Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

Classification and clustering of video fingerprints: preliminary results

Relatore:
Chiar.mo Prof.
Danilo Montesi

Correlatore:
Ph.D. Flavio Bertini

Correlatore:
Ph.D. Rahimeh Rouhi

Presentata da:
Lorenzo Massimiliani

Sessione III
Anno Accademico 2019-2020

Contents

List of Figures

List of Tables

Introduction

In this period everyone certainly has at least one smartphone and probably has a tablet or some other device with a camera. Many photos and videos are produced and uploaded to the Internet every day. Even though this is a small part of the total, a large amount of them is illegal. For example, their content may depict illegal acts such as assault or child abuse.

Once digital content has been distributed online, it is often difficult to re-associate the photo or video to the device that produced it or to the user who initially shared it. In the event that the material is illegal, the user who disseminates it has, naturally, an interest in ensuring that it is not possible to trace the source.

To counter the spread of illegal content, there is a branch of studies called "source camera identification", which aims to reconnect a photo or video to the device that developed it. Unfortunately, it is not possible to rely on file metadata, such as content creation date and GPS location, as they are easily editable. The idea behind source camera identification is that each camera, having imperfections that make it unique, gives a digital fingerprint to the content it produces. The noise of a digital content, which represents a variation of intensity that cannot be found in the recorded content, contains the fingerprint along with some random factors. The noises, which are extracted through denoising algorithms, can be used directly to identify the device that produced the content, or they can be used to estimate the fingerprint.

This thesis works in the source camera identification of video content. Two datasets are considered: one called Vision, which is considered the reference dataset

in this area and one made available by the University of Bologna [1]. The Vision dataset also contains videos that have been uploaded and downloaded on social networks. Working on these contents simulates a real scenario, where the available items are uploaded online and therefore are subject to compression.
The work carried out in this thesis was to extract the noises on those datasets, and calculate the fingerprints, comparing different approaches present in the state of the art. The approach that was chosen has yielded the best results through a classification algorithm. Once the noises were extracted and the fingerprints calculated, classification and clustering techniques were applied. Some of them worked directly on the noises extracted, others on the estimates of the fingerprints. Two classification techniques have been developed one through convolutional neural network and another using a function called Peak-to-correlation energy (PCE). Clustering algorithms have been applied, already developed to work in this area, one that considers a known number of classes and another that considers an unknown number. The project was developed on the MATLAB computing environment, using a license made available by the University of Bologna.
The thesis is organized in the following way:

Chapter 1

The first chapter explains the state of the art of the studies carried out in the area of camera identification. We will discuss what kind of fingerprint a camera leaves on the content it records. Algorithms able to isolate these fingerprints from images will be presented. The source camera identification techniques of the images and the results obtained will be shown. Finally, the last sub-chapter will refer to videos and explain the differences between videos and images in the source camera identification process.

[1] http://smartdata.cs.unibo.it/datasets#videos

Chapter 2

The second chapter will examine the classification and clustering techniques used to address the problem of camera identification. The first one includes the application of a neural convolutional network, the use of an algorithm based on the PCE function and a combined approach of both. Subsequently, there will be a clustering technique with a known cluster number and one with an unknown cluster number.

Chapter 3

This chapter will present the database used for the experiments and all the pre-processing operations performed. These operations are necessary to obtain, starting from the videos, the noises and the estimates of the fingerprints that will be the input elements of the classification and clustering. Among these we find the extraction of frames, the extraction of noises, the problem management of video stabilization and the calculation of the fingerprint. In the state of the art there are different approaches to deal with these pre-processing operations. In this chapter they are described and compared using a classification algorithm.

Chapter 4

The fourth chapter presents the results of classification and clustering on the dataset presented in the previous chapter, through some performance measures described. Of course, the pre-processing operations, for the results described in this chapter, are those that have shown the best performance. We will also see the results obtained on datasets composed of videos uploaded to social networks and on a dataset collected by the University of Bologna. Finally, the results will be compared and commented on.

Chapter 5

The last chapter will report the conclusions of the project done in this thesis. We will also discuss possible developments starting from this work.

Chapter 1

State of the art

Source camera identification is a kind of process that identifies image or video source. The first goal of this process was to determine the model of the camera, later the goal became to identify the specific camera. The second goal is more ambitious, because it renounces to exploit the specific features that a camera model provides to the content it record, considering only those relating to the model instance. The work carried out in this thesis arises precisely in the latter context, in which source camera identification refers to the identification of the specific camera that has captured an image or a video.
This research area has interesting relevance in a digital forensic analysis context. It is difficult to establish the origin of content spread on the internet, especially if it is illegal, and those who share it have an interest in hiding their identity. For the purpose of a forensic investigation it may be useful to understand whether two videos uploaded to a social network from two different accounts have been recorded by the same camera. It is interesting to be able to address this problem, considering only the shared content, omitting all other factors, such as file metadata or social account data.
The next sub-chapter will talk about which elements can be considered to identify the source camera.

1.1 Sensor Pattern Noise

Most of the studies in this field deal with the research of the source camera of the photographs. For videos, there are additional difficulties that complicate this operation. For simplicity we will start by considering only the case of the photographs.
Source camera identification is based exclusively on the image itself without any information about file generation equipment. In addition to the metadata, which cannot be considered, the camera leaves other information about the content it acquires. In fact, in each photo we can find imperfections called noises. Especially the Sensor Pattern Noise (SPN), mainly due to camera imperfections, makes a device recognizable [1]. The SPN contains the Fixed Pattern Noise (FPN) [2] and the Photo-Response Non-Uniformity (PRNU) noise.
The FPN is an uncertainty due to some manufacturing defects of the camera in one or more sensors, such as CCD (charged coupled device) and CMOS (complementary metal oxide semiconductor). This noise has the drawback of being easily removable from an image by subtracting a black photo from it. This is due to the fact that this noise is the same for each image and, therefore, by making this subtraction the noise disappears [3].
The PRNU was found to be more useful for the purpose of SCI [4]. In fact, it does not depend on manufacturing defects, but on the difference in sensitivity of each pixel subjected to light. This also makes it much more difficult to discard of the PRNU from a photo. This pattern noise is always present due to the device and is proven to be unique per sensor [5].
Once extracted from videos or images, it can act as a fingerprint for a particular camera and can be used to verify the existence of that pattern in other images.
Other studies have considered accelerometers [6], gyroscopes [7], magnetometers [8] and microphones and speakers [9] as sensors on which to identify a fingerprint.

1.2 Residual noises and fingerprints

There are several algorithms used to remove noise from images, from state of the art. They were thought for the purpose of cleaning images and not within the scope of source camera identification. But with a little trick it is possible to exploit them for that purpose.
We can consider an image composed in this way:

$$I = I_{(o)} + K * I_{(o)} + \Theta$$

where $I_{(o)}$ is the image without noise, $K * I_{(o)}$ is the PRNU and Θ is a generic noise term which considers other noisy contributions, such as quantization noise and dark current. We can get the residual noise of an image by calculating the difference between that image and the same image that has had the noise removed.

$$RN = I - d(I)$$

The letter 'd' indicates any denoising algorithm.
A camera fingerprint can also be estimated at this point. Having more photographs available, the image obtained as an average of the residual noises can be considered as an approximate fingerprint.

$$F = \frac{1}{n}\sum_{i=1}^{n} RN_i$$

In this way, we can consider the fingerprint (F) as the signature of a camera on a video or photo it produces. The idea is that adding more residual noises can eliminate some of the random noise. Of course, this is an estimate and the more photographs will be considered the more the estimated fingerprint should be similar to the real one.

1.3 Video denoising filters

Once it has been determined how to proceed to calculate residual noise and fingerprints, it is necessary to consider what kind of denoising algorithms exist and how they work.

The Block-matching and 3D filtering (BM3D) algorithm [10] is the current state of the art for image denoising. Most of the denoising algorithms, which were used prior to the BM3D output, were based on computing the noise as approximated by a linear combination of few basic elements. Instead, the BM3D algorithm is based on non-linear combinations, in particular, on an enhanced sparse representation in transform domain.

The improvement is given by the idea of grouping fragments of the 2D image (called blocks) in 3D data arrays (called groups). This is done through three stages: 3D transformation of a group, shrinkage of the transformation spectrum and inverse 3D transformation. The second stage applies a filter through a shrinking transformation (collaborative filter) to 3D blocks. In the last stage, the filtered blocks are then returned to their original positions and aggregated in the final estimate of the image.

There is also a version of this algorithm called Block-Matching and 4D filtering (BM4D), which has the difference of grouping 3D rather than 2D groups [11]. By applying this algorithm, better image denoising is achieved.

We can consider a video, recorded in any format, as a sequence of frames. Each frame is similar to a photograph, but there is a difference. After a video is recorded, a compression process takes place, which has the aim of reducing the memory space occupied by the file. For example, if we consider a 4k video lasting one minute recorded at 30 frames per second, and considering that a 4k photo occupies 10 MB, the entire video, if it were to be stored as a sequence of photographs, would occupy 18 GB. This explains the need for a compression process, which causes a loss of detail [12] (lossy compression).

Most compression algorithms have three types of frames: I-frame (Intra-coded picture), P-frame (Predicted picture) and B-frame (Bidirectional predicted picture). An I-frame is a complete image, and can be decompressed without any other infor-

mation. P-frame and B-frame instead, contain only information on how the image evolves with respect to the following and previous frames. These frames contain little information compared to an I-frame and allow us to save a lot of space when storing a video.
A widely used algorithm for video denoising is video block-matching and 3D filtering (V-BM3D) [13], which is a video adaptation of BM3D. V-BM3D in addition to aggregating similar groups together at the frame level, as it happens in BM3D, also aggregates taking into consideration the temporal evolution of the frames. Considering the time axis, the algorithm gives very good denoising results. The similarities along the time axis tend to be much more pronounced than those within the same frame. A limitation of this method is that it is unable to distinguish between local and temporal similarity.
An algorithm that overcomes this limitation is block-matching and 4D filtering (V-BM4D) [14]. In fact, it separately exploits the temporal and spatial redundancy of the video sequence. Hence, groups in V-BM4D are 4-D stacks of 3-D volumes, and the collaborative filtering is then performed via a separable 4-D spatiotemporal transform. Hence, groups in V-BM4D are 4-D stacks, and the collaborative filtering is performed via a separable 4-D spatiotemporal transform. Three types of similarity are considered: local similarity within the same frame, temporal similarity, and local and temporal similarity within volumes. These features make V-BM4D a more performing denoising algorithm than V-BM3D.
Of course, the main task of these algorithms is to clean the images from noise, not to extract noises for the purpose of source camera identification. For this second purpose, it is not clear which is the best algorithm at the state of the art. For this reason, research on this point has been done on the thesis work.

1.4 Source camera identification using videos

We will now see the techniques and approaches used in the literature to deal with camera identification considering photographs. The most used approaches for camera identification are two categories of techniques: classification and clustering. Although both are part of the learning arena, there are some differences. Classification consists in building a model that is able to associate an element with a class. This occurs through two phases: the first of training and the second of testing. Clustering does not include a training phase and is part of unsupervised learning and consists in associating objects in groups. Usually, the results obtained through classification techniques are better than those obtained through clustering. Some clustering techniques provide for the distribution on a fixed number of clusters known by the algorithm, while other techniques do not know this value a priori and obtain it during execution.

