



POLITECNICO
MILANO 1863

Dipartimento di Elettronica, Informazione e Bioingegneria

Master Degree in Music and Acoustic Engineering

A Multi-Modal Approach to Forensic Audio-Visual Device Identification

by:
Davide Dal Cortivo

matr.:
920199

Supervisor:
Prof. Paolo Bestagini

Co-supervisor:
Dr. Sara Mandelli

Academic Year
2019-2020

Abstract

The advent of the internet and social media has determined a rapid diffusion of digital multimedia content online. Images and videos are often shared to convey strong messages. If done maliciously, diffusion of biased or altered content may lead to severe social consequences. For this reason, it is important to develop forensic detectors capable of assessing the origin and the integrity of multimedia objects.

In this thesis, we focus on the problem of camera model identification for video sequences. This is, given a video under analysis, detect the camera model used for its acquisition. This problem has gained a significant importance in multimedia forensics as it allows to trace back a video to its creator, thus enabling to solve copyright infringement cases as well as to expose the authors of hideous crimes.

In order to solve the problem of determining the smartphone model used to acquire a video of unknown provenance, we develop two different detectors working in a multi-modal scenario. Both detectors are based on the use of convolutional neural networks (CNNs) and jointly exploit audio and visual information from the video under analysis in different ways. The first detector applies a voting procedure on top of two mono-modal CNNs that analyze the audio and visual streams separately. The second detector is composed by a single CNN that takes decision jointly analyzing audio and visual data.

The proposed solutions are tested on the well known Vision dataset, which contains a series of videos belonging to different devices. Experiments are performed considering original videos directly coming from the acquisition devices, videos uploaded on YouTube, videos shared through WhatsApp, and videos re-encoded using modern coding standards. Results show that the proposed multi-modal approaches outperform their mono-modal counterparts.

Sommario

L'avvento di internet e dei social media ha determinato una rapida diffusione di contenuti multimediali digitali online. Le immagini e i video sono spesso condivisi per trasmettere messaggi forti. Se fatto in modo doloso, la diffusione di contenuti di parte o alterati può portare a gravi conseguenze sociali. Per questo motivo è importante sviluppare dei rilevatori forensi in grado di valutare l'origine e l'integrità degli oggetti multimediali.

In questa tesi, ci concentriamo sul problema del camera model identification per sequenze video. Si tratta, dato un video in analisi, di rilevare il modello di fotocamera utilizzato per la sua acquisizione. Questo problema ha acquisito una notevole importanza nell'ambito forense multimediale in quanto permette di ricondurre un video al suo creatore, consentendo così di risolvere casi di violazione del copyright nonché esporre gli autori di crimini orribili.

Per risolvere il problema della determinazione del modello di smartphone utilizzato per acquisire un video di provenienza sconosciuta, sviluppiamo due diversi rilevatori che lavorano in uno scenario multimodale. Entrambi i rilevatori si basano sull'uso di reti neurali convoluzionali (CNN) e sfruttano congiuntamente le informazioni audio e visive del video analizzato in modi diversi. Il primo rilevatore applica una procedura di voto su due CNN monomodali che analizzano separatamente i flussi audio e video. Il secondo rilevatore è composto da un'unica CNN che prende la decisione analizzando congiuntamente i dati audio e video.

Le soluzioni proposte vengono testate sul noto dataset Vision, che contiene una serie di video appartenenti a diversi dispositivi. Gli esperimenti vengono eseguiti considerando video originali provenienti direttamente dai dispositivi di acquisizione, video caricati su YouTube, video condivisi tramite WhatsApp e video ricodificati utilizzando moderni standard di codifica. I risultati mostrano che gli approcci multimodali proposti superano le loro controparti monomodali.

Ringraziamenti

Sono rare le occasioni in cui si ha la possibilità di ringraziare pubblicamente tutte le persone che, in un modo o nell'altro, mi hanno aiutato ad arrivare alla fine di questo percorso universitario.

Vorrei innanzitutto ringraziare il prof. Paolo Bestagini e la dott.ssa Sara Mandelli, che con grande professionalità e disponibilità mi hanno guidato lungo tutta la stesura della tesi.

Ringrazio il prof. Massimiliano Musone, che con la sua bravura nell'insegnamento mi ha fatto appassionare alle materie scientifiche e ha contribuito alla scelta del mio corso di laurea.

Ringrazio i miei genitori, le mie sorelle e i miei parenti, che hanno sempre creduto in me e ci sono sempre stati nel momento del bisogno. Siete e sarete per sempre il mio porto sicuro.

Ringrazio la mia fidanzata Morgana, una persona splendida che mi è sempre stata accanto. Grazie per tutto il tempo che mi hai dedicato, per tutta la pazienza che hai avuto, per tutta l'energia positiva che mi hai trasmesso, per farmi sentire speciale.

Ringrazio i miei amici, senza i quali non sarei mai riuscito a vivere con serenità e spensieratezza tutti questi anni. Un grazie particolare a Gabriele, un fratello con cui ho condiviso tanto e che mi ha donato tanto; non gli sarò mai grato abbastanza.

Davide

Contents

Abstract	i
Sommario	ii
Ringraziamenti	iii
List of Figures	viii
List of Tables	x
Introduction	xi
1 Theoretical Background	1
1.1 Neural Networks	1
1.2 Convolutional Neural Networks	4
1.3 Digital Image Acquisition	6
1.4 Log-Mel Spectrogram	7
1.5 Conclusive Remarks	11
2 Problem Formulation	12
2.1 Mono-Modal Model Attribution	13
2.2 Multi-Modal Model Attribution	14
2.3 Conclusive Remarks	15
3 State of the Art	16
3.1 Model-based Approaches	16
3.1.1 CFA Configuration	16
3.1.2 CFA Interpolation	17
3.1.3 Lens Effects	17
3.1.4 Other Processing and Defects	17
3.2 Data-driven Approaches	17
3.2.1 Hand-crafted Features	18
3.2.2 Learned Features	18

3.3	Conclusive Remarks	18
4	Methodology	19
4.1	Mono-Modal Model Attribution	19
4.1.1	CNN for Camera Model Identification	20
4.1.2	CNN Architectures	22
4.1.3	Data Preprocessing	25
4.2	Multi-Modal Model Attribution	28
4.2.1	Late Fusion	29
4.2.2	Early Fusion	30
4.3	Conclusive Remarks	31
5	Simulations and Tests	33
5.1	Vision Dataset	33
5.2	Experimental Setup	35
5.3	Results	36
5.3.1	Visual Patches	36
5.3.2	Audio Patches	43
5.3.3	Late Fusion	50
5.3.4	Early Fusion	52
5.4	Conclusive Remarks	55
6	Conclusions and Future Works	57

List of Figures

1.1	A generic neural network [1].	2
1.2	The curve indicates the loss function with respect to a single weight. The ordinates of the dots indicate the values assumed by the loss function after each weight update; one is yellow because its ordinate also indicates the minimum point of the loss function. The distance between the abscissas of adjacent points is equal to the related learning step. The arrows between adjacent points indicate the weight update direction [2].	3
1.3	A general convolutional neural network [3].	5
1.4	Typical steps of a common image acquisition pipeline. . .	6
1.5	A sensor showing a layer of the photosites with a Bayer CFA [4].	6
1.6	Profile/cross-section of a sensor covered by a Bayer CFA [5].	7
1.7	Representation of the digital audio signal $s(n)$ in time domain.	8
1.8	Representation of the magnitude spectrum $ S_w(k) $	9
1.9	Representation of the log spectrogram $ S(m, k) ^2$	10
1.10	Representation of the log-mel spectrogram of $s(n)$	10
2.1	Granularity levels in forensic source identification.	13
4.1	Black box block diagram of the whole process.	22
4.2	Simplified representation of the EfficientNetB0.	22
4.3	A general residual block [6].	23
4.4	Representation of a residual block and an inverted residual block [7].	23
4.5	EfficientNetB0 with last layer modified.	24
4.6	Simplified representation of the VGGish.	25
4.7	VGGish with one extra fully connected layer.	25
4.8	Representation of the extracted frames.	26
4.9	Extraction of the visual patches.	26

4.10	Visual information preprocessing block diagram.	27
4.11	Extraction of the audio patches.	27
4.12	Audio information preprocessing block diagram.	28
4.13	EfficientNetB0 further modified for audio data.	28
4.14	Late Fusion block diagram.	29
4.15	EfficientNetB0 + VGGish multi-input neural network. . .	30
4.16	EfficientNetB0 + EfficientNetB0 multi-input neural network.	31
5.1	Complete list of devices in Vision dataset; DStab shows the presence or absence of digital stabilization on the acquired content, HDR indicates whether the device supports it, VR stands for video resolution and IR for image resolution [8].	34
5.2	Frames respectively extracted from a flat video, an indoor video and an outdoor video shot with the device “D06_Apple_iPhone6” [8].	35
5.3	Training Set: Original; Testing Set: Original; Accuracy: 0.8202.	37
5.4	Training Set: WhatsApp; Testing Set: WhatsApp; Accuracy: 0.6739.	37
5.5	Training Set: YouTube; Testing Set: YouTube; Accuracy: 0.7404.	38
5.6	Training Set: Original; Testing Set: WhatsApp; Accuracy: 0.3579.	39
5.7	Training Set: Original; Testing Set: YouTube; Accuracy: 0.4869.	39
5.8	Classification accuracies as a function of training/testing sets and H.264 compression quality factors.	41
5.9	Classification accuracies as a function of training/testing sets and H.265 compression quality factors.	41
5.10	Classification accuracies as a function of training/testing sets and H.264 and H.265 compression quality factors. . .	42
5.11	VGGish - Training Set: Original; Testing Set: Original; Accuracy: 0.6578.	43
5.12	VGGish - Training Set: WhatsApp; Testing Set: WhatsApp; Accuracy: 6757.	44
5.13	VGGish - Training Set: YouTube; Testing Set: YouTube; Accuracy: 0.7010.	44
5.14	VGGish - Training Set: Original; Testing Set: WhatsApp; Accuracy: 0.5304.	45

5.15 VGGish - Training Set: Original; Testing Set: YouTube; Accuracy: 0.6654.	46
5.16 Classification accuracies as a function of training/testing sets and MP3 compression quality factors.	47
5.17 Classification accuracies as a function of training/testing sets and MP3 compression quality factors - EfficientNetB0 ₆₄	48
5.18 Classification accuracies as a function of training/test- ing sets and MP3 compression quality factors - EfficientNetB0 ₁₉₂	49

List of Tables

4.1	A general application of the Late Fusion with 3 patch pairs and 4 camera models. We indicate in bold the numbers analyzed in the text.	30
5.1	Table that collects the accuracy achieved in the various tests.	40
5.2	Table that collects the classification accuracies of mono-modal methods as a function of training/testing sets; for audio patches we have a VGGish. We indicate in bold the numbers analyzed in the text.	46
5.3	Table that collects the classification accuracies of mono-modal methods as a function of training/testing sets; VGGish, EfficientNetB0 ₆₄ , EfficientNetB0 ₁₉₂ are for audio patches and EfficientNetB0 for visual patches. We indicate in bold the numbers analyzed in the text.	50
5.4	Table that collects the classification accuracies of Late Fusion 1 and mono-modal methods as a function of training/testing sets; for Late Fusion 1 we have an EfficientNetB0 + VGGish, for audio patches we have a VGGish. We indicate in bold the numbers analyzed in the text.	51
5.5	Table that collects the classification accuracies of Late Fusion 2 and mono-modal methods as a function of training/testing sets; for Late Fusion 2 we have an EfficientNetB0 + EfficientNetB0 ₆₄ , for audio patches we have an EfficientNetB0 ₆₄ . We indicate in bold the numbers analyzed in the text.	51

5.6	Table that collects the classification accuracies of Late Fusion 3 and mono-modal methods as a function of training/testing sets; for Late Fusion 3 we have an EfficientNetB0 + EfficientNetB0 ₁₉₂ , for audio patches we have a EfficientNetB0 ₁₉₂ . We indicate in bold the numbers analyzed in the text.	52
5.7	Table that collects the classification accuracies of Early Fusion 1 and mono-modal methods as a function of training/testing sets; for Early Fusion 1 we have an EfficientNetB0 + VGGish, for audio patches we have a VGGish. We indicate in bold the numbers analyzed in the text. . .	53
5.8	Table that collects the classification accuracies of Early Fusion 2 and mono-modal methods as a function of training/testing sets; for Early Fusion 2 we have an EfficientNetB0 + EfficientNetB0 ₆₄ , for audio patches we have an EfficientNetB0 ₆₄ . We indicate in bold the numbers analyzed in the text.	53
5.9	Table that collects the classification accuracies of Early Fusion 3 and mono-modal methods as a function of training/testing sets; for Early Fusion 3 we have an EfficientNetB0 + EfficientNetB0 ₁₉₂ , for audio patches we have a EfficientNetB0 ₁₉₂	54
5.10	Table that collects the classification accuracies of Late Fusion 1, Late Fusion 2 and Late Fusion 3 as a function of training/testing sets; for Late Fusion 1 we have an EfficientNetB0 + VGGish, for Late Fusion 2 we have an EfficientNetB0 + EfficientNetB0 ₆₄ , for Late Fusion 3 we have an EfficientNetB0 + EfficientNetB0 ₁₉₂ . We indicate in bold the numbers analyzed in the text.	54
5.11	Table that collects the classification accuracies of Early Fusion 1, Early Fusion 2 and Early Fusion 3 as a function of training/testing sets; for Early Fusion 1 we have an EfficientNetB0 + VGGish, for Early Fusion 2 we have an EfficientNetB0 + EfficientNetB0 ₆₄ , for Early Fusion 3 we have an EfficientNetB0 + EfficientNetB0 ₁₉₂ . We indicate in bold the numbers analyzed in the text.	55

Introduction

Camera model identification has gained significant importance in multimedia forensic investigations, as digital multimedia contents (i.e., digital images, videos and audio sequences) are increasingly widespread and will continue to spread in the future with the advance of technological progress. This phenomenon is mainly attributable to the advent of the internet and social media, which allowed a very rapid diffusion of digital contents and, consequently, made it extremely difficult to trace their origin.

In forensic analysis, tracing the origin of digital contents can be essential, for example, to identify the perpetrators of crimes such as rape, drug trafficking or acts of terrorism. There is also the possibility that certain private contents become viral through the internet, as sadly happened in recent times with revenge porn; tracing the origin therefore assumes a fundamental role.

The aim of this thesis is to determine the smartphone model used to acquire digital video sequences by exploiting visual and audio information of the videos themselves. We mainly focus on this, because little work has been done specifically for digital video sequences in the state-of-the-art forensic source identification literature [9]. The analysis of digital images, in the other hand, is widely addressed [10, 11, 12, 13, 14, 15, 16].

The two main families of approaches related to image camera model identification are model-based and data-driven. Model-based approaches specifically rely on exploiting the traces released by the digital image acquisition process in order to identify the camera model. To this family belong methods that exploit the peculiar traces released by color filter array (CFA) configuration, by demosaicing process or by unwanted effects generated by the lens. On the other hand, data-driven approaches consist in extracting from the images under investigation certain statistical features that capture the difference caused by camera structure and various in-camera image processing algorithms, followed by machine

learning and pattern recognition algorithms for similarity measures of extracted features. Methods based on hand-crafted features (i.e., manually engineered by data scientists) or learned features (i.e., automatically obtained from a machine learning algorithm) belong to this family. As we can see, the camera model identification methods related to digital images are numerous; however many of these are considered obsolete.

We rely on the most advanced data-driven approaches that use learned features, in order to develop effective methods for video sequences. These methods involve the use of convolutional neural networks (CNNs) capable of classifying videos by extracting suitable features from their visual and audio content. Given a video, as visual content to provide to the CNNs we use patches cropped from its video frames, while as audio content we use patches cropped from the log-mel spectrogram of its audio track.

