POLITECNICO DI MILANO
School of Industrial and Information Engineering
Master of Science Degree in Mathematical Engineering
Applied Statistics

A STUDY ON BAGGING-VORONOI ALGORITHM FOR TAMPERING LOCALIZATION

Supervisor: Prof. Paolo BESTAGINI
Co-supervisor: Prof. Marco COMPAGNONI
Co-supervisor: Ing. Luca BONDI

Master thesis of:
Corinne Elena CEREGHETTI
matricola 864050

Academic Year 2017-2018

A coloro che credono in quello che fanno
e lottano per raggiungere i propri obiettivi.

Abstract

In the last decades, it's always become easier to modify an image. For this reason the forensic community has dedicated significant attention to the development of image authenticity verification algorithms.
The aim of this thesis is to propose a new technique to solve the tampering localization problem, using the Bagging-Voronoi algorithm. This algorithm was originally developed by Secchi, Vantini, and Vitelli [1] for spatial dependents data. We will focus on the investigation of the optimal parameters under which the algorithm gives us the best results in a real-case tampering localization scenario.
We will see that looking at a single image it's very difficult to find a priori, the optimal parameters, since any image has different characteristics, as the dimension of the whole image and the dimension of the tampered area.
Then we will propose some improvements, trying to exploit all the possible information we have about the input data.
We will notice that knowing the geometry of our data, and using this information in the Bagging-Voronoi algorithm, we will obtain much better results, while there are some images for which it's difficult to obtain the expected good results.

The algorithm and all the analysis are done using *R*.

Sommario

Negli ultimi anni è diventato sempre più facile modificare il contenuto di un'immagine. Per questo motivo è stato necessario, da parte della comunità forense, sviluppare delle tecniche in grado di determinare l'autenticità delle immagini.
Lo scopo di questa tesi è quella di proporre una nuova tecnica per la risoluzione del problema di ricerca di modifiche in immagini, ovvero l'algoritmo Bagging-Voronoi, tecnica originariamente sviluppata da Secchi, Vantini, and Vitelli [1] per dati spazialmente dipendenti.
Ci concentreremo sulla ricerca di condizioni di ottimalità in corrispondenza delle quali la sua accuratezza di classificazione è massima. Quello che otterremo è che prese le singole immagini è assai difficile trovare a priori i parametri ottimali poiché ogni immagine presenta caratteristiche differenti, come la dimensione dell'immagine o dell'area modificata.
Proporremo poi qualche miglioramento, cercando di sfruttare al meglio tutte le informazioni possibili che abbiamo sui dati in ingresso.
Scopriremo che conoscendo la geometria dei dati (ovvero in che modo sono stati costruiti i dati) e utilizzando questa informazione nell'algoritmo Bagging-Voronoi, otterremo risultati decisamente migliori, sebbene rimangono alcune immagini per le quali non si ottengono i risultati sperati.

L'algoritmo e tutte le analisi sono state svolte con l'utilizzo di *R*.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In the last decades, due to the imaging software availability, we can easily modify the content of an image. As a consequence, the diffusion of tampered content has become a widespread phenomenon, and for this reason we need to discern between original or tampered content.
Fortunately, when an image is edited through non-reversible operations, it is possible to reconstruct information about its past history for authenticity detection.
The goal of this thesis is to develop a mathematical algorithm able to perform tampering localization. The proposed one is the Bagging-Voronoi algorithm, that was originally developed by Secchi, Vantini, and Vitelli [1] for the exploration of georeferenced functional data. Suppose to observe a set of data on a spatial grid, for which the characteristics of each observation is influenced by the neighboring data. In other words, there is a correlation between neighboring data. We are working with images, that can be represented as a bi-dimensional grid. We divide our image in patches of dimension 64×64 pixel, and the union of all this patches give us the grid. We have spatial dependency between data, since we expected that all the patches belonging to the edited area are one close to the other, and for this reason it is useful to use an algorithm that takes into account the spatial dependency between data.
We focus our analysis on the search to the optimal choice of some influential parameters motivating these choices.
For the experimental analysis and the implementation of the Bagging-Voronoi algorithm we use R, that is a programming language for developing statistical software and data analysis.

The thesis is structured as follow.

In Chapter 2 we provide a background on clustering techniques that we used and on image forensic, describing some of the principal techniques developed by the forensic community.

In Chapter 3 we formalize the problem that we want to solve, with a brief motivation on the importance to solve this type of problem. We also provide a brief description of the dataset that we have. Then, we report the current state-of-the-art.

In Chapter 4 we describe the proposed solution to solve the problem; in particular, we focus on the Bagging-Voronoi algorithm, following what is done by Secchi, Vantini, and Vitelli [1]. Then we propose some improvement to be applied.

In Chapter 5 we report all the performed experiments, applying all the solution proposed in Chapter 4.

In Chapter 6 we draw our conclusions on the performed work and we outline the future possible directions of work.

Chapter 2

Background

This Chapter contains some basic notions necessary to fully understand the main contributions of this work. Section 2.1 describes what is clustering and how the proposed algorithms works. Section 2.2 presents an overview on image forensic.

2.1 Clustering Techniques

The aim of this Section is to give an overview on clustering techniques, and how they work.
Clustering techniques search for a structure of "natural" grouping, and grouping is important because can provide an informal tool to evaluate the dimension, identifying outliers and suggesting hypotheses concerning relationships. In cluster analysis there is no need to make assumptions on the number of groups or on the group structure, and grouping is done using similarities or dissimilarities.
For the section related to K-means we will follow the structure given in Eynard [2], and for the description of PAM we will follow Struyf, Hubert, and Rousseeuw [3].

2.1.1 Basic notions

To create a simple group structure from complex dataset a measure of "closeness" or "similarity" is required. Recall first the notation of a distance measure and the definition of Euclidean distance between p-dimensional observations $\mathbf{x}' = [x_1, x_2, \ldots, x_p]$ and $\mathbf{y}' = [y_1, y_2, \ldots, y_p]$

Definition 2.1. A distance measure d on a set X is a function $d : X \times X \to [0, \infty)$, such that $\forall x, y, z \in X$, the following conditions are satisfied:

1. $d(x, y) \geq 0$ (non-negativity)
2. $d(x, y) = 0 \Leftrightarrow x = y$ (identity)
3. $d(x, y) = d(y, x)$ (symmetry)
4. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

Definition 2.2. The Euclidean distance between p-dimensional observations $\mathbf{x}' = [x_1, x_2, \ldots, x_p]$ and $\mathbf{y}' = [y_1, y_2, \ldots, y_p]$ is given by

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_p - y_p)^2} = \sqrt{(\mathbf{x} - \mathbf{y})'(\mathbf{x} - \mathbf{y})}$$

We use distances to measure the similarity and the dissimilarity between two data objects, that tell us how much two object are (or not) similar.
An example of similarity is the cosine similarity distance, defined, for two vectors $\mathbf{A}$ and $\mathbf{B}$ as:

$$similarity = cos(\theta) = \frac{A \cdot B}{||A||||B||}. \tag{2.1}$$

Higher is the value of *similarity* and more two objects are similar. This similarity take values in $[-1, 1]$, and we can obtain the dissimilarity distance in the following way:

$$dissimilarity = 1 - \frac{1 + similarity}{2}. \tag{2.2}$$

In case of the *dissimilarity*, an higher value means that the two objects are not similar.

2.1.2 K-means algorithm

The K-means algorithm is a non hierarchical clustering technique designed to group elements, rather than variables, into K clusters.
The algorithm works in the following way:

1. Considering the space represented by the objects that are being clustered, select K random points, that represent initial group centroids;

2. Assign each element to the group whose centroid is nearest, with respect to a specific distance (not necessarily the Euclidean distance);

3. When all the elements have been assigned, compute again the positions of the K centroids;

4. Repeat steps 2 and 3 until the centroids no longer change.

Advantages in using K-means algorithm are that it is a simple algorithm and that often terminates at a local optimum. The disadvantages are that we need to specify the number of clusters K in advance, that is an algorithm sensible to outliers (the presence of an outlier might produce at least one group with very disperse elements) and that the results depend both on the metric used to measure distances and on the value of K.

2.1.3 Partitioning Around Medoids

This algorithm is very similar to K-means, because both the algorithms divide the dataset into groups and try to minimize a cost function based on a specific distance. The difference between the two is that PAM works with medoids, that are entity of the dataset that represent the groups in which they are inserted, and K-means works with centroids, which are artificially created points that represent its clusters.
The algorithm works with a dissimilarity matrix, whose goal is to minimize the overall dissimilarity between the representatives of each cluster and its members, i.e. the goal is to find a subset of medoids $\{m_1, \ldots, m_k\} \subset \{1, \ldots, n\}$ which minimizes the objective function

$$\sum_{i=1}^{n} \underset{t=1,\ldots,k}{min}\, d(i, m_t). \tag{2.3}$$

As input this algorithm can take both a dissimilarity matrix or a data matrix. The algorithm works in the following way:

1. Calculate the dissimilarity matrix, if not given as input;

2. Compute K medoids as follow:

   - m_1 is the object with the smallest $\sum_{i=1}^{n} d(i, m_1)$;

- m_2 decreases the objective in Eq. (2.3) as much as possible;

$$\vdots$$

- m_K decreases the objective in Eq. (2.3) as much as possible.

3. Assign every object to its closest medoid, with respect to a specific distance (not necessarily the Euclidean distance);

4. When all the elements have been assigned, recalculate the positions of the K medoids, selecting the ones that minimize the objective function in Eq. (2.3);

5. Repeat steps 3 and 4 until the medoids no longer change.

We introduce PAM algorithm because, choosing the initial points among those present, allows us to better respect the description of our data.

2.2 Image Forensic

In this section we want to give an overview on image forensics, following what is done by Piva [4].
These techniques have been designed to identify the source of a digital image or to understand if the content has been modified or not, without having any prior information on the image. Throughout history we have always trusted the authenticity of the images, for example an image printed in a newspaper is synonymous of the truthfulness of the news. With the rapid technological growth, nowadays is very easy to use software tools for image editing, and to alter the content of the images. During its lifetime, a visual digital object might go from acquisition to its fruition, through several processing stages, with the aim of improving the quality, putting together several images or tampering with the content. This situation creates the need for methods that allow the reconstruction of the history of a digital image in order to verify its truthfulness and test its quality.
Knowing the original image, it's easy to understand if the image is acquired with the device that is detected and if the image shows the original captured scene. In practical case, we have not a priori information about the original image, so investigators need to authenticate the image history in a blind way. There are several approaches that can be classified into active and passive

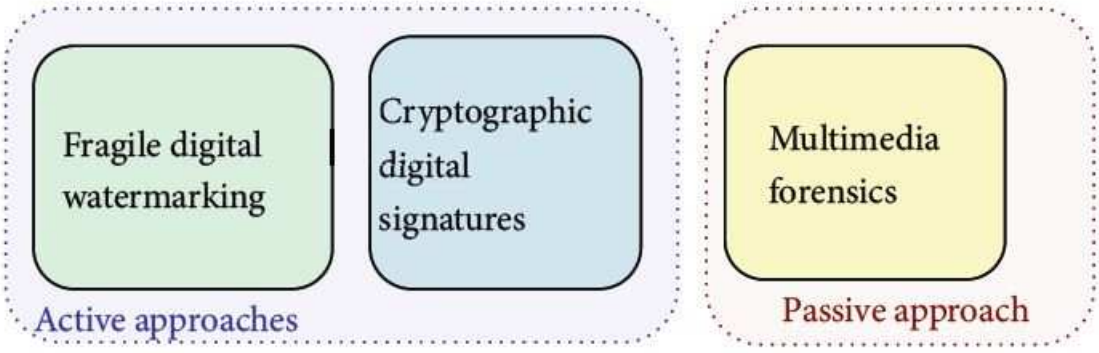

Figure 2.1: Possible approaches for the assessment of the history and credibility of a digital image.

technologies as shown in Figure 2.1, where for "active" we indicate that for the evaluation of reliability, some information that has been calculated at the source (i.e. in the camera) during the acquisition step, are exploited, while with the "passive" term we indicate that the reliability evaluation is made having only the digital content available.

A major problem of active solutions is that digital cameras are specially equipped with a watermarking chip or a digital signature chip that, exploiting a private key hard-wired in the camera itself, authenticates every image the camera takes before storing it on its memory card. To overcome this problem, a new set of methods for authenticating the contents of digital images without using any prior information, defined as passive, have evolved quickly. This technology is defined as multimedia forensic, and the idea is that each phase of the image history (the acquisition process, its storing in a compressed format and any post processing operation), leaves a distinctive trace on the data, like a digital fingerprint. Is therefore possible to identify the source of the digital image or determine if it is authentic or not, looking if the features intrinsically tied to the digital content are present, absent or incongruous. Multimedia forensics descends from the classical forensic science, which studies the use of scientific method to obtain significant data from physical or digital evidences. The goal of multimedia forensic is to expose the traces left in the multimedia content in each step of its life, exploiting existing knowledge on digital image and in multimedia security research. Here we give some examples on how multimedia forensics works:

2.2.1 Photo-Response Non-Uniformity (PRNU)

Photo-response non-uniformity is an intrinsic property of all digital imaging sensors, due to slight variation among individual pixels in their ability to convert photons to electrons. Pixel are made of silicon and capture light by converting photons into electrons, using the photoelectric effect. We use a matrix $\mathbf{K}$, that has the same dimensions as the sensor, to capture the differences among pixel. By illuminating the imaging sensor with ideally uniform light intensity I, without noise sources, the sensor would register a noise-like signal $I + I\mathbf{K}$. The term $I\mathbf{K}$ is usually referred to as the pixel-to-pixel non-uniformity or PRNU. The matrix $\mathbf{K}$ is responsible for the major part of what we call the camera fingerprint, that can be estimated experimentally by taking many images of a uniformly illuminated surface, and averaging the images to isolate the systematic component of all the images. $I\mathbf{K}$ is the PRNU term, and it's only weakly present in dark areas, where $I \approx 0$ or in completely saturated areas of an image, that produce a constant signal and do not carry any traces of PRNU.
The factor $\mathbf{K}$ is very useful, and its responsible for a unique sensor fingerprint; has the properties of dimensionality (the fingerprint is stochastic in nature and has a large information content, which makes it unique to each sensor), universality (all imaging sensors exhibit PRNU), generality (the fingerprint is present in every picture, with the exception of completely dark images), stability (is stable in time and under wide range of environmental conditions, like temperature or humidity) and robustness (it survives lossy compression, filtering, and many other typical processing).
The fingerprint can be used for many forensic tasks; for example, by testing the presence of a specific fingerprint in the image we can prove that a certain camera took a given image, or prove that two images comes from the same camera. We can also see if the image is natural or modified: by testing the absence of the fingerprint in individual image regions, we can find areas of the image that has been replaced. By detecting the strength or form of the fingerprint, we can reconstruct some of the processing history. We can use the spectral and spatial characteristics of the fingerprint to identify the camera model or distinguish between a scan and a digital camera image.
For more details, refer to Fridrich [5]

2.2.2 Camera Model Identification

A problem that we aim to solve is to detect the camera model (and the brand of this camera) used to shoot a picture to solve some forensic problems, like copyright infringement to ownership attribution. The forensic community has developed a set of camera model identification algorithms that exploit characteristic traces left on acquired images, by the processing pipelines specific of each camera model.
The idea is that each camera model implements a series of characteristic complex operations at acquisition time, from focusing light rays through lenses, to interpolation of color channels by proprietary demosaicing filters, to brightness adjustment and others. These are non-invertible operations, and for this reason they introduce some unique artifacts on the final image, that are footprints that help to detect the used camera model. The problem of camera model identification consists in detecting the model L (within a set of known camera models) used to shoot the image I. A possible solution for this problem is to use a convolutional neural networks (CNN). We will give an overview on CNN in Section 3.2.2.
For further details look at Bondi et al. [6].