In the past, some works have extracted the features considered useful, more or less manually, and used an algorithm for the classification. For example, establishing some similarity measures such as features and applying a kNN classifier algorithm [15]. A study was also carried out that considers statistical data from the wavelet domain as features and applies SVM classifiers [16]. The problem with these solutions is that the use of features extraction based on some aspects, however accurate, meant that some information was not considered. The growth of hardware and software has allowed the birth of deep learning techniques, thanks to which the manual extraction of the features is no longer necessary. Some works use clustering directly on extracted noise [17].

After the success of neural networks obtained in various fields, such as text and image classification, information extraction, machine translation, and speech recognition, they have also been used in the field of camera identification. For example, some works have used convolutional neural networks, often used when working with images, in order to classify photographs residual noises [18].

Several studies use a function, called Peak to Correlation Energy (PCE) [19], which compares the correlation between two PRNU patterns. PCE defines a measure of cross-correlation between 2D data arrays, which in this case are represented by residual noises or fingerprints. This function measures the correlation considering the possible presence of a shift that would maximize the result. This aspect is optimal as, often, the imprinted pattern of the fingerprint of the same camera on two different photographs is not completely aligned.

One study [20] uses a clustering algorithm called Hybrid Markov Clustering (HAL) capable of dividing images by associating them with a social profile. This method combines hierarchical and Markov clustering algorithms and does not require a priori knowledge of the number of cameras in the dataset.

Camera identification through videos can take place in two ways: by extracting the noises frame by frame, with a denoising algorithm for images, or by extracting the noises through a video denoising algorithm. The first way allows us to object 3D objects [21] in which the third dimension represents the temporal succession of the frames, while the second way a collection of 2D noises. The two elements are similar, the difference is that through a video denoising algorithm the time dimension is also exploited to obtain a better noise reduction.
A major problem when working with video is the possible presence of video stabilization. It is an algorithm used to improve video quality by removing involuntary video shaking due to hand movement during recording. Nowadays, practically all smartphones have this feature turned on by default. This is done by applying geometric transformations to frames, such as translation, scale, rotation [22]. This causes a misalignment between individual pixels between frames, making sure that a camera pixel does not always cover the same position between frames.
Several works have been done in this field in order to reduce this problem. Attempts have been made to calculate the fingerprints by realigning the frames, trying to eliminate the effect of video stabilization. For example, aligning misaligned frames using an inverse affine transformation [23].
One element that can be exploited is the fact that typically the first frame of a video sequence is not subject to stabilization. Starting from this observation, all the other potentially stabilized frames can be corrected by comparing them with the first frame of the video [24]. It is also possible not to assume that the first frame is stabilized (for example the video could be cut) and in that case the frame, whose noise is more similar to all the others, can be considered as the reference frame. The comparison between the frames is done considering a correlation function, for example, Peak-to-Correlation Energy ratio (PCE). The PCE function measures the correlation between two noises and allows us to identify how much

is the shift needed to maximize the correlation. Applying it between each noise and the reference noise, it is possible to estimate the geometric transformation that maximizes the correlation, and which is approximately the one to apply to reduce the effect of video stabilization. Another possible solution is to eliminate the noises that are found to have a low correlation with most of the noises present in the video. This approach could eliminate, for example, noises from frames that were recorded during a major camera movement.
Another problem is the rotation at which the videos were recorded. For example, having to estimate a fingerprint of a device having two videos available, it is necessary to be able to recognize the rotation at which they were shot. Otherwise, the camera pixels do not affect the same coordinate in the two videos. To align the videos, a fingerprint can be independently calculated for each video and using a correlation algorithm the rotation that maximizes the value can be determined. Or another solution is to represent the fingerprint so that it does not vary according to the rotation [25].
As regards classification and clustering, once the above problems have been solved, the same approaches used for source camera identification using images can be applied. For each video it is possible to calculate an approximate fingerprint and apply the same techniques applied to the images on the individual residual noises. Another possibility is to build a neural network able to individually classify each noise of a video [26]. Once a classifier of this kind has been built, the classification of the individual noises of which it is composed could be considered as video classification. The most predicted class on all noises can be considered as the model prediction on the video.

The data sets, on which the problem of camera identification is addressed, are sometimes composed of native images and videos, namely files directly recorded and saved on the device. But most of the time, files uploaded to social networks are examined, to simulate a context of digital forensics, where resources are uploaded to the web. The images and videos uploaded to social networks undergo a compression process that decreases the quality and consequently makes it more difficult to identify the device fingerprint. For this reason, many studies concern content uploaded to social media, such as YouTube [27], WhatsApp and Facebook.

Chapter 2

Video classification and clustering methods

This chapter addresses the classification and clustering techniques applied in this thesis project, which revolves around the source camera identification area. Having available a dataset of videos recorded by cameras of different smartphones, the goal will be to associate videos from the same smartphone to the same class.
Both techniques, classification and clustering, are machine learning methods, with some differences. As regards the classification, it is necessary to divide the dataset into training set and test set. The first training phase will refer to the training set, which will be used to train a model so that it is able to classify the test set. Clustering does not involve an initial training phase, and the dataset is clustered by distance on a multidimensional data plane.

2.1 Path to extract noises and calculate fingerprints

Actually, the videos will not really be the object on which learning will be done. In fact, as described in the previous chapter, to address the problem of source camera identification, reference will be made to the noises extracted from the videos and fingerprints. This is the path that allows us to obtain residual noises and fingerprints.

1. Resize all videos in the dataset to the same resolution
   This step is necessary to ensure that all videos in the dataset have the same resolution so that they are comparable to each other.

2. Video rotation
   If for the same device there are videos recorded in both landscape and portrait mode, you need to rotate them so to make sure that the pixels match between the videos.

3. Frame extraction
   In this way a set of frames is obtained from a video. Denoising algorithms work on objects of this type.

4. Noise extraction
   The noise can be extracted directly on all frames or only on some.

5. Fingerprint calculation
   You can estimate the fingerprint of the device that recorded a video by considering the noises extracted. The simplest solution to calculate the fingerprint is to average pixel by pixel of the residual noise.

2.2 Peak-to-Correlation-Energy based classification algorithm

Peak-to-Correlation-Energy (PCE) is a function that measures cross-correlation between 2D data arrays. To determine the PCE value of two matrices w_1 and w_2 of size m x n it is first necessary to calculate their correlation. The correlation c(m, n) can be expressed as the inverse 2D Fourier Transform (FT) of the conjugate product in the frequency domain.

$$c(m,n) = FT^{-1}\{W_1(k,l)W_2^*(k,l)\}$$

W_1 and W_2 represent the 2D Fourier Transform of w_1 and w_2 respectively. the output is a correlation plane and will have a point (i_{peak}, j_{peak}) where the correlation will be maximum. At this point it is possible to calculate the PCE function as follows:

$$PCE = \frac{c(i_{peak}, j_{peak})^2}{\sum_{i,j} c(i,j)^2}$$

In practice, the PCE function compares the point where the plane c has its peak with the remaining points. The greater the difference, i.e. the higher its peak, the higher the value of the PCE function will be. Otherwise, the flatter the c plane, the lower the PCE value will be. The point (i_{peak}, j_{peak}) can be considered as the shift that allows the maximum correlation between the w_1 and w_2 matrices. From the point of view of noises, this function is useful as it searches for a pattern between the two matrices.
Starting from the idea that the pattern between two estimated fingerprints of two videos recorded by the same device are similar, it is easy to build a classifier. Having available videos in the training set, it is possible to calculate a single fingerprint for each class. Fingerprinting between multiple videos can be done in the same way as calculating a video fingerprint. However, the next chapter of the thesis will deal with this aspect in detail.
At this point you will have a fingerprint for each class in the training set and a fingerprint for each video in the test set. Let's consider F_test the fingerprint we want to classify and F_1 ... F_k the fingerprints we have in the training set. The class predicted by the model will be the one that:

$$max_{(i)} PCE(F_{test}, F_i)$$

I built a function in MATLAB that behaves in this way and allows to classify all the elements of the test set.

2.3 Convolutional Neural Network classification

A convolutional neural network (CNN) is a neural network often used in image classification. In the field of machine learning, a neural network is a computational model composed of neurons and inspired by a biological neural network. The advantage these models offer is the ability to classify non-linear patterns that humans would not be able to recognize. A CNN is characterized by the presence of convolutional layers that perform a very important job as they extract features from images through the use of filters. Since it is usually better to use many elements to train a neural network, it is reasonable to consider noises, rather than fingerprints, as a dataset. There are many CNN networks already built for the purpose of classifying images, so we have chosen to use one of them rather than building one from scratch. After trying several of them, the one that gave the best results, in terms of classification, is GoogLeNet [29].

type	**output size**	**depth**	**params**
convolution	112x112x64	1	2.7K
max pool	56x56x64	0	
convolution	56x56x192	2	112K
max pool	28x28x192	0	
inception (3a)	28x28x256	2	159K
inception (3b)	28x28x480	2	380K
max pool	14x14x480	0	
inception (4a)	14x14x512	2	364K
inception (4b)	14x14x512	2	437K
inception (4c)	14x14x512	2	463K
inception (4d)	14x14x528	2	580K
inception (4e)	14x14x832	2	840K
max pool	7x7x832	0	
inception (5a)	7x7x832	2	1072K
inception (5b)	7x7x1024	2	1388K
avg pool	1x1x1024	0	
dropout(40%)	1x1x1024	0	
linear	1x1x1000	1	1000K
softmax	1x1x1000	0	

Table 2.1: GoogleNet network architecture

The levels shown in table 2.1 have the following characteristics:

- Convolutionary layer:
  A convolutionary layer applies one or more filters to an input matrix. A filter is usually a small matrix that flows to cover all the input. The scalar product between the filter and the input window covered by the filter is output. Once the calculation is done, the filter scrolls by a number of steps determined by a parameter and the scalar product is repeated until all the input has been covered.

- Pooling layer:
  The pooling layer reduces the size of the input. A window is scrolled on the input and the maximum value in the window (in the case of max pooling), or the average value (in the case of average pooling), is produced in output.

- Inception module:
  Inception Modules are used to reduce computational complexity and to have deeper networks. In practice, it allows us to operate convolutions with filters of different sizes by ordering them so that they work at the same level. Once these filters are applied, a max pooling operation can be performed and finally the outputs are concatenated.

- Dropout:
  Dropout is a technique that reduces the effect of over fitting and is practiced by randomly deleting a certain percentage of connections between neurons.

- Linear layer:
  Linear layer contains a number of neurons equal to the classes of the dataset.

- Softmax layer:
  The final layer is the softmax layer. It uses the softmax function which outputs a probability distribution that the input drop into each of the classes.

2.3.1 Using the single noise

The network foresees as input images of the size 224-by-224. Noises are usually larger in size. There are two ways to match the dataset with CNN. It is possible to resize the images to the size required by the network. Or, on the contrary, modify the network so that it accepts images of different sizes. The first solution, unfortunately, makes the images lose information as they have to be resized to a smaller size. So, the second approach was chosen. To adapt the network to the input data I added a convolutional layer on top of the network.This layer has been constructed so as to accept the size of the input images and produce an output of the size expected from the layer below. In this way, however, a resizing of the input takes place, but it takes place within the network and all the pixels of the images can be exploited.
Another problem is that the input noises are in grayscale, while the network expects RGB images. The added layer also solves this aspect, since it is composed of three filters. In this way, it produces an output matrix of three channels.

