | X) = how many pixels of value X are in the image corresponding
to w / the total number of pixels with a value of X

    p_w1_if_x <- h1$counts[X + 1] / h$counts[X + 1]
    p_x_and_w1 <- p_w1_if_x * px
    p_x_if_w1 <- p_x_and_w1 / probs[1]

    p_w2_if_x <- h2$counts[X + 1] / h$counts[X + 1]
    p_x_and_w2 <- p_w2_if_x * px
    p_x_if_w2 <- p_x_and_w2 / probs[2]

    c(px, p_x_if_w1, p_x_if_w2)
}

print (calculateConditionalProbability(141, probs))
```

LISTING 2.5
Calculating conditional probabilities

Running the above function for $X = 141$, we get the following probabilities:

$$P(X|I) = 0.011800000$$
$$P(X|w_1) = 0.008947368$$
$$P(X|w_2) = 0.015581395$$

where $P(X|I)$ refers to the probability of a pixel to have the gray value of X in the whole (merged) image. Of course, $P(X|w_1) * P(w_1) + P(X|w_2) * P(w_2) = P(X|I)$.

2.6 AUTOMATIC THRESHOLDING USING BAYES

Having a threshold $\hat{X}$, we can define an assignment error as being the following:

$$P(\epsilon|\hat{X}) = \underset{x}{\text{argmin}}(\sum_{X \in \hat{w}_2} P(X|w_1) * P(w_1) + \sum_{X \in \hat{w}_1} P(X|w_2) * P(w_2))$$

Our goal is to find the $\hat{X}$ that minimizes $P(\epsilon|\hat{X})$. For this, we can try each value for $\hat{X}$ and see which is the best.

```
automaticSegmentationUsingBayes <- function(){
    # calculate these probabilities only once
    probs <- calculateClassAPrioriProbabilities()
    # only get the imageData, it will be faster for comparisons
    image <- imageData(image)
    # calculate these dimensions only once
    dimImage <- dim(image)
    dimAbove <- dim(above)
    dimBelow <- dim(below)

    # build our reference segmentation
    perfect <- matrix (ncol=dim(image)[1], nrow=dim(image)[2])
    # dim(image)[1] = number of columns
    for (i in 1:dim(image)[1])
      for (j in 1:dim(above)[2]){
        perfect[i, j] <- FALSE
      }
    for (i in 1:dim(image)[1])
      for (j in dim(above)[2]:dim(image)[2]){
        perfect[i, j] <- TRUE
      }
    display(perfect, method="raster", all=TRUE)

    # search for the best threshold
    # max error is (hypotethically) when all probabilities are 1 and all pixels
are missclassified
    min_error <- 2 * dimImage[1] * dimImage[2]
    best_threshold <- 0

    # precalculate all probabilities
    condProbs <- array(dim=c(dimImage[1], dimImage[2], 3))
    for (i in 1:dimImage[1])
      for (j in 1:dimImage[2]){
        res <- calculateConditionalProbability(image[i, j] * 255, probs)
        condProbs[i, j, 1] <- res[1]
        condProbs[i, j, 2] <- res[2]
        condProbs[i, j, 3] <- res[3]
      }

    for (X in 0:255){
      binary <- (image - X/255) >= 0
      # only get the data, otherwise the comparisons will be really slow
      binary <- imageData(binary)

      error <- 0
      for (i in 1:dimImage[1])
        for (j in 1:dimAbove[2])
          if (perfect [i, j] != binary [i, j])
            error <- error + condProbs[2] * probs[1]
      for (i in 1:dimImage[1])
        for (j in dimAbove[2]:dimImage[2])
          if (perfect [i, j] != binary [i, j])
            error <- error + condProbs[3] * probs[2]

      if (error < min_error){
        min_error <- error
        best_threshold <- X
      }
    }

    print (c (min_error, best_threshold))
    # segment using best_threshold
    binary <- (image - best_threshold/255) >= 0
    display(binary, method="raster", all=TRUE)
  }
```

LISTING 2.6
Automatic segmentation using Bayes

Below we can look at the "perfect" image and the one we got using this automatic segmentation. To calculate the "perfect" segmentation, we simply assigned all pixels from the first (above) image to a class, and the rest to another class. According to the script, the minimum error is 6.22581 and the best threshold for the gray value is $X = 142$.

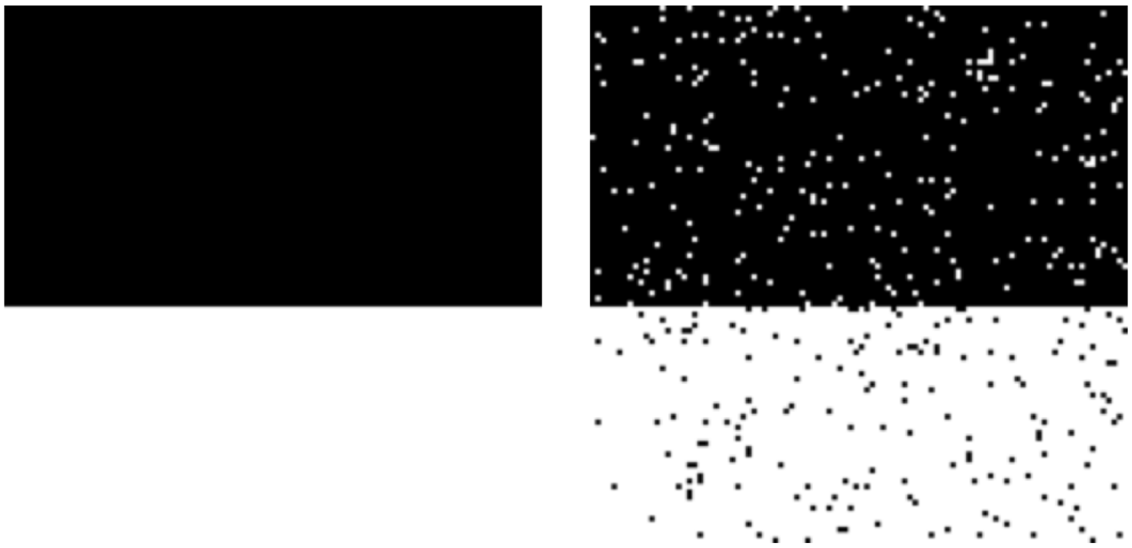

FIGURE 2.5
Perfect segmentation and our best result

Chapter 3

Conclusion

Using only the gray values or the texture values as attributes did not result in a great segmentation. As we've seen in the experiments, the error rate, when using each attribute individually, can be pretty high in certain cases, even though for some textures the methods are really precise.

However, where one of them did bad, the other did well. By combining the two, we have a more precise way of segmenting the image. In conclusion, we may state that the combination of the two attributes is a worthy contender for binary segmentation.

Bibliography

Wikipedia (2020). *Bayes' Theorem*. Wikimedia Foundation. URL: `https://en.wikipedia.org/wiki/Bayes%27_theorem` (visited on 27th Feb. 2020).