2.2.3 Double JPEG compression

When we save an image, usually it is saved in JPEG format. For this reason, when we manipulate the content, a JPEG re-compression happen. Often, manipulation takes place on small parts of the image, and therefore, being able to detect DJPEG on small image patches is important for localization of manipulated regions in image forgery detection problems.
JPEG is an image transform coding technique based on block-wise Discrete Cosine Transform (DCT). An image is split into 8×8 non-overlapping blocks, then each block is DCT transformed and quantized, and at the end the entropy is coded and packed into the bitstream. Quantization cause the loss of informations, and is driven by pre-defined quantization tables scaled by a quality factor (QF). If QF is low, a strong quantization occurs, and the final decompressed image has lower quality. We have a double compression when an image compressed with a quality factor QF1 is first decompressed, and then compressed again with quality factor QF2. If between the two compression no operations are made, we speak of A-DJPEG compression, since 8×8 JPEG blocks of the first and second compressions are perfectly aligned. If

the second compression 8×8 grid is shifted with respect to the previous one, we have a NA-DJPEG compression (e.g. cropping or cut and paste operation between the two compressions). We want to build a detector which is able to classify between single and double compressed images: if H_0 correspond to the hypothesis of single compressed image, and H_1 to the hypothesis of double compressed image, given a $B \times B$ pixel image I, we want to detect if H_0 or H_1 is verified. We consider the following case: only A-DJPEG, only NA-DJPEG or both A-DJPEG and NA-DJPEG. We need to use different approaches because aligned and non-aligned DJPEG compression leave different footprints and then cannot be detected in the same way.
For more details look at Barni et al. [7].

Chapter 3

Problem Formulation

In this Chapter we define the goal of our work and the characteristics of the studied problem, providing a general idea on how we can find the solution. Then, we describe the dataset on which our analysis are made, and at the end we introduce the main background concepts and state-of-the art algorithms.

3.1 Motivation and problem statement

It's always easier to modify the content of an image, thanks to the many image processing software, both for personal computer and smartphones, and for this reason the forensic community has developed a series of techniques to asses image authenticity. Another important problem to be solve from the forensic community is the copyright infringement to ownership attribution. We can detect if an image has been edited because all non-invertible operations applied to a picture leave peculiar footprints, which can be detected to expose forgeries.
Given an image, we want to be able to detect if there are any modified areas, and where they are located.

3.1.1 The dataset

The dataset that we have is composed of a total of 93 images with different dimensions. Each image is divided into N patches, that are made by 64×64 pixel. In Table 3.1 we have the number of images that have a certain number of patches, where N is the total number of patches present in the image and the second row tell us how many images have dimension N. We can see that

N	**408**	**850**	**1230**	**1380**	**1764**	**1850**	**2301**	**2745**	**2852**	**3690**
images	17	21	14	13	3	4	5	7	7	2

Table 3.1: Number of images and dimension

Figure 3.1: Real image and suddivision of the two groups: in blue the original image and in red the modified area

we have a majority of images with small dimension.

A single patch is characterized by the coordinates (y, x), a variable gt that is equal to 0 if the patch belong to the original image, and is equal to 1 if the patch belong to the modified area, and a vector of 419 elements of features $\mathbf{f}$, that represent the characteristics of the patches. The feature vector is extracted from each patch through a slightly modified version of the Convolutional Neural Network presented in [6]. In Fig. 3.1 we have an example of an image, and the true classification. The blue part is the original image, and the red part is the tampered area. In this case we can see that there are two areas that are modified, one is the sky and the other is the crocodile.

3.2 State of the Art

In this section an overview of the literature methods on image tampering detection and localization, and on Convolutional Neural Networks for camera model identification is given.

3.2.1 Image tampering detection and localization

Image tampering detection and localization techniques have been developed focusing, for example, on copy-move forgeries, splice forgeries, impainting and image-wise adjustments (re-sizing, histogram equalization, cropping, etc.). We focus on algorithms that do not need any prior information.

ADQ1 Aligned Double Quantization detection, developed by Lin et al. [8] aims to discover the absence of double JPEG compression traces in tampered 8×8 image blocks. Generate for each block the posterior probabilities to being possibly tampered, through voting among discrete cosine transform (DCT) coefficients histograms collected for all blocks in the image.

BLK Due to its nature, JPEG compression introduces 8×8 periodic artifacts on images that can be highlighted thanks to Block Artifact Grids (BAG) technique from Li, Yuan, and Yu [9]. We can find traces of image cropping (grid misalignment), painted regions and copy-move positions by detecting disturbances in the BAG.

CFA1 At aquisition time, color images undergo Color Filter Array (CFA) interpolation due to the nature of aquisition process. Ferrara et al. [10] build a tampering localization system detecting anomalies in the statistics of CFA interpolation patterns. A mixture of Gaussian distributions is estimated from all the 2×2 blocks of the image, resulting in a fine-grained tamper probability map.

CFA2 Dirik and Memon [11] show, by estimating CFA patterns within four common Bayer patterns, that if none of the candidate patterns fits sufficiently better than the other in a neighborhood, then, it's like tampering occurred. In fact, weak traces of interpolation left by the de-mosaicking algorithm are hidden by typical splicing operations like re-sizing and rotations.

DCT In Ye, Sun, and Chang [12] are detected inconsistencies in JPEG DCT coefficients histograms by first estimating the quantization matrix from trusted image blocks, then evaluating suspicious areas with a blocking artifact measure (BAM).

ELA Error Level Analysis, developed by Krawetz [13], aims to detect parts of the image that have undergone fewer JPEG compression than the rest of the image. It works by re-saving the image at a known error rate and then computing the difference between the original and the re-compressed image. Local minimums in image difference indicate original regions, and local maximums indicate that a tampering occurs.

GHO JPEG Ghosts detection by Farid [14] is an effective technique to identify parts of the image that were previously compressed with smaller quality factors. The method is based on finding local minimum in the sum of squared differences between the image under analysis and its re-compressed version with varying quality factors.

NOI1 It's a tampering detector build by Mahdian and Saic [15] upon the hypothesis that usually Gaussian noise is added to tampered regions, with the goal of deceiving classical image tampering detection algorithms. A segmentation method based on homogeneous noise levels is used to provide an estimate of the tampered regions within an image, modeling the local image noise variance trough wavelet filtering.

NOI4 Fontani et al. [16] presented a statistical framework aimed at fusing several sources of information about image tampering. Information provided by different forensic tools yields to a global judgement about the authenticity of an image. Outcomes from several JPEG-based algorithms are modeled and fused using Dempster-Shafer Theory of Evidence, being capable of handling uncertain answers and lack of knowledge about prior probabilities.

3.2.2 CNN for Camera Model Identification

Convolutional Neural Networks are complex nonlinear interconnection of neurons inspired by biology of human vision system. The use of CNNs in forensics is recent. Inspired by results obtained by Bondi et al. [17], we take the proposed CNN as a feature extractor for each patch of the input image. The structure of a CNN is divided into several blocks, called layers. Each layer $\mathcal{L}_i$ takes as input either an $H_i \times W_i \times P_i$ feature map or a feature vector of size P_i and produces as output either an $H_{i+1} \times W_{i+1} \times P_{i+1}$ feature map or a feature vector of size P_{i+1}.
There are various type of layer, for example:

- **Convolutional layer** Performs convolution, with stride S_h and S_w along the first two axes, between input feature maps and a set of P_{i+1} filters with size $K_h \times K_w \times P_i$. Output feature maps have size

$$H_{i+1} = \left\lfloor \frac{H_i - K_h + 1}{S_h} \right\rfloor,\ W_{i+1} = \left\lfloor \frac{W_i - K_w + 1}{S_w} \right\rfloor,\ and\ P_{i+1}.$$

- **Max-pooling layer** Perform maximum element extraction, with stride S_h and S_w along first two axes, from a neighborhood of size $K_h \times K_w$ over each $2D$ slice of input feature map. Output feature maps have size

$$H_{i+1} = \left\lceil \frac{H_i - K_h + 1}{S_h} \right\rceil,\ W_{i+1} = \left\lceil \frac{W_i - K_w + 1}{S_w} \right\rceil,\ and\ P_{i+1} = P_i.$$

- **Fully-connected layer** Performs dot multiplication between input feature vector and weights matrix with P_{i+1} rows and P_i columns. Output feature vector has P_{i+1} elements.

- **ReLU layer** Performs element-wise non linear activation. Given a single neuron x, it is transformed in a single neuron y with $y = max(0, x)$.

- **Softmax layer** Turns an input feature vector to a vector with the same number of elements summing to 1. Given an input vector x with P_i neurons, $x_j \in [1, P_i]$, each input neuron produces a corresponding output neuron $y_j = \frac{e^{x_j}}{\sum_{k=1}^{k=P_i} e^{x_k}}$.

The network proposed by Bondi et al. [17] is an 11 layer CNN structured as follows:

- An RGB color input patch of size 64×64 is fed as input to the first Convolutional layer with kernel size $4 \times 4 \times 3$ producing 32 feature maps as output. Filtering is applide with stride 1.

- The resulting $63 \times 63 \times 32$ feature maps are aggregated with a Max-Pooling layer kernel size 2×2 applied with stride 2, producing $32 \times 32 \times 32$ feature maps.

- A second Convolutional layer with 48 filters of size $5 \times 5 \times 32$ applied with stride 1 generates $28 \times 28 \times 48$ feature maps.

- a Max-Pooling layer with kernel size 2×2 appied with stride 2 produces a $14 \times 14 \times 48$ feature maps.
- A third Convolutional layer with 64 filters of size $5 \times 5 \times 48$ applied with stride 1 generates $10 \times 10 \times 64$ feature map.
- A Max-Pooling layer with kernel size 2×2 applied with stride 2 produces a $5 \times 5 \times 64$ feature map.
- A fourth Convolutional layer with 128 filters of size $5 \times 5 \times 64$ applied with stride 1 generates a vector of 128 elements.
- A fully connected layer with 128 output neurons followed by a ReLU layer produces the 128 element feature vector.
- A last fully connected layer with N_{cams} output neurons followed by a Softmax layer acts as logistic regression classifier during CNN training phase. N_{cams} is the number of camera model used at CNN training stage.

A shown in Bondi et al. [17], the trained CNN is able to extract meaningful information about camera models even from images belonging to cameras never used for training the network. This property enables exposing forgeries operated also with unknown camera models.

3.2.3 Splicebuster

Splicebuster is a feature-based algorithm to detect image splicing, proposed by Cozzolino, Poggi, and Verdoliva [18]. Splicing is the insertion of material taken from a different source, without any prior information on the host camera, on the splicing, or on their processing history.
The problem of methods based on machine learning and feature modeling, is that they need a large training set.
The approach used to localize possible forgeries in the image is based on three major steps:

- defining an expressive feature that captures the traces left locally by in-camera processing;
- based on a suitable training set, compute synthetic feature parameters (mean vector and covariance matrix) for the class of images under test;

- use these statistics to discover where the features computed locally depart from the model, pointing to some possible image manipulation.

We have a single image with no prior information, so we must localize the forgery and estimate the parameters of interest at the same time. We consider the supervised case, in which the user has to select a tentative training set to learn the model parameters, and the unsupervised case, where simultaneous parameter estimation and image segmentation are done jointly by means of the expectation-maximization (EM) algorithm.

Co-occurrence based local feature Feature extraction is base on three main steps:

1. computation of residuals through high-pass filtering: we use a linear high-pass filter of the third order, that is defined as $r_{ij} = x_{i,j-1} - 3x_{i,j} + 3x_{i,j+1} - x_{i,j+2}$, where x and r are origin and residual images, and i, j indicate spatial coordinates. This type of filter assure us a good performance for both forgery detection and camera identification.

2. quantization of the residuals: we perform quantization and truncation as $\hat{r}_{ij} = trunc_T(round(r_{ij}/q))$[1], where q is the quantization step and T the truncation value.

3. computation of a histogram of co-occurrences, on four pixels in a row, that is: $C(k_0, k_1, k_2, k_3) = \sum_{i,j} I(\hat{r}_{i,j} = k_0, \hat{r}_{i+1,j} = k_1, \hat{r}_{i+2,j} = k_2, \hat{r}_{i+3,j} = k_3)$, where $I(A)$ is the indicator function of event A, equal to 1 if A holds and equal to 0 otherwise.

Supervised scenario In this case, the user has an active role in the process, by selecting a bounding box that will be subject to the analysis (including the possible forgery); the rest of the image is used as training set. The N blocks taken from the training area are used to estimate mean and covariance of the feature vector $\mathbf{h}'$, extracted from each block of the test area, as follow:

$$\mu = \frac{1}{N} \sum_{n=1}^{N} \mathbf{h}_n$$

[1]Remember that truncation is defined in the following way: given a number $x \in \mathbb{R}_+$ to be truncated and $n \in \mathbb{N}_0$ the number of elements to be kept behind the decimal point, the truncated value of x is: $trunc_n(x) = \frac{\lfloor 10^n \cdot x \rfloor}{10^n}$

$$\Sigma = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{h}_n - \mu)(\mathbf{h}_n - \mu)^T$$

Then the Mahalanobis distance with respect to the reference feature μ is computed:

$$D(\mathbf{h}', \mu; \Sigma) = (\mathbf{h}' - \mu)^T \Sigma^{-1} (\mathbf{h}' - \mu)$$

The greater the distance, the more the blocks deviate from the model. Then, the user provide an output map, where to each block is given a color associated with the computed distance.

Unsupervised scenario In this case, after the feature extraction phase, we use an automatic algorithm to jointly compute the model parameters and the two-class image segmentation. As input, we need the mixture model of the data (number of classes, their probabilities, and the probability model of each class). Consider two cases for the class models:

1. both the original area (hypothesis H_0) and the tampered area (hypothesis H_1) are modeled as multivariate Gaussian:
   $p(\mathbf{h}) = \pi_0 \mathcal{N}(\mathbf{h}|\mu_0, \Sigma_0) + \pi_1 \mathcal{N}(\mathbf{h}|\mu_1, \Sigma_1)$;

2. the genuine area of the image is modeled as Gaussian, while the tampered area is modeled as Uniform over the feature domain Ω,
   $p(\mathbf{h}) = \pi_0 \mathcal{N}(\mathbf{h}|\mu_0, \Sigma) + \pi_1 \alpha_1 I(\Omega)$;

The Gaussian model is a simplification, and is conceived for the case when the edited area is relatively large with respect to the whole image.
However, when the modified region is very small the intra-class variability may become dominant with respect to inter-class differences, leading to wrong results. Therefore, we consider Gaussian-Uniform model, which can be expected to deal better with these situations, and in fact has been often considered to account for the presence of outliers.

Chapter 4

Proposed solutions

In this Chapter we will describe the Bagging-Voronoi algorithm, showing its structure, the theoretical details and the properties.
In Section 4.1 the Bagging-Voronoi algorithm for clustering spatially dependent data is introduced and described. In section 4.2 the algorithm is explained in detail. In Section 4.3 are shown the changes to be applied to the algorithm in presence of data whose geometry is known and finally, in Section 4.4 we propose a set of improvements.

4.1 Geometry-aware clustering algorithm

Let call site $\mathbf{x}$ each patch of our image, that are the elements that we want to group using the clustering algorithm, and let be the latent label $l \in \{1, \ldots, L\}$ be the group assigned to the site $\mathbf{x}$. We call the whole image S_0.
The problem consists in associating to each site $\mathbf{x} \in S_0$ a latent label $l \in \{1, ..., L\}$, such that sites that present similar characteristics are labeled the same. We aim to identify different homogeneous macro-areas. Using the standard clustering techniques, such as K-means, do not properly account for spatial dependence, and so, when we assign a site to a cluster, information carried by neighboring sites are not considered. For a given number K of clusters, the proposed spatial clustering procedure generates and analyzes bootstrap samples in three basic steps: generation of a spatial random Voronoi tessellation with dimension n, identification of a local representative for each of the n tessellation, considering all the elements contained in the tessellation and clustering of the local representatives.