2.3.2 Using the entire video

Once you have built a model that can classify a video noise, it is easy to build a model for the entire video. A video can be considered composed of several noises.

$$V = N_1 + N_2 + ... + N_m$$

During the classification of a noise N_k, the penultimate layer of the CNN produces a probability vector $\bar{\mathbb{Y}}_k$, which represents the assumed probability from the model that the noise drops into each class. By adding together the probability vectors, class by class, we get a final vector and the class with the highest value is the class assigned by the model to the video.
Of course it is expected that the classification of videos with respect to the classification of single noise will give better results. This is because considering more elements could reduce the errors. The longer the video, the more information you will have and the better the classification. The heterogeneity of the video must

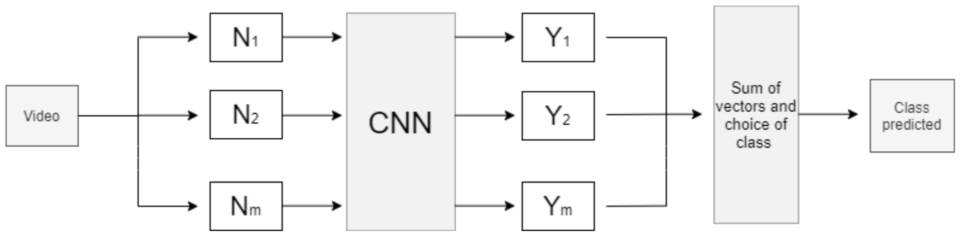

Figure 2.1: Path of video classification through CNN

also be considered. If the video is static, the first frame will be similar to the last one and the same will be true for noises. So, in the latter case, if the first noise is classified it is likely that the whole video will be classified badly.

2.3.3 Using a combination of neural networks

Another possibility is the use of a combination of neural networks to classify a video. We can train other neural networks as we did in the previous section. In this case we have chosen to use GoogleNet and two other networks: VGG16 and SqueezeNet. The training of the networks takes place in the usual way. For classification, each of the networks will output a vector, which represents the probability that the video falls into each of the classes. By adding class by class of the three carriers, the class with the greatest weight will be the prediction of the three CNN system. Using three neural networks could correct mistakes made by a single network. This improvement could be achieved by creating a single larger neural network, but the complexity would greatly increase.

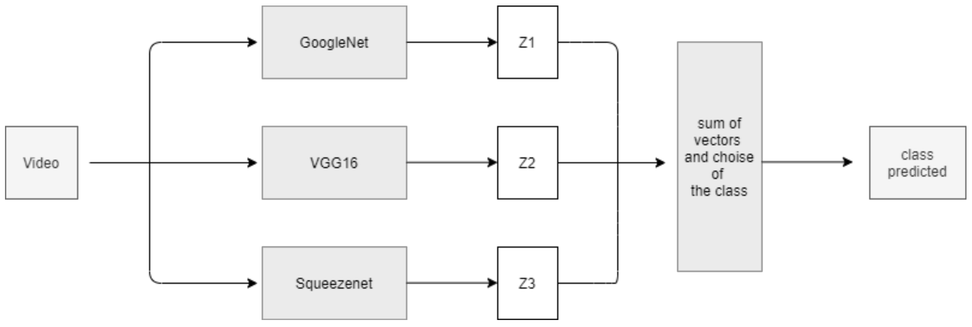

Figure 2.2: Path of video classification through three CNN

2.4 Combined Peak-to-Correlation-Energy based classification algorithm and Convolutional Neural Network classification

A third classification method can be obtained by combining CNN and PCE classifications. Being two methods that use different techniques for classification, their combination could combine the advantages of both. The input of the CNN classification is noises, while the PCE classification uses fingerprints. This third method, of course, uses both.
To create this classifier it is sufficient to train the two classifiers separately. To classify a video V, it is necessary to extract the noises R1 ... Rm and calculate the fingerprint F. Both models will classify the input elements separately and produce two probability vectors. At this point, the two vectors can be added class by class and the class with the highest value can be considered the prediction of this new model.

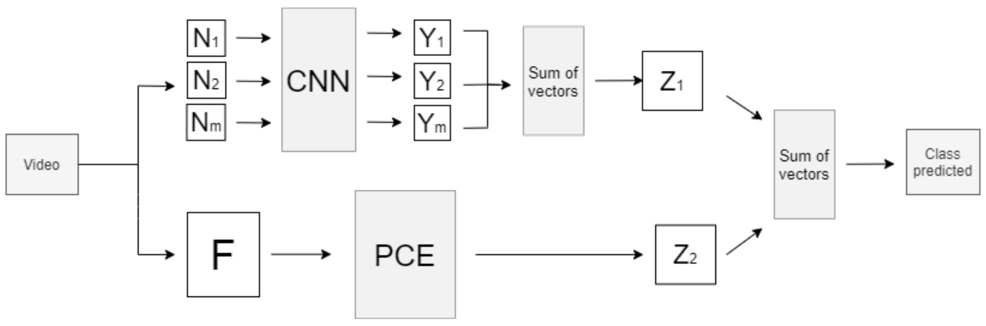

Figure 2.3: Path of video classification through PCE and CNN

2.5 Clustering

Clustering is a data analysis technique that is based on dividing data into homogeneous groups. By homogeneous group we mean a set of elements that resemble each other, i.e. that are not far away in a multidimensional space. The belonging of an element to a set depends on its proximity to the set itself. In this technique, the choice of the metric, that is how the distance between the data is calculated, has an crucial importance. A relevant difference between the various clustering techniques is whether the number of clusters into which you want to divide the data is known. If so, the number of clusters which equals the number of classes in the dataset provides an important aid to the algorithm. Otherwise the algorithm must be able to estimate how many classes there are in the dataset. In this case it will be necessary to indicate some more parameters, such as the distance and the number of data needed to create a new cluster.

2.5.1 Clustering with known number of cameras

The clustering algorithm with a known number of cameras used is k-medoids. This method is implemented in MATLAB's default library and, therefore, it was not necessary to re-implement it. K-medoids is similar to k-means, whose goal is to divide the observations into k subsets so that the sum of the distances between the observations and the center of the cluster to which they are assigned is as small as possible. The difference is that k-medoids use observations (medoids) as cluster centers.
There are several algorithms to implement k-medoids. The one used by MATLAB is called Partitioning Around Medoids (PAM).
The algorithm goes like this:

1. Randomly select k observations as medoids.

2. Assign the closest medoid to each data point.

3. In each cluster, each point o is tested as a potential medoid by checking if the sum of the distances within the cluster is reduced by using o as a medoid. If so, o replaces the old Medoid.

4. Repeat step 2,3 until there are no changes in the medoids.

As for the calculation of the distance between two noises, a MATLAB function called "crosscorr" was used. This function uses Fourier transform to calculate the correlation between two variables.

2.5.2 Clustering with unknown number of cameras

As for clustering with an unknown number of cameras, a method called "Hybrid Markov Clustering" (HAL) was used. It was proposed in a doctoral thesis awarded in Bologna. This method groups fingerprints based on the combination of Hierarchical and Markov clustering algorithms and on a threshold that is generated based on the results. The problem with hierarchical clustering is that assigning data to a cluster cannot be undone. If it is not correct, the error is propagated in subsequent iterations. HAL method which uses a hybrid approach overcomes this limitation.
The algorithm follows these steps:

1. At first, each observation is considered a cluster.
2. The dataset is partitioned into small batches.
3. A correlation matrix is calculated for each batch.
4. Cluster Markov is applied to each correlation matrix. The new clusters are selected and the threshold for joining the clusters is defined.
5. Hierarchical clustering takes place and at the end of the process similar observations are merged into the same cluster.
6. Repeat 2,3,4,5 until there are no changes in the clusters.

2.6 Evaluation Measures

There are several evaluation measures of classification and clustering models. The following were used in this project: precision(2.1), recall(2.2), F1-score(2.3), True Negative Rate(2.4) and False Positive Rate(2.5). In a multiclass classification, these measures can be defined for each class. Before proceeding with these definitions it is necessary to define other measures.

- True Positive (TP) of class k: It refers to the number of predictions where the classifier correctly predicts data of class k as being part of class k.
- True Negative (TN) of class k: It refers to the number of predictions where the classifier correctly predicts data of class not k as not being part of class k.
- False Positive (FP) of class k: It refers to the number of predictions where the classifier incorrectly predicts data of class not k as being part of class k.
- False Negative (FN) of class k: It refers to the number of predictions where the classifier incorrectly predicts data of class k as not being part of class k.

Through these formulas it is possible to define precision(2.1), recall(2.2), F1-score(2.3), True Negative Rate(2.4) and False Positive Rate(2.5).
Again the formulas are defined for each class.

$$Precision = \frac{TP}{(TP + FP)} \tag{2.1}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{2.2}$$

$$F1 = 2 \times \frac{Precision \times Recall}{(Precision + Recall)} \tag{2.3}$$

$$TNR = \frac{TN}{(TN + FP)} \tag{2.4}$$

$$FPR = \frac{FP}{(FP + TN)} \tag{2.5}$$

Once the values for each class have been calculated, there are two ways to obtain the average values for the entire classification. We can consider the average over all observations (micro average) or the average over the classes (macro average). The micro average is basically a weighted average on the number of observations for each class. Having a lot of data available, it is usually preferable to consider the macro average. For this reason, in this project we have chosen to refer to macro average.

We define the Random Index (RI) as follows:

$$RI = \frac{TP + TN}{(TP + FP + TN + TP)} \tag{2.6}$$

For two random classes we can define $\overline{RI}$ as the mean of RI of those two classes. We can define the adjusted rand index (ARI):

$$ARI = \frac{RI - \overline{RI}}{1 - \overline{RI}} \tag{2.7}$$

And defining as $|C|$ the number of the obtained classes, $\hat{c}_i$ denotes the number of observations with the dominant class label in the cluster c_i, and $|c_i|$ is the total number of observations in c_i. In this way we can define purity:

$$Purity = \frac{\sum_{i=1}^{|C|} \frac{|\hat{c}_i|}{|c_i|}}{|C|} \tag{2.8}$$

Chapter 3

Noise extraction and fingerprint calculation

This chapter describes the operations carried out in the thesis project on a dataset to extract the noises and to estimate the fingerprints. These operations are necessary to obtain the objects that will be classified. In particular, we will refer to a specific video dataset, called "Vision dataset", considered the reference in the area of source camera identification.
In previous works, in the field of camera source identification, there are various solutions to obtain noises and fingerprints. Most of these solutions, however, refer to photographs, and therefore can only partially be used for videos. In this thesis work several state of the art solutions were evaluated in order to obtain the approach that gave the best results in terms of classification. To compare two alternative solutions, it was decided to evaluate both approaches with the PCE-based classification and, of course, to choose the one that gave the best results.