We propose two multi-modal camera model identification approaches: in the first one we compare the scores individually obtained from two trained mono-modal CNNs (one only with visual patches, the other only with audio patches), in the second one we build multi-input networks and train them with visual/audio patch pairs. For completeness, before proceeding with the resolution of the multi-modal model attribution, we investigate the mono-modal model attribution using CNNs trained only with video patches or audio patches. Investigating both problems is also useful for us to understand which approach leads to better results.

The dataset from which we take videos is the Vision dataset, a recent image and video dataset, purposely designed for multimedia forensics investigations. The videos on which we experience are not only the original ones; we also use those compressed by the WhatsApp and YouTube algorithms and those further compressed in other ways (e.g., H.264 or H.265 codecs) so as to obtain many results with different configurations.

In general, multi-modal methods are more effective than mono-modal methods and, depending on our needs, the first multi-modal method can be better or worse than the second one. We achieve very good results in non-cross tests (i.e., when training and testing sets are selected from the same dataset), reaching accuracy values up to 99% in the first multi-modal method. In cross tests (i.e., when training and testing sets belong to different datasets) the achieved accuracy depends on the kind of datasets involved (e.g., WhatsApp data are more challenging to model than YouTube's) and the applied compression factors (e.g., stronger compressions usually correspond to worse results). Both multi-modal methods achieve similar results; depending on the specific

cross-test, one method may be preferred over the other. Possible future works could concern the improvement of cross test results thanks to the realization of systems that are more stable and less sensitive to data compression.

This thesis is organized as follows. In Chapter 1, we introduce some general concepts in order to better understand the tackled problem and the proposed methodology. In Chapter 2, we report the formulation of the problems of mono-modal and multi-modal model attribution, and in Chapter 3 we present different camera model identification methods documented in literature. In Chapter 4, we report a detailed explanation of the resolution methods used; in Chapter 5, we analyze the results obtained from the application of the methods. Eventually, Chapter 6 draws some conclusions.

1

Theoretical Background

This chapter introduces some general background concepts in order to better understand what will be explained in the following pages. In particular, we focus on the concepts of neural networks and log-mel spectrogram and take a first look at how a digital image is captured.

1.1 Neural Networks

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning algorithms and are at the heart of deep learning [17, 18]. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

ANNs are composed by layers containing multiple nodes (or neurons). They are typically composed by an input layer, one or more hidden layers, and an output layer (as shown in Figure 1.1). Each node, or artificial neuron, connects to another one and has an associated weight and activation (e.g., a threshold or some other non-linear function). If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

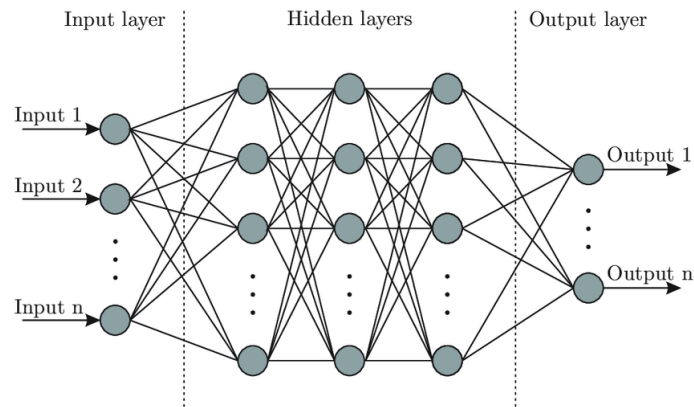


Figure 1.1: A generic neural network [1].

Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned to minimize some cost function and maximize some metric (e.g., accuracy), they are powerful tools in computer science and artificial intelligence, allowing us to classify data at a high velocity.

Let us see how a neural network works. Think of each individual j -node as its own linear regression model, composed of input data s_i , weights w_{ij} , a bias b_j (or threshold), and an output s_j . The formulas would look something like this:

$$z_j = \sum_{i=0}^{n-1} w_{ij} s_i, \quad (1.1)$$

$$g_j(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0, \\ -1 & \text{if } x < 0 \end{cases}, \quad (1.2)$$

$$s_j = g_j(z_j - b_j). \quad (1.3)$$

Initially, weights and bias are assigned randomly or with a specific criterion. These weights help determine the importance of any given variable, with larger ones contributing more significantly to the output compared to other inputs. All inputs are then multiplied by their respective weights and then summed. Afterward, this sum z_j and bias b_j are passed through an activation function $g_j(x)$, which determines the output s_j . The activation function can be different from network to network (in this example we have reported a step function) and determines the passage of data from one node to the next layer of the network. This results in the output of one node becoming in the input of the next node.

This process of passing data from one layer to the next layer defines this neural network as a feedforward network.

Naturally, the network will not work well with the initial weights and biases; we have to update them. We will focus on updating related to supervised learning (i.e., the desired outcome is known for each input data). Suppose we have a network with one input and one output:

- y_i is the outcome associated to a particular input data x_i ; we want that y_i is predicted from the network;
- \hat{y}_i is the predicted outcome from x_i ;
- N is the number of samples.

We can evaluate the accuracy of the network with a loss (or cost) function. A very common loss function is the mean square error (MSE) defined in this way:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2. \quad (1.4)$$

The goal is to minimize the value of the loss function to ensure correctness of fit for any given observation. This value depends on weights and biases that the network presents. With each training example, the parameters of the model adjust allowing the value of the loss function to gradually converge at the minimum, as shown in Figure 1.2.

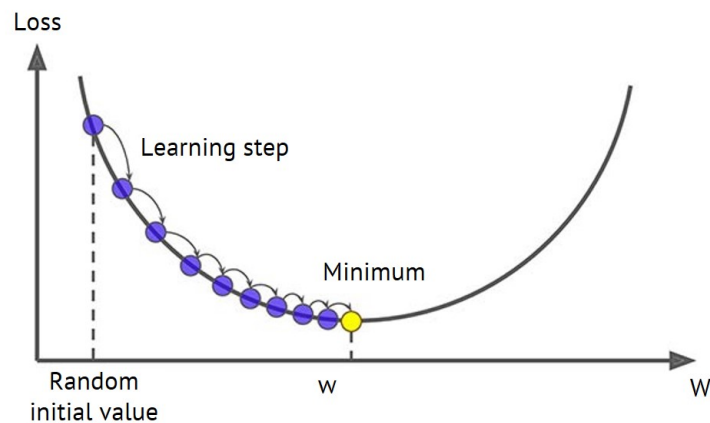


Figure 1.2: The curve indicates the loss function with respect to a single weight. The ordinates of the dots indicate the values assumed by the loss function after each weight update; one is yellow because its ordinate also indicates the minimum point of the loss function. The distance between the abscissas of adjacent points is equal to the related learning step. The arrows between adjacent points indicate the weight update direction [2].

The process is called backpropagation and consists in updating the parameters of the network using this formula:

$$\mathbf{w}^{n+1} = \mathbf{w}^n + \Delta\mathbf{w} = \mathbf{w}^n - \eta \cdot \left. \frac{\partial E}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^n}, \quad (1.5)$$

where \mathbf{w}^n indicates the weights vector (including biases) at the time step n , $\Delta\mathbf{w}$ the learning step, η the learning rate and E the loss function. To better understand this formula, let us consider the case in which we have to update only one parameter like in Figure 1.2.

The parameter is updated by adding a certain quantity, the learning step, which can be positive or negative. As we can see in Figure 1.2, by gradually enlarging the parameter we will decrease the value of the loss function; if instead we do the reverse operation, the value of the loss function will increase. The increase and decrease of the parameter is dictated by the sign of the learning step. The way to choose the right sign is to exploit the sign of the derivative of the loss function with respect to the parameter calculated at the parameter point: if it is positive, it means that the loss function is increasing; we will therefore decrease the parameter to end up with a lower loss function value. The operation is opposite for the negative sign. The idea to update the network weights is to proceed in small increments to avoid exceeding the minimum point and witness an increase in the loss function value. The learning step will therefore be composed of the derivative with opposite sign (to define the increase or decrease of the parameter) multiplied by a very small number, the learning rate (to proceed gradually in decreasing the value of the loss function).

In the equation (1.5) we report the general case in which the network has several parameters. The reasoning remains unchanged, clearly we will apply the partial derivatives of the loss function with respect to the different parameters. This process is called backpropagation because the updating of weights and biases starts from those closest to the output and then proceeds towards the initial ones.

1.2 Convolutional Neural Networks

Suppose we have to build a neural network able to classifying images. A first approach could be to use an input layer with a number of nodes equal to the number of pixels in the input images (every node is associated to a particular pixel), some hidden layer and an output layer. It is a simple approach, but not optimal, due to three main reasons:

1. in image classification, we want to recognize patterns corresponding to objects inside the image. We are more interested in how nearby pixels are arranged, and less interested in how pixels, that are far from each other, appear in combination. In the above approach, we consider always the entire image: missing locality property;
2. the network must recognize objects regardless of their position in the image: translation invariant property. In the above approach, also this property is missing;
3. the above approach involves a network with a very very large amount of parameters, therefore very difficult to train.

A convolutional neural network (CNN) [19, 20] is a particular type of neural network that presents locality and translation invariant properties and it is realized with fewer parameters: it is perfect for image classification.

A CNN is mainly made up of layers called convolutional layers; they perform an operation on images called convolution. Convolution is defined for a 2D image as the scalar product of a weight matrix, the kernel, with every neighborhood in the input:

$$g(x, y) = \omega * f(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b \omega(s, t) f(x - s, y - t), \quad (1.6)$$

where $g(x, y)$ is the output image obtained from convolution, $f(x, y)$ is the original image (the value of $f(x, y)$ is 0 outside the image), ω is the kernel.

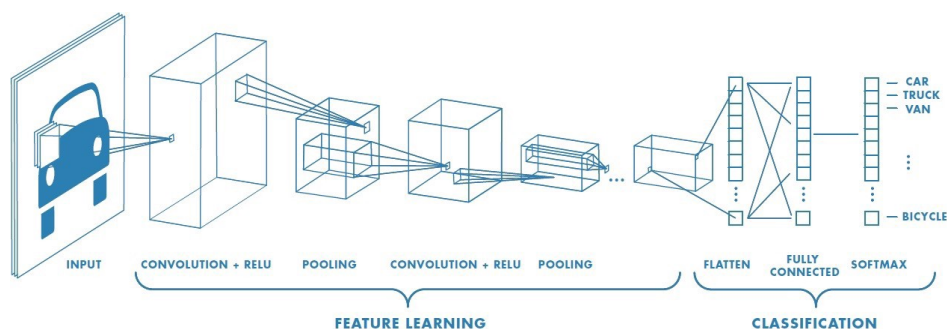


Figure 1.3: A general convolutional neural network [3].

Figure 1.3 shows a CNN that includes convolutional layers (each with more than one kernel) + ReLU (the activation function), pooling layers used to reduce the dimension of images and a final classification section with some flatten layers like those in Figure 1.1.

Again, we have to compute the loss function and adjust weights and bias of each layer, in order to train the network. In convolutional layers, weights are the elements inside kernels.

1.3 Digital Image Acquisition

Whenever we take a photograph with a digital camera or smartphone, we trigger an elaborate process consisting of several operations called digital image acquisition. This process, which lasts a few fractions of a second, starts when we press the shutter button and ends when we can visualize the shot we took.

The digital image acquisition pipeline varies according to the device we are considering. However, it is reasonable to assume that a typical digital image acquisition pipeline is composed by a series of common steps [21], as shown in Figure 1.4.

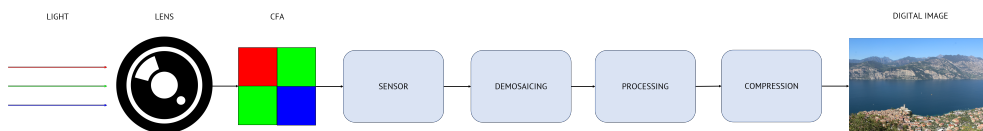


Figure 1.4: Typical steps of a common image acquisition pipeline.

Ray lights hit a lens that focus them on the sensor [4]. The surface of a sensor is covered by a grid of microscopic pits called photosites which represent the pixels of a digital image. Each photosite has an electronic device called photodiode inside which transforms light intensity into an electric charge.

Most sensors use color filters, because the photosites themselves capture only the light intensity (not the color). The most common color filter array is the Bayer color filter array (Bayer CFA).

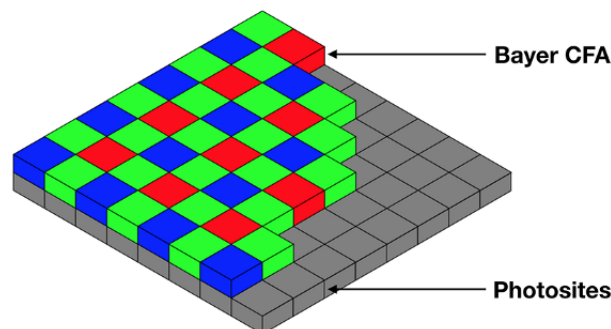


Figure 1.5: A sensor showing a layer of the photosites with a Bayer CFA [4].

As we can see in Figure 1.5, the Bayer CFA covers each photosite with a colored filter (red, green or blue), specializing it in capturing that particular color. It typically consists of 50% green, 25% red and 25% blue (as shown in Figure 1.6). This type of pattern is also called RGGB (sometimes also arranged as BGGR, RGBG or GRGB). The green color represents the luminance-sensitive elements of an image and the red and blue represent the chrominance-sensitive elements. The Bayer CFA was developed to mimic the way the human eye interprets color. This is because according to study the luminance perception of the human eye is most sensitive to the color green. The colors of an image on digital cameras are based on this concept.

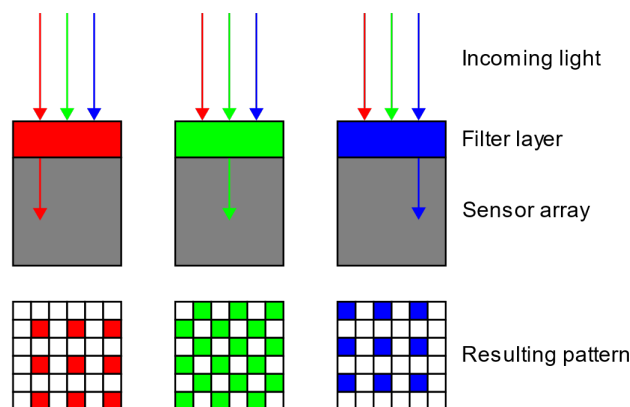


Figure 1.6: Profile/cross-section of a sensor covered by a Bayer CFA [5].

In order to detect the color of a pixel associated to a specific photosite p , an interpolation is made between the color captured by p and the colors captured by the photosites placed around p . This procedure is called demosaicing and allows to obtain a raw version of color images. After that, we have a processing phase consisting of additional operations. We generally find white balance, color correction and lens distortion correction among the most common processing operations. Finally, lossy image compression is typically applied through the JPEG standard.

1.4 Log-Mel Spectrogram

A signal is a variation in a certain quantity over time. For audio, the quantity that varies is air pressure. To digitally capture this information, we can sample and quantize the air pressure over time. The rate at which we sample the data can vary, but is most commonly 44.1kHz (e.g., Audio CDs use this sample rate).

Suppose we have a digital audio signal $s(n)$ where n indicates the time; $s(n)$ describes the signal in terms of air pressure, or more generally, in terms of amplitude in time domain (as shown in Figure 1.7).

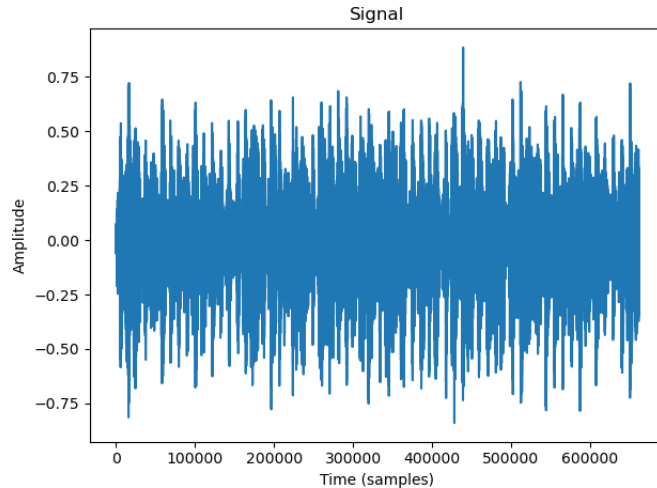


Figure 1.7: Representation of the digital audio signal $s(n)$ in time domain.