Lets describe the Bagging-Voronoi algorithm, following the pseudo-code scheme in Algorithm 1.

Algorithm: Bagging-Voronoi classifier.

Data: A matrix that contain, for each patch, the coordinate and the characteristics

input : B,n,θ,K

output: A final classification map

Bootstrap:

for $b \leftarrow 1$ **to** B **do**

step 1. randomly generate a set of nuclei $\phi_n^b = \{Z_1^b, ..., Z_n^b\}$ among the sites in S_0: for $i = 1, ..., n$, $Z_i^b \overset{iid}{\sim} U(S_0)$, where U is the uniform distribution on the lattice. Obtain a random Voronoi tessellation of S_0, $\{V(Z_i^b|\phi_n^b)\}_{i=1}^n$, by assigning each site $\mathbf{x} \in S_0$ to the nearest nucleus Z_i^b, according to the euclidean distance;

step 2. for $i = 1, ..., n$, compute the local representative g_i^b, by summarizing information carried by the data associated to sites belonging to the i-th element of the tessellation $V(Z_i^b|\phi_n^b)$;

step 3. perform clustering on the local representatives $\{g_1^b, \ldots, g_n^b\}$ using an unsupervised method;

end

Aggregation

step 1. perform cluster matching: for $k = 1, ..., K$, and $b = 1, ..., B$; indicate with C_k^b the set of $\mathbf{x} \in S_0$ whose label is equal to k at the b-th replicate, and match the cluster labels across bootstrap replicates, to ensure identifiability;

step 2. assign each $\mathbf{x}$ to the most frequent label;

step 3. compute spatial entropy η_x^K for each site $\mathbf{x} \in S_0$;

Algorithm 1: Pseudo-code of the Bagging-Voronoi classifiers algorithm

Suppose a latent field of labels $\Lambda_0 : S_0 \to \{1, ..., L\}$ is defined on the image S_0; $\Lambda_0(\mathbf{x})$ is the true unknown label associated to the site $\mathbf{x} \in S_0$,

where S_0 is a subset of $\mathbb{R}^p$. In our case S_0 is a subset of $\mathbb{R}^2$, and represents the whole image. A single image contains N different patches, that are made by 64×64 pixels. So each site $\mathbf{x}$ corresponds to a single patch.

For each site $\mathbf{x}$ we have some information, given in a vector of features $\mathbf{f} \in [0, 1]$, that sum up to 1. The algorithm perform first a bootstrap sampling phase, that is divided in three steps, and then an aggregation phase, where at each iteration of the algorithm a single weak classifier is found, which exploits a specific structure of spatial dependence. After B different runs of the algorithm, bagging together all single classifiers, we can obtain a more accurate global classifier.
In step 1 of the bootstrap sampling part, we generate a random Voronoi tessellation, to capture potential spatial dependence. In step 2 we identify a local representative for each tessellation, summing up local information from all the elements contained in the block: we expect that neighboring site are drawn from the same distribution. Then the local representatives are grouped in K clusters according to a suitable unsupervised method (we will use K-means clustering and Partitioning Around Medoids), and all the sites $\mathbf{x}$ contained in the tessellation are labeled the same. We have to perform cluster matching to ensure coherence of cluster assignments across replicates. After having done all the iterations, each site $\mathbf{x}$ is assigned to the most frequent label. The result depends on the choice of some parameters: the number of iterations B (the higher is B less random are the results), the number of tessellation n (the higher is n, the less the spatial dependence is taken into account), the variance of the Gaussian θ (that takes into account the spatial dependence range) and the number of clusters K.
In order to select the most appropriate value of n, the number of cells produced by the Voronoi tessellation, if we don't have any other information on the correct label of the data, we can try to use the spatial entropy criterion, as suggested by Secchi, Vantini, and Vitelli [1]. Consider the frequency distribution of assignment $\pi_{\mathbf{x}} = (\pi_{\mathbf{x}}^1, \ldots, \pi_{\mathbf{x}}^K)$ of each site $\mathbf{x} \in S_0$ to each of the K clusters. The entropy associated to the final classification in the site $\mathbf{x} \in S_0$ is obtained as

$$\eta_{\mathbf{x}}^K = -\sum_{k=1}^{K} \pi_{\mathbf{x}}^k \cdot \log(\pi_{\mathbf{x}}^k) \tag{4.1}$$

which assumes minimum value 0 when there is an r such that $\pi_{\mathbf{x}}^r = 1$ while $\pi_{\mathbf{x}}^k = 0 \; \forall k \neq r$, with $k, r \in \{1, \ldots, K\}$, and a maximum value of $log(K)$ when $\pi_{\mathbf{x}}^k = 1/K$ for $k = 1, \ldots, K$. The use of the index in Eq. (4.1) is based on the

following idea: the more the frequency distribution $\pi_{\mathbf{x}}$ is concentrated on one particular label, the more the classification is precise and stable along replicates. A global evaluation index can be computed as the *average normalized entropy*

$$\eta^K = \frac{\sum_{\mathbf{x}\in S_0} \eta^K_{\mathbf{x}}}{\log(K) \cdot |S_0|} \tag{4.2}$$

including the contribution to the final classification quality of all sites in S_0. For comparison over different choices of K, the quantity $\eta^K_{\mathbf{x}}$ in Eq. (4.2) has been normalized by the factor $log(K)$. We expect the value of η^K to be low if n is properly chosen in accordance to the (unknown) spatial dependence in the latent field of labels: for an optimal choice of n, few elements of the Voronoi tessellation will cross the borders among regions associated to different labels in the latent field, and high values of the entropy will be very localized along these borders. The entropy criterion generally leads to a choice for K more parsimonious than necessary.

4.2 Details on the Bagging-Voronoi algorithm

Now we will see each step of the Bagging-Voronoi algorithm.

4.2.1 Step 1: Voronoi tessellation

We need to unequivocally define a random tessellation, so we have to select a set of points in S_0, where $S_0 \subset R^2$, as nuclei for the Voronoi tessellation. Let $\Phi_n = \{\mathbf{Z}_1, \cdots, \mathbf{Z}_n\}$ be a set of n points in S_0 sampled from a uniform distribution $U(S)$; these points will be the nuclei of the Voronoi tessellation. For each $\mathbf{Z}_i \in \Phi_n$ define the polyhedron

$$V(\mathbf{Z}_i|\Phi_n) = \{\mathbf{x} \in S_0 : d(\mathbf{x}, \mathbf{Z}_i) \leq d(\mathbf{x}, \mathbf{Z}_j), \forall \mathbf{Z}_j \in \Phi_n, i \neq j\} \tag{4.3}$$

to be the Voronoi cell with nucleus $\mathbf{Z}_i$ for the Voronoi tessellation induced by Φ_n, i.e. the set of $\mathbf{x} \in S_0$ which distance $d(\cdot,\cdot)$ from $\mathbf{Z}_i$ is smaller than each other nucleus $\mathbf{Z}_j \neq \mathbf{Z}_i$. The collection $\{V(\mathbf{Z}_i|\Phi_n)\}_{i=1}^n$ forms a Voronoi tessellation of S_0. The final tessellation will depend on the choice of the metric; for our analysis we have taken $d(\cdot,\cdot)$ as the euclidean distance. For more details on the properties of the Voronoi tessellation look at Penrose [19].

The most interesting property for our purposes is the *coverage* property. Consider the collection of Lebesgue measurable sets $\{A_l\}_{l=1}^L$, $A_l \in S_0$ for $l = 1, \ldots, L$, and let $V_i = V(\mathbf{Z}_i|\Phi_n)$; moreover, let

$$A_l^n = \bigcup_{\mathbf{Z}_i \in A_l} V_i \ ,$$

be an approximation of A_l given by the Voronoi cells whose nuclei belong to A_l. The coverage property states that, for $l = 1, \ldots, L$, the set A_l^n is a consistent estimator of the unknown set A_l in the sense that

$$\mu(A_l^n \Delta A_l) \overset{a.s.}{\to} 0, \ n \to \infty, \tag{4.4}$$

where Δ denotes the symmetric difference between two sets and μ is the Lebesgue measure.
This property represents a strong law of large numbers in the context of Voronoi tessellations. The Eq. (4.4) is essential for the validity of our algorithm, since it states that, when the tessellation becomes less and less coarse, subsets of the domain S_0 associated to the same label are well approximated by suitable sets of Voronoi elements. Indeed, we define the collection of sets $\{A_l\}_{l=1}^L$ by setting, for $l = 1, \ldots, L$,

$$A_l = \{\mathbf{x} \in S_0 : \Lambda(\mathbf{x}) = l\},$$

4.2.2 Step 2: local representatives

For each element V_i of the Voronoi tessellation, we sum up the information contained in the sub-sample $\{\mathbf{f_x}\}_{\mathbf{x} \in S_0}$, where $\mathbf{f}$ is a vector containing some information for $\mathbf{x}$, by calculating a *local representative* trough a method that exploits spatial dependence of neighboring data.
For $i = 1, \ldots, n$, the local representative g_i of the sub sample $\{f_\mathbf{x}\}_{\mathbf{x} \in V_i}$ is computed as a weighted mean with a Gaussian kernel:

$$g_i(t) = \frac{\sum_{\mathbf{x} \in V_i} w_\mathbf{x}^i f_\mathbf{x}(t)}{\sum_{\mathbf{x} \in V_i} w_\mathbf{x}^i} \ , \tag{4.5}$$

where $w_\mathbf{x}^i$ is a Gaussian weight centered in $\mathbf{Z}_i$ and decreasing with respect to $d(\mathbf{x}, \mathbf{Z}_i)$. Intuitively, we assume that spatial dependence between two sites decreases when the distance between them increases. We take as kernel covariance the matrix $\sigma^2\Theta$, where Θ is a bi-dimensional matrix that has value

$\Theta_{ij} = 0$, if $i \neq j$ and $\Theta_{ij} = \theta$ if $i = j$, and where $\sigma = d_{max}/d_{min}$, with d_{max} and d_{min} the maximum and minimum distance between two nuclei of the Voronoi tessellation, respectively.
The choice of n, that is the number of tessellation to be generated, has an influence on the algorithm behavior. The optimal choice of n is the one that finds a good compromise between variance and bias:

- as n decreases, noise is reduced in the computation of the local representatives in Eq. (4.5), since the sub-samples are, in average, larger (minimal variance). However, at the same time the associated Voronoi tessellation follows less accurately the boundaries in the true latent field of labels, thus including different mixture components in the computation of local representatives (maximal bias). The limiting case is $n \equiv 1$, when all sites in the finite lattice belong to the same Voronoi element, and are all used to compute a single representative.

- as n increases, the resulting Voronoi tessellation approximates more accurately the boundaries of the latent field of labels (minimal bias), but at the same time the variability reduction due to spatial smoothing is smaller (maximal variance). The limiting case is $n \equiv |S_0|$, when all sites in the finite lattice are nuclei, and thus no spatial smoothing is performed.

The optimal value of n determined by this trade-off depends both on the strength of spatial dependence, and on the mixture components of the distribution generating the data.

4.2.3 Step 3: Clustering and aggregation

Once we have obtained the local representatives, we need to search for clusters over them and obtain a classification map. For our problem we have decided to use the K-means algorithm.
At each iteration b of the bootstrap phase of the algorithm, a label $\hat{l}_1^b, \ldots, \hat{l}_n^b$ is assigned for each local representative, where $\hat{l}_i^b \in \{1, 2, \ldots, K\}$ corresponds to the cluster assignment of the representative g_i^b to one of the K clusters, for $i = 1, \ldots, n$. This assignment will be the same for all the sites $\mathbf{x} \in V_i^b$.
At each iteration a different classification map is generated, and the problem is that there can be inconsistency between two different iterations. Let C_k^b be the set of site $\mathbf{x} \in S_0$ whose label is equal to k at the b-th replicate. To

obtain the final classification map, we consider the frequency distribution of assignment of each site to each of the K clusters along the B replicates. To compute this frequency distribution we need to assume that cluster labels $\{C_1^b, \ldots, C_K^b\}$ are coherent with $\{C_1^1, \ldots, C_K^1\}$, for all $b \geq 2$ and $b \leq B$. We can do this performing *cluster matching* between the first replicate and the current one. The algorithm searches the label permutation that minimizes the total sum of the off-diagonal frequencies in the contingency table describing the joint distribution of sites along the first classifications and the current one. It's possible to use also other cluster matching procedure.

4.3 Algorithm knowing the geometry of the data

In this section are shown the changes that we have to do in our algorithm when we know the property of our data. In our case we know that the features are similar in accordance to the cosine similarity distance, that is, for two vectors **A** and **B**:

$$similarity = cos(\theta) = \frac{A \cdot B}{||A||||B||}. \tag{4.6}$$

For our image, at each site $\mathbf{x}$ is associated a vector $\mathbf{f}$ of features, that contain some characteristics for the site; if we consider $\mathbf{f}_1, \ldots, \mathbf{f}_n$ be the vectors that contain, for each feature the value associated to all the sites $\mathbf{x} \in S_0$, then more are $\mathbf{f}_i$ and $\mathbf{f}_j$ similar and more the value of *similarity* will be high.
The idea is to compute the similarity distance on the local representatives, before clustering. Instead using k-means, we use partitioning around medoid, so we can take better into account the description of our data; since the Eq. (4.6) give us a similarity matrix, which elements are $\in [-1, 1]$, first we standardize the result to be in $[0, 1]$, and then we compute the dissimilarity matrix doing $1 - similarity$, as shown in Eq. (2.2)
Let consider the local representative $\mathbf{g}_i^b$, for $i = 1, \ldots, n$ at the iteration b. The matrix of *similarity* is a symmetric matrix of dimension $n \times n$, where n is the number of Voronoi tessellations that we build. We construct the dissimilarity matrix in the following way:

$$\mathbf{R}[i, j] = 1 - \frac{1}{2} \cdot \left(\frac{\mathbf{g}_i^b \cdot \mathbf{g}_j^b}{||\mathbf{g}_i^b||||\mathbf{g}_j^b||} + 1 \right). \tag{4.7}$$

From Eq. (4.7) we can see that more the two local representative would have similar characteristics and more $\mathbf{R}[i,j]$ would be close to 0. On the diagonal, all the elements are equal to 0.

4.4 Improvements

In this section we will explain some improvements that we included in our algorithm.

4.4.1 Bagging-Voronoi exploiting more the euclidean distance

We expect that sites contained in the modified area are all close one to the other, and that there is not a single site that belong to the edited area, if all the neighboring sites belong to the original image. To solve this problem we consider, in addition to the geometry of the data taken into account by the Voronoi tessellation, also the euclidean distance that there is between two sites. So, we will give in input to the pam function a matrix

$$\mathbf{M} = \alpha \cdot \mathbf{R} + (1 - \alpha) \cdot \mathbf{D},$$

where $\mathbf{R}$ represent the matrix with the dissimilarity described in Section 4.3, and take into account the geometry of the data, and

$$\mathbf{D}[i,j] = \begin{cases} \dfrac{1}{\sqrt{\mathbf{Z}_i^b[i,1] - \mathbf{Z}_i^b[j,1])^2 + (\mathbf{Z}_i^b[i,2] - \mathbf{Z}_i^b[j,2])^2}}, & if\ i \neq j \\ 1, & if\ i = j \end{cases}$$

take into account the euclidean distance between the nuclei Z of the i-th and the j-th tessellation, at the b-th iteration. $\mathbf{D}$ is, like $\mathbf{R}$, a symmetric matrix with dimension $n \times n$, and take values $\in (0, 1)$.
Higher is $\mathbf{D}[i,j]$ and smaller is the distance between the nuclei $\mathbf{Z}_i$ and $\mathbf{Z}_j$, and higher is the probability that the two sites belong to the same group. Since the algorithm require as input a dissimilarity matrix, we have to do $\mathbf{D} = \mathbf{I} - \mathbf{D}$, where $\mathbf{I}$ is a matrix that has values equal to 1 in each element.
α is the weight that we give to the geometry of the data and to the euclidean

distance: $\alpha = 0$ means that we are using only the euclidean distance between nuclei, and $\alpha = 1$ means that we are using only the geometry of the data, and correspond to the case explained in Section 4.3.