3.1 Dataset

The project of this thesis took two datasets into consideration: Vision Dataset [28], a dataset regarded as the standard in this field and Smart video, a dataset collected by the University of Bologna. The Vision dataset is made up of both

photographs and videos, although we will only consider videos. The videos of the Vision dataset are present in three formats: native, uploaded to WhatsApp and uploaded to YouTube. Native videos correspond to how videos are stored on smartphone after being recorded. The other two formats are obtained through the upload and subsequent download of the same videos on two social platforms. Uploading videos to social networks involves compressions that decrease the quality of the video. The presence of this kind of videos is motivated by simulating a more common scenario where the videos, from which we want to derive the origin, are downloaded from social networks. All videos were recorded with a rear camera, each video has a duration of approximately one minute and there is a total of 646 videos for each group (native, WhatsApp and YouTube).
To compare the different methods, it was decided to use only the Vision dataset composed of native videos. It was decided to divide the dataset into training set and test set, with a 70-30 ratio.
In the next chapter the results of the datasets composed of the videos uploaded on social networks and Smart Video will be shown.
The basic steps to extract residual noises and calculate fingerprints from a dataset are as follows:

1. Rotation of videos to match pixels.
2. Resizing of all the videos to the same resolution.
3. Extraction of frames
4. Exclusion of some frames.
5. Noise extraction.
6. Countering the problem of video stabilization.
7. Noise normalization.
8. Calculation of the estimated fingerprints.

ID	Brand	Model	Native resolution	WhatsApp resolution	YouTube resolution	Number of videos
1	Apple	iPad 2	1280x720	848x480	1280x720	22
2	Apple	iPad mini	1920x1080	848x480	1920x1080	13
3	Apple	iPhone 4	1280x720	848x480	1920x1080	19
4	Apple	iPhone 4S	1920x1080	800x480	1920x1080	19
5	Apple	iPhone 4S	1920x1080	848x480	800x480	18
6	Apple	iPhone 5	1920x1080	848x480	1920x1080	17
7	Apple	iPhone 5	1920x1080	848x480	1920x1080	18
8	Apple	iPhone 5c	1920x1080	848x480	1280x720	37
9	Apple	iPhone 5c	1920x1080	848x480	1280x720	19
10	Apple	iPhone 5c	1920x1080	848x480	1280x720	15
11	Apple	iPhone 6	1920x1080	848x480	1920x1080	19
12	Apple	iPhone 6	1920x1080	848x480	1920x1080	19
13	Apple	iPhone 6 Plus	1920x1080	848x480	1920x1080	16
14	Asus	Zenfone 2 Laser	640x480	848x480	1280x720	19
15	Huawei	Ascend G6-U10	1280x720	848x480	1920x1080	18
16	Huawei	Honor 5C	1920x1080	848x480	1920x1080	19
17	Huawei	P8 GRA-L09	1920x1080	848x480	1920x1080	10
18	Huawei	P9 EVA-L09	1920x1080	848x480	1920x1080	13
19	Huawei	P9 Lite VNS-L31	1920x1080	848x480	1920x1080	19
20	Lenovo	Lenovo P70-A	1280x720	848x480	1920x1080	16
21	LG	electronics D290	800x480	848x480	1920x1080	11
22	Microsoft	Lumia 640 LTE	1920x1080	848x480	1920x1080	16
23	ePlus	A3000	1920x1080	640x480	1280x720	19
24	ePlus	A3003	1920x1080	848x480	640x480	19
25	Samsung	Galaxy S III	1280x720	848x480	1920x1080	19
26	Samsung	Galaxy S III	1280x720	848x480	1920x1080	16
27	Samsung	Galaxy S3	1920x1080	848x480	1280x720	19
28	Samsung	Galaxy S4	1920x1080	848x480	1920x1080	19
29	Samsung	Galaxy S5	1920x1080	848x480	1920x1080	29
30	Samsung	Galaxy Tab 3	1280x720	848x480	1920x1080	19
31	Samsung	Galaxy Tab A	1280x720	848x480	1920x1080	19
32	Samsung	Galaxy Trend Plus	1280x720	848x480	1920x1080	19
33	Sony	Xperia Z1 Compact	1920x1080	848x480	1920x1080	19
34	Wiko	Ridge 4G	1920x1080	848x480	1280x720	32
35	Xiaomi	Redmi Note 3	1920x1080	848x480	1280x720	16

Table 3.1: Characteristics of Vision dataset smartphones

3.2 Pre-processing

3.2.1 Rotation of videos to match pixels

A problem in the Vision dataset, and which could be present in any dataset, concerns the orientation of the video recording. The fingerprint is dependent on the orientation of the video, and therefore it is necessary to correct their rotation, in case the videos are recorded with different orientations.
As it can be seen in the image below, a video can be recorded in four different orientations. Taking the top of the smartphone as a reference point, it is necessary to rotate it so that the top of the smartphone is in the same direction in all videos. Of course, it is not fundamental to have all the videos in the dataset aligned to the same orientation, but only those coming from the same smartphone must have this requisite. In this way, the fingerprints extracted from videos of the same smartphone will be comparable.

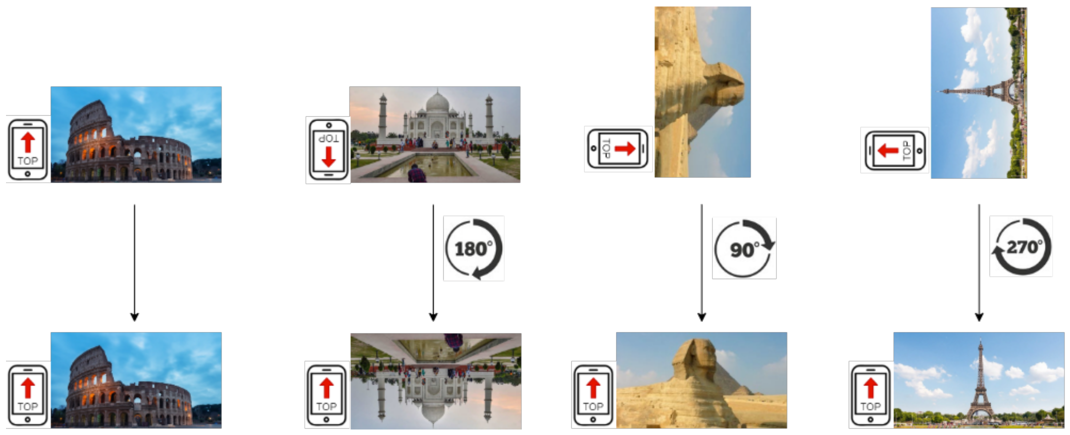

Figure 3.1: Rotations required to align videos to a single orientation.

Figure 3.1 shows the rotation required to align the videos in the four possible scenarios. There are several ways to identify the orientation in which a video was recorded. Having two videos V_1 and V_2 it is possible to extract the fingerprint from both and see with which rotation the two fingerprints have a higher correlation (using for example the PCE function). Actually, in the Vision dataset the orientation of each video is indicated, and this information was used to evaluate the method. In all cases, the rotation that maximizes the correlation was that indicated in the dataset. Although the orientation was indicated for each video, we chose to find a method that solves this problem without using this information to consider a more general case. Another possibility to solve this problem is to represent the fingerprints in such a way that they are independent of rotation [30].

3.2.2 Resizing of all the videos to the same resolution

Another operation required to make the videos comparable is resizing to the same resolution. In the vision dataset there are videos recorded with different resolutions, but the videos coming from the same smartphone have the same resolution. Considering the native dataset there are 4 different resolutions: 1280x720 (HD), 1920x1080 (Full HD), 640x480 and 800x480. A test was carried out to see how the classification results change when resizing all videos to a single resolution.

Resolution	**Precision**	**Recall**	**F1**	**TNR**	**FPR**	**ARI**	**Purity**
640x480	0.640	0.639	0.605	0.989	0.010	0.401	0.639
800x480	0.663	0.667	0.628	0.989	0.010	0.428	0.662
HD	0.719	0.741	0.698	0.991	0.008	0.508	0.715
Full HD	0.807	0.824	0.791	0.994	0.006	0.656	0.808

Table 3.2: Comparison considering different frame resizing

As expected, the results showed that scaling to a higher resolution leads to better classification.

3.2.3 Extraction of frames

A video is made up of a sequence of frames. Although frames may resemble photographs there are some important differences. In fact, some types of frames correspond to photographs, but others, which are the majority, contain little information on how the video evolves. In this way, a compression that drastically reduces the memory weight of the video occurs.
Most compression algorithms have three types of frames:

- **I-frame** (Intra-coded picture):
  An I-frame is a complete image, and can be decompressed without any other information.
- **P-frame** (Predicted picture):
  A P-frame holds only the changes in the image from the previous frame. To be decompressed in image it needs the precedent frame.
- **B-frame** (Bidirectional predicted picture)
  A P-frame holds the changes in the image from both the previous frame and the following frame. To be decompressed in image, it needs the precedent frame and the following frame.

In an I-frame, also called key frame, there is no compression, while in the other two, there is compression.

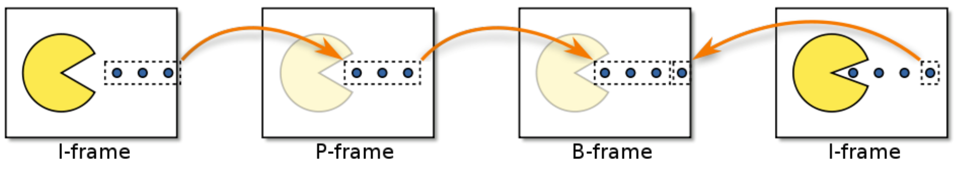

Figure 3.2: The different types of frames

In the figure 3.2, Pac Man images better show the difference between frames. The P-frame contains only the information on the balls, the Pac Man image comes from the previous frame. In The B-frame the image comes from the previous frame and the new ball comes from the next frame.
A program, called FFMpeg, was used to extract the frames. A one-minute video from the Vision dataset consists of approximately 1500 frames. Of these frames, 8% are I-frames, 66% B-frames and 26% P-frames.

3.2.4 Exclusion of some frames

Considering thousands of frames per video can be very expensive, especially for the subsequent noise extraction step. For this reason, one may wonder whether it is possible to give up on some frames. The first consideration to be made is to observe whether we can limit ourselves to considering I-frames. This is because the decompression of P-frames and B-frames uses the data coming from the key-frames and, therefore, the decompressed images closely resemble those obtainable from the key-frames. Furthermore, I-frames are much less present than other types of frames, about one in thirty, and limiting ourselves to considering them would allow us to greatly reduce the computational calculation.

Frames considered:	**Precision**	**Recall**	**F1**	**TNR**	**FPR**	**ARI**	**Purity**
I-frames	0.807	0.824	0.791	0.994	0.006	0.656	0.808
I-frames, P-frames	0.807	0.824	0.791	0.994	0.006	0.656	0.808
I-frames, P-frames, B-frames	0.807	0.824	0.791	0.994	0.006	0.656	0.808

Table 3.3: Comparison considering different types of frames

The table shows the same results for all configurations. This can lead us to limit ourselves to key-frames only. To extract the key-frames, the FFMpeg program has always been used with a specific flag to indicate the type of frame to extract.
Furthermore, saturated and dark images are not useful for calculating the fingerprint and can be eliminated. The value of each pixel in a grayscale image is between 0 and 255, where 0 represents black and 255 represents white. A saturated image can be defined as an image that has a given percentage of k pixels with at least a value above a threshold. The choice of threshold and number of pixels determines how strict we are in judging a saturated an image as saturated. Similarly, we can define a dark image.