An audio signal is comprised of several single-frequency sound waves. When taking samples of the signal over time, we only capture the resulting amplitude. The Fourier transform (FT) is a mathematical formula that allows to decompose a signal into its individual frequency components; in other words, it converts the signal from the time domain into the frequency domain.

$$S_w(k) = \sum_{n=0}^{N-1} s_w(n) \cdot e^{-\frac{j2\pi}{N}kn}. \quad (1.7)$$

The equation (1.7) defines the discrete Fourier transform (DFT) applied to a windowed segment of the signal $s(n)$ indicated as $s_w(n)$. Here, j is the imaginary unit and N is the length of the window. We use DFT instead of classic FT because we work in discrete time and frequency domains. The result $S_w(k)$ is called spectrum and it is a function of the frequency k . If we compute the absolute value of $S_w(k)$, we obtain the magnitude spectrum $|S_w(k)|$ which is related to the amplitude of various frequency components of $s_w(n)$ (as shown in Figure 1.8).

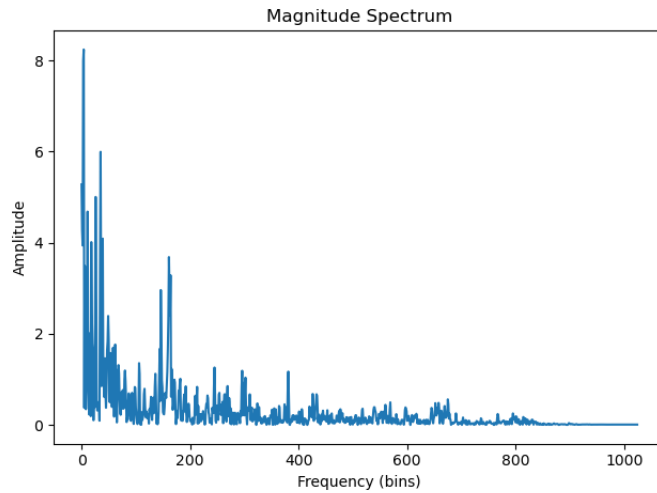


Figure 1.8: Representation of the magnitude spectrum $|S_w(k)|$.

If the frequency content of our signal varies over time, we have to compute several spectrums by performing the FT on several windowed segments of the signal. This process is called short-time Fourier transform (STFT). We define the discrete STFT for the same reasons of the DFT:

$$S(m, k) = \sum_{n=-\infty}^{\infty} s(n) \cdot w(n - m) \cdot e^{-\frac{j2\pi}{N}kn}. \quad (1.8)$$

In the equation (1.8), m indicates a time shift and $w(n - m)$ indicates a particular window that multiplies the signal $s(n)$. The spectrogram of $s(n)$ is defined as $|S(m, k)|^2$ and can be used to represent the spectral content of an audio signal over time in a Cartesian coordinate system: on the x -axis we have time, on the y -axis we have frequency and on the z -axis we have amplitude. Generally, the y -axis is converted to a log scale and the z -axis is converted to decibels. This is because humans do not perceive frequencies and amplitudes on a linear scale, but on a log scale (we are better at detecting differences in lower frequencies and amplitudes than higher frequencies and amplitudes). From this rescaling of the axis, we obtain the log spectrogram illustrated in Figure 1.9.

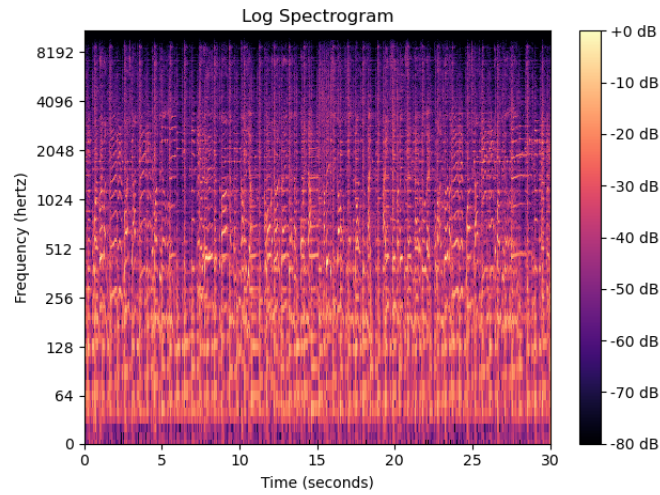


Figure 1.9: Representation of the log spectrogram $|S(m, k)|^2$.

In 1937, Stevens, Volkman, and Newmann proposed a perceptual scale of pitches judged by listeners to be equal in distance from one another: the mel scale [22]. The relation between the pitch and the frequency on which this scale is based is as follows:

$$\text{Mel}(k) = 2595 \cdot \log \left(1 + \frac{k}{700} \right), \quad (1.9)$$

where $\text{Mel}(k)$ indicates the perceived pitch of a sound at frequency k . Based on this relation, we can rescale the y -axis of the log spectrogram, in order to obtain a distribution of frequencies that best approximates the human perception of them. From this rescaling of the y -axis, we obtain the so-called log-mel spectrogram [23], illustrated in Figure 1.10.

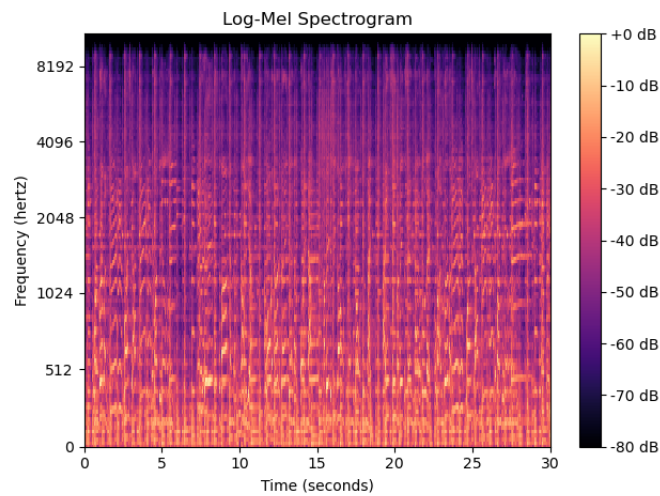


Figure 1.10: Representation of the log-mel spectrogram of $s(n)$.

1.5 Conclusive Remarks

In this chapter we have introduced the basic concepts useful for understanding the next part of the thesis. In particular we have shown:

- what a neural network is and how it works,
- what a CNN is,
- how digital image acquisition works,
- what a log-mel spectrogram is and how we can extract it from an audio signal.

Now that these concepts have been defined, we can formally introduce the problem we aim to solve in our work and which are some possible solutions in the next chapters.

2

Problem Formulation

The problem we address in this thesis is that of audio-visual camera model identification. Camera model identification subsumes the broad class of forensic source identification techniques that aim to determine the camera (e.g., digital camera, smartphone camera, etc.) model used to acquire some multimedia content of unknown provenance [16]. We mainly focus on identifying the source camera model of digital video sequences (considering both audio and visual cues), as the analysis of digital images is widely addressed in the forensic source identification literature, with excellent results [24, 25]. Camera model identification seeks to answer one or both of these questions:

- “What is the camera model that (most likely) acquired this data content?”;
- “Was this content captured with a camera of this particular make and model?”.

The premise of camera model identification is that contents acquired with the same camera model share common characteristics, which can be exploited for inference about the source.

Depending on the specific case and available reference information, forensic source identification may be approached at different levels of identification granularity.

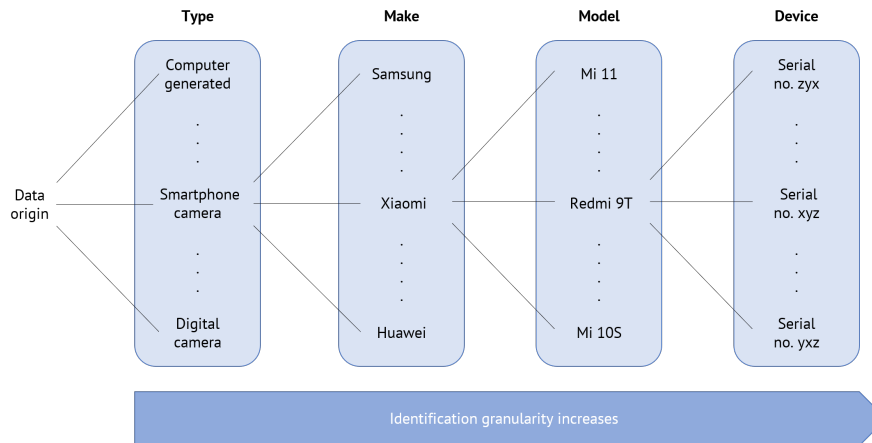


Figure 2.1: Granularity levels in forensic source identification.

In Figure 2.1 we can distinguish between methods to determine the type of acquisition device, its make or model, and ultimately also the actual device itself. The vertically stacked instances of (groups of) acquisition devices and/or acquisition methods allude to possible class sets of each of the horizontally arranged granularity levels. Granularity increases from left to right: at the lowest level, we might be interested in separating between computer generated and camera shot material, whereas device-level identification aims to differentiate individual digital cameras (or other devices).

We stay on the granularity model-level, working with video sequences recorded from different smartphone models and proposing an innovative approach that combines visual and audio information of the considered videos.

2.1 Mono-Modal Model Attribution

The problem of mono-modal model attribution consists in detecting which is the device model used to acquire a specific kind of media at a single modality. For instance, given a photograph, to understand which is the model of the camera used to take it. Alternatively, given an audio recording, to detect the used recorder model. Given a video, which is the case of our interest, the mono-modal model attribution consists in identifying the device model that shot it, using the visual or audio information of the video itself. Formally, we can define this problem as follows.

Let \mathcal{M} be the set of device models and \mathcal{D} the set of videos. Each element $\mathbf{d} \in \mathcal{D}$ is a tuple $(v^{(m)}, a^{(m)})$, where $v^{(m)}$ and $a^{(m)}$ respectively indicate the visual and audio content of a particular video shot with the model $m \in \mathcal{M}$. We define:

- the set of visual contents $\mathcal{V} = \{\pi_1(\mathbf{d}) : \mathbf{d} = (v^{(m)}, a^{(m)}) \in \mathcal{D}\}$;
- the set of audio contents $\mathcal{A} = \{\pi_2(\mathbf{d}) : \mathbf{d} = (v^{(m)}, a^{(m)}) \in \mathcal{D}\}$;

where $\pi_1(\mathbf{d})$ and $\pi_2(\mathbf{d})$ respectively indicate the first and second elements of the tuple \mathbf{d} .

According to the chosen modality (visual information or audio information), the mono-modal model attribution problem on video sequences consists in one of these two tasks:

- compute the function $f : \mathcal{V} \rightarrow \mathcal{M}$ such that, $\forall v^{(m)} \in \mathcal{V}$, $f(v^{(m)}) = m$ with $m \in \mathcal{M}$;
- compute the function $g : \mathcal{A} \rightarrow \mathcal{M}$ such that, $\forall a^{(m)} \in \mathcal{A}$, $g(a^{(m)}) = m$ with $m \in \mathcal{M}$.

However, solving the problem defined in this way is very difficult in practice. We can redefine it to be more accessible. Again, according to the chosen modality, the redefined problem consists in one of these tasks:

- compute the function $f : \mathcal{V} \rightarrow \mathcal{M}$ such that, taken any $v^{(m)} \in \mathcal{V}$ and $m \in \mathcal{M}$, the probability $P(f(v^{(m)}) \neq m)$ is as small as possible;
- compute the function $g : \mathcal{A} \rightarrow \mathcal{M}$ such that, taken any $a^{(m)} \in \mathcal{A}$ and $m \in \mathcal{M}$, the probability $P(g(a^{(m)}) \neq m)$ is as small as possible.

2.2 Multi-Modal Model Attribution

Given a video sequence, the problem of multi-modal model attribution converts in identifying the device model that shot it, using both visual and audio information of the video itself. Formally, we can define this problem as follows.

Let \mathcal{M} , \mathcal{D} , \mathcal{V} , \mathcal{A} be defined as in mono-modal model attribution case. The problem consists in one of these tasks:

- compute the function $h_1 : \{\mathcal{V} \times \mathcal{A}\} \rightarrow \mathcal{M}$ such that, $\forall (v^{(m)}, a^{(m)}) \in \mathcal{D}$, $h_1(v^{(m)}, a^{(m)}) = m$ with $m \in \mathcal{M}$;
- compute the function $h_2 : \{\mathcal{A} \times \mathcal{V}\} \rightarrow \mathcal{M}$ such that, $\forall (v^{(m)}, a^{(m)}) \in \mathcal{D}$, $h_2(a^{(m)}, v^{(m)}) = m$ with $m \in \mathcal{M}$.

Again, solving the defined problem may be very difficult in practice. The redefined and more accessible problem consists in one of these tasks:

- compute the function $h_1 : \{\mathcal{V} \times \mathcal{A}\} \rightarrow \mathcal{M}$ such that, taken any $(v^{(m)}, a^{(m)}) \in \mathcal{D}$ and $m \in \mathcal{M}$, the probability $P(h_1(v^{(m)}, a^{(m)}) \neq m)$ is small as possible;
- compute the function $h_2 : \{\mathcal{A} \times \mathcal{V}\} \rightarrow \mathcal{M}$ such that, taken any $(v^{(m)}, a^{(m)}) \in \mathcal{D}$ and $m \in \mathcal{M}$, the probability $P(h_2(a^{(m)}, v^{(m)}) \neq m)$ is small as possible.

2.3 Conclusive Remarks

In this chapter we have formulated the problem to be addressed. We have seen how the camera model identification can be approached at different granularity levels (e.g., model, device, etc.), sources (e.g., video, audio, etc.) and modalities (i.e., mono-modal and multi-modal). In the next chapter we will understand how the state of the art can help us in the realization of the methods used to address the main problem.

3

State of the Art

In this chapter we present different camera model identification methods widely documented in state-of-the-art literature. We focus on model-based and data-driven approaches, reporting some methods for each.

3.1 Model-based Approaches

As we have seen in Section 1.3, digital image acquisition is a process composed of several operations that can characterize a specific camera brand and model. This process varies from device to device, therefore the operations that compose it can leave traces within the images produced that are peculiar to the device in question. In this section, we provide an overview of the so-called model-based methods, which are camera model identification methods based specifically on exploiting those traces. They assume that each artifact of the acquisition chain can be modeled, in order to detect the camera model. These methods are historically the first ones being documented in the literature [26].

3.1.1 CFA Configuration

We can exploit information from characteristics of the color filter array (CFA) to infer some model-specific features. As CFA patterns are different among different camera models, the artifacts introduced by CFA

configuration can reflect model-specific to some extent [10, 27].

3.1.2 CFA Interpolation

Digital cameras acquire an image with one sensor overlaid with a CFA, capturing at each spatial location one sample from the three necessary color channels. The missing pixels must be interpolated in a process known as demosaicing. This process is highly nonlinear and can vary greatly between different camera brands and models. A camera model classification method based on this process use correlations between color channels to construct a characteristic map that is useful in matching an image to its source [11, 28].

3.1.3 Lens Effects

The type of lens mounted on a camera model generates unwanted effects in the images acquired. In particular, we can distinguish three main effects:

- distortion: image is stretched in some way;
- chromatic aberration: color fringe and uneven colors around edge details in high contrast scenes;
- vignetting: darkening of corners of an image.

A camera model classification method based on lens effects tries to exploit the differences between one or more effects generated by distinct camera models [12, 29].