4.4.2 Changing the acceptance criterion

Up to now, we decided to assign a site $\mathbf{x}$ to the most frequent label. If we consider the local entropy associated to the final classification in each site $\mathbf{x}$, defined in Eq. (4.1), we have that when the site $\mathbf{x}$ is almost always classified in the same way, then this value is small, and when at each iteration we assign the site $\mathbf{x}$ to a different label, then the value of the local entropy would be very high.
If the Bagging-Voronoi cannot detect the two groups, this means that the most frequent label in the site $\mathbf{x}$ is not the correct one, but can be that the entropy in the modified area is very high, because in certain iterations the algorithm classify the related sites in a correct way. It's reasonable to think that this may happen specially when the tampered area is small or divided in two areas, since in the Voronoi block there will be more sites coming from the original image.
The idea is, instead to assign a site $\mathbf{x}$ to the most frequent label, to assign it to the group of the modified area, if the local entropy is greater than a certain value β, and assign it to the group of the original image otherwise.

Chapter 5

Experimental results

In this section the experimental evaluation of our algorithm is provided. First, In Section 5.1, the result of our analysis are shown when the data geometry is not known. In Section 5.2, supposing to know the geometry of the data, the most significant analysis obtained in Section 5.1 are repeated. In Section 5.3 is shown an attempt of improvement carried out by us. In Section 5.4 an interesting study considering the false positive and false negative is made. In Section 5.5 consideration on the optimal number of clusters K is made, and finally, in Section 5.6, a study considering only some images is done.

5.1 Simulation using Bagging-Voronoi algorithm

In this section are shown the results obtained using the Bagging-Voronoi algorithm described in Section 4.1-4.2.
We evaluate our result using the Matthews correlation coefficient (MCC) that is a metric widely used in bioinformatics.

Having defined some metrics in Table 5.1, where Y represent the assigned group and X the original group, we can define the Matthews correlation coefficient as follow:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FN)(TP + FP)(TN + FP)(TN + FN)}}. \tag{5.1}$$

For us, 1 corresponds to the modified area and 0 corresponds to the original area of the image. We decided to use this evaluation metric because it takes

Metric	Definition	Description
TP	P(Y=1,X=1)	True Positive (correctly identified)
TN	P(Y=0, X=0)	True Negative (correctly rejected
TPR	TP/(TP+FN)	True Positive Rate
FN	P(Y=0,X=1)	False Negative
FP	P(Y=1, X=0)	False Positive
TNR	TN/(FP+TN)	True Negative Rate

Table 5.1: Definitions of the elementary metrics used to formulate the evaluation metrics

into account true and false positives and negatives, and it's a metric that can be used even if the classes are of very different sizes, like may happen in our case.
For the first analysis that we made, we fix the number of clusters K equal to 2.
After a set of preliminary tests, we fix a number of iterations B equal to 31, because further increasing this value did not improve the results. Then, we set intuitively the number of tessellation $n = 300$ and the variance of the Gaussian distribution $\theta = 1$. Using these values for the parameters, we perform a simulation of all the 93 images contained in our dataset, and we obtain a mean MCC equal to 0.542, that represent our baseline.

5.1.1 Optimal length of the tessellation

The aim of this first test is to search for an optimal number of tessellation to select, inspired by what was done by Volpe [20] . Since in our dataset we have images of different sizes, instead to consider the number of tessellation n to generate we consider the average length of a block l, that is defined as $l = N/n$, where N is the dimension of the whole image, i.e. the number of patches that we have. Therefore, the optimal number of tessellation will be $n = \lfloor N/l \rfloor$.
We consider $l = \{1, 3, 5, 10, 15, 25, 40, 60, 90, 125, 200, 400\}$ and we evaluate, for each length l of each image, the MCC and the *average normalized entropy*, that from now we will call ANE and that was defined in Eq. 4.2.
We want to verify if the ANE is a good criterion for the choice of the optimal

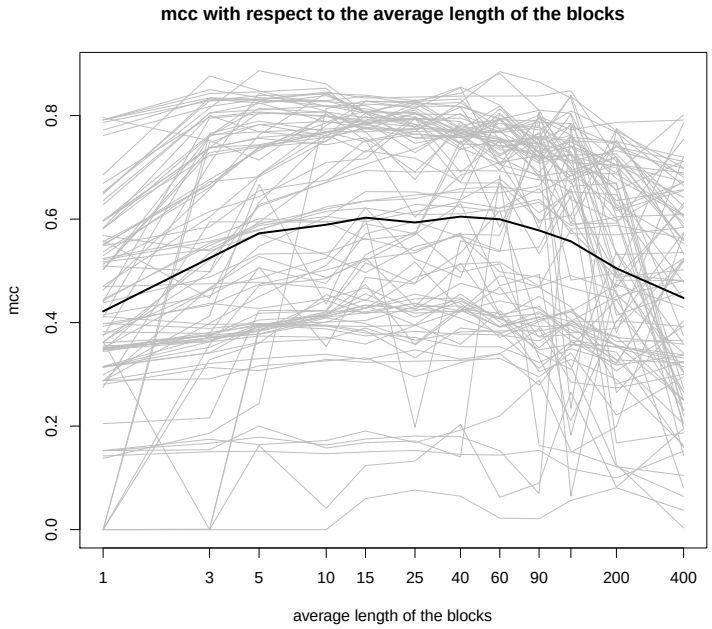

Figure 5.1: MCC value with respect to the average length in a block, considering B=31, K=2 and Gaussian variance equal to 1

average length, as suggested by Secchi, Vantini, and Vitelli [1] and verified by Volpe [20], also when real data are used instead of synthesized ones.. We obtain the result shown in Figure 5.1, where the black lines represent the mean values and the gray lines represents the value obtained for each single image.

Looking at Figure 5.1 we can see that at the beginning the mean value of the MCC is increasing, then it remains approximately constant between $l = 5$ and $l = 60$ and then it decreases. This trend justifies the advantages given by using the Bagging-Voronoi algorithm, since for $l = 1$ (that means that each tessellation contain a single element) we have smaller values than taking larger values of l. Otherwise, if a single tessellation has to many elements (when l increases), when we compute the local representative we are losing too much information, thus, also in this case the results are not so good.
The mean curve present a maximum for $l = 15$, that correspond to $MCC = 0.603$, that is greater than the value found in the first simulation done with a generic $n = 300$. Although the mean curve has an ideally shape, we cannot

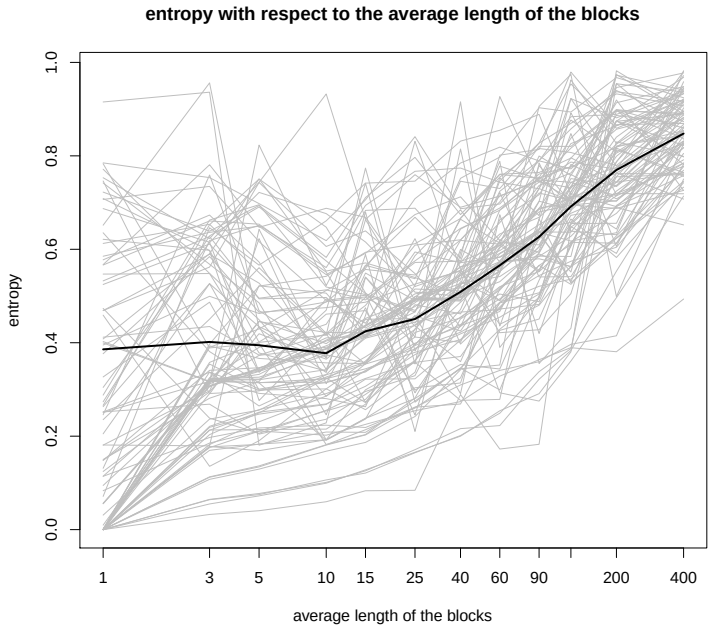

Figure 5.2: MCC value with respect to the average length in a block, considering B=31, K=2 and Gaussian variance equal to 1

say this for the gray lines, which are the values of the MCC for the single images. In particular, we can see that there are some images for which the algorithm it's not able to detect the tampered area for all the values of l and other images for which the lines fluctuate. Looking at the images for which we have obtained the best results we notice that the MCC is not much greater than 0.8.

Let analyze now Figure 5.2, that shows the ANE with respect to the average length of a block. We notice that the mean line starts in a flat way, presents a minimum for $l = 10$ and then increases. For the mean point in $l = 1$ we have not considered the points with $ANE = 0$, that are the points clustered, at each iteration, in the same way. When l is large, also the value of the ANE is very high, and this make sense because the local representatives that we build have a lot of variance, and for this reason the result depend a lot of the random initialization of the nuclei; the result is that most of the patches are assigned at one or another group with high uncertainty.
Observing the gray lines, that correspond to the single images, we notice that there are few images for which the ANE value is low, and other images for

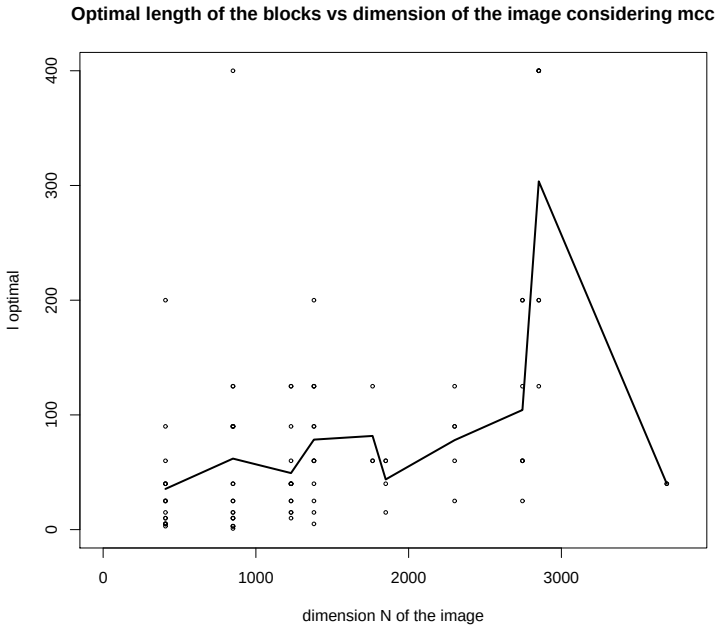

Figure 5.3: l optimal vs dimension N of the image, considering B=31, K=2 and Gaussian variance equal to 1

which the ANE value oscillates.
Looking at the two plots it seems that there is a small relation between the goodness of classification and the ANE, since taking the smallest value for the mean ANE, that is $l = 10$, we have one of the highest value of the MCC (we obtain 0.589 versus 0.603 obtained with $l = 15$). The problem is that this reasoning is true only looking at the mean line. For the single images it is more complicated to search for the optimal value, and it's not always true that taking the minimum value from the ANE and selecting the corresponding l, give us an high MCC.

5.1.2 Optimal length of the tessellation with respect to the dimension of the image

As we have seen, our dataset contain images that have different dimension. A question we asked ourselves is if there is a relation between the optimal average length of a block and the dimension of the image. We proceed in the following way: first we calculate all the values of MCC for all the average

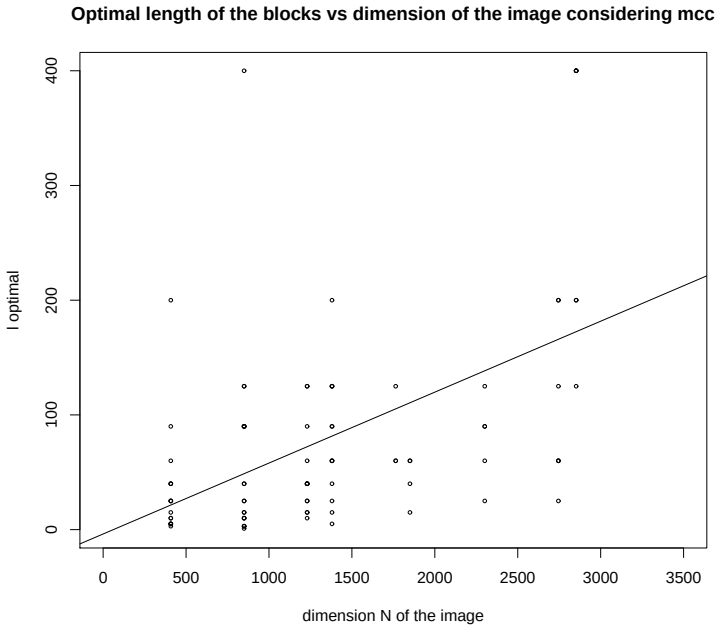

Figure 5.4: l optimal vs dimension N of the image, considering B=31, K=2 and Gaussian variance equal to 1, with the regression line

lengths of the block $l = \{1, 3, 5, 10, 15, 25, 40, 60, 90, 125, 200, 400\}$; then, for each image, we have searched the highest value obtained in terms of MCC, and we have saved the corresponding optimal average length l^* of the block. The result is the plot in Figure 5.3, where the black line represents the mean optimal length of the tessellation for each N. We can notice that for the same dimension N of the image, we have very different average optimal length of the tessellation. This happen because even if two images have the same dimension N, their content and the dimension of the edited area may be very different, and so, for some images K-means works better with small l while for the other K-means work better with a larger l.
If we don't consider the points for $N = 3680$, that are only 2, it seems that there is a little increasing of l^*, when N increases. We can see that there are some outliers like the two points which result is $l^* = 400$.
The idea was to search for a regression line between, but looking at Figure 5.3 it seems that there is not a linear relation between the optimal average length of the tessellation and the size of the image. To confirm our suppositions, we build it, and the best regression model that we get, after trying also to not

consider the outliers, is the one in Figure 5.4, where we don't have considered the points in $N = 3680$. For this model we obtained a $R^2 = 0.2697$ that is very low: this means that only 26.97% of all the data are well explained by this model. This regression line correspond to the following equation:

$$l^* = -3.80555 + 0.06182 \cdot N \tag{5.2}$$

We also tried to do this analysis considering, instead of the highest value of the MCC for each image, the first 3 highest value of the MCC, if they where sufficiently good, because for some images the Bagging-Voronoi algorithm gives good result for more than one value of l. But also in this case there is not a relation between the optimal value of l and the dimension N of an image.
Since this result is not significant, we will no longer consider this type of analysis, and we will set $l = 15$ for all the following experiments.