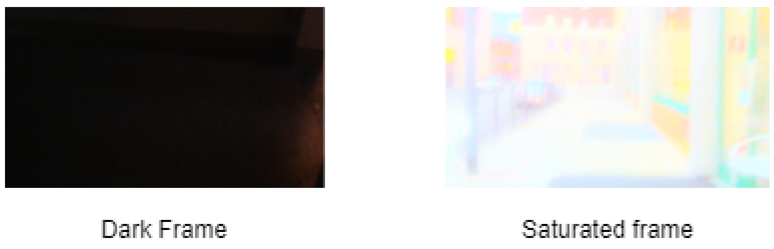

Figure 3.3: Example of a saturated and dark image

Using a part of the training set as a training set we determined the parameters that allowed to consider the frames saturated or dark and to eliminate them. The best result was obtained by eliminating the frames that have 81% of the pixels below 41 and those that have 25% of the pixels above 249.

Method	**Precision**	**Recall**	**F1**	**TNR**	**FPR**	**ARI**	**Purity**
Without elimination of saturated and dark frames	0.773	0.785	0.752	0.993	0.006	0.573	0.773
Elimination of saturated and dark frames	0.807	0.824	0.791	0.994	0.006	0.656	0.808

Table 3.4: Comparison considering elimination of dark and saturated frames

3.2.5 Noise extraction

The reference algorithm for the extraction of image noises is BM3D in the field of source camera identification. This algorithm can also be applied to single video frames as well as to photographs. Working with videos, we also have other algorithms such as V-BM4D which also exploit the temporal dimension of the frames in the noise extraction.

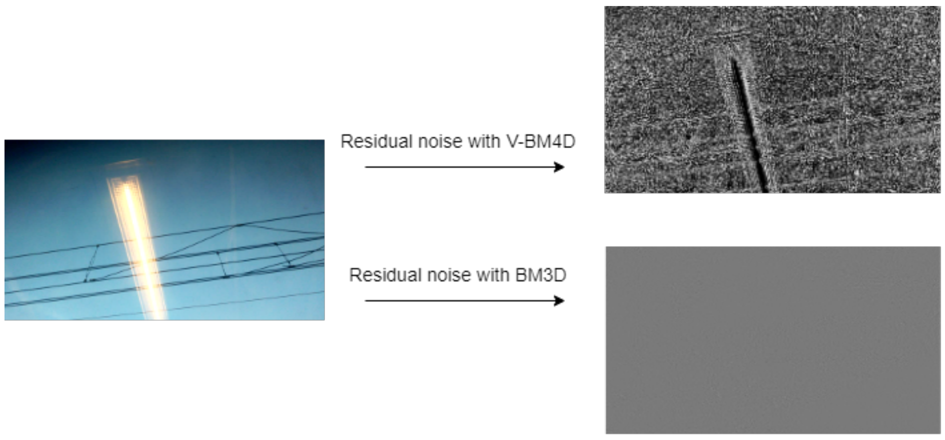

Figure 3.4: Comparison of residual noises using BM3D and V-BM4D

Since the extraction time with the V-BM4D algorithm is very long, in the order of tens of hours for a one-minute video, in order to make the comparison between the two extraction techniques, it was decided to use another dataset on the residual noises extracted with V-BM4D which were already available. The extraction of noises on that dataset was carried out in another thesis work of the University of Bologna. The dataset is a part of the one collected by the University of Bologna.

ID	Brand	Model	Video Resolution	Number of videos
1	Huawei P9 Lite	Huawei VNS-L31	1920x1080	20
2	LG	Nexus 5	1920x1080	20
3	LG	Nexus 5	1920x1080	20
4	motorola	moto g2	1280x720	20
5	Apple	iPhone SE	3840x2160	20

Table 3.5: Composition of the dataset used to compare BM3D and V-BM4D

The table below shows the comparison results. While the results can be compromised by a rather small data set, they are clear enough. This comparison prompted us to use BM3D extraction also in the case of the Vision dataset.

Noise extraction method	Precision	Recall	F1	TNR	FPR	ARI	Purity
V-BM4D	0.766	0.760	0.753	0.941	0.058	0.449	0.760
BM3D	0.920	0.926	0.909	0.981	0.018	0.813	0.920

Table 3.6: Comparison between different noise extraction techniques

3.2.6 Counter the problem of video stabilization

About half of the devices in the vision dataset, 16 out of 35, have video stabilization. This effect correcting the flickering of the shot means it is not always the same pixel of the camera that captures the same portion of the image in the succession of frames. So, this makes it harder to find the device fingerprint in the residual noise.

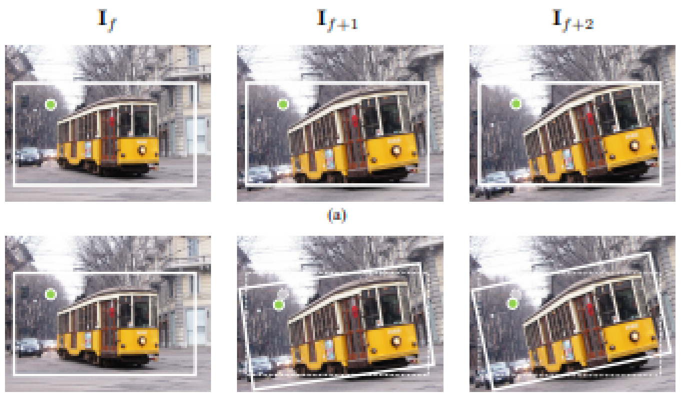

Figure 3.5: Comparison between unstabilized(a) and stabilized(b) video

Figure 3.5 shows that in the absence of stabilization, the coordinates of the pixels in the images remain the same, while with stabilization they can change, through transformations such as translation, scale and rotation.
There are three ways to overcome this obstacle:

- Ignoring video stabilization and proceeding as if it was not there. The fingerprint calculation is done as a simple average of the residual noises.
- Recognizing and eliminating the frames the most subject to video stabilization. The fingerprint is the sum of the remaining noises.
- Making geometric transformations to the frames in order to try to cancel the effect of stabilization. The fingerprint occurs as the sum of the noises after they have undergone the transformations.

To recognize the frames most subject to the effect of video stabilization, it is possible to evaluate the correlation between the noise of the first frame of a video and the following ones. Low correlation may indicate strong stabilization. The first frame is taken as a reference as it is not subject to stabilization: if the video was cut and the first frame was not available, the frame that has the highest correlation with all the others could be taken as a reference. In order to do this I wrote a program that simply calculates the correlation between noises and considers a noise in the fingerprint calculation only if the correlation reaches a certain k value.

Deleting frames with PCE below	Precision	Recall	F1	TNR	FPR	ARI	Purity
0 (no frames deleted)	0.598	0.637	0.588	0.989	0.01	0.45	0.668
5	0.692	0.721	0.676	0.99	0.009	0.508	0.703
10	0.807	0.824	0.791	0.994	0.006	0.656	0.808
20	0.779	0.807	0.763	0.993	0.006	0.604	0.779
30	0.715	0.744	0.694	0.991	0.008	0.531	0.720

Table 3.7: Comparison of different thresholds to eliminate stabilized frames

Similarly, we can determine what geometric transformations are needed to maximize correlation and then apply them to try to eliminate the effect of video correlation.
As a correlation function we have chosen to use the PCE function. To follow the third approach, a program available on GitHub [31] and described in a paper was used. That program allows us to calculate the fingerprint by performing geometric operations such as shifts, rotations and scaling on residual noises.
The following table presents the results of the three methods listed above. The second method eliminates the frames that have a PCE value (calculated with the first frame) lower than 10, as it was the best threshold. The third method corrects the video stabilization by making geometric transformations.

Method	Precision	Recall	F1	TNR	FPR	ARI	Purity
Ignore video stabilization	0.655	0.701	0.647	0.989	0.011	0.421	0.655
Elimination of more stabilized frames	0.807	0.824	0.791	0.994	0.006	0.656	0.808
Correct the stabilization	0.688	0.736	0.672	0.990	0.009	0.464	0.680

Table 3.8: Comparison of different methods to manage video stabilization

The table shows that the second way, that is the one which consists in eliminating the frames more subject to stabilization, produces the best results. Applying geometric transformations probably alters frames too much and goes beyond correcting video stabilization.

3.2.7 Noise normalization

The extracted residual noises have a brightness dependent on the frame from which they are extracted. As brightness we can consider the average value of the intensity of the pixels in an image. The different brightness between the noises can be used as a characteristic on which the classification takes place. However, this feature is determined more by the brightness of the video than by the fingerprint of the device that recorded the video. What has been done to eliminate this factor is to apply a normalization to the noises.

Given a noise R, we define a pixel of coordinates (x, y) as R (x, y). Given that the intensity of a pixel is in a range between 0 and 255, the average value is 127.5. Furthermore, we consider AVG the average of the intensity of all pixels of N. We can apply this formula on each pixel of the normalized noise N':

$$N'(x,y) = 127,5 + (N(x,y) - AVG) \tag{3.1}$$

Applying this formula to all noises in the dataset, they will have the same average intensity. In this way, the difference in brightness will no longer be considered a characteristic on which the classification takes place. Whereas, the aspect on which the classification must take place, determined by the fingerprints of the cameras, i.e. the difference in intensity between the pixels of the noises, is maintained.
Now let's see how the classification changes by performing the noise normalization.

Normalization	**Precision**	**Recall**	**F1**	**TNR**	**FPR**	**ARI**	**Purity**
no	0.771	0.792	0.76	0.993	0.006	0.612	0.779
yes	0.807	0.824	0.791	0.994	0.006	0.656	0.808

Table 3.9: Comparison between the use and non-use of normalization.

As we can see, normalization allows for slightly better results.

3.3 Fingerprint calculation

To estimate the fingerprint of a series of photographs, it is sufficient to make the average, pixel by pixel, of the residual noise of the photographs. To do it in a video there are a few more possibilities. We can proceed as in the case of photographs, but also with more sophisticated approaches.
Using the PCE function it is possible to determine the shift that maximizes the correlation between two images.

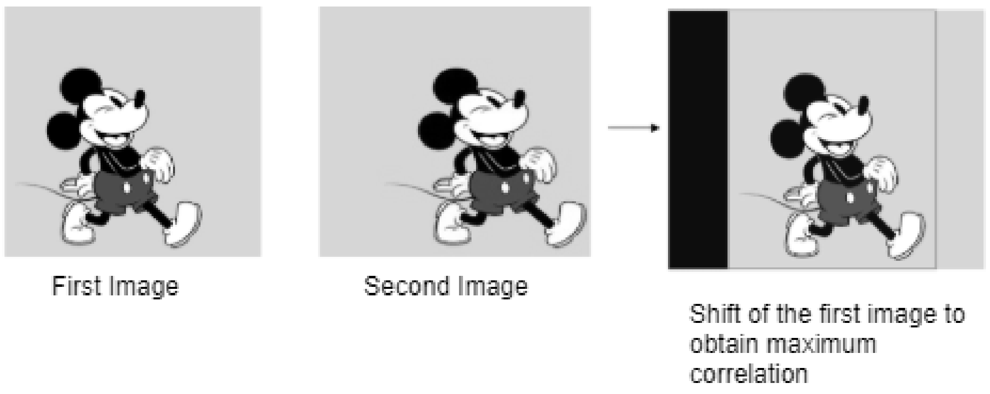

Figure 3.6: Shift that maximizes the correlation between two images

In the same way it is possible to determine the shift that maximizes the correlation between two noises. By doing this, we can try to correct the effect of video stabilization. The problem with this operation is that by shifting a noise, a part of it will not fit into the image frame, as seen in image 3.6. It is possible to solve this problem by adding a black frame around each noise, which will allow us to perform shifts while keeping the noise within the frame.