3.1.4 Other Processing and Defects

There are also other camera model classification methods less reported in literature that exploit other artifacts or characteristics, such as the dust particles in front of the camera sensor that create a persistent pattern in the captured images [30] or the Auto-White Balance algorithm used inside the camera [31].

3.2 Data-driven Approaches

These approaches consist in extracting from the images under investigation certain statistical features that capture the difference caused by

camera structure and various in-camera image processing algorithms, followed by machine learning and pattern recognition algorithms for similarity measures of extracted features. Features can be hand-crafted (i.e., manually engineered by data scientists) or learned (i.e., automatically obtained from a machine learning algorithm). In this section, we provide an overview of camera model identification methods that work in this way.

3.2.1 Hand-crafted Features

There are methods that use uniform gray-scale invariant local binary patterns [13] or are based on generalized noise model developed by following a image processing pipeline of a digital camera [32, 33]; several hand-crafted features can be extracted.

3.2.2 Learned Features

These are the most recent methods, which are best suited to the latest generation cameras. Most of them use convolutional neural network (CNN) that are able to extract features suitable for camera model classification [14, 15], learn similarity features [34], deal with open sets of camera models [35] or be part of generative adversarial networks (GANs) [36]. There are also methods for video camera model identification [9], but they have not been covered much in the literature.

3.3 Conclusive Remarks

As we can see, there are several camera model classification methods, but many of them are obsolete nowadays; both because more advanced methods based on CNNs have outperformed older detectors and because the latest generation cameras have different characteristics with respect to those on which these methods were based. Our analysis is developed based on the most recent methods: data-driven approaches using learned features.

4

Methodology

We have seen what mono-modal and multi-modal model attribution mean, especially if we use video sequences. Now let us see how to implement useful algorithms to solve these problems, starting from a data preprocessing and arriving at a model prediction.

4.1 Mono-Modal Model Attribution

As mentioned in Section 2.1, given a video, the mono-modal model attribution consists in identifying the device model that shot it, using the visual or audio information of the video itself. Data from the same model must exhibit similar characteristics in order to classify them; more commonly called features.

Let us take into consideration the visual information modality. For each $v^{(m)} \in \mathcal{V}$ we can extract a feature vector \mathbf{f} belonging to the set of feature vectors \mathcal{F} and then, map each \mathbf{f} to a model $m \in \mathcal{M}$. Therefore, we can visualize the function f that we want to compute as a composite function $f_B \circ f_A$, where:

- $f_A : \mathcal{V} \rightarrow \mathcal{F}$ extracts feature vectors from visual contents;
- $f_B : \mathcal{F} \rightarrow \mathcal{M}$ maps feature vectors to models.

Computing the functions f_A and f_B would mean solving the problem, but unfortunately it is not so immediate; also because we do not know apriori which are the useful features to extract in order to identify a model. The solution we have adopted is to use a convolutional neural network (CNN).

Both functions are learned in the CNN training phase from tuples of training data $(v_{\text{train}}^{(m)}, m) \in \{\mathcal{V}_{\text{train}} \times \mathcal{M}_{\text{train}}\} \subseteq \{\mathcal{V} \times \mathcal{M}\}$, where the notations $\mathcal{V}_{\text{train}}$ indicates the subsets of \mathcal{V} used for training. In our case, $\mathcal{M}_{\text{train}} = \mathcal{M}$, as we exploit all the available camera models during the training process. Notice that too homogeneous training data is likely to result in a network that does not generalize well. Therefore, before practical use, any camera model identification algorithm should additionally be tested against validation data $(v_{\text{val}}^{(m)}, m) \in \{\mathcal{V}_{\text{val}} \times \mathcal{M}_{\text{val}}\} \subseteq \{\mathcal{V} \times \mathcal{M}\}$ and refined based thereon, to rule out a mismatch with new input data. In this case as well, $\mathcal{M}_{\text{val}} = \mathcal{M}$.

After training, the network can be used for practical camera model identification in the test phase by predicting the source camera models (more generally called labels) from never seen data $v_{\text{test}}^{(m)} \in \mathcal{V}_{\text{test}} \subseteq \mathcal{V}$, with $m \in \mathcal{M}$.

The algorithm is analogous as regards the audio information modality; clearly the extracted features will differ as the type of information is different.

4.1.1 CNN for Camera Model Identification

Now let us see how a CNN built for classification, given a generic input data x , selects the predicted camera model $\hat{m} \in \mathcal{M}$.

The final layer of the network is a fully connected layer with a number of nodes equal to the total number of models; each node is associated with a particular model and an output value calculated each time a new input data x is provided. These values are the result of the various transformations applied to x along the network.

We call \mathbf{y} the vector that contains the output values of all the final layer nodes associated with the input data x . In the classification process, \mathbf{y} is passed to the softmax function, which maps its values into a range from 0 to 1 so that their sum gives 1. Softmax is defined as:

$$\sigma(\mathbf{y})_i = \frac{e^{y_i}}{\sum_{j=0}^{M-1} e^{y_j}}. \quad (4.1)$$

The notation $\sigma(\mathbf{y})_i$ indicates the i -th value of \mathbf{y} mapped in the new range and is called score, y_i indicates the i -th value of \mathbf{y} and M indicates

the total number of camera models. Therefore a score $\sigma(\mathbf{y})_i$ produced by softmax, which we remember being associated with the i -node, represents the probability that input data has been acquired by the model associated with that node. To extract the predicted model \hat{m} , we have to select the node associated with the maximum score obtained:

$$\hat{m} = \underset{i}{\operatorname{argmax}} \sigma(\mathbf{y})_i. \quad (4.2)$$

In order to train and test a CNN for camera model identification, we must first divide the available input data into training, validation and testing data. Then, we go to provide the network with a certain amount of training data, called batch. Based on the score vectors obtained from the network and associated with each training data in the batch, we can calculate the loss function. In our case we use the cross-entropy loss (CE), which is defined in this way:

$$\text{CE} = \frac{1}{N} \sum_{n=1}^N \left(- \sum_{i=0}^{M-1} b_{ni} \cdot \ln(\sigma(\mathbf{y}_n)_i) \right), \quad (4.3)$$

where N indicates the batch size, \mathbf{y}_n the output vector of the n -th input data and b_{ni} a binary indicator equal to 1 when i is equal to the true camera model of the n -th data; 0 otherwise.

Thanks to the loss function we can update weights and biases of the network through the backpropagation, as described in Section 1.1. This procedure will be repeated starting from a new batch of different data, until we have provided all the training data to the network. At this point we are going to calculate a global CE on the validation data, called validation loss.

The process that goes from providing the CNN with training data to the calculation of the validation loss is repeated for a certain number of epochs. At the end of each epoch, the value of the validation loss is compared with the lowest one calculated previously. Therefore, validation data are used to understand how the CNN behaves with data that has not been used to adjust its parameters. When the number of epochs is finished, we consider the network trained with the parameters that allowed us to obtain the lowest value of the validation loss.

Generally, if the validation loss tends not to decrease, the learning rate is lowered to approach the minimum point more gradually; if even in this way it does not decrease, the training stops. Once the network is trained, we can test its effectiveness based on the predicted camera models of testing data.

After describing how a CNN for camera model identification works, let us see in detail the CNN architectures that we use in our analysis. Then, we focus on the concrete data that we provide as input to the networks. Visual and audio contents are in fact general concepts that do not precisely define a type of data. For now we only anticipate that these input data are extracted from video/audio tracks of videos in a preprocessing phase. The general procedure is described in Figure 4.1.

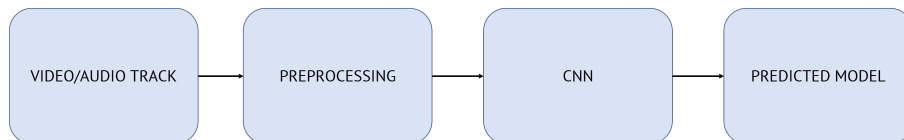


Figure 4.1: Black box block diagram of the whole process.

4.1.2 CNN Architectures

The CNNs we use to solve the problem are called EfficientNetB0 and VGGish. We use the first one for both visual and audio content, while the second one only for audio content.

4.1.2.1 EfficientNetB0 Architecture

The EfficientNetB0 belongs to the recently proposed EfficientNet family of CNN models [37], which has achieved very good results both in computer vision and multimedia forensics tasks. It is the simplest EfficientNet model; we have selected this in order to have faster training phases and, consequently, much more time to experiment with different configurations.

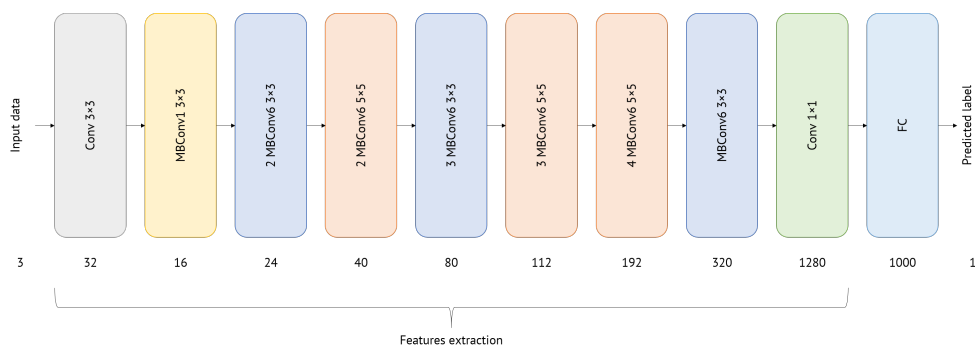


Figure 4.2: Simplified representation of the EfficientNetB0.

As we can see in Figure 4.2, the EfficientNetB0 is composed of several convolutional layers and a final fully connected layer. The notation $n \times n$ indicates the kernel size of that layer, while the number at the bottom

indicates the number of channels (the number of nodes for the fully connected layer). The successive layers which are the same are merged; the number placed before the name in some layers indicates the number of occurrences of that layer.

MBConv1 and MBConv6 are particular convolutional blocks belonging to a family of blocks called inverted residual blocks [7]. For instance, a general (i.e., not inverted) residual block is structured as shown in Figure 4.3.

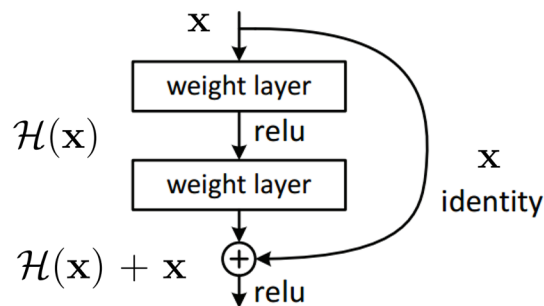


Figure 4.3: A general residual block [6].

As we can see, the peculiarity of this block consists in an additional connection that allows input \mathbf{x} to add directly to $\mathcal{H}(\mathbf{x})$ skipping the intermediate layers. Let us see how this additional connection, commonly called skip-connection, can be useful. A neural network try to best approximate a particular function, which can be more or less complex. A deep network can approximate very complex functions that would be out of reach for a too simple network, but would have much more trouble approximating simpler functions. Since the function to approximate is initially unknown, it is difficult to quantify its complexity and consequently choose a suitable network. The introduction of residual blocks in a deep network can help it to learn simpler functions, thanks to the opportunity to skip some layers [7]. The differences between a classic residual block and an inverted residual block can be seen in Figure 4.4.

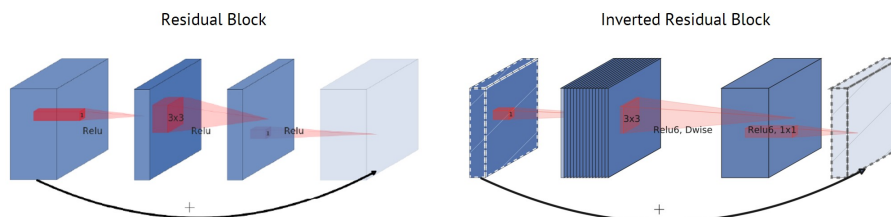


Figure 4.4: Representation of a residual block and an inverted residual block [7].

We can notice that a classic residual block follows a wide-narrow-wide approach concerning the number of channels [38] and is characterized by three convolutions + ReLU operations. On the other hand, an inverted residual block follows a narrow-wide-narrow approach and is characterized by a convolution + ReLU6, a depth-wise separable convolution (i.e., convolution that reduces the computational cost) + ReLU6 and a final convolution. The lightly colored layers indicate the beginning of the next block, while the diagonally hatched textures indicate linear bottlenecks, particular layers that contain few channels compared to the previous ones and are not preceded by an activation function. In the names MBConv1 and MBConv6, the numbers 1 and 6 refer to the so-called expansion factor t , which affects the number of channels of the intermediate layers of the block [7].

For our purposes, we replace the last fully connected layer with 1000 nodes with another fully connected layer with a number of nodes equal to that of camera models (i.e., labels) used for the classification, as shown in Figure 4.5.

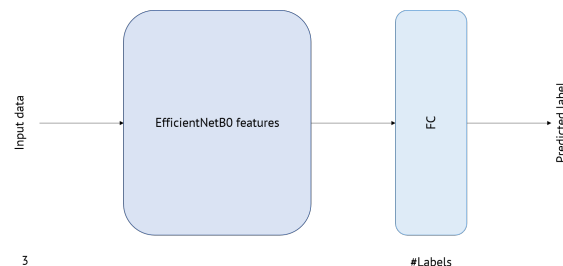


Figure 4.5: EfficientNetB0 with last layer modified.

4.1.2.2 VGGish Architecture

The VGGish [39] is a pretrained CNN from Google used for audio classification and inspired to the famous VGG networks [40] used for image classification. It has been used in several recent projects concerning audio classification; we can cite [41, 42]. The structure is much more compact than that of the EfficientNetB0 and is composed of several convolutional layers followed by some fully connected layer, as shown in Figure 4.6.

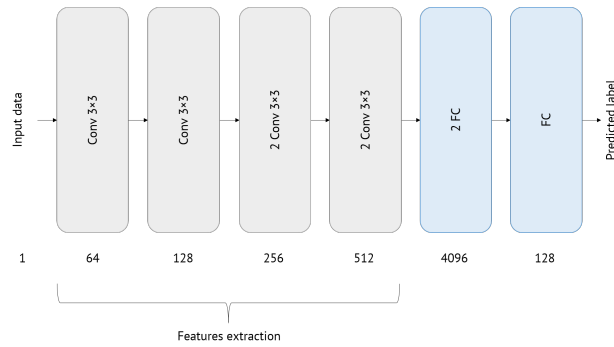


Figure 4.6: Simplified representation of the VGGish.

For our purposes, we add a final fully connected layer with a number of nodes equal to the number of labels, as shown in Figure 4.7.

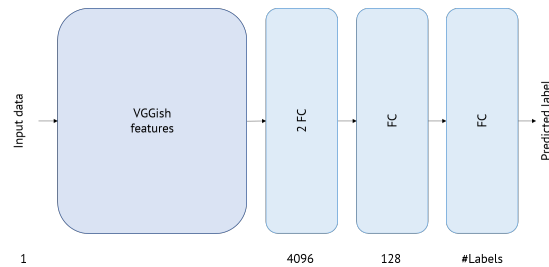


Figure 4.7: VGGish with one extra fully connected layer.

4.1.3 Data Preprocessing

The preprocessing phase consists of video and audio content extraction and data normalization. As far as audio is concerned, this process includes a further modification to the EfficientNetB0 in order to match the shape of the input data.

4.1.3.1 Visual Information

The visual content of a video can be seen as a set of images, called frames, placed one after the other over time. We define it as an array of shape $N \times C \times H \times W$, where N is the number of frames, C is the number of color channels and H and W respectively are the height and the width (i.e., the number of pixels along rows and columns) of a video frame. Since we adopt the RGB representation, C is always equal to 3. For each video, we extract a certain number N_F of frames (as shown in Figure 4.8) equally distant in time and distributed over its entire duration. In this way, we capture part of the visual information along the entire length of the video.