5.1.3 Gaussian variance choice

In this section we will do some considerations about the variance of the Gaussian, that is another parameter that may change our results. Recall that the local representatives g_i, for $i = 1, \dots, n$ of the sub-sample $\{f_{\mathbf{x}}\}_{\mathbf{x} \in V_i}$ were computed as a weighted mean with a Gaussian kernel (see Eq. 4.5).
Taking a smaller value of the variance, means that we are shrinking the Gaussian kernel, and that the values near the nuclei (in our case the Gaussian distribution is centered in the nuclei) are considerate much more than distant points. Otherwise, taking a bigger value of the variance, means that we are spreading the Gaussian kernel, and there is less difference in the weight of the values of $f_{\mathbf{x}}$ between near and far points from the nuclei.
We consider $B = 31$, $l = 15$ (that was the best value that we obtained in Section 5.1.1), $K = 2$, and we computed, for all the 93 images and for the Gaussian variance $\theta = (0.5, 1, 5, 10)$, the mean value of the MCC and of the ANE. We obtained the results shown in Table. 5.2

Looking at this results, we can see that it seems that an higher variance give us better results. This happen because we have, on average, $l = 15$ elements par tessellation, and a too small variance cannot be able to consider appropriately all the elements of the block.
An interesting thing is that it seems to be a relation between the optimal variance value and the ANE: in fact, except for $\theta = 10$, where the increment

variance	MCC	ANE
0.5	0.57769	0.48765
1	0.60573	0.41637
5	0.61596	0.33820
10	0.61789	0.34688

Table 5.2: Value of MCC and ANE when the variance of the Gaussian varies

of the MCC with respect to $\theta = 5$ is very low, higher is the MCC and lower is the ANE. So, if we have a set of images without any prior information, a good way to select the optimal value of the Gaussian variance is taking the value that correspond to the lowest ANE, since when the value of the MCC increase a bit, the ANE decreases much more.

5.1.4 Final consideration

The significant result that we obtained from this analysis are the following:

- looking at the minimum mean value of the ANE, we can select the optimal value of the length of the tessellation;
- a good average length of the tessellation is $l^* \in (5, 90)$;
- there is no relation between the number of tessellation to take and the dimension of the image;
- the variance of the Gaussian kernel may influence the results, and can be selected by taking the minimum value of the ANE;

A final consideration is that deal with real data is not easy; our results are fine if we look at all together, but if we consider the single images, the results can be very good or not.

5.2 Simulation knowing the geometry of the data

In this section we suppose to known the geometry of our data; in our case the data are distributed according to cosine similarity distance. We have to

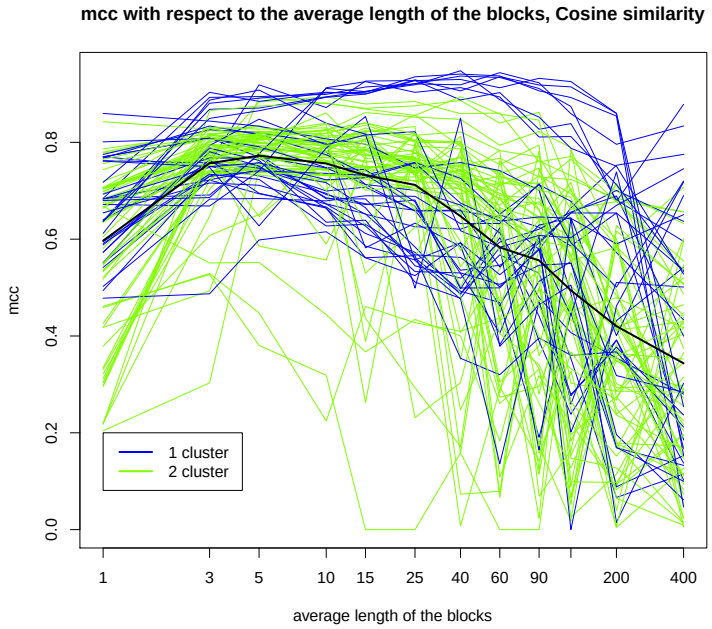

Figure 5.5: MCC value with respect to the average length in a block, considering the cosine similarity distance and B=31, K=2 and a Gaussian variance equal to 1

make some small changes in the algorithm, as described in section 4.3.
As in section 5.1, at first we do a simulation of all the 93 images considering $B = 31$ iterations, a dimension of the tessellation $n = 300$, the number of cluster $K = 2$ and a Gaussian variance $\theta = 1$. We obtain, taking the mean MCC of all the images, an $MCC = 0.7534923$. In section 5.1 the MCC value was 0.542; this means that if we know how our data are distributed, then we can obtain much better results.

5.2.1 Optimal length of the tessellation

In this section we will repeat the analysis done in section 5.1.1, where there was an optimal average length of the tessellation. Consider $l = N/n$ be the average length of a block, with $l = \{1, 3, 5, 10, 15, 25, 40, 60, 90, 200, 400\}$. Let consider also in this case the ANE, that was defined in equation 4.2. The result of this simulation can be see in Figure 5.5. The black line represent the mean MCC value, and we can notice that present a maximum in $l = 5$, equal to 0.77248, that is not much greater than the preliminary simulation done with $n = 300$. The optimal value of l is lower than in the situation

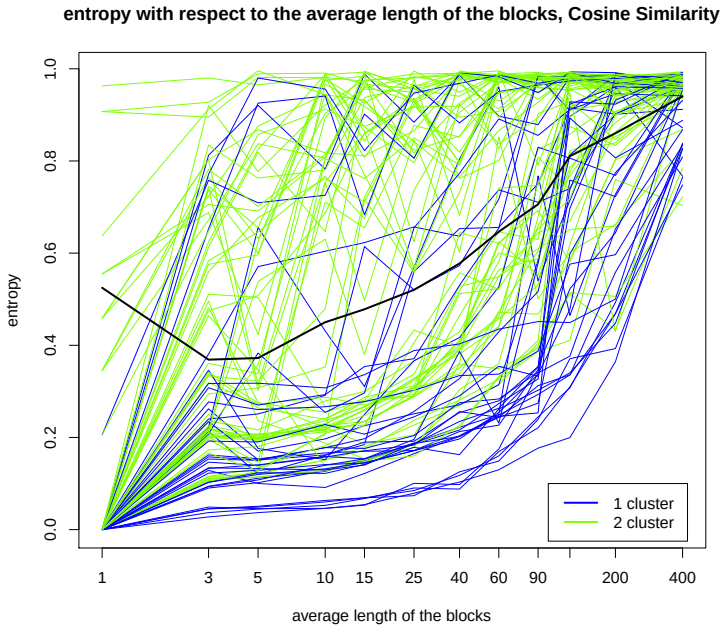

Figure 5.6: ANE value with respect to the average length in a block, considering the cosine similarity distance and B=31, K=2 and.

studied in 5.1.1; the reason is that if we have, on average, too many elements in the tessellation, averaging to compute the local representative we lost the information about the geometry of the data.
We can see that at the beginning this line increase a lot, then it remain stable from approximately $l = 3$ to $l = 25$, and then it decreases. This line, present value much lower for $l = 400$ than for $l = 1$, and this is because knowing the geometry of our data, also without considering the spatial relation between near patches ($l = 1$ case), allows k-means to understand better which patches are similar. On the other hand, if l is high, we are loosing the geometry information by averaging to compute the local representatives. Although the black line presents a good trend, we cannot say the same for the single images (blue and green lines) that we can see fluctuate a lot. The blue lines represent the images where a single area has been modified and the green ones represent the images where two areas have been modified. With respect to the study in Section 5.1.1 there are less images for which the algorithm is always bad. We obtain the worse MCC values for images containing two areas modified. We will see in section 5.6 that the reason is that often the

variance	MCC	ANE
0.5	0.771913	0.408305
1	0.776804	0.381916
5	0.774061	0.395678

Table 5.3: Value of MCC and ANE when the variance of the Gaussian varies

algorithm cannot catch both the areas.
In Figure 5.6 we have the ANE with respect to the average length l in a block. We can see that there is a minimum in $l = 3$, that is much lower with respect to the value obtained in 5.1, that was $l = 15$. When l increases, also the ANE increase. This make sense because increasing l means that we have a lower number of blocks n, and at each iteration we classify incorrectly a lot of patches. In $l = 1$ we have a lot images with $ANE = 0$; this is because at each iteration the algorithm has classified all the patches in the same way. We don't have considered this images, for the mean value in $l = 1$. An interesting thing is the fact that for $l = 1$, more or less all the images that have entropy $\eta \neq 0$ are those that contain two modified areas. This happen because sometimes the algorithm recognizes one group, and other times recognizes the other group.
Looking at the single images, there are a lot of images that present a value of ANE very high, and images with a single modified area present a lower ANE. Taking the value of l that present a minimum for the ANE give us a good result in terms of MCC, but also in this case, like in section 5.1 it's correct only considering the mean values, and not considering the single images.

5.2.2 Gaussian variance choice

We repeat the simulation done in section 5.1.3 about the variance of the Gaussian. In this case we will consider $l = 5$, that is the best value obtained in section 5.2.1, $B = 31$ and $K = 2$. The results are shown in Table 5.3. In this case we obtain the best results when $\theta = 1$, and this value correspond also to the minimum of the ANE. Since having only, on average, 5 elements per block, the variance has less influence on the results with respect to the algorithm without knowing the geometry of the data (where the optimal l

was 15).

5.2.3 Final consideration

From the analysis done in this section, we have obtained the following important information:

- knowing the geometry of the data, give us better results;
- a good average length of the tessellation is $l^* \in (3, 10)$;
- looking at the mean value of the ANE, and selecting the corresponding value of l, we can get good results;
- if we consider a low number of average length of the tessellation, the variance of the Gaussian don't influence too much the results.

5.3 Simulation considering euclidean distance before clustering

In this section we want to show the experiment that we have done considering the algorithm as in section 4.4.1.
As we have done in section 5.1 and 5.2, first we do a simulation using a tessellation $n = 300$, $B = 31$ iterations, $K = 2$ clusters and a variance $\theta = 1$. We obtain a mean $MCC = 0.738578$; this value is smaller of that obtained in 5.2, that was 0.7534923.

5.3.1 Optimal length of the tessellation

As in Section 5.1.1 and 5.2.1 we want to study the trend of the MCC and the ANE when the average length of the tessellation vary, considering $l = \{1, 3, 5, 10, 15, 25, 40, 60, 90, 125, 200, 400\}$.
In Figure 5.7 we have the related plot. The mean value of the MCC, that is represented by the black line, at the beginning increases up to $l = 5$, then is more or less constant from $l = 5$ to $l = 10$, and then it decreases. We have a maximum in $l = 5$, that correspond to 0.7290666, that is a bit lower with respect to the initial simulation with $n = 300$. Looking at the other lines, that correspond to the single images, we can see that there are some lines

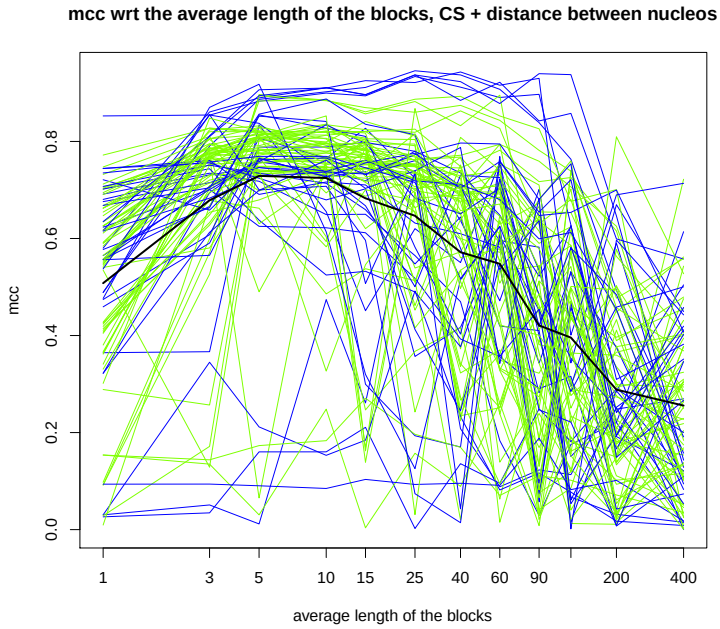

Figure 5.7: MCC with respect to the average length of the tessellation, with B=31, K=2 and a Gaussian variance equal to 1, using the Bagging-Voronoi algorithm weighing the cosine similarity distance and the geometric distance

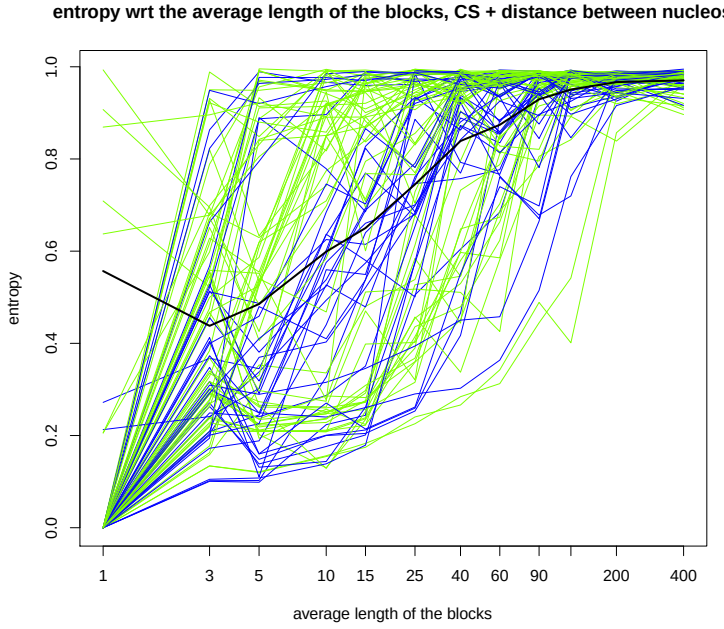

Figure 5.8: ANE with respect to the average length of the tessellation, with B=31, K=2 and a Gaussian variance equal to 1, using the Bagging-Voronoi algorithm weighing the cosine similarity distance and the geometric distance

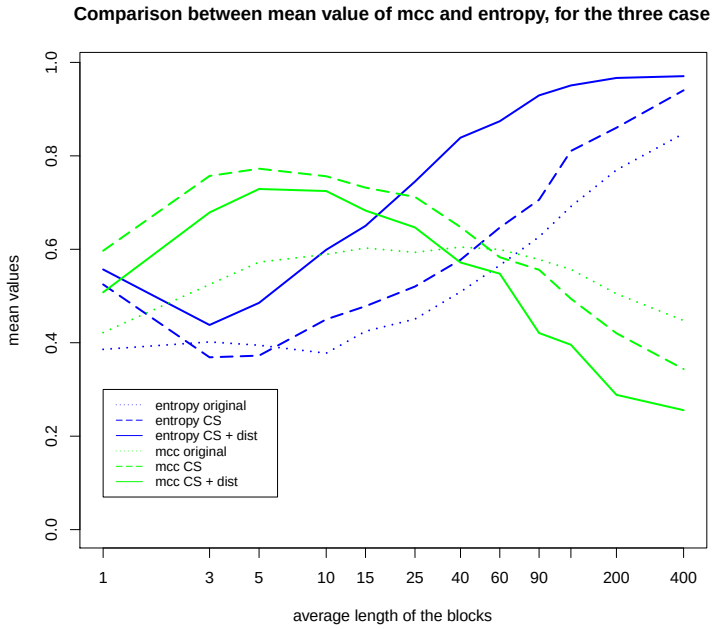

Figure 5.9: comparison between the Bagging-Voronoi simulation whitout knowing the geometry of the data, knowing the geometry, and considering the distance between patches and cosine similarity distance before clustering, with B=31, K=2 and a Gaussian variance equal to 1

which give us not so good results, for each value of l. Here we can see that there is not a division between images with one group (blue lines) or two group (green lines), as in Section 5.2.1.
In this analysis we are considering the weight of the geometric distance and of the cosine similarity distance equally, but it's reasonable to think that the cosine similarity distance is more important than the geometric distance. In section 5.3.2 we will do this analysis.