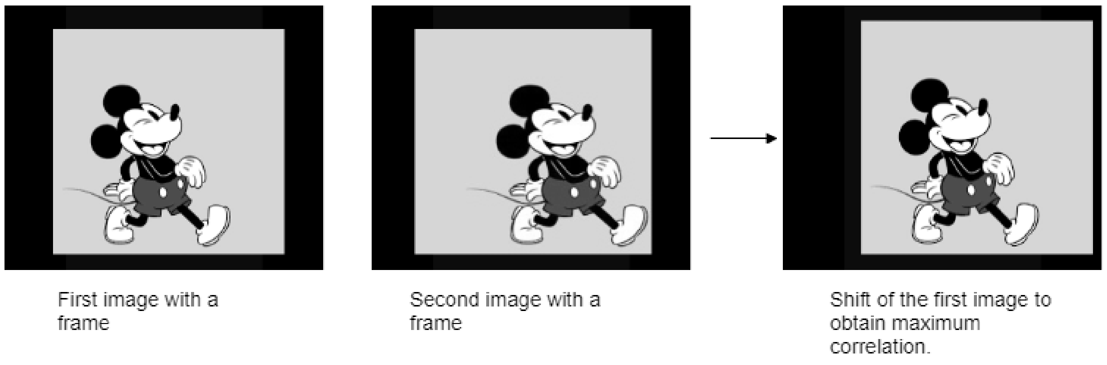
First image with a frame
Second image with a frame
Shift of the first image to obtain maximum correlation.

Figure 3.7: Shifting to the right does not take the image out of the frame

Three different ways of calculating video fingerprints were evaluated:

- fingerprint calculation as an average of the noises (a).
- fingerprint calculation as an average of the shifted noises to correct the video stabilization (b).
- fingerprint calculation as the average of the shifted noises, with the addition of a frame (c).

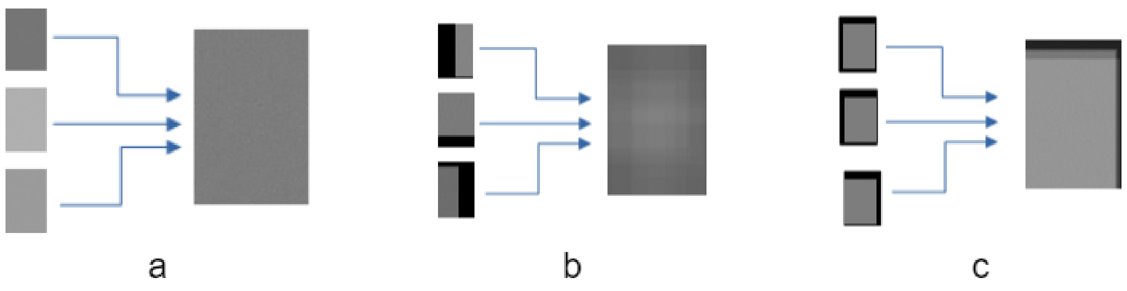

Figure 3.8: Different methods of calculating the estimated fingerprints

fingerprint calculation method	**Precision**	**Recall**	**F1**	**TNR**	**FPR**	**ARI**	**Purity**
Method a	0.807	0.824	0.791	0.994	0.006	0.656	0.808
Method b	0.763	0.78	0.74	0.992	0.007	0.555	0.761
Method c	0.801	0.815	0.782	0.994	0.006	0.642	0.801

Table 3.10: Comparison of different techniques for calculating fingerprints

The results show that using the simple average to calculate the fingerprint is the best solution.

Chapter 4

Experimental results

This chapter shows the classification and clustering results in the context of source camera identification. Two datasets were used: Vision dataset, the dataset considered the reference in this area, and Smart Video dataset, collected by the University of Bologna. In reality, the Vision dataset consists of native videos and videos uploaded to YouTube and WhatsApp.
The pre-processing procedures seen in chapter 3 will be applied to each dataset. The classification and clustering techniques are presented in chapter 2.

4.1 Results on Vision's native dataset

This dataset was the one taken as a reference in the choice of pre-processing operations to be performed on the videos before classification and clustering. The tables below show the results of classification and clustering with the techniques described in the previous chapters.

Considering	**Precision**	**Recall**	**F1**	**TNR**	**FPR**	**ARI**	**Purity**
PCE	0.807	0.824	0.791	0.994	0.006	0.656	0.808
CNN	0.817	0.831	0.810	0.994	0.005	0.671	0.813
Combination of three CNN	0.866	0.880	0.864	0.996	0.003	0.740	0.866
PCE + CNN	0.879	0.896	0.876	0.996	0.003	0.771	0.877

Table 4.1: Classification on Vision's native dataset

The results in Table 4.1 are similar for the first two classification techniques, while they are better using the two methods combined.

Considering	**Precision**	**Recall**	**F1**	**TNR**	**FPR**	**ARI**	**Purity**
k-medoids	0.266	0.705	0.386	0.972	0.028	0.358	0.407
HAL	0.541	0.496	0.517	0.987	0.013	0.503	0.536

Table 4.2: Clustering on Vision's native dataset

As it can be seen, the results obtained with the classification techniques are better than those obtained with clustering techniques. This can be explained by the fact that the classification includes a training phase, while clustering does not.

4.2 Results on Vision's YouTube dataset

It is also interesting to consider datasets composed of videos uploaded on social networks. In this case Vision's YouTube dataset is made up of the same videos from the native dataset with the difference that these videos have been uploaded, and subsequently downloaded, from YouTube. Following these steps, the videos undergo compression operations that significantly reduce resolution and quality.

Considering	**Precision**	**Recall**	**F1**	**TNR**	**FPR**	**ARI**	**Purity**
PCE	0.492	0.520	0.455	0.985	0.014	0.306	0.548
CNN	0.645	0.676	0.637	0.989	0.01	0.467	0.682
PCE + CNN	0.671	0.694	0.658	0.991	0.009	0.509	0.695

Table 4.3: Classification on Vision's YouTube dataset

Considering	**Precision**	**Recall**	**F1**	**TNR**	**FPR**	**ARI**	**Purity**
k-medoids	0.052	0.331	0.081	0.794	0.202	0.085	0.094
HAL	0.168	0.151	0.159	0.955	0.45	0.136	0.167

Table 4.4: Clustering on Vision's YouTube dataset

By performing the classification and clustering techniques on the same videos, but uploaded to a social network, the performance drops significantly. In particular, the results of clustering have significantly deteriorated.

4.3 Results on Vision's WhatsApp dataset

In this case the dataset is made up of videos uploaded and downloaded from WhatsApp.

Considering	**Precision**	**Recall**	**F1**	**TNR**	**FPR**	**ARI**	**Purity**
PCE	0.513	0.557	0.493	0.985	0.014	0.328	0.543
CNN	0.534	0.618	0.530	0.986	0.013	0.317	0.567
PCE + CNN	0.548	0.629	0.545	0.987	0.012	0.360	0.590

Table 4.5: Classification on Vision's WhatsApp dataset

Considering	**Precision**	**Recall**	**F1**	**TNR**	**FPR**	**ARI**	**Purity**
k-medoids	0.070	0.457	0.131	0.824	0.176	0.075	0.141
HAL	0.230	0.111	0.150	0.964	0.036	0.140	0.172

Table 4.6: Clustering on Vision's WhatsApp dataset

In this case, the performance is even worse than the YouTube dataset, as the compression carried out on WhatsApp is even stronger.

4.4 Results on Smart Data dataset

The dataset in question consists of about 520 videos, recorded by some students of the University of Bologna who collaborated in the creation of the dataset. Videos were recorded from 13 devices. For each device there are 20 videos recorded with the rear camera and 20 with the front camera. All videos are native, i.e. they have not been uploaded to social platforms.

ID	Brand	Model	Front Camera	Rear Camera
1	Samsung	Galaxy Core Prime	640x480	1280x720
2	Huawei	VNS-L31	1280x720	1920x1080
3	LG	Nexus 5x	1920x1080	1920x1080
4	LG	Nexus 5	1280x720	1920x1080
5	OnePlus	OnePlus 3	1920x1080	1920x1080
6	LG	X screen	1920x1080	1920x1080
7	LG	Nexus 5	1280x720	1920x1080
8	Apple	iPhone 7	1920x1080	3840x2160
9	Nokia	635	absent	1280x720
10	One Plus	One Plus 3	1920x1080	1920x1080
11	motorola	moto g2	1280x720	1280x720
12	Apple	iPhone SE	1280x720	3840x2160
13	meizu	mx4	1920x1080	3840x2176

Table 4.7: Characteristics of Smart Video dataset smartphones

Only videos recorded with the rear camera were considered in the tests.

Considering	**Precision**	**Recall**	**F1**	**TNR**	**FPR**	**ARI**	**Purity**
PCE	0.884	0.904	0.889	0.990	0.009	0.753	0.884
CNN	0.926	0.935	0.927	0.993	0.006	0.843	0.926
PCE + CNN	0.938	0.946	0.939	0.994	0.005	0.864	0.938

Table 4.8: Classification on Smart Data dataset

The performance on this dataset is in line with the Vision native dataset, considering the different number of devices.

4.5 Comparison with other studies

In the field of source camera identification there are several studies done on photographs and a few carried out on videos. Almost all of the studies on videos are concerned with showing the correlation between the fingerprints extracted from two videos rather than classifying the videos of an entire dataset. In December 2020, a paper was published proposing the use of a neural network for the classification of the videos of the Vision dataset[32]. The study has some similarities with the work done in this thesis: in both works the neural network classifies the single frames of the videos, and the most predicted class for frames is the prediction of the video. But there are also some imported differences: the paper classifies the frames directly, while our work classifies the noise extracted from the frames. Furthermore, the paper does not consider some aspects such as video stabilization or the choice of which frames to use.
In order to make a comparison between the two methods, I repeated the classification under the conditions described in the article. Taking the native Vision dataset as a reference, considering only 28 devices out of 35 available and maintaining a training-test split of 55-45. The method proposed by the paper shows an accuracy of 66.5%, while the classification method by CNN proposed in this thesis has an accuracy of 82.3%. Probably, the greatest difference is given by the classification

made on the noises extracted from the frames rather than directly on the frames. However, this comparison justifies the effectiveness of the model proposed in this thesis.