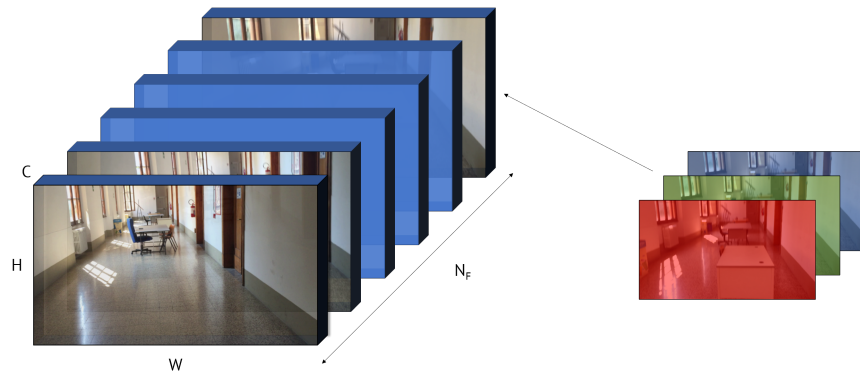


Figure 4.8: Representation of the extracted frames.

From each frame, we randomly crop a number N_P of color patches of size $H_P \times W_P$ (as shown in Figure 4.9); the patches are exactly the elements $v^{(m)} \in \mathcal{V}$. We use patches instead of frames as input data, so as to have a greater number of data available with the same size.

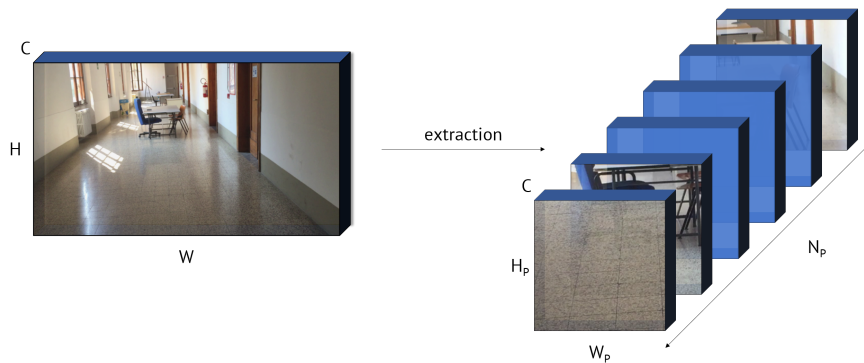


Figure 4.9: Extraction of the visual patches.

It is a good practice to normalize data before providing them to neural networks; this is because it can help the training phase [43]. There are several ways to normalize data; we used the following one:

$$(z_i)_c = \frac{(x_i)_c - \mu_c}{\sigma_c}. \quad (4.4)$$

This type of normalization, which is called z-score, is applied at the pixel-level of the patches: the notations $(x_i)_c$ and $(z_i)_c$ respectively indicate the $c \in \{\text{Red, Green, Blue}\}$ value of the raw and the normalized i -pixel of a patch; the notations μ_c and σ_c respectively indicate the mean and the standard deviation calculated with respect to the c value of all pixels of all patches. The whole process is summarized in Figure 4.10.

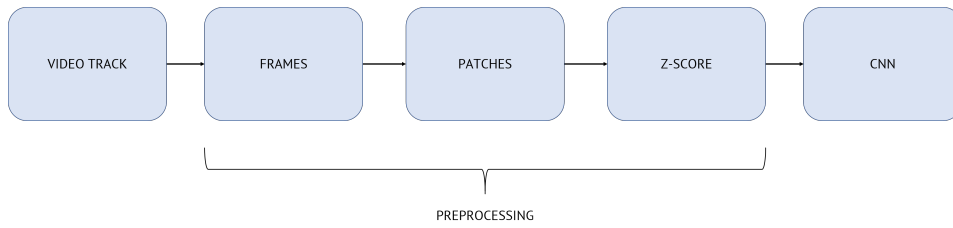


Figure 4.10: Visual information preprocessing block diagram.

4.1.3.2 Audio Information

From the audio tracks of the videos, we extract the log-mel spectrograms. Their information can be stored in 2D matrices $H \times W$, where values at each position (i.e., i -th row, j -th column) correspond to values of log-amplitudes that we find at coordinates (x_i, y_j) in their Cartesian representation; the x -axis indicates the time information, while the y -axis indicates the mel-frequency information. Since the networks we use admit arrays of shape $C \times H \times W$, we add a dimension C equal to 1 before the two dimensions H and W mentioned above. From each log-mel spectrogram, we randomly crop a number N_P of patches of size $C \times H_P \times W_P$ (as shown in Figure 4.11) and then we normalize them with the z-score (equation (4.4)).

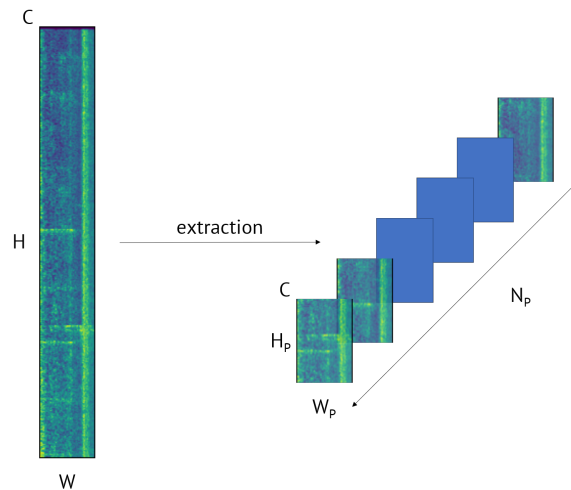


Figure 4.11: Extraction of the audio patches.

The patches are exactly the elements $a^{(m)} \in \mathcal{A}$. The whole process is summarized in Figure 4.12.

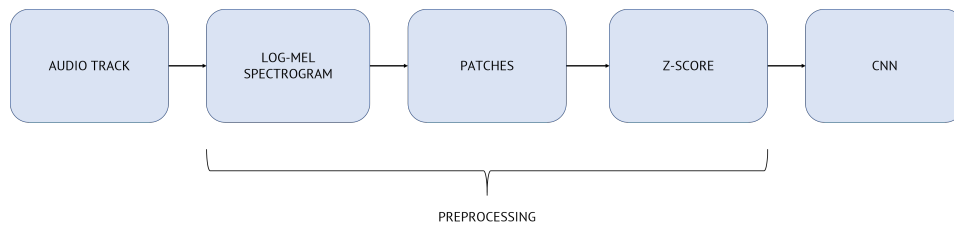


Figure 4.12: Audio information preprocessing block diagram.

In the EfficientNetB0 (as shown in Figure 4.13) we add an initial convolutional layer to switch from a 1-channel image to a 3-channel image as required by the network.

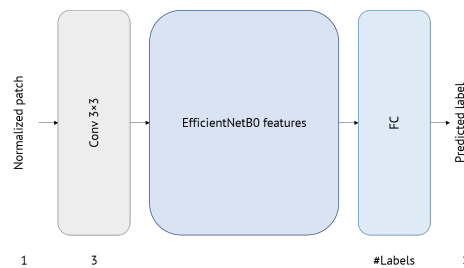


Figure 4.13: EfficientNetB0 further modified for audio data.

4.2 Multi-Modal Model Attribution

As mentioned in Section 2.2, given a video sequence, the problem of multi-modal model attribution converts in identifying the device model that shot it, using both visual and audio information of the video itself.

We adopt two methods to solve the problem of multi-modal model attribution:

1. Late Fusion method: compare the scores of each visual/audio patch pair separately obtained from two single-input CNNs;
2. Early Fusion method: build a multi-input CNN and train it with visual/audio patch pairs.

In both methods, we need a one-to-one correspondence between visual and audio patches. In other words, each visual patch must be paired with one and only one audio patch and vice versa. This is because we must ensure that no visual or audio patch is excluded and that any visual or audio patch is taken no more than once; in this way they all have the same importance.

Let N_F and N_P respectively be the number of frames and the number of patches per frame extracted from a video. In order to obtain a one-to-one correspondence, we extract $N_F \cdot N_P$ patches per log-mel spectrogram making sure that, for each video frame, its audio information is contained in one and only one group of N_P patches. In this way, we got the same number of visual and audio patches associated with a certain frame. The generation of pairs is random; it is sufficient that the patches belong to the same frame (i.e., N_P random pairs per frame).

4.2.1 Late Fusion

In the first method, we follow three steps to determine the predicted model \hat{m} for a visual/audio patch pair:

1. separately train a CNN with visual patches and a CNN with audio patches;
2. given a new visual/audio patch pair, provide the visual patch to the first network and the audio patch to the second one, obtaining the respective score vectors $\mathbf{s}_v = (\sigma(\mathbf{y}_v)_i)$ and $\mathbf{s}_a = (\sigma(\mathbf{y}_a)_i)$, $i \in [0, M - 1]$. Precisely, \mathbf{y}_v refers to the output vector related to the visual patch and \mathbf{y}_a to the output vector related to the audio patch (refer to Section 4.1.1);
3. select the score vector that contains the highest score; \hat{m} is given by the index in which that score is found: $\hat{m} = \operatorname{argmax}_i s_i$, where s_i is the i -th element of the vector \mathbf{s} defined as

$$\mathbf{s} = \begin{cases} \mathbf{s}_v & \text{if } \max_i \sigma(\mathbf{y}_v)_i \geq \max_i \sigma(\mathbf{y}_a)_i \\ \mathbf{s}_a & \text{if } \max_i \sigma(\mathbf{y}_v)_i < \max_i \sigma(\mathbf{y}_a)_i \end{cases}. \quad (4.5)$$

For the sake of clarity, Figure 4.14 shows the proposed pipeline for the Late Fusion multi-modal attribution.

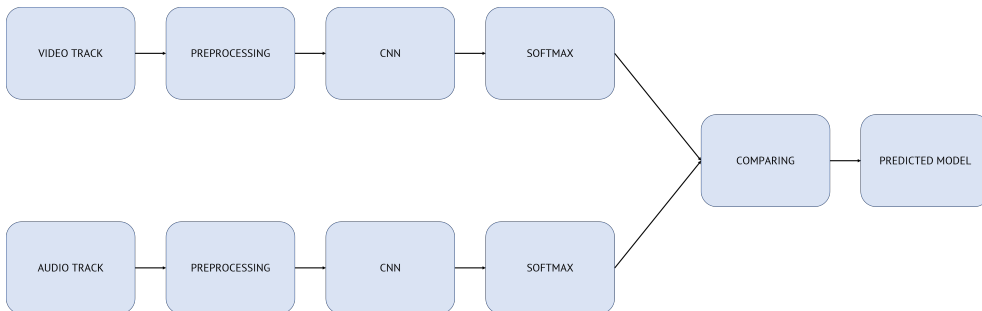


Figure 4.14: Late Fusion block diagram.

Table 4.1 shows an application of Late Fusion multi-modal attribution with 4 visual/audio patch pairs and 4 reference camera models. For example, if we analyze the second row, the highest score among visual and audio patches is associated with an audio patch and is equal to 1. Therefore, the predicted camera model for that patch pair is related to the position of this score in the audio patch score vector, i.e., the predicted model is that with label equal to 1.

Table 4.1: A general application of the Late Fusion with 3 patch pairs and 4 camera models. We indicate in bold the numbers analyzed in the text.

Visual Patch Scores	Audio Patch Scores	Max Score	Predicted Model
(0.8, 0.0, 0.0, 0.2)	(0.6, 0.0, 0.0, 0.4)	0.8	0
(0.5, 0.4, 0.1, 0.0)	(0.0, 1.0 , 0.0, 0.0)	1.0	1
(0.2, 0.1, 0.1, 0.6)	(0.2, 0.1, 0.2, 0.5)	0.6	3

4.2.2 Early Fusion

In the second method, we build a multi-input CNN by joining together an EfficientNetB0 with a VGGish or two EfficientNetB0. The union is made by concatenating the final fully connected layers of the two networks and is followed by some additional fully connected layer up to the prediction of the model. The resulting network then has two inputs: one for visual patches and one for audio patches.

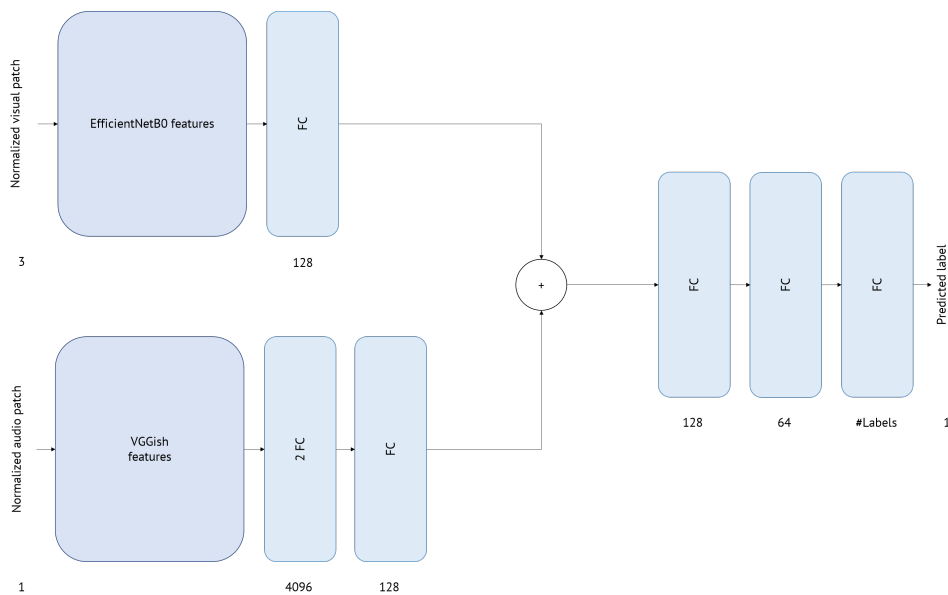


Figure 4.15: EfficientNetB0 + VGGish multi-input neural network.

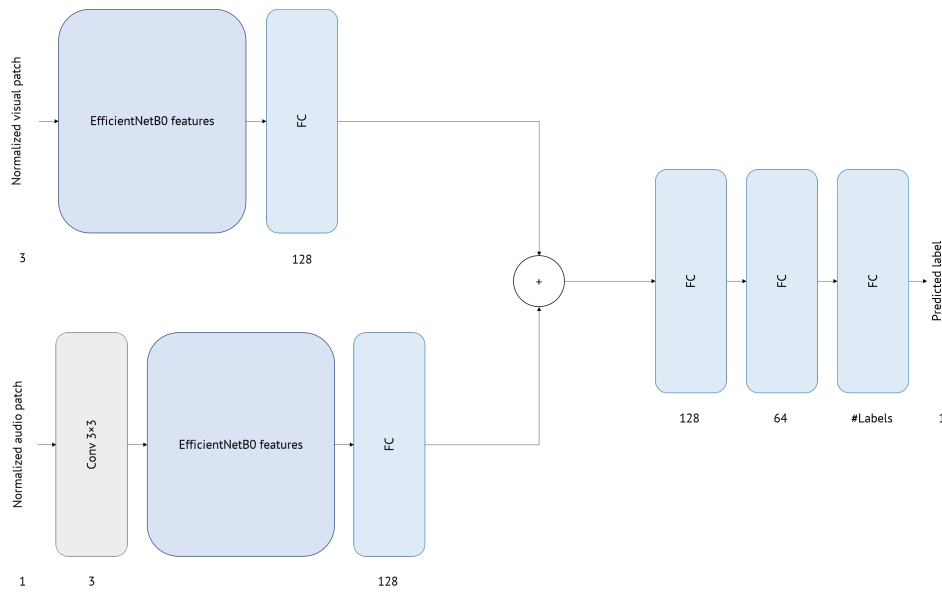


Figure 4.16: EfficientNetB0 + EfficientNetB0 multi-input neural network.

In Figures 4.15 and 4.16, we can see that the changes made to the individual single-input networks are different compared to the mono-modal case: the EfficientNetB0 networks, in fact, end with a fully connected layer of 128 nodes and the VGGish used is the classic one, without any modification. Both layers that go to concatenate have 128 nodes so as to give the same importance to both patches. For the sake of brevity, we do not report the preprocessing phase, which remains the same as mono-modal case.