In Figure 5.8 we have the plot of the ANE with respect to the average length in the tessellation. Looking at the black line, that represents the mean ANE of all the images, we can see that the trend is decreasing at the beginning, present a minimum for $l = 3$, and then always increases. Also in this case, like in Figure 5.6, for $l = 1$, more or less all the images that have entropy $\eta \neq 0$ are those that contain two modified areas.
In figure 5.9 we have the comparison between the three simulations that we have done. The blue lines represent the mean values of the ANE, and the

l	MCC base	MCC CS	MCC d	ANE base	ANE CS	ANE d
1	0.42176	0.59691	0.50795	0.38581	0.52477	0.55685
3	0.52460	0.75696	0.67879	0.40171	**0.36882**	**0.43799**
5	0.57250	**0.77248**	**0.70522**	0.39464	0.37238	0.48413
10	0.58912	0.75633	0.69697	**0.37758**	0.45002	0.54889
15	**0.60273**	0.73196	0.69205	0.42449	0.47816	0.59948
25	0.59963	0.71207	0.65731	0.45070	0.52031	0.65564
40	0.60480	0.64760	0.64823	0.50888	0.57743	0.69379
60	0.59963	0.58300	0.59848	0.56572	0.64645	0.72927
90	0.57811	0.55607	0.52386	0.62657	0.70596	0.82159
125	0.5570	0.49424	0.47806	0.69150	0.81048	0.85714
200	0.50481	0.42027	0.40279	0.76971	0.86016	0.92556
400	0.44751	0.34358	0.26406	0.84772	0.94026	0.96532

Table 5.4: Mean value of the ANE and the MCC for the simple Bagging-Voronoi (base), the algorithm when we know the geometry of the data (CS) and the algorithm using the geometry of the data and the geometric distance (d). The best result for each column is highlighted

green ones the mean values of the MCC; the continuous curves are related to the cosine similarity and geometric distance, the dashed lines show the result knowing the geometry of the data, and the dotted ones are the simulation without knowing the geometry of the data. Looking at the MCC, we can see that for small values of l we have that the dashed line is much higher, that means that knowing the geometry of the data increase a lot the goodness of the results; for high values of l, we have that are better the results without knowing the geometry of the data. This happens because if we have a lot of elements in the tessellation, when we compute the local representatives, in averaging over all the patches in the tessellation, we lost the information about the geometry of the data and we make things worse.

Looking at the ANE, we notice that from $l = 10$ the three lines have approximately the same trend (specially the lines related to the first two simulations). An interesting thing is that the ANE is smaller considering the simulation without knowing the geometry of the data, for all the value of l except for $l = 3, 5$ (that are the values that give us the best results). This means that there is less variation between the classification in different

iterations.
In conclusion we can say that knowing the geometry of the data assure us to get better result if we take the value of l where we have a low ANE, always looking at the mean line. This is shown also in Table 5.4, which shows the values of MCC and ANE for the 3 case that we have considered, where we can see that taking the minimum of the ANE give us one of the highest MCC value. We have highlighted the best values for each column: for the MCC are better high values, and for the ANE are better small values.

5.3.2 Study changing the weight between the geometric distance and the cosine similarity

In this section we will study how changes the value of the MCC and of the ANE when we change the weight of the distance. We build the dissimilarity matrix on which we then apply pam algorithm in the following way:

$$\mathbf{M} = \alpha \cdot \mathbf{R} + (1 - \alpha) \cdot \mathbf{D}, \tag{5.3}$$

where $\mathbf{R}$ is the matrix containing the dissimilarities between features and $\mathbf{D}$ is the matrix containing the geometric distance between patches. This means that when $\alpha = 0$ we are using only the geometric distance and when $\alpha = 1$ we are considering only the dissimilarities between features. We want to find an optimal value of $\alpha \in [0, 1]$.

Figure 5.10 shows the values of the MCC when α vary, for images with one modified area (in blue) and images with two modified areas (in green); the black line is the mean value of all the images. As we expected, when $\alpha = 0$ we don't obtain good results. When α increase, our result become more and more better, and from $\alpha = 0.6$ to $\alpha = 1$ we have good results, which are more or less constant. Looking at the single images, we notice that images with two modified areas tend to reach convergence early than the other images.

Figure 5.11 shows the ANE when α vary. We can see that in $\alpha = 0$ the ANE is very high for all the images. This is reasonable because the distance depends on the initialization of the nuclei, that at each iteration is random. Then it rapidly decreases, and stabilizes around $\alpha = 0.8$. Looking at the two plots, we can notice that when the MCC increases, the ANE decreases, but the MCC reach convergence earlier than the ANE. Despite this fact, looking at the mean value, the ANE is a good criterion for take the best value of α: we select the one for which we have the minimum. Looking at

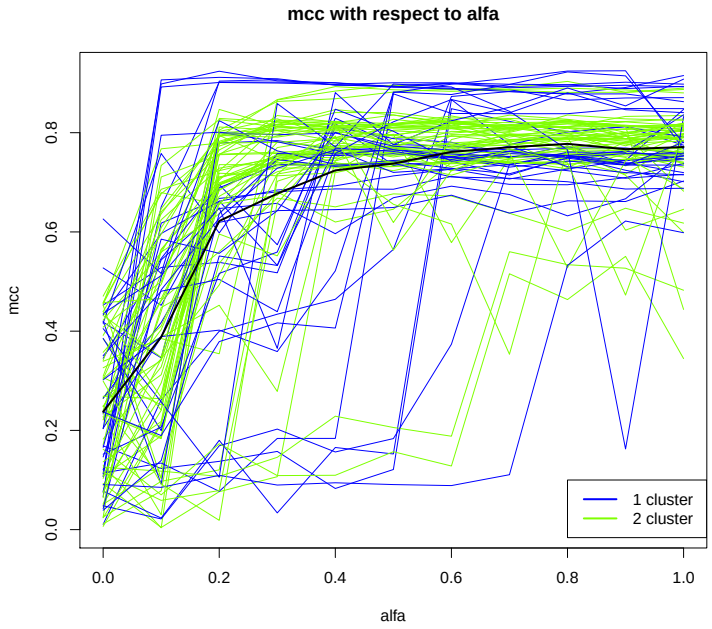

Figure 5.10: MCC varying the weight of the geometric distance and dissimilarities between features, considering B=31, K=2 and a Gaussian variance equal to 1

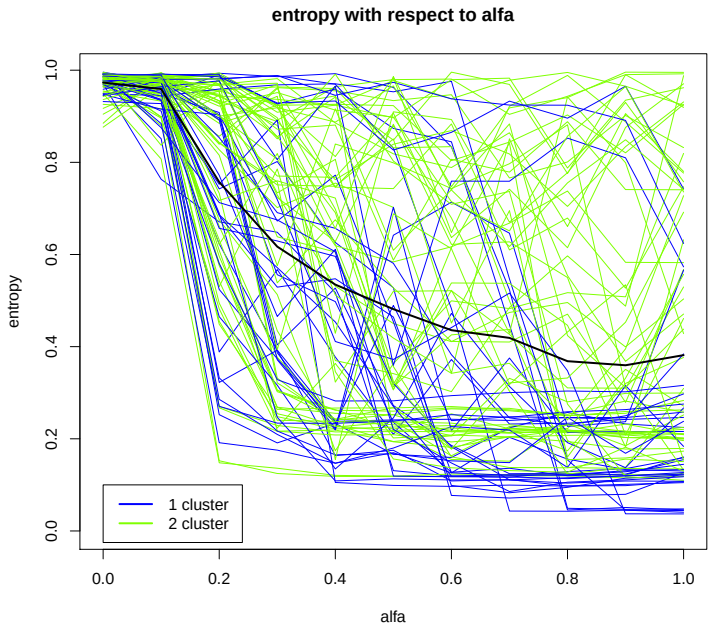

Figure 5.11: ANE varying the weight of the geometric distance and dissimilarities between features, considering B=31, K=2 and a Gaussian variance equal to 1

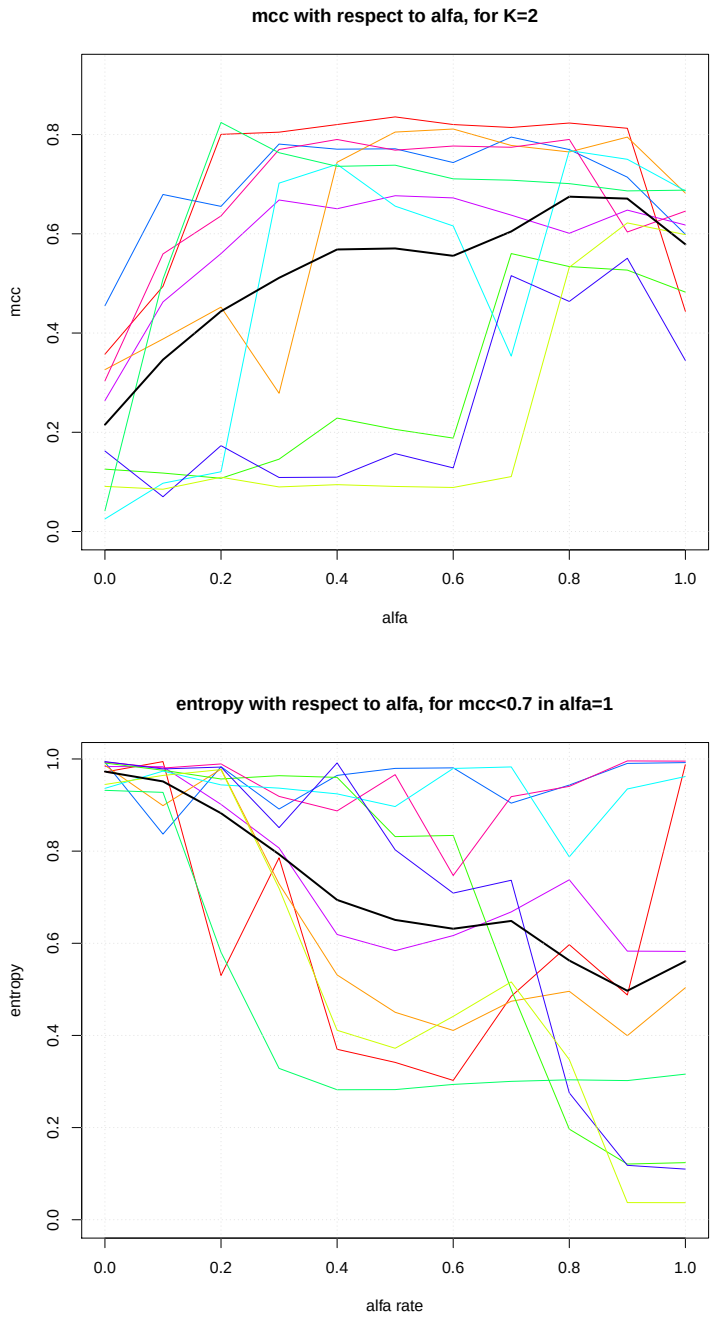

Figure 5.12: MCC and ANE with respect to alfa, for images that in alfa=1 gives an MCC lower than 0.7. The black lines represent the mean value and the colored ones represent the value for the single images

dim	408	850	1230	1380	1764	1850	2301	2745	2852	3690
FN	8	13	9	3	2	2	1	2	1	2
%	47.06	72.22	64.29	25.00	66.67	50.00	25.00	33.33	14.29	100
dim	408	850	1230	1380	1764	1850	2301	2745	2852	3690
FP	0	3	0	0	0	0	0	4	1	0
%	0	16.67	0	0	0	0	0	66.67	14.29	0

Table 5.5: Number of false negative and false positive that occur, and the percentage value of all the images with this dimension.

the two groups of images, we can see that for images with a single modified area the ANE goes down to low values more faster than for the other images, except for some.

In Figure 5.12 we have the MCC and the ANE for those images that for $\alpha = 1$ gave us an $MCC \leq 0.7$. The black line represents the mean value and the colored ones are related to the single images. From the first plot we can see that taking $\alpha = 0.8$ or $\alpha = 0.9$ we have greater results. Looking at the second plot, we can notice that the mean value present a minimum for $\alpha = 0.9$, for which corresponds a great value of MCC. Looking at the colored lines, there seems to be a relation between the maximum value of the MCC and the minimum value of the ANE: except for the blue line, where small values of the ANE don't correspond to high value in the MCC, all the lines seems to present the bests value of the MCC in correspondence to the lowest values of the ANE. So, having a single image, we can select the optimal value of α taking that value that minimize the ANE.

5.4 False positives and false negatives rate

In this section we want to study which type of error we do, to see if it is better to change the threshold used to classify the patches.
Let consider the *cosine similarity* case, which was the situation that gave us the best results. We fix $l = 5$, $B = 31$, $K = 2$ and $\theta = 1$, and we do a simulation over the 87 images, without considering the images that give us an $MCC \leq 0.7$. We obtain a mean rate of $FN = 0.6724536$ and $FP = 0.327544$, where FN and FP are defined in Table 5.1.

In Table 5.5 are reported the number of images, divided by the num-

ber of patches, that present a miss-classification error type of false negative (patch that would belong to the modified area are classified as belonging to the original image) or false positive (patch classified as modified but which would belong to the original image) greater than 80% of all the miss-classified patches.
From Table 5.5 we can see that there is a majority of false negative. Another interesting observation is that all the images with dimension 3690, 2745, 850 obtain a total miss-classification error with a predominance of a type of error (predominance of false negative for the images with 3690 patches, and mixed predominance for the other two).
Having an high number of false negative means that we are not so able to detect the tampered area, so we have to lower the acceptance threshold: we now assume that a patch belongs to the modification if has been classified in this way for 40% of the B iterations. Computing the false positive and false negative rate for all the images we obtain a false negative rate of 49.39 and a false positive rate of 50.61. Doing again a simulation of all the images, with $l = 5$, we obtain a mean $MCC = 0.7$, so, balance the number of false positive and false negative do not improve the results.

5.4.1 Receiver Operating Characteristic curve

The Receiver Operating Characteristic (ROC) curve is used to evaluate machine learning algorithms. The area under the ROC curve (AUC) can be used as a measure of binary classifier performance. We use this curve to select the optimal threshold of assignment. In Table 5.6 we have the classification of all the patches, that are 130702, when the threshold of acceptance was 0.5. This give us an accuracy of $(26017 + 76046)/130702 = 0.78088$, that means that 78% of all the patches, in all the images, are classified correctly.
For build the ROC curve we need the sensitivity, that is defined using the

X\Y	**1**	**0**
1	26017	20695
0	7944	76046

Table 5.6: Confusion matrix considering all the 93 images, that contain a total of 130702 patches

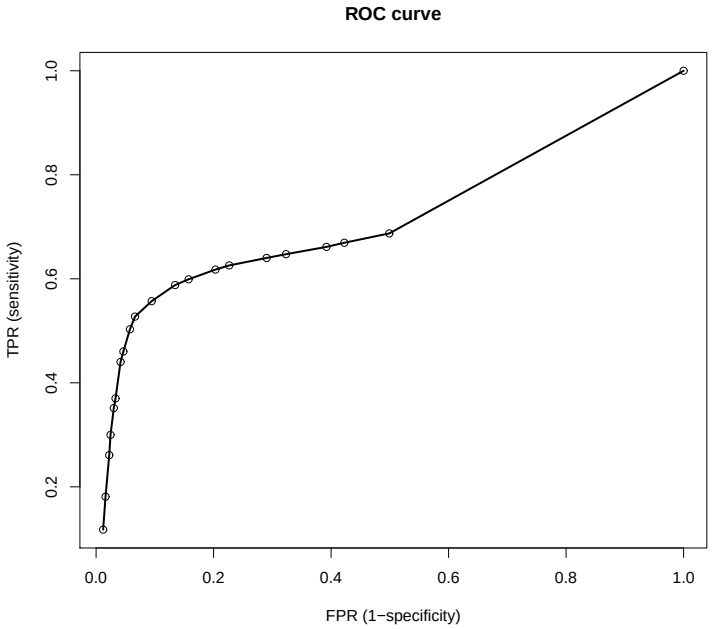

Figure 5.13: ROC curve

metrics given in 5.1 as:

$$sensitivity = TPR = \frac{TP}{TP + FN} = 0.55697,$$

and the specificity that is:

$$specificity = 1 - FPR = 1 - \frac{FP}{FP + TN} = 0.90542$$

High sensitivity means that we are using a low threshold, and so we pick up a large number of edited patches but also a many original patches; high specificity indicate that we are using an high threshold, that means that we select correctly the modified patches, but we don't see a lot of other modified patches. A good fit of the model is when both sensitivity and specificity are high. Here we have an high specificity but a not so good sensitivity.
We generate the ROC curve by plotting the TPR against the FPR at various threshold. More high is the AUC value and better is the model (1 is the ideal value). In Figure 5.13 we can see the ROC curve related to our data. Computing the value of the AUC, we get 0.7171914, that is not so high, so the model can surely been improved.