Conclusions

This thesis focused on the field of video source camera identification. In particular, classification and clustering techniques are proposed in order to group videos from the same device into the same class. This association occurs thanks to the fact that each device leaves its own fingerprint in the contents it records, in the form of noise. The noise represents that part of the image that does not depend on the photographed or recorded object, but on the fingerprint and some other random factors as well. Once we have extracted noises from the videos (typically a noise for each frame of the video), we can try to estimate the fingerprint of the device starting from the noises and trying to eliminate the random component that does not depend on the fingerprint.
A significant part of the work is dedicated to finding the best way to extract the noises from each video and calculate the fingerprint. To do this, the various approaches present at the state of the art are evaluated through a classification algorithm and the one that gave the best results is chosen.
Once the noises and the fingerprints are obtained, the classification and clustering algorithms can be applied. As regards the classification, three techniques are proposed. One is based on a correlation function between images, called Peak to Correlation Energy (PCE). The correlation between the video test fingerprint and the training video fingerprint is calculated and the class showing the highest value will be the predicted class. Another classification technique is based on the convolutional neural network. The network is trained on the noises of the training set videos. To predict the class of a video it is necessary to predict the class of all the noises that make up the video. The most predicted class will correspond

to the class attributed to the video. A similar technique was presented in a paper in December 2020 and the one proposed in this thesis shows better results. The latest classification technique combines classification with CNN and with PCE and is found to be the one that produces the best results.
Two clustering techniques are applied, one that does not know the number of classes present and another that knows them. The first technique was proposed in a doctoral thesis of the University of Bologna in the field of source camera identification of photographs, and has therefore been adapted to work on videos. The second is an algorithm natively present on the MATLAB platform. Classification results are better than clustering probably because clustering does not include a training phase.
Two datasets are used: Vision, the dataset considered the reference in this field of study, and Smart Data, a dataset collected by the University of Bologna. Vision also includes videos uploaded to two social platforms: You-
Tube and WhatsApp. The videos uploaded to social networks are compressed and therefore lose information, but this aspect can help us simulate a real scenario of digital forensics in which the videos are downloaded from the web.
This work could be expanded by building a classification system that combines videos and photographs. In this way both multimedia contents would be exploited, and it would certainly be interesting from a forensic point of view. Having more computational power available, it would also be possible to build more complex neural networks that would probably give better results.

Appendix A

Source code for classification

In this appendix we find the codes, written in MATLAB language, used for classification by PCE, with the convolutional network and with both approaches.

A.1 Classification with Peak to Correlation Energy

The first function calculates the PCE correlation between two fingerprints. The function takes as input two fingerprints, x and y, calculates the correlation matrix between the two, and passes the matrix to the library function PCE which determines the correlation PCE. The returned value y will be the numeric value expressing the PCE correlation.

```
function y = PCEVALUE(x,z)
    C = crosscorr(x, z);
    detection = PCE(C);
    y = detection.PCE;
end
```

The Fingerprint Comparison function uses the PCEVALUE function to compare a test fingerprint with all other training fingerprints. Each time a correlation is computed it is added to the FingCom array and is eventually restored after changing the absolute values of the correlation with the relative values of each pair relative to the others.

```
function FingCom = FingerprintComparison (train, test)
    test_img = imread(strcat(test));

    FingCom = [];
    files = dir(train);
    dirFlags = [files.isdir];
    subFolders = files(dirFlags);

    for k=3:length(subFolders)
        files2 = dir(strcat(train, subFolders(k).name));
        train_img = strcat(train,"", subFolders(k).name , "\",
            files2(3).name);
        train_img = imread(train_img);

        y = PCEVALUE(train_img,test_img);
        FingCom = [FingCom y];
    end

    tot = sum (FingCom);
    FingCom = arrayfun(@(x) (x/tot)*100, FingCom);

end
```

Finally, the PCE classification function that iterates over all the videos in the test set and saves the predicted class and the correct class in an array for each one.

```
function [y_val,y_pred] = PCE_classification (train, test)
y_val = [];
y_pred = [];

files = dir(test);
dirFlags = [files.isdir];
subFolders = files(dirFlags);

    for k = 3 : length(subFolders)

        path2 = strcat (test,subFolders(k).name, "\" );
        files2 = dir(path2);
        dirFlags2 = [files2.isdir];
        subFolders2 = files2(dirFlags2);

        for kk = 3: length(subFolders2)

            test2 = strcat(test, subFolders(k).name,"\",
               subFolders2(kk).name, "\fingerprint.png");
            FingCom = confrontoFingerprint (train, test2);

            [_,predicted] = max(FingCom);

            y_val = [y_val, str2num(subFolders(k).name)];
            y_pred = [y_pred, predicted];
        end
    end
end
```

A.2 Classification with Convolutional Neural Network

The following code allows us to train the convolutional neural network with the noises extracted from the videos. In this way, a model is built that can predict which device a single noise belongs to.

```
imds = imageDatastore('D:\Dataset Native\train\','IncludeSubfolders',true,'LabelSource','foldernames', 'FileExtensions','.tiff');
imds_test = imageDatastore('D:\Dataset Native\test\','IncludeSubfolders', true,'LabelSource','foldernames', 'FileExtensions','.tiff');

augimdsTrain = augmentedImageDatastore([1080 1920],imds);
augimdsValidation = augmentedImageDatastore([1080 1920],imds_test);

miniBatchSize = 2;
valFrequency = floor(numel(augimdsTrain.Files)/miniBatchSize);

options = trainingOptions('sgdm', ...
    'ExecutionEnvironment','gpu', ...
    'MiniBatchSize',miniBatchSize, ...
    'MaxEpochs',30, ...
    'InitialLearnRate',3e-4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',augimdsValidation, ...
    'ValidationFrequency',valFrequency, ...
    'Verbose',false, ...
    'shuffle', 'every-epoch', ...
    'Plots','training-progress', ...
    'CheckpointPath', 'D:\CheckPoint\' );

```

```

TrainedNetVisionDataset = trainNetwork(augimdsTrain,lgraph_2,
    options);
```

Once we have built a classifier for individual noises it is easier to build one for the entire video. Function CNN_video_classification does that: it takes as input the directory containing the noises of a video and the already trained CNN network. Following the prediction of the model, for each noise a vector is produced which represents the probability that the noise is part of each of the classes of the dataset. We add up the vectors of each noise and we obtain the probability that the video is part of each class.

```
function y_pred = CNN_video_classification (test_path,
    TrainedNetNoise)

     Files=dir(path);
     nElements = length(Files);

     imds_val = imageDatastore(test_path,'IncludeSubfolders',
         true,'LabelSource','foldernames', 'FileExtensions','.png
         ');
     imds_val = augmentedImageDatastore( [1080 1920],imds_val);
     y_pred = predict(TrainedNetNoise, imds_val);

     y_pred =  (sum  (y_pred(1:nElements,:)));
     tot = sum (y_pred);
     S = arrayfun(@(x) (x/tot)*100, y_pred);

end
```

A.3 Classification with Peak to Correlation Energy plus Convolutional Neural Network

Once built a classifier that uses the PCE function and one that uses a neural network, it is enough have to join them to build a third one. It is needed to take the vector of predictions produced by the first method and add it to the vector obtained by the second method and normalize it. In this way the two predictions are considered equally.

```
function [y_val,y_pred] = PCEplusCNN_classification (train,
    fingerprint_test, noise_test)
y_val = [];
y_pred = [];

files = dir(test);
dirFlags = [files.isdir];
subFolders = files(dirFlags);

    for k = 3 : length(subFolders)

        path2 = strcat (test,subFolders(k).name, "\" );
        files2 = dir(path2);
        dirFlags2 = [files2.isdir];
        subFolders2 = files2(dirFlags2);

        for kk = 3: length(subFolders2)

            test2 = strcat(fingerprint_test, subFolders(k).name
                ,"\",subFolders2(kk).name, "\fingerprint.png");
            test3 = strcat(noise_test, subFolders(k).name, "\",
                subFolders2(kk).name, "\" );

            ResultsPCE = PCE_classification (train, test2);
            ResultsCNN = CNN_video_classification(test3,
                TrainedNetNoise);

            for k=1:size(ResultsPCE, 2)
                x =  (ResultsPCE(k)+ResultsCNN(k));
                Results = [Results, x];
            end

            tot = sum (Results);
            Results = arrayfun(@(x) (x/tot)*100, Results);
            [M,I] = max(Results);

            [~,predicted] = max(FingCom);
            y_val = [y_val, str2num(subFolders(k).name)];
            y_pred = [y_pred, predicted];

        end
    end
end
```

Appendix B

Code for pre-processing operations

This appendix presents the code used to extract noises from videos and calculate fingerprints.

B.1 Video rotation and resizing

The code below is needed to rotate the videos in order to have the same orientation. This is necessary as the fingerprint is dependent on the video orientation and you cannot compare fingerprints from videos with different orientations. In the Vision dataset there is a file called Orientations.csv which contains the orientation at which each video was recorded. The following program reads this information and applies reverse rotation to return a video to its default orientation. Another operation that the code does is to resize each video to a fixed size (full HD).

```
orientations = readtable('Orientations.csv');
path = "Vision_noises\";
files = dir(path);
dirFlags = [files.isdir];
subFolders = files(dirFlags);
```

```

for k = 3: length(subFolders)

    path2 = strcat(path, subFolders(k).name, "\");
    files2 = dir(path2);
    dirFlags2 = [files2.isdir];
    subFolders2 = files2(dirFlags2);

    for kk = 3: length(subFolders2)

        name = subFolders2(kk).name;
        u = (t(strcmp(orientations.FileName, name), :));
        rot = (u.Rotation);

        if (rot~=0)
            path3 = strcat(path2, subFolders2(kk).name, "\");
            files3 = dir(path3);
            dirFlags3 = [files3.isdir];

            path4 = strcat(path3, "\key-frames\");
            files4 = dir(path4);

            for kkk=3: length(files4)
                path5 = strcat (path4, files4(kkk).name);
                I = imread(path5);
                I = imrotate(I,rot);
                I = imresize(I, [1920 1080]);
                imwrite(I, path5);
            end
        end
    end
end
```

B.2 Extraction of key frames from a video

The following cmd command uses the FFmpeg library to extract keyframes from a video.

```
ffmpeg -skip_frame nokey -i "video.mp3" -vsync 0 -frame_pts
    true "frame-%02d.png"
```

B.3 Removal of saturated and dark frames

The ImageIsNotDarkOrSaturated function is used to determine if an image is saturated, dark or not according to some given parameters. If the image has a percentage greater than perc1 of pixels below an underbound threshold or a percentage greater than perc2 of pixels above the overbound threshold, the image is considered dark and saturated, respectively. The function therefore allows us to determine which frames can be eliminated as they are not useful for classifications, because they are compromised by too high or too low brightness of the image.

```
function ok = ImageIsNotDarkOrSaturated(img, underbound, perc1,
     overbound, perc2)
img = imread(img);
img = rgb2gray(img);
Dark = 0;
White = 0;
X = size(img,1);
Y = size(img,2);
totpixel = X*Y;

for x=1:X
        for y=1:Y
            % reverse
            if(img(x,y) > underbound)
```

```
                Dark = Dark + 1;
            end
            if(img(x,y) < overbound)
                White = White + 1;
            end
        end
end

v1 = (perc1/100);
v2 = (perc2/100);
LIMIT1 = totpixel*v1;
LIMIT2 = totpixel*v2;

if(LIMIT1 > Dark && LIMIT2 > White)
    ok = 1;
else
    ok = 0;
end

end
```