Training and testing phases are analogous to that of the mono-modal model attribution, but this time we provide the CNN with visual/audio patch pairs; not single patches. For each patch pair the network predicts a model always based on the scores given by the application of a softmax function to the output vector \mathbf{y}_{v+a} (refer to Section 4.1.1):

$$\hat{m} = \underset{i}{\operatorname{argmax}} \sigma(\mathbf{y}_{v+a})_i. \quad (4.6)$$

4.3 Conclusive Remarks

In this chapter we have described the algorithms used to solve these problems, dwelling on the data preprocessing and the networks used for classification. We can notice how the most evident differences between visual and audio information methods reside in the preprocessing phase; once the patches are obtained, the residual process is almost identical.

We just have to concretely apply the methods and discuss the results obtained; we will see everything in the next chapter.

5

Simulations and Tests

Now let us concretely apply the methods described in Chapter 4. We first of all take a look at the dataset we work with and at the experimental setup (i.e., numerical values and configurations that we use in the experiments); after that we report and comment on the various results obtained and draw conclusions.

5.1 Vision Dataset

In our analysis we use the Vision dataset [8]; a recent image and video dataset, purposely designed for multimedia forensics investigations. Specifically, Vision dataset has been designed to follow the trend on image and video acquisition and social sharing. In the last few years photo-amateurs have rapidly converted to portable devices as preferred mean to capture images and videos. Then, the acquired content is typically shared on social media platforms, e.g., WhatsApp, Facebook, etc. In this vein, Vision dataset collects almost 12,000 native images of similar scenes captured by 35 modern smartphones/tablets, including also their related social media version. We can see the complete list of devices in Figure 5.1.

Brand	Model	ID	DStab	HDR	VR	#Videos	IR	#Images	#Flat	#Nat
Apple	iPad 2	D13	Off	F	1280 × 720	16	960 × 720	330	159	171
Apple	iPad mini	D20	On	F	1920 × 1080	16	2592 × 1936	278	119	159
Apple	iPhone 4	D09	Off	T	1280 × 720	19	2592 × 1936	326	109	217
Apple	iPhone 4S	D02	On	T	1920 × 1080	13	3264 × 2448	307	103	204
Apple	iPhone 4S	D10	On	T	1920 × 1080	15	3264 × 2448	311	133	178
Apple	iPhone 5	D29	On	T	1920 × 1080	19	3264 × 2448	324	100	224
Apple	iPhone 5	D34	On	T	1920 × 1080	32	3264 × 2448	310	106	204
Apple	iPhone 5c	D05	On	T	1920 × 1080	19	3264 × 2448	463	113	350
Apple	iPhone 5c	D14	On	T	1920 × 1080	19	3264 × 2448	339	130	209
Apple	iPhone 5c	D18	On	T	1920 × 1080	13	3264 × 2448	305	101	204
Apple	iPhone 6	D06	On	T	1920 × 1080	17	3264 × 2448	281	149	132
Apple	iPhone 6	D15	On	T	1920 × 1080	18	3264 × 2448	337	110	227
Apple	iPhone 6 Plus	D19	On	T	1920 × 1080	19	3264 × 2448	428	169	259
Asus	Zenfone 2 Laser	D23*	On	F	640 × 480	19	3264 × 1836	327	117	210
Huawei	Ascend G6-U10	D33	Off	T	1280 × 720	19	2448 × 3264	239	84	155
Huawei	Honor 5C NEM-L51	D30	Off	T	1920 × 1080	19	4160 × 3120	351	80	271
Huawei	P8 GRA-L09	D28	Off	T	1920 × 1080	19	4160 × 2336	392	126	266
Huawei	P9 EVA-L09	D03	Off	F	1920 × 1080	19	3968 × 2976	355	118	237
Huawei	P9 Lite VNS-L31	D16	Off	T	1920 × 1080	19	4160 × 3120	350	115	235
Lenovo	Lenovo P70-A	D07	Off	F	1280 × 720	19	4784 × 2704	375	158	217
LG electronics	D290	D04	On	F	800 × 480	19	3264 × 2448	368	141	227
Microsoft	Lumia 640 LTE	D17	Off	T	1920 × 1080	10	3264 × 1840	285	97	188
OnePlus	A3000	D25	On	T	1920 × 1080	19	4640 × 3480	463	176	287
OnePlus	A3003	D32	On	T	1920 × 1080	19	4640 × 3480	386	150	236
Samsung	Galaxy S III Mini GT-I8190	D26	Off	F	1280 × 720	16	2560 × 1920	210	60	150
Samsung	Galaxy S III Mini GT-I8190N	D01	Off	F	1280 × 720	22	2560 × 1920	283	78	205
Samsung	Galaxy S3 GT-I9300	D11	Off	T	1920 × 1080	19	3264 × 2448	309	102	207
Samsung	Galaxy S4 Mini GT-I9195	D31	Off	T	1920 × 1080	19	3264 × 1836	328	112	216
Samsung	Galaxy S5 SM-G900F	D27	Off	T	1920 × 1080	19	5312 × 2988	354	100	254
Samsung	Galaxy Tab 3 GT-P5210	D08	Off	F	1280 × 720	37	2048 × 1536	229	61	168
Samsung	Galaxy Tab A SM-T555	D35	Off	F	1280 × 720	16	2592 × 1944	280	126	154
Samsung	Galaxy Trend Plus GT-S7580	D22	Off	F	1280 × 720	16	2560 × 1920	314	151	163
Sony	Xperia Z1 Compact D5503	D12	On	T	1920 × 1080	19	5248 × 3936	316	100	216
Wiko	Ridge 4G	D21	Off	T	1920 × 1080	11	3264 × 2448	393	140	253
Xiaomi	Redmi Note 3	D24	Off	T	1920 × 1080	19	4608 × 2592	486	174	312

Figure 5.1: Complete list of devices in Vision dataset; DStab shows the presence or absence of digital stabilization on the acquired content, HDR indicates whether the device supports it, VR stands for video resolution and IR for image resolution [8].

For each device, the dataset presents different categories of videos. In particular, a video can be flat, indoor or outdoor. A flat video is made up of almost monochromatic frames very similar to each other; indoor and outdoor videos are respectively videos shot inside a building and videos shot outdoors. In Figure 5.2 we report three frames each belonging to a different category. For each video, or almost, the Vision dataset additionally provides two compressed copies: one with the WhatsApp algorithm and the other with the YouTube algorithm.



Figure 5.2: Frames respectively extracted from a flat video, an indoor video and an outdoor video shot with the device “D06_Apple_iPhone6” [8].

In the literature, Vision dataset has often been used for investigations on the camera model identification problem. Among them, we can cite [44, 45].

5.2 Experimental Setup

To perform our experiments we select non-flat native videos from Vision Video Dataset; both the original ones and those compressed by WhatsApp and YouTube algorithms. Since our analysis is aimed at the granularity model-level, we group videos from different devices that belong to the same model. Videos from devices “D04_LG_D290”, “D12_Sony_XperiaZ1Compact”, “D17_Microsoft_Lumia640LTE” and “D22_Samsung_GalaxyTrendPlus” are excluded because they give problems in the extraction of the frames or the audio track. We also exclude the original videos that do not feature a WhatsApp or YouTube compressed version. Therefore, we work with 1110 videos of about 1 minute, belonging to 25 different camera models. We extract 50 frames per video equally distant in time and distributed over its entire duration and 10 random patches per frame, for a total of 500 patches per video. As for audio, we extract 100 random patches per log-mel spectrogram in the mono-modal case and 500 random patches per log-mel spectrogram in the multi-modal case. We apply data augmentation on visual patches, in particular: horizontal flip, vertical flip, 90 degree rotation and JPEG compression [46] at six different quality levels, from 75 to 100 with a step of 5.

Following a common procedure applied in convolutional neural network (CNN) training, we initialize the EfficientNetB0 weights using those trained on ImageNet database [47], while we initialize the VGGish ones using those trained on AudioSet database [48]. We initialize in the same way also the weights of the EfficientNetB0 and of the VGGish networks that are part of the multi-input CNNs in the Early Fusion. All CNNs are trained using cross-entropy loss (CE) and Adam optimizer with default parameters. The learning rate is initialized to 0.001 and is decreased by

a factor 10 whenever the validation loss does not improve for 10 epochs. We train the networks for at most 50 epochs, and training is stopped if the validation loss does not decrease for more than 20 epochs. The model providing the best validation loss is selected.

Concerning the dataset split policy, we always keep 80% of the data for training phase (further divided in 85% – 15% for training set and validation set, respectively), leaving the remaining 20% to the evaluation set; we use a batch-size of 20. All tests run on a workstation equipped with one Intel® Xeon E5-2687W v4 (48 Cores @3 GHz), RAM 252 GB, one TITAN V (5120 CUDA Cores @1455 MHz), 12 GB, running Ubuntu 20.04.2. We resort to Albumentations [49] as data augmentation library for visual patches and we use Pytorch [50] as Deep Learning framework.

5.3 Results

Now let us see the results obtained with various configurations, starting from the mono-modal methods to then arrive at the multi-modal methods.

5.3.1 Visual Patches

We train the EfficientNetB0 using visual patches of shape $3 \times 256 \times 256$ from original videos, videos compressed by the WhatsApp algorithm and videos compressed by the YouTube algorithm, obtaining three network models. We test these models not only using patches from videos of the same category, but also from different categories (e.g., test patches from WhatsApp videos on the model trained with patches from original videos). This last type of test is called cross test.

Let us start by analyzing the confusion matrices produced by non-cross tests.

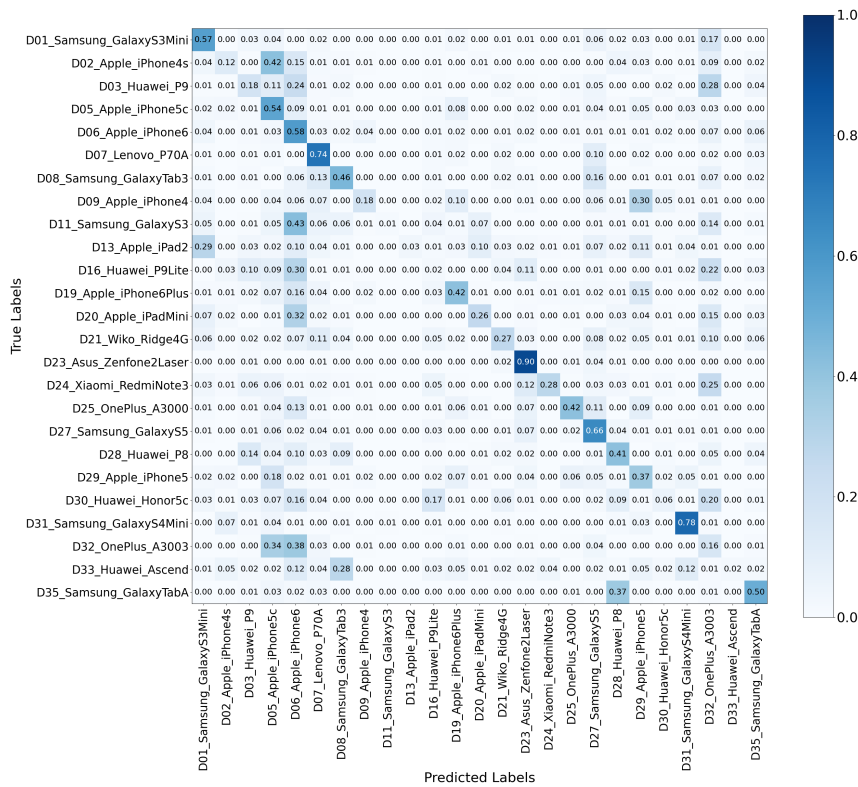


Figure 5.6: Training Set: Original; Testing Set: WhatsApp; Accuracy: 0.3579.

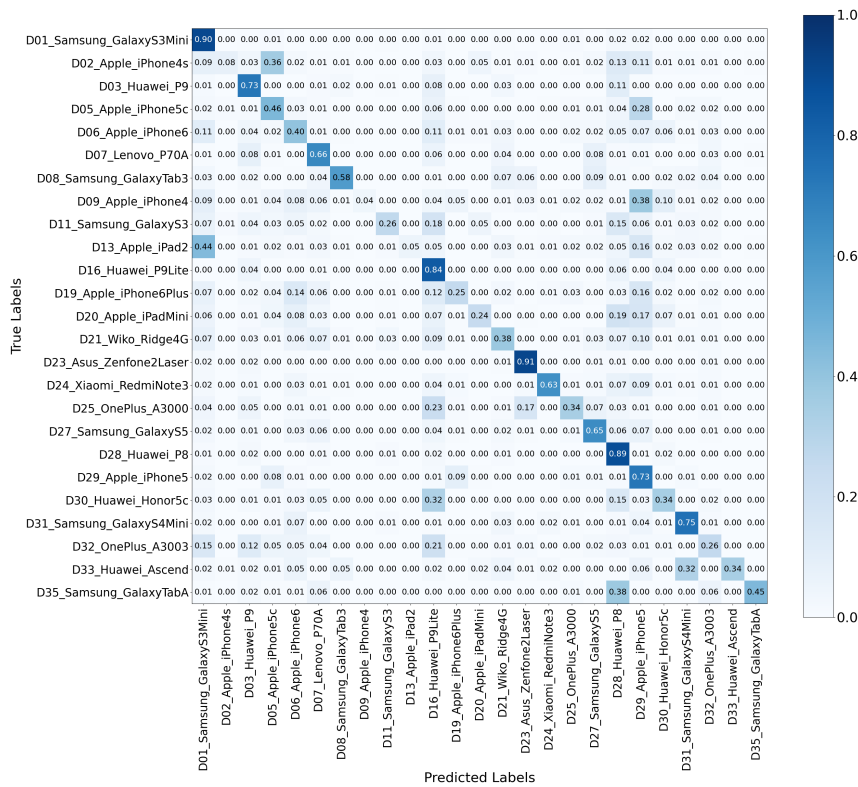


Figure 5.7: Training Set: Original; Testing Set: YouTube; Accuracy: 0.4869.

As shown in Figures 5.6 and 5.7, the algorithm presents some difficulties in the classification; only some models manage to classify them very well, while others tend to confuse them (especially smartphones of similar models of the same brand).

We summarize in Table 5.1 the accuracy given by the various tests in order to draw conclusions.

Table 5.1: Table that collects the accuracy achieved in the various tests.

Testing Set → Training Set ↓	Original	WhatsApp	YouTube
Original	0.8202	0.3579	0.4869
WhatsApp	0.5599	0.6739	0.5158
YouTube	0.7271	0.5531	0.7404

We note how the algorithm works worse in the case in which patches are coming from WhatsApp videos. Apparently, the compression applied by WhatsApp is more massive than that of YouTube, making it difficult to extract the features suitable for classification.

Let us investigate further by compressing videos with the H.264 [51] and H.265 [52] codecs at different quality factors. For the compression, we use FFmpeg [53] at five different Constant Rate Factor (CRF) values [53]: 7, 15, 23, 31 and 39; smaller values correspond to better quality, larger values to lower. For the sake of brevity, we report only the classification accuracies as a function of the used dataset (i.e., original, WhatsApp-compressed, YouTube-compressed) and CRF values. Figure 5.8 shows the accuracies concerning the data compressed with the H.264 codec, Figure 5.9 reports accuracies concerning the data compressed with the H.265 codec.