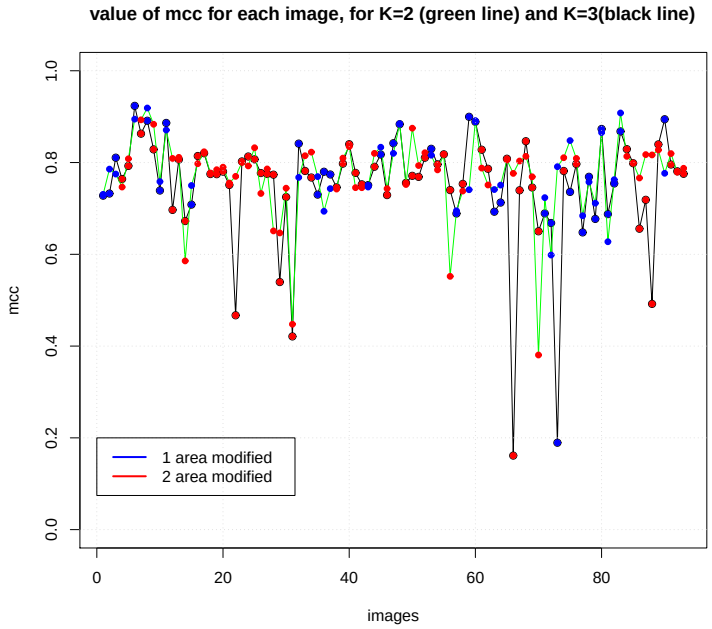

Figure 5.14: MCC values for all the images, for K=2 or K=3, B=31, l=5, Gaussian variance equal to 1

5.5 Choice of the number of clusters K

In this section is shown a comparison between setting $K = 2$ or $K = 3$ clusters. We consider the *cosine similarity* case, which was the best one.

In the images with two modified areas, despite we have a total of 3 groups, we have only the variable 0 for the original area and 1, for the tampered area. Once we have obtained the final classification, since we are using the algorithm with $K = 3$ we have 3 labels. We need to group two of them in the most appropriate way, and we do this by analyzing all the possible combination and taking the one that maximize the MCC, supposing that the user can look at the original picture and choice the aggregation more reasonable. In Figure 5.14 we can see the MCC for all the images; we have highlighted the images with one modified area (in blue) and those with two modified area (in red). The green lines represent the simulation done with $K = 3$, and the black lines represent the simulation with $K = 2$.

We expected that the simulation with 3 clusters give us better result when we have effectively 3 groups (the original image, the first modified area and the second one). Effectively seems to be so, since those points that give a

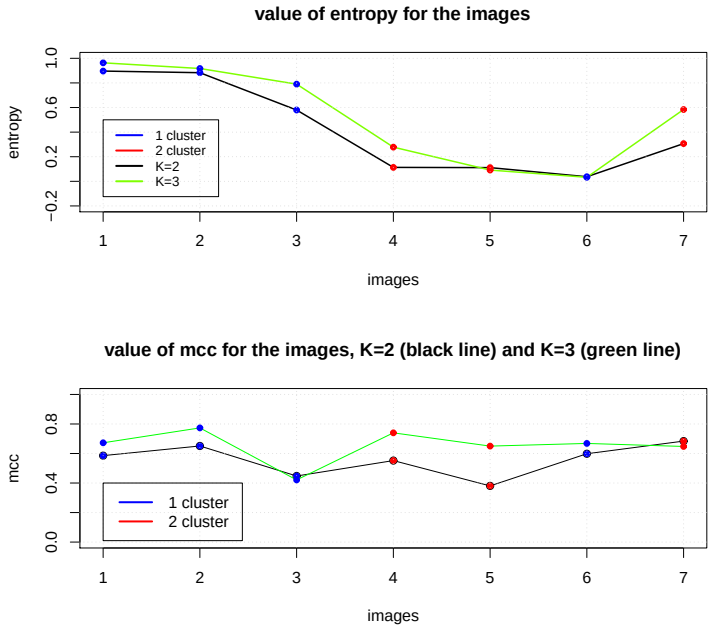

Figure 5.15: value of ANE and MCC for the 7 worse images in the cosine similarity case, with B=31, l=5 and a Gaussian variance equal to 1

low MCC for $K = 2$ (that are all images with 3 groups), with $K = 3$ give us better results. On the other hand, there are some images which give us worse results using $K = 3$, and all of those except one are also images with 3 groups. There are then other images for which the results are good in both cases.

So the problem is to find a criterion for the optimal choice of K. We have tried to use the ANE, but unfortunately this is not a correct criterion, because it happen a lot of times that the ANE has the minimum value in $K = 2$, as we will see in the next section.

Figure 5.15 representS the value of ANE and MCC for the 7 worse images: the black line represent the simulation with $K = 2$ clusters and the green one the simulation with $K = 3$. Red points are images with two modified areas and blue points images with only one modified area. Unfortunately, we can notice that also if the results, in terms of MCC are better using 3 clusters, the ANE is not always lower, so we cannot use this as a criterion to select the optimal number of clusters K.

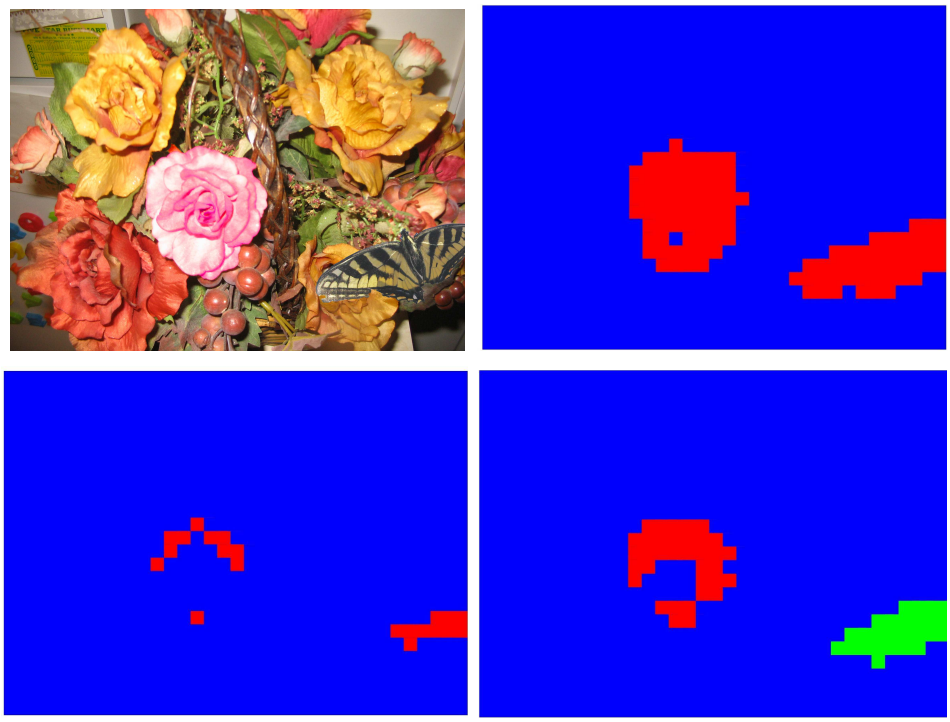

Figure 5.16: At the top we have the original image (left) and the related mask (right) for image c53d947607a4db8e320dd488de2a7be9; bottom we have on the left the classification obtained using 2 clusters, and on the right the classification obtained using 3 clusters

5.6 Analysis on a single image

In this section we will study more in detail a single image, that give us not good results in terms of MCC, in section 5.2, and make some considerations. As we have seen in section 5.5, if we get bad result with $K = 2$, use $K = 3$ will help us. Since we are working with images, one can visualize the output result of the classification, and try to understand, looking at the original image, which classification seems to be better. For example, looking at Figure 5.16, were we have on the top the original image and the related mask, bottom left the result with $K = 2$ clusters and on the bottom right the result with $K = 3$ clusters, one can try to understand which are the element that are not original in the image: with $K = 2$ we cannot see a lot, but looking at the plot for $K = 3$ one can guess that the butterfly and the rose flower are added manually, because it's unrealistic to think that only half of the

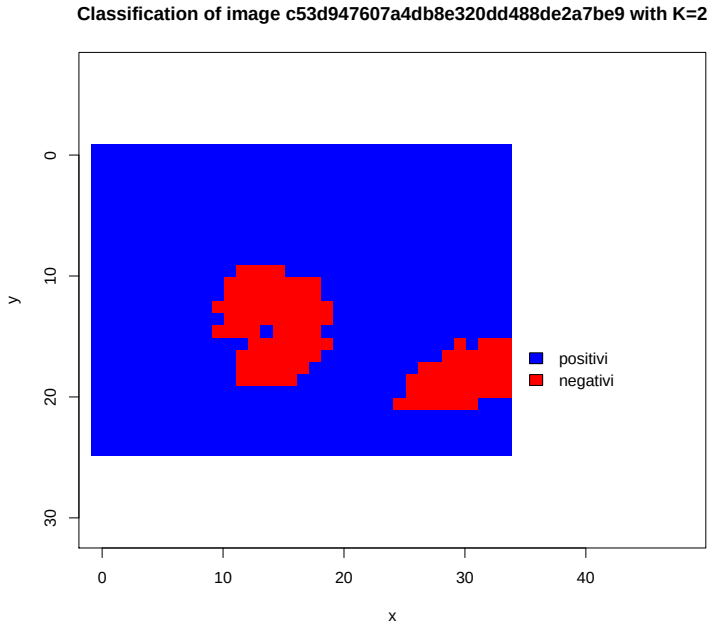

Figure 5.17: Classification of the image c53d947607a4db8e320dd488de2a7be9 with the entropy criterion with a threshold of 0.3

butterfly is tampered. Since we have a predominance of false negative (as show in Table 5.5), that means that an edited patch it's classified as original, we can guess that the regions classified as tampered are really been modified. Both simulation are made with $l = 5$, $B = 31$ and $\theta = 1$.

In Figure 5.17 we have the final classification obtained considering the algorithm as explained in Section 4.4.2. In this case we have considered a threshold of 0.3: all the patches with a local entropy greater than 0.3 are classified as an edited area. In this way we obtained a $MCC = 0.877$, that is much better than the results shown in 5.16. Although we got good results for this image, this criterion is not appropriate for most images.

At the end we show the curve for the MCC and the ANE, in Figure 5.18 and the related mask, in Figure 5.19, always for the image under consideration before, doing a simulation in the cosine similarity case, with $B = 31$, $K = 2$ and a Gaussian variance equal to 1.
From Figure 5.18 we can see that we have the best values for the MCC for small or big values of l. Comparing to Figure 5.19, we can notice that when l

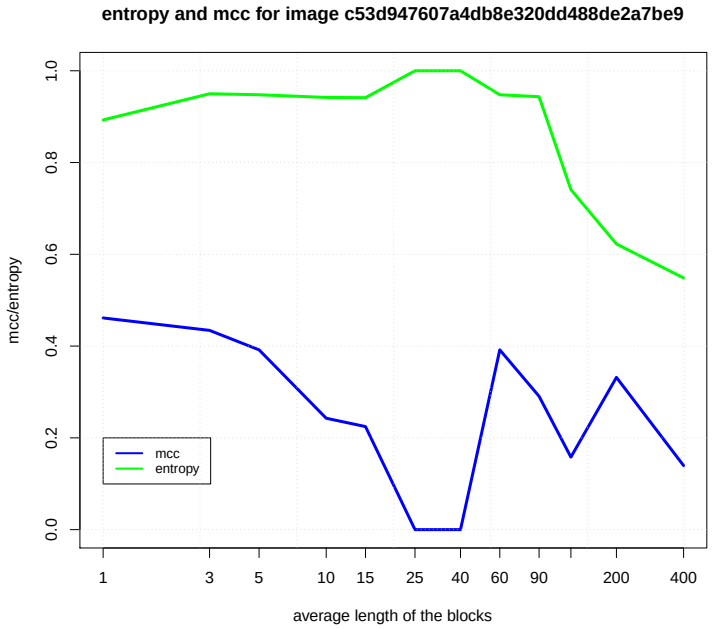

Figure 5.18: Curve of the MCC and the ANE in the cosine similarity case, for image c53d947607a4db8e320dd488de2a7be9, with B=31, K=2, l=5 and Gaussian variance equal to 1

is small the algorithm recognize better one group (the butterfly), then when l increases the algorithm is able to detect better the other group (the flower) and if l become too big the result is messy. For $l = 25$ and $l = 40$ the algorithm is not able to detect the tampered area, and the corresponding MCC value is 0. The original image is shown in Figure 5.16.