B.4 Noise extraction

The noise_extraction function extracts the noises from a series of frames given in input. Inside the code, the BM3D function is called which returns the image with the noise removed. To get the noise, it is enough to calculate the difference between the original image and the clean image.

```
function Noise_extraction(input, output, output2, video)

S = dir(fullfile(input,'*.png'));

```

```
    for k = 1:numel(S)
        F = fullfile(input,S(k).name);
        IMG = imread(F);
        IMG = rgb2gray(IMG);

        [A, y_est] = BM3D(1,IMG,4, 'profile', 'lc');
        rn = im2double(IMG)-y_est;
        rn = mat2gray(rn);

        A = NoiseExtractFromImage(IMG,2);

        imwrite(rn, strcat(output,'\',erase(video,"mp4"), S(k).
            name));
        imwrite(A, strcat(output2,'\',erase(video,"mp4"), erase
            (S(k).name, 'tiff'), '.jpg'));
    end

end
```

B.5 Removal of highly stabilized frames

The Elimination_stabilized_frames function only saves those frames that have a correlation (measured by the PCE function) greater than 10 respect to the first frame (considered the reference as the first frame of a video is not usually subject to stabilization). The value of 10 was chosen as it was found to be the best threshold for eliminating a frame in terms of the classification results. Furthermore, if the video is cut, we can consider as a reference not the first frame, but the frame that shows a higher correlation with all the others.

```
function Elimination_stabilized_frames(frames, noises, output)

addpath(genpath('CameraFingerprint'));
n_frames = size(frames, 3);

%considering the first frame as reference
reference_frame = frames(1);
reference_noise = noises(1);
imwrite(reference_noise, strcat(output, "\frame1.tiff"))

for k=2:n_frames

    C = crosscorr((ref_noise.*double(frames(k))), noises(k));
    detection = PCE(C, size(reference_noise)-1);
    PCEvalue= detection.PCE;

    if(PCEvalue>10)
        imwrite(noises(k), strcat(output, "\frame", num2str(k),
            ".tiff"))
    end

end
```

B.6 Noise normalization

The Image Normalization function receives an image as input and returns the same normalized. This is done by calculating the average of the pixel intensity and updating the value of each pixel in relation to the average.

```
function img = ImgNormalization(img)

    dim1 = size(img,1);
    dim2 = size(img,2);
    avg = sum(img, 'All')/(dim1*dim2);

    for k=1:dim1
        for kk=1:dim2
            img(k,kk) = (255/2) +( img(k,kk) - avg);
        end
    end
end
```

B.7 Fingerprint calculation

The GetFingerprint function takes as input a directory of noises and saves the fingerprint in the folder defined by the second parameter. This is done by reading all the noises and updating the fingerprint as an average of them.

```
function GetFingerprint(input, output)

   files2 = dir(input2);

  C = zeros(1920,1080);

   for k=3:length(files2)

       img = imread(strcat(input2, files2(k).name));
       img = imresize(img, [480 848]);
       I = im2double(img);
      C = C + (I/(length(files2)-2));
```

```

    end

    mkdir(output);
    imwrite(C,strcat(output, "\fingerprint.tiff") );

end
```

Bibliography

[1] Luka, J., Fridrich, J., & Goljan, M. (2006). Digital Camera Identification From Sensor Pattern Noise. IEEE Transactions on Information Forensics and Security, 1(2), 205-214. https://doi.org/10.1109/tifs.2006.873602

[2] El Gamal, A., Fowler, B. A., Min, H., & Liu, X. (1998). Modeling and estimation of FPN components in CMOS image sensors. Solid State Sensor Arrays: Development and Applications II. https://doi.org/10.1117/12.304560

[3] Holst GC., Lomheim TS. (2007). CMOS/CCD Sensors and Camera Systems. Bellingham, Wash.; The International Society for Optical Engineering 2007; JCD Publishing

[4] Dierickx B. (2015). Imperfections of high-performance image sensors. Conference: Lorenz WorkshopAt: Leiden (NL)Volume: Artefacts in X-Ray Tomography.

[5] Goljan, M., Fridrich, J., & Filler, T. s. (2009). Large scale test of sensor fingerprint camera identification. Media Forensics and Security. https://doi.org/10.1117/12.805701

[6] Dey, S., Roy, N., Xu, W., Choudhury, R. R., & Nelakuditi, S. (2014). AccelPrint: Imperfections of Accelerometers Make Smartphones Trackable. Proceedings 2014 Network and Distributed System Security Symposium. https://doi.org/10.14722/ndss.2014.23059

[7] Willers, O., Huth, C., Guajardo, J., & Seidel, H. (2016). MEMS Gyroscopes as Physical Unclonable Functions. Proceedings of the 2016

ACM SIGSAC Conference on Computer and Communications Security. https://doi.org/10.1145/2976749.2978295

[8] Jin, R., Shi, L., Zeng, K., Pande, A., & Mohapatra, P. (2016). MagPairing: Pairing Smartphones in Close Proximity Using Magnetometers. IEEE Transactions on Information Forensics and Security, 11(6), 1306-1320. https://doi.org/10.1109/tifs.2015.2505626

[9] Das, A., Borisov, N., & Caesar, M. (2014). Do You Hear What I Hear? Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 1-2. https://doi.org/10.1145/2660267.2660325

[10] Dabov, K., Foi, A., Katkovnik, V., & Egiazarian, K. (2007). Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering. IEEE Transactions on Image Processing, 16(8), 2080-2095. https://doi.org/10.1109/tip.2007.901238

[11] Maggioni, M., Katkovnik, V., Egiazarian, K., & Foi, A. (2013). Nonlocal Transform-Domain Filter for Volumetric Data Denoising and Reconstruction. IEEE Transactions on Image Processing, 22(1), 119-133. https://doi.org/10.1109/tip.2012.2210725

[12] Abomhara, M., Zakaria, O., Khalifa, O. O., Zaidan, A. A., & Zaidan, B. B. (2010). Enhancing Selective Encryption for H.264/AVC Using Advanced Encryption Standard. International Journal of Computer and Electrical Engineering, 223-229. https://doi.org/10.7763/ijcee.2010.v2.141

[13] Dabov K, Foi A, Egiazarian K. (2007). Video denoising by sparse 3D transform-domain collaborative filtering. In EUSIPCO.

[14] Maggioni, M., Boracchi, G., Foi, A., & Egiazarian, K. (2012). Video Denoising, Deblocking, and Enhancement Through Separable 4-D Nonlocal Spatiotemporal Transforms. IEEE Transactions on Image Processing, 21(9), 3952-3966. https://doi.org/10.1109/tip.2012.2199324

[15] Celiktutan, O., Sankur, B., & Avcibas, I. (2008). Blind Identification of Source Cell-Phone Model. IEEE Transactions on Information Forensics and Security, 3(3), 553-566. https://doi.org/10.1109/tifs.2008.926993

[16] Kharrazi, M., Sencar, H. T., & Memon, N. (2004). Blind source camera identification. 2004 International Conference on Image Processing, ICIP '04. https://doi.org/10.1109/icip.2004.1418853

[17] Rouhi, R., Bertini, F., Montesi, D., Lin, X., Quan, Y., & Li, C.-T. (2019). Hybrid Clustering of Shared Images on Social Networks for Digital Forensics. IEEE Access, 7, 87288-87302. https://doi.org/10.1109/access.2019.2925102

[18] Freire-Obregón, D., Narducci, F., Barra, S., & Castrillón-Santana, M. (2019). Deep learning for source camera identification on mobile devices. Pattern Recognition Letters, 126, 86-91. https://doi.org/10.1016/j.patrec.2018.01.005

[19] Vijaya Kumar, B. V. K., & Hassebrook, L. (1990). Performance measures for correlation filters. Applied Optics, 29(20), 2997. https://doi.org/10.1364/ao.29.002997

[20] Rouhi, R. (2020) Classification and Clustering of Shared Images on Social Networks and User Profile Linking, [Dissertation thesis], Alma Mater Studiorum Universitá di Bologna. Dottorato di ricerca in Computer science and engineering, 32 Ciclo. DOI 10.6092/unibo/amsdottorato/9403.

[21] Valguarnera, E. (2019). Estrazione del Pattern Noise da video per un processo di identificazione di una fotocamera sorgente. [Laurea magistrale], Universitá di Bologna, Corso di Studio in Informatica [LM-DM270]

[22] Grundmann, M. Kwatra, V. Essa, I. (2018). Cascaded camera motion estimation, rolling shutter detection, and camera shake detection for video stabilization. uS Patent 9,888,180

[23] Taspinar, S., Mohanty, M., & Memon, N. (2016). Source camera attribution using stabilized video. 2016 IEEE International Workshop on Information Forensics and Security (WIFS). https://doi.org/10.1109/wifs.2016.7823918

[24] Mandelli, S., Bestagini, P., Verdoliva, L., & Tubaro, S. (2020). Facing Device Attribution Problem for Stabilized Video Sequences. IEEE Transactions on Information Forensics and Security, 15, 14-27. https://doi.org/10.1109/tifs.2019.2918644

[25] Lin, X., & Li, C.-T. (2018). Rotation-invariant Binary Representation of Sensor Pattern Noise for Source-Oriented Image and Video Clustering. 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), 1. https://doi.org/10.1109/avss.2018.8639161

[26] Timmerman, D., Bennabhaktula, S., Alegre, E., & Azzopardi, G. (2020). Video Camera Identification from Sensor Pattern Noise with a Constrained ConvNet. arXiv:2012.06277v1

[27] van Houten, W., & Geradts, Z. (2009). Source video camera identification for multiply compressed videos originating from YouTube. Digital Investigation, 6(1-2), 48-60. https://doi.org/10.1016/j.diin.2009.05.003.

[28] Shullani, D., Fontani, M., Iuliani, M., Shaya, O. A., & Piva, A. (2017). VISION: a video and image dataset for source identification. EURASIP Journal on Information Security, 2017(1). https://doi.org/10.1186/s13635-017-0067-2

[29] Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1. https://doi.org/10.1109/cvpr.2015.7298594

[30] Lin, X., & Li, C.-T. (2018). Rotation-invariant Binary Representation of Sensor Pattern Noise for Source-Oriented Image and Video Clustering. 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS). https://doi.org/10.1109/avss.2018.8639161

[31] https://github.com/polimi-ispl/TIFS2019-stabilized-video-attribution

[32] Timmerman, D., Bennabhaktula, S., Alegre, E., & Azzopardi, G. (2020). Video Camera Identification from Sensor Pattern Noise with a Constrained ConvNet.

Acknowledgements

I want to thank my supervisor, Professor Danilo Montesi, and my co-supervisors, Ph.D. Flavio Bertini and Ph.D. Rahimeh Rouhi, for the support they showed me during the thesis work and for making themselves available.
Without my parents' moral support, I could have never gotten this far. I must thank them for everything.
I thank my girlfriend Francesca for always making me smile, even in the darkest moments.
And finally, I thank my best friends Alessio, Andrea B. Andrea C., Andrea N. and Simone. Thanks for always being there.