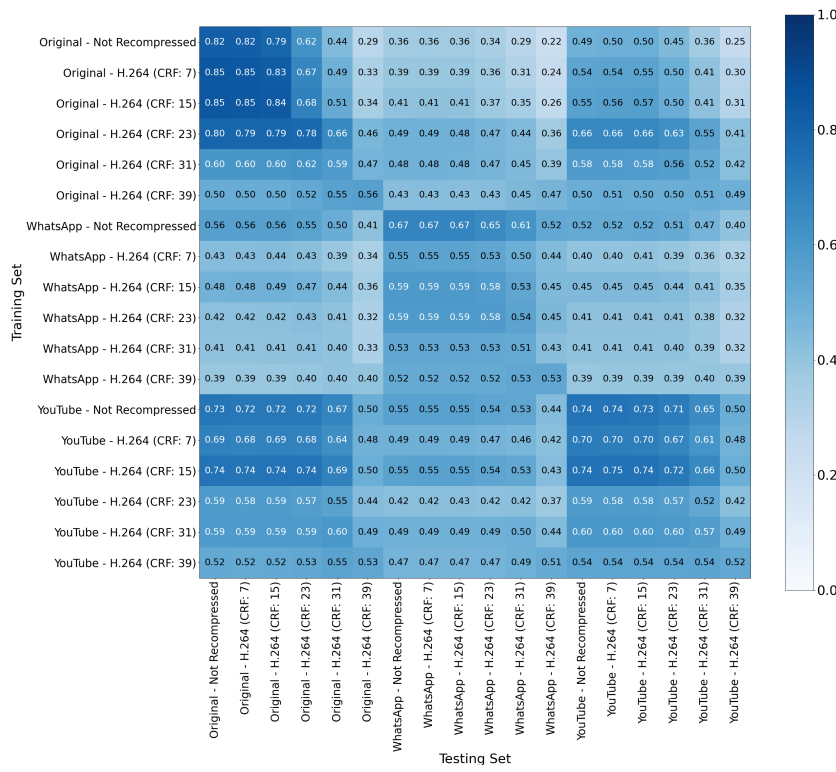


Figure 5.8: Classification accuracies as a function of training/testing sets and H.264 compression quality factors.

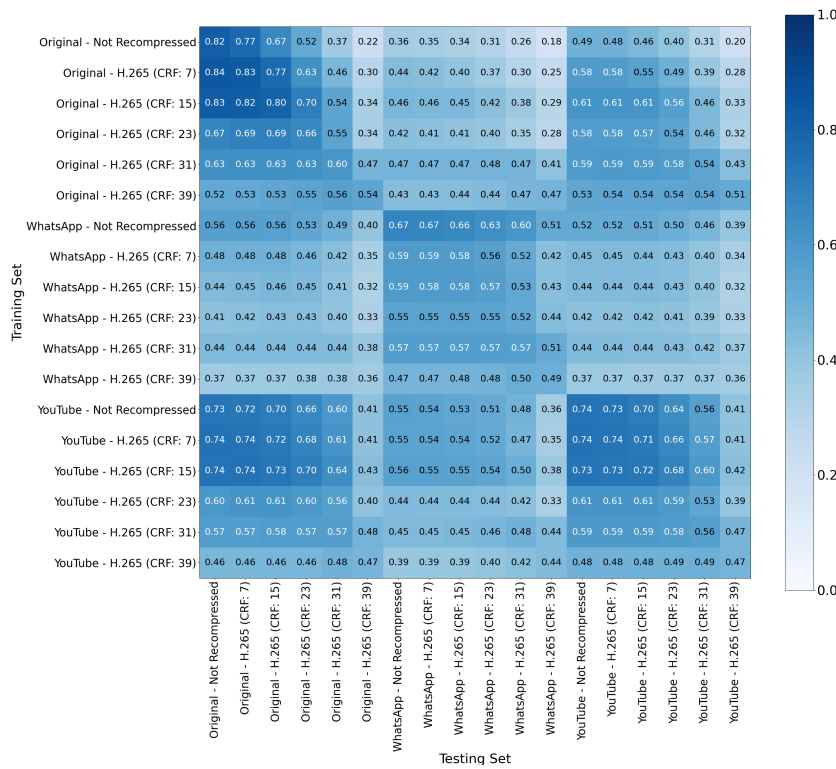


Figure 5.9: Classification accuracies as a function of training/testing sets and H.265 compression quality factors.

As we can see, the matrices are quite comparable and, predictably, the accuracy drops as the compression factor increases.

To conclude, we report in Figure 5.10 the accuracies concerning the data compressed with H.264 and H.265 codecs.

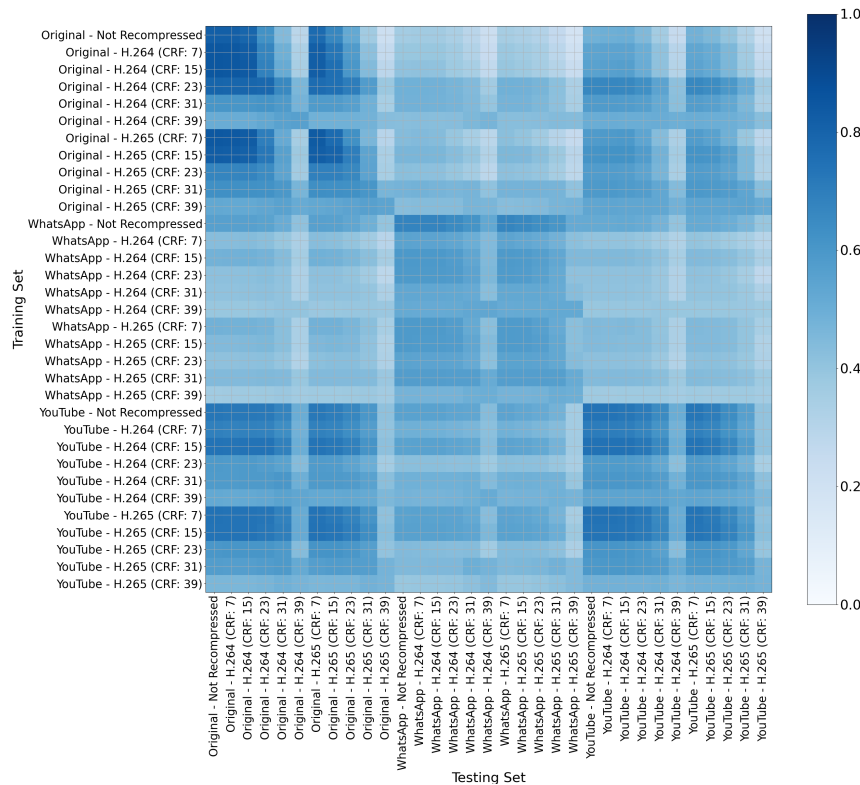


Figure 5.10: Classification accuracies as a function of training/testing sets and H.264 and H.265 compression quality factors.

Since the matrix is excessively large, we report only the colors of the boxes and not the numbers (that would have been illegible); in this case it is much more important to rely on the glance. The areas concerning the different training set/testing set pairs are clearly distinguishable. We achieve the best results for the pairs of original/original, YouTube/original and YouTube/YouTube sets. The same cannot be said for the original/YouTube pair of sets, where the classification algorithm works worse. Comparing the cross tests H.264/H.265 and H.265/H.264, we note how the results are very similar in general. The most evident difference is found in the area with the original/original set pairs, where a significant decrease in accuracy occurs at lower CRF values in H.265 compared to H.264. The tests involving WhatsApp data are confirmed as the worst, representing the weakness of the model. Finally, we can see how better results are obtained in cross tests if the training set is made up of more compressed data than those of the testing set (e.g., YouTube/original

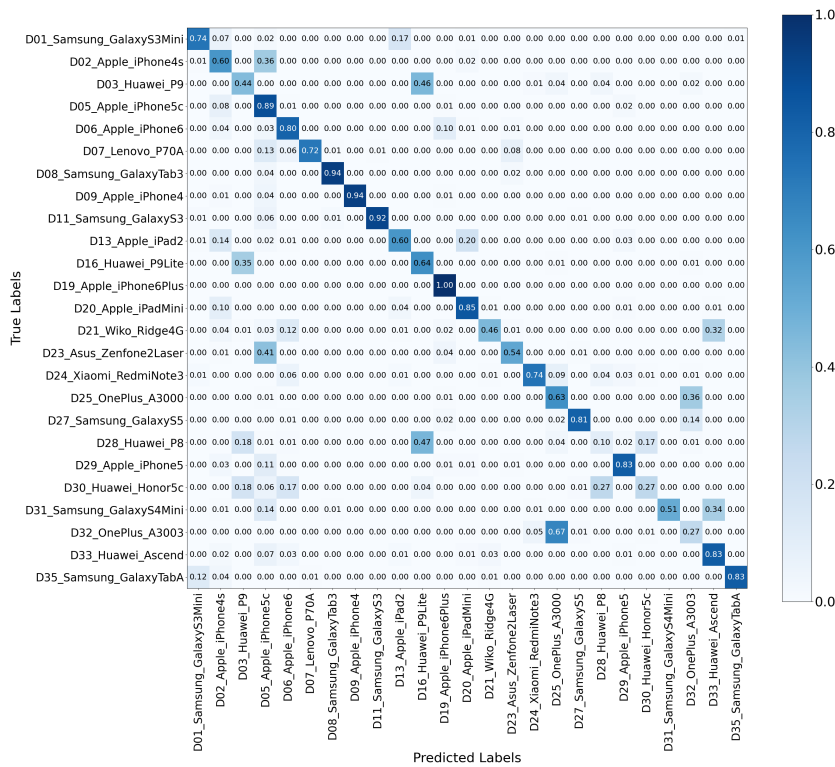


Figure 5.12: VGGish - Training Set: WhatsApp; Testing Set: WhatsApp; Accuracy: 6757.

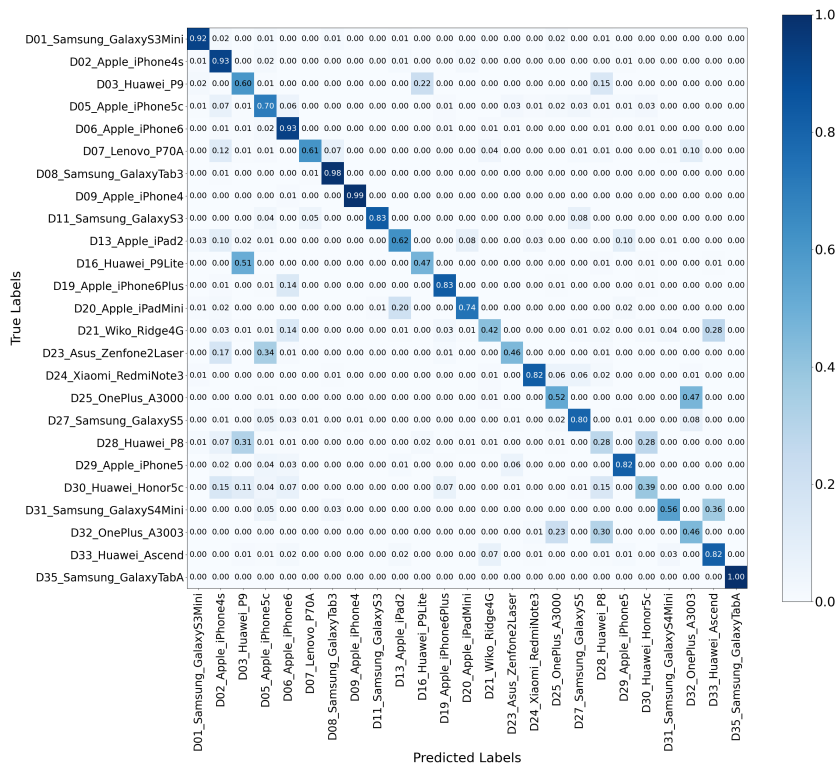


Figure 5.13: VGGish - Training Set: YouTube; Testing Set: YouTube; Accuracy: 0.7010.

As shown in Figures 5.11, 5.12 and 5.13, the classification algorithm works quite well, beyond a bit of uncertainty about some model.

Now let us see how it behaves in cross tests (for the sake of brevity, we show only two tests).

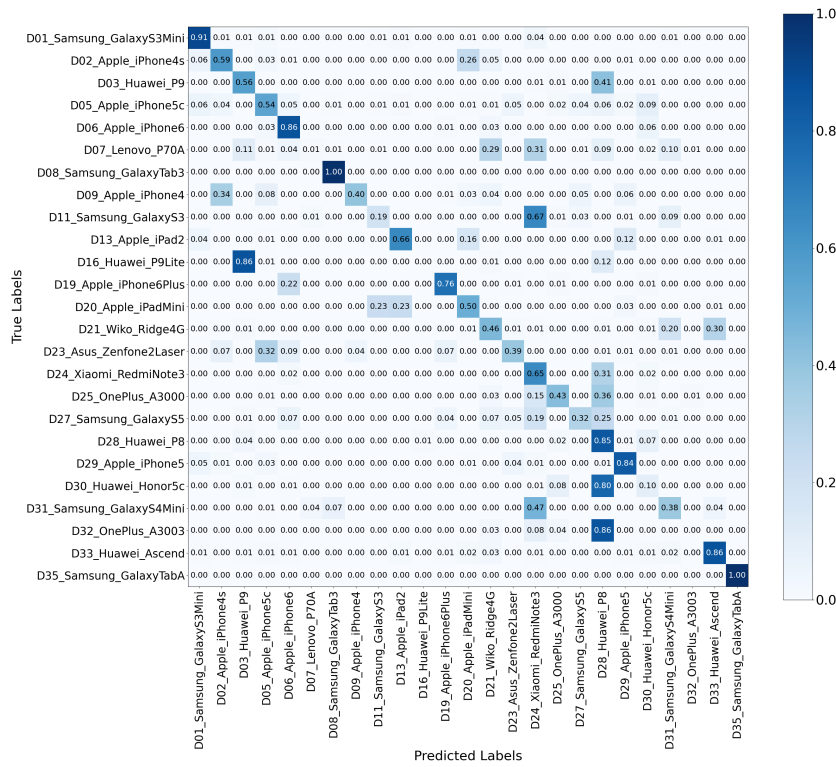


Figure 5.14: VGGish - Training Set: Original; Testing Set: WhatsApp; Accuracy: 0.5304.

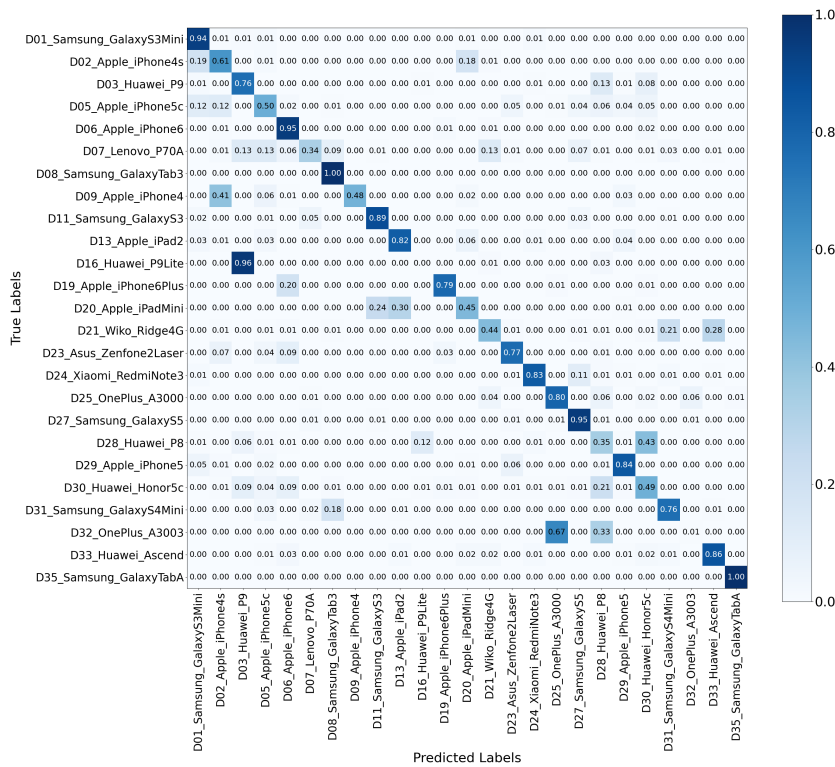


Figure 5.15: VGGish - Training Set: Original; Testing Set: YouTube; Accuracy: 0.6654.