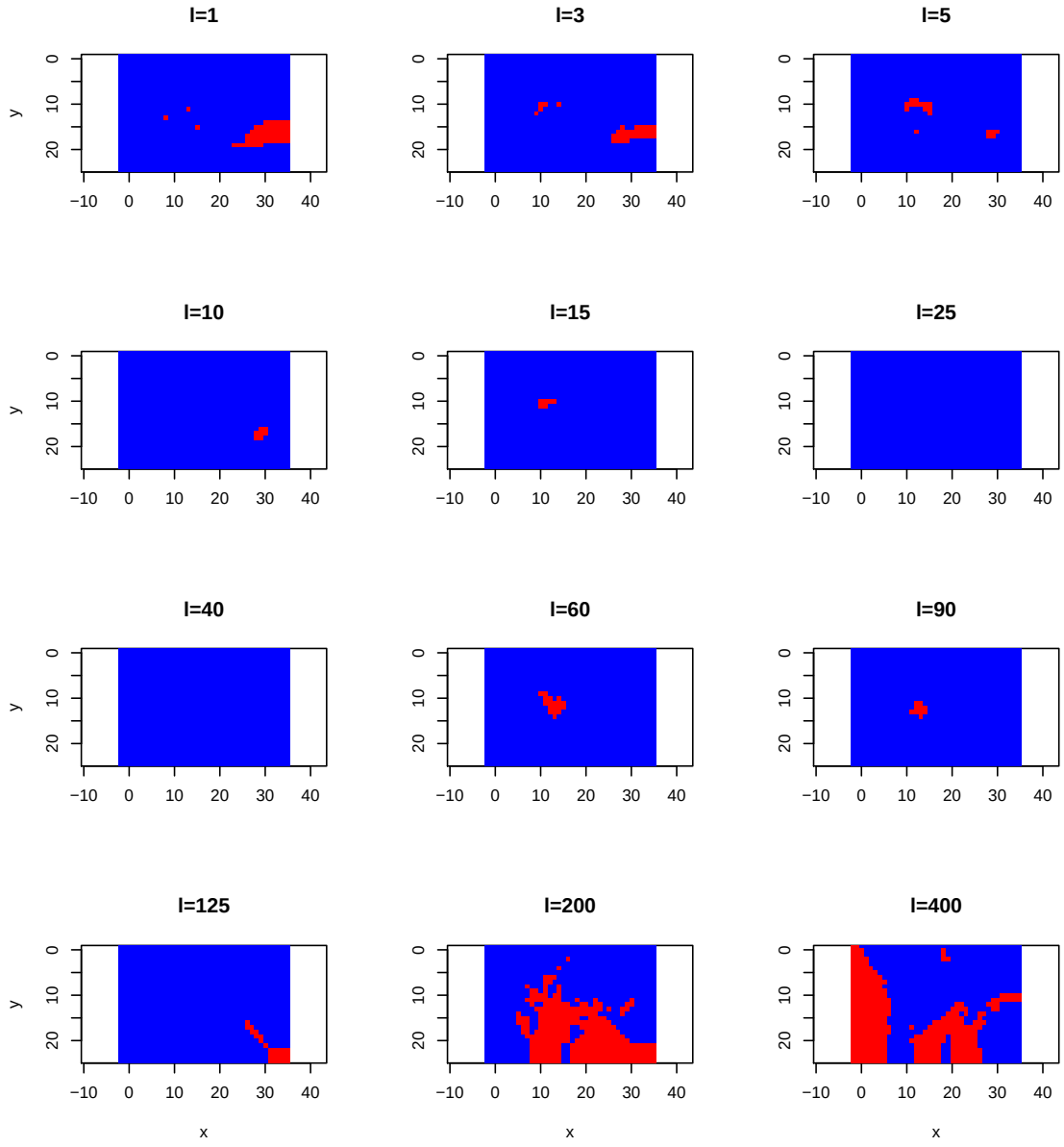

Figure 5.19: Final classification for all the values of l, in the cosine similarity case, with B=31, K=2, l=5 and Gaussian variance equal to 1

Chapter 6

Conclusions and future work

6.1 Conclusions

In this thesis we have used a new approach to solve the tampering localization problem, by implementing the Bagging-Voronoi algorithm.
We put most of our effort in the search of the optimal parameter for the algorithm and in the search to improvement for our algorithm. It's not so easy to find the optimal parameter, since each image is different from each other, and so there is not an optimal criterion to detect the best value for the parameters. Also each tampered area is different from each other, and also this may condition the results.
We have understand that if we know how the data are generated, then we can get better results, considering the geometry of the data in the Bagging-Voronoi algorithm. Another thing that we have learned is that the Voronoi tessellation must not be to big, so, as optimal value for run the algorithm one has to take a value of $l \in (5, 15)$.
In general, the Gaussian variance can be selected taking the minimum value given in the *average normalized entropy*.
Considering the euclidean distance between patches, as well as the geometry of the data, can improve the results for some images, and the optimal weight α can be selected by looking at the minimum *average normalized entropy*.

6.2 Future research

In this work we focused our analysis on the optimal length of the blocks, but there is a lot of work that can be done to improve the algorithm. First of all, the search for a criterion that can guarantee the best choice of the length of the tessellation and of the number of clusters K, since we have seen that the *generalized average entropy* criterion give us good results only when we look at the results averaging over all the images, but not for the single ones.
We also show briefly how changing the acceptance criterion, using the local entropy, can give good results when the algorithm do not work correctly, but there can be done analysis to understand which is the optimal β value and if there is a way to detect when it's better to use the majority criterion and when the local entropy criterion.
Another interesting thing is the use of the value of the entropy in the single patches to detect where is the division between the areas, since an high value of the entropy means that at each iteration the algorithm classify in a different way the corresponding patch, and so we are on the edge, and a low value of the entropy means that at almost all the iterations the algorithm assign the patch at the same group.

Bibliography

[1] Piercesare Secchi, Simone Vantini, and Valeria Vitelli. "Bagging Voronoi classifiers for clustering spatial functional data". In: *International Journal of Applied Earth Observation and Geoinformation* 22 (2013). Spatial Statistics for Mapping the Environment, pp. 53–64. ISSN: 0303-2434. DOI: https://doi.org/10.1016/j.jag.2012.03.006. URL: http://www.sciencedirect.com/science/article/pii/S0303243412000505 (cit. on pp. V, VII, 1, 2, 21, 31).

[2] Davide Eynard. *Machine Learning, Lecture Note on Clustering (I).* 2017-2018. URL: http://davide.eynard.it/teaching/2018_ML/slides-lecture-e1.pdf (cit. on p. 3).

[3] Anja Struyf, Mia Hubert, and Peter Rousseeuw. "Clustering in an Object-Oriented Environment". In: *Journal of Statistical Software, Articles* 1.4 (1997), pp. 1–30. ISSN: 1548-7660. DOI: 10.18637/jss.v001.i04. URL: https://www.jstatsoft.org/v001/i04 (cit. on p. 3).

[4] Alessadro Piva. "An Overview on Image Forensics". In: *ISRN Signal Processing* 2013.4 (2013), p. 22. URL: http://dx.doi.org/10.1155/2013/496701 (cit. on p. 6).

[5] J. Fridrich. "Digital image forensics". In: *IEEE Signal Processing Magazine* 26.2 (Mar. 2009), pp. 26–37. ISSN: 1053-5888. DOI: 10.1109/MSP.2008.931078 (cit. on p. 8).

[6] L. Bondi et al. "First Steps Toward Camera Model Identification With Convolutional Neural Networks". In: *IEEE Signal Processing Letters* 24.3 (Mar. 2017), pp. 259–263. ISSN: 1070-9908. DOI: 10.1109/LSP.2016.2641006 (cit. on pp. 9, 12).

[7] M. Barni et al. “Aligned and non-aligned double JPEG detection using convolutional neural networks”. In: *Journal of Visual Communication and Image Representation* 49 (2017), pp. 153–163. ISSN: 1047-3203. DOI: https://doi.org/10.1016/j.jvcir.2017.09.003. URL: http://www.sciencedirect.com/science/article/pii/S104732031730175X (cit. on p. 10).

[8] Zhouchen Lin et al. “Fast, automatic and fine-grained tampered JPEG image detection via DCT coefficient analysis”. In: *Pattern Recognition* 42.11 (2009), pp. 2492–2501. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2009.03.019. URL: http://www.sciencedirect.com/science/article/pii/S0031320309001198 (cit. on p. 13).

[9] Weihai Li, Yuan Yuan, and Nenghai Yu. “Passive detection of doctored JPEG image via block artifact grid extraction”. In: *Signal Processing* 89.9 (2009), pp. 1821–1829. ISSN: 0165-1684. DOI: https://doi.org/10.1016/j.sigpro.2009.03.025. URL: http://www.sciencedirect.com/science/article/pii/S0165168409001315 (cit. on p. 13).

[10] P. Ferrara et al. “Image Forgery Localization via Fine-Grained Analysis of CFA Artifacts”. In: *IEEE Transactions on Information Forensics and Security* 7.5 (Oct. 2012), pp. 1566–1577. ISSN: 1556-6013. DOI: 10.1109/TIFS.2012.2202227 (cit. on p. 13).

[11] Ahmet Emir Dirik and Nasir Memon. “Image Tamper Detection Based on Demosaicing Artifacts”. In: *Proceedings of the 16th IEEE International Conference on Image Processing*. ICIP’09. Cairo, Egypt: IEEE Press, 2009, pp. 1481–1484. ISBN: 978-1-4244-5653-6. URL: http://dl.acm.org/citation.cfm?id=1818719.1819143 (cit. on p. 13).

[12] Shuiming Ye, Qibin Sun, and Ee-Chien Chang. *Detecting Digital Image Forgeries by Measuring Inconsistencies of Blocking Artifact*. Aug. 2007 (cit. on p. 13).

[13] N. Krawetz. *A Picture’s Worth*. 2007 (cit. on p. 14).

[14] H. Farid. “Exposing Digital Forgeries From JPEG Ghosts”. In: *IEEE Transactions on Information Forensics and Security* 4.1 (Mar. 2009), pp. 154–160. ISSN: 1556-6013. DOI: 10.1109/TIFS.2008.2012215 (cit. on p. 14).

[15] Babak Mahdian and Stanislav Saic. "Using noise inconsistencies for blind image forensics". In: *Image and Vision Computing* 27.10 (2009). Special Section: Computer Vision Methods for Ambient Intelligence, pp. 1497–1503. ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2009.02.001. URL: http://www.sciencedirect.com/science/article/pii/S0262885609000146 (cit. on p. 14).

[16] Marco Fontani et al. "A Framework for Decision Fusion in Image Forensics Based on Dempster-Shafer Theory of Evidence". In: 8 (Apr. 2013), pp. 593–607 (cit. on p. 14).

[17] L. Bondi et al. "First Steps Toward Camera Model Identification With Convolutional Neural Networks". In: *IEEE Signal Processing Letters* 24.3 (Mar. 2017), pp. 259–263. ISSN: 1070-9908. DOI: 10.1109/LSP.2016.2641006 (cit. on pp. 14–16).

[18] D. Cozzolino, G. Poggi, and L. Verdoliva. "Splicebuster: A new blind image splicing detector". In: *2015 IEEE International Workshop on Information Forensics and Security (WIFS)*. Nov. 2015, pp. 1–6. DOI: 10.1109/WIFS.2015.7368565 (cit. on p. 16).

[19] Mathew D. Penrose. "Laws of large numbers in stochastic geometry with statistical applications". In: *ArXiv e-prints* (Nov. 2007). arXiv: 0711.4486 (cit. on p. 22).

[20] Alessandro Volpe. "L'algoritmo Bagging-Voronoi per la classificazione di dati temporalmente dipendenti". MA thesis. http://hdl.handle.net/10589/92454: Politecnico di Milano, 2014 (cit. on pp. 30, 31).

[21] Richard A. Johnson and Dean W. Wichern. *Applied multivariate statistical analysis*. 6th ed. Pearson Education, Inc., 2007.

[22] Sabri Boughorbel, Fethi Jarray, and Mohammed El-Anbari. "Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric". In: *PLOS ONE* 12.6 (June 2017), pp. 1–17. DOI: 10.1371/journal.pone.0177678. URL: https://doi.org/10.1371/journal.pone.0177678.

Appendix A

Bagging-Voronoi code

```
# Variables:

# data: a matrix for each image, containing: coordinates y,x and a number of features feat_num
# B: number of iterations
# l: average length of the tessellation
# K: number of clusters
# var_gauss: the Gaussian variance
# group: final output classification

Bagging_Voronoi <- function(data,B,l,K,var_gauss){

    data=as.matrix(data)
    m=dim(data)[1]
    n=m/l
    feat_num=dim(data)[2]-2
    clusters_result=matrix(0,m,B)
    group=vector(mode="numeric",length=m)
    x.1 <- seq(1,2*max(data[,1]),1)
    x.2 <- seq(1,2*max(data[,2]),1)
    w <- matrix(NA, length(x.1), length(x.2))
    for(jj in 1:length(x.1)) {
      for(kk in 1:length(x.2)) {
          w[jj,kk] <- dmvnorm(c(x.1[jj],x.2[kk]),c(max(data[,1]),max(data[,2])),diag(x =1, 2,2))
        }
```

```
25     }
26
27    for (b in 1:B) {
28       z <- sample(1:m, n)
29       nuclei_coord=data[z,1:2]
30       V=matrix(list(),n,1)
31       el[]=1
32
33       for (i in 1:m) {
34         k=which.min(sqrt((data[i,2]-data[z,1:2][,2])^2+(
              data[i,1]-data[z,1:2][,1])^2))
35         V[[k]][el[k]]=i
36         el[k]=el[k]+1
37       }
38
39       for (j in 1:n) {
40         kernel_mean=as.vector(t(nuclei_coord[j,]))
41         for (k in 1:n) {
42       dist_nuclei[k]=sqrt((nuclei_coord[k,2]-nuclei_
           coord[j,2])^2+(nuclei_coord[k,1]-nuclei_coord[j
           ,1])^2)
43         }
44         diff0=dist_nuclei>0
45         kernel_cov=max(dist_nuclei)/min(dist_nuclei[
              diff0])
46         ww<-kernel_cov*w;
47         el_tassello=V[[j]]
48         if(length(el_tassello)==1){
49           coord_el=as.matrix(t(data[el_tassello,1:2]))
50         } else {
51             coord_el=as.matrix(data[el_tassello,1:2])
52         }
53      weight_sum=0
54
55         for (p in 1:length(el_tassello)) {
56           rapp[j,]=rapp[j,]+as.vector(data[el_tassello[p
                ],3:(feat_num+2)])*ww[-kernel_mean[1]+coord
                _el[p,1]+max(data[,1]),-kernel_mean[2]+
                coord_el[p,2]+max(data[,2])]
```

```
57            weight_sum=weight_sum+ww[-kernel_mean[1]+coord
                 _el[p,1]+max(data[,1]),-kernel_mean[2]+
                 coord_el[p,2]+max(data[,2])]
58          }
59          rapp[j,]=rapp[j,]/weight_sum;
60    }
61
62    rapp_sim=matrix(data=NA,nrow=dim(rapp)[1],ncol=dim(
         rapp)[1])
63
64      for (i in 1:dim(rapp)[1]){
65        for (j in 1:dim(rapp)[1]){
66              rapp_sim[i,j]=sum(rapp[i,]*rapp[j,])/(sqrt(
                   sum(rapp[i,]^2))*sqrt(sum(rapp[j,]^2)))
67          }
68    }
69      rapp_sim=(rapp_sim+1)/2
70      rapp_sim=1-rapp_sim
71    matrix_dist_nuclei=matrix(data=NA,nrow=dim(rapp)[1],
         ncol=dim(rapp)[1])
72
73    for (i in 1:dim(rapp)[1]){
74        for (j in 1:dim(rapp)[1]){
75            matrix_dist_nuclei[i,j]=ifelse(i==j,1,1/sqrt((
                 nuclei_coord[i,1]-nuclei_coord[j,1])^2+(
                 nuclei_coord[i,2]-nuclei_coord[j,2])^2))
76        }
77    }
78      matrix_dist_nuclei=1-matrix_dist_nuclei
79      matrix_sim=matrix(data=NA,nrow=dim(rapp)[1],ncol=dim
           (rapp)[1])
80      matrix_sim=1/2*(rapp_sim+matrix_dist_nuclei)
81
82      result.k <- pam(as.dist(matrix_sim),K)
83
84  # cluster matching
85
86    if (b==1)  {
```

```
87          clustSample_ref = ClusteredSample(labels=result.
               k$clustering, sample=as.data.frame(matrix_sim
               ))
88          for (ll in 1:n){
89            clusters_result[V[[ll]],b]=result.k$clustering
                 [ll]
90          }
91      }
92      if (b!=1){
93          clustSample_new=ClusteredSample(labels=result.k$
               clustering, sample=as.data.frame(matrix_sim))
94          mmatch=match.clusters(clustSample_new,
               clustSample_ref, dist.type='Euclidean',
               unmatch.penalty=9999)
95          clusters_temp=result.k$clustering
96          for (kk in 1:K) {
97            clustt=result.k$clustering==kk
98            a=get.match21(mmatch)[[kk]]
99            if (a!=kk)
100             clusters_temp[clustt]=a
101           }
102           for (ll in 1:n){
103             clusters_result[V[[ll]],b]=clusters_temp[ll]
104           }
105       }
106     }
107
108 # final classification
109
110     for(pixel in 1:m) {
111       group[pixel]=as.numeric(names(which.max(table(
             clusters_result[pixel,])))
112       }
113
114     return(group)
115
116 }
```