As shown in Figures 5.14 and 5.15, the algorithm presents some difficulties; but results are more stable with respect to the classification with visual patches.

We summarize in Table 5.2 the accuracy given by the various tests in order to draw conclusions.

Table 5.2: Table that collects the classification accuracies of mono-modal methods as a function of training/testing sets; for audio patches we have a VGGish. We indicate in bold the numbers analyzed in the text.

-	Audio Patches			Visual Patches		
Testing Set → Training Set ↓	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube
Original	0.6578	0.5304	0.6654	0.8202	0.3579	0.4869
WhatsApp	0.5028	0.6757	0.5245	0.5599	0.6739	0.5158
YouTube	0.6954	0.5924	0.7010	0.7271	0.5531	0.7404

The algorithm does not present particular weaknesses, but very high accuracy are never achieved. Certainly, the biggest improvement with respect to the mono-modal case with visual patches is obtained in the

original/WhatsApp and original/YouTube cross tests. The results of the remaining cross-tests, on the other hand, are very similar in both configurations. As regards the non-cross tests, we continue to have the best results with the visual patches, reaching a maximum accuracy of 82% in the original/original test; with audio patches we only reach a maximum of 70% in the YouTube/YouTube test. Intuitively, the network model obtained with audio patches is more general, because there are no particular cases in which it behaves much better or much worse than the others.

Let us investigate further by compressing audio in MP3 format [54] at different quality factors. For the compression, we used Ffmpeg at five different qscale values [53]: 0, 2, 4, 6 and 8; smaller values correspond to better quality, larger values to lower. For the sake of brevity, we report in Figure 5.16 the classification accuracies as a function of the used dataset and qscale values.

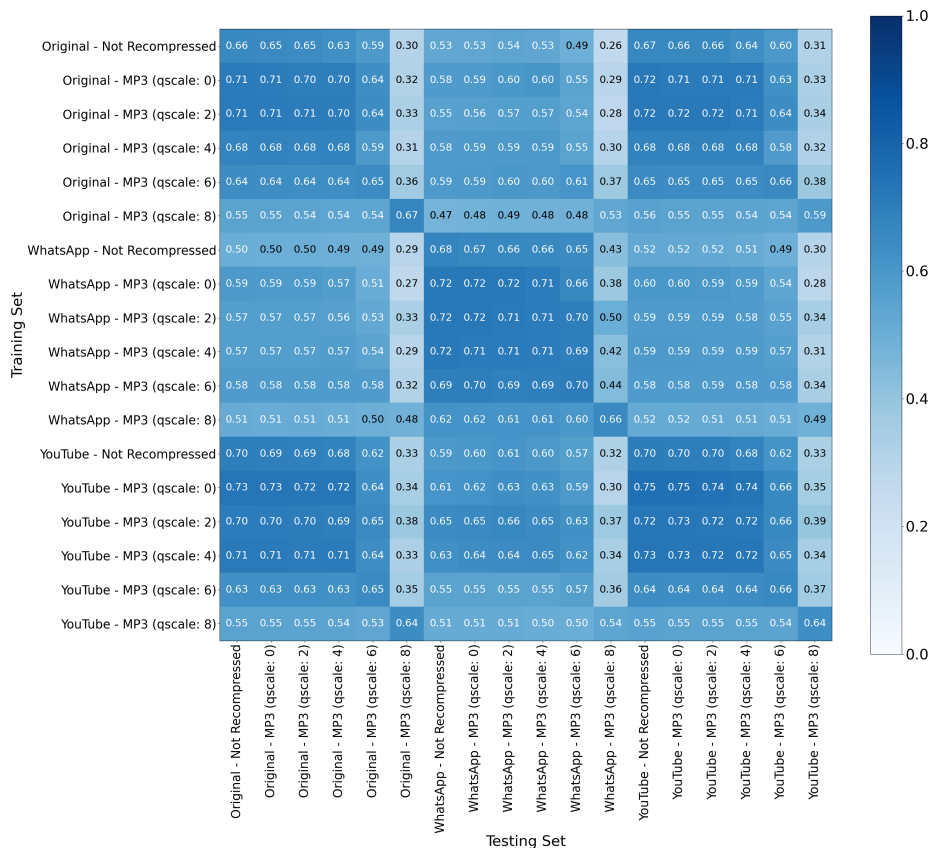


Figure 5.16: Classification accuracies as a function of training/testing sets and MP3 compression quality factors.

The generality of the network model is reconfirmed, in fact there are no major changes between one compression and another; a noticeable

decrease in accuracy occurs only when the compression is really high.

5.3.2.2 Results of EfficientNetB0 on Audio Patches

Passing to the EfficientNetB0, patches are extracted both as for the VG-Gish, and by expanding the frequency range to the entire audio band using 192 mel beans. We call the EfficientNetB0 trained with patch $1 \times 96 \times 64$ EfficientNetB0₆₄, while we call the EfficientNetB0 trained with patch $1 \times 96 \times 192$ EfficientNetB0₁₉₂. For the sake of brevity, we report the classification accuracies as a function of the used dataset and MP3 values. Figure 5.17 shows the accuracies concerning the EfficientNetB0₆₄, Figure 5.18 reports accuracies concerning the EfficientNetB0₁₉₂.

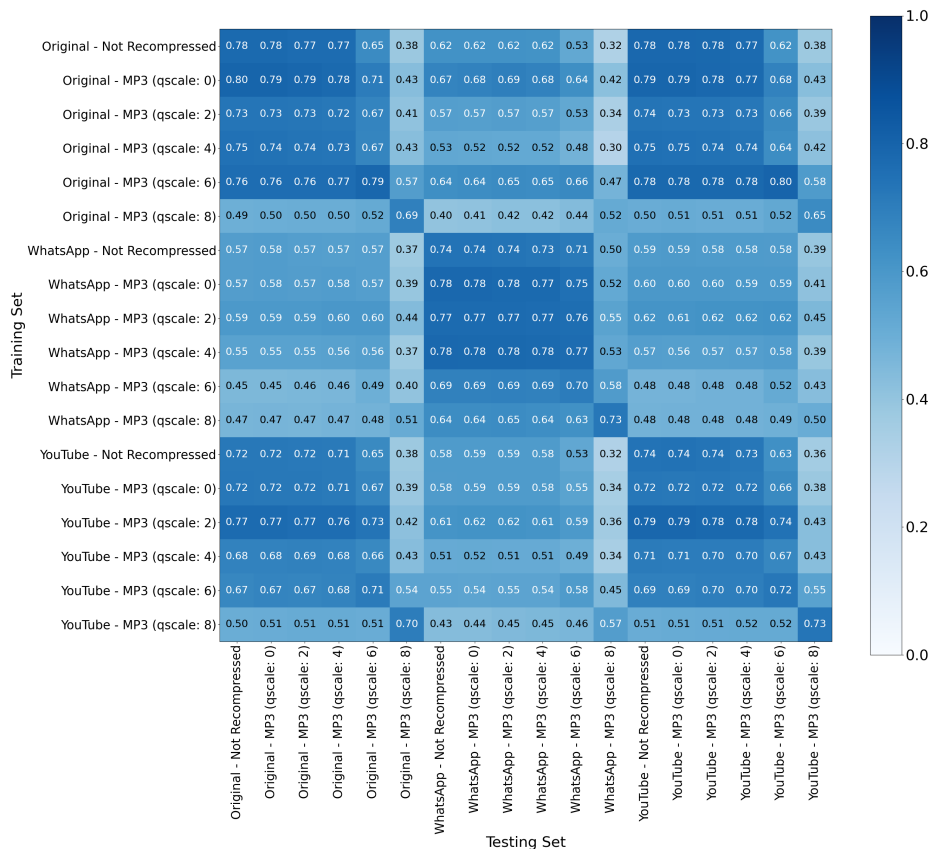


Figure 5.17: Classification accuracies as a function of training/testing sets and MP3 compression quality factors - EfficientNetB0₆₄.

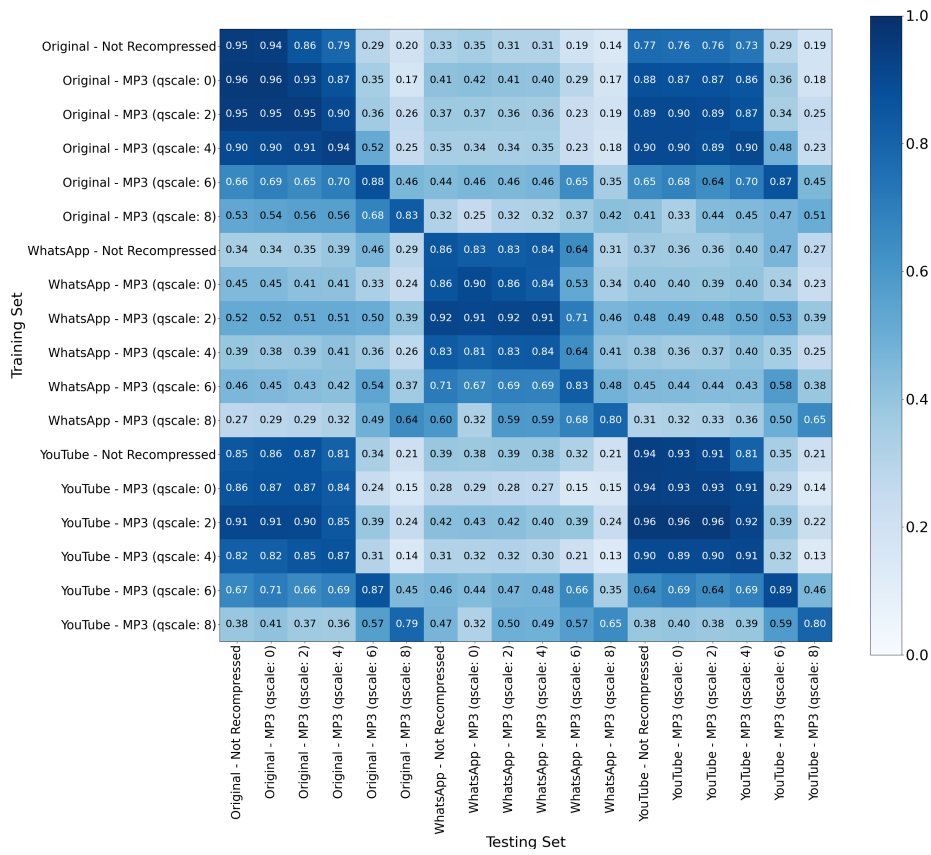


Figure 5.18: Classification accuracies as a function of training/testing sets and MP3 compression quality factors - EfficientNetB0192.

Starting from Figure 5.17, we notice how the differences with respect to VGGish are minimal. The results are generally a little better, but we have more sensitivity on the compression factor.

Expanding the spectrum, the differences become considerable. In Figure 5.18 we note very high accuracy with regard to non-cross tests and cross tests including only original and Youtube data, but very low accuracy with regard to cross tests including WhatsApp data.

We summarize in Table 5.3 the accuracy given by the various tests in order to draw conclusions.

Table 5.3: Table that collects the classification accuracies of mono-modal methods as a function of training/testing sets; VGGish, EfficientNetB0₆₄, EfficientNetB0₁₉₂ are for audio patches and EfficientNetB0 for visual patches. We indicate in bold the numbers analyzed in the text.

-	VGGish			EfficientNetB0 ₆₄			EfficientNetB0 ₁₉₂			EfficientNetB0		
Test. Set → Train. Set ↓	O	WA	YT	O	WA	YT	O	WA	YT	O	WA	YT
O	0.66	0.53	0.67	0.78	0.62	0.78	0.95	0.33	0.77	0.82	0.36	0.49
WA	0.50	0.68	0.52	0.57	0.74	0.59	0.34	0.86	0.37	0.56	0.67	0.52
YT	0.70	0.59	0.70	0.72	0.58	0.74	0.85	0.39	0.94	0.73	0.55	0.74

As we can see, the best network model from the point of view of non-cross tests is undoubtedly the one made with the EfficientNetB0₁₉₂, which reaches a maximum accuracy value of 95%. However, it is also the one that presents the worst results in cross tests with WhatsApp data. This network model is therefore the least general, whose performance drastically worsens in the case of more powerful compressions (e.g., WhatsApp). Apparently, it comes closest to approximating the function that maps the training data to their labels.

As for the original/YouTube and YouTube/original cross tests, the best results are found in the EfficientNetB0₆₄ and in the EfficientNetB0₁₉₂; while for WhatsApp cross tests the best results are found in VGGish and in the EfficientNetB0₆₄. Finally, if we get better cross test results for visual patches with a training set consisting of more compressed data than the testing set (e.g., YouTube/original results are better than original/YouTube), the same does not happen for audio patches where the results remain very similar (e.g., YouTube/original results are very similar to original/YouTube).

5.3.3 Late Fusion

Given the previous results, we focus on trying to improve the cases where mono-modal methods performed worse (i.e., cross-tests with WhatsApp data). In order to improve these results, intuitively it is more reasonable to combine the visual results with those of an audio network that has proved to be more general: VGGish or EfficientNetB0₆₄. We call the Late Fusion with the VGGish Late Fusion 1; Late Fusion 2 the one with EfficientNetB0₆₄. For the sake of brevity, we directly report the classification accuracies as a function of the used dataset for both configurations;

omitting further compressions (i.e., H.264, H.265 and MP3).

Table 5.4: Table that collects the classification accuracies of Late Fusion 1 and mono-modal methods as a function of training/testing sets; for Late Fusion 1 we have an EfficientNetB0 + VGGish, for audio patches we have a VGGish. We indicate in bold the numbers analyzed in the text.

-	Late Fusion 1			Visual Patches			Audio Patches		
Testing Set → Training Set ↓	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube
Original	0.9039	0.5960	0.7069	0.8202	0.3579	0.4869	0.6578	0.5304	0.6654
WhatsApp	0.6413	0.7610	0.6368	0.5599	0.6739	0.5158	0.5028	0.6757	0.5245
YouTube	0.8163	0.6595	0.8274	0.7271	0.5531	0.7404	0.6954	0.5924	0.7010

Table 5.5: Table that collects the classification accuracies of Late Fusion 2 and mono-modal methods as a function of training/testing sets; for Late Fusion 2 we have an EfficientNetB0 + EfficientNetB0₆₄, for audio patches we have an EfficientNetB0₆₄. We indicate in bold the numbers analyzed in the text.

-	Late Fusion 2			Visual Patches			Audio Patches		
Testing Set → Training Set ↓	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube
Original	0.8945	0.6020	0.8039	0.8202	0.3579	0.4869	0.7758	0.6214	0.7847
WhatsApp	0.6262	0.8198	0.6208	0.5599	0.6739	0.5158	0.5730	0.7408	0.5857
YouTube	0.8321	0.6976	0.8390	0.7271	0.5531	0.7404	0.7203	0.5823	0.7404

In Tables 5.4 and 5.5 we note how Late Fusion 1 and Late Fusion 2 perform better than their mono-modal counterparts. The only result in which the mono-modal has a slightly higher accuracy is in the original/WhatsApp cross test with respect to Late Fusion 2, as reported by the bold numbers in Table 5.5.

Based on the results considered, there is no multi-modal configuration that is totally better than the other. In a general sense, we can consider Late Fusion 2 better, because the results in which it is worse are few and not far from those obtained by the other configuration. On the contrary, in the more numerous results where Late Fusion 2 is better, the gap compared to those of the other configuration is greater. An example of this we can find in the original/YouTube cross test (as reported by the bold numbers in Tables 5.4 and 5.5), where the gap is about 10%.

In both multi-modal configurations, we can see how better results are obtained in cross tests if the training set is made up of more compressed

data than those of the testing set (e.g., YouTube/original results are better than original/YouTube ones). WhatsApp cross tests still give the worst results.

To conclude, we combine the scores of the EfficientNetB0 and the EfficientNetB0₁₉₂, calling this configuration Late Fusion 3. For the sake of brevity, we directly report the classification accuracies as a function of the used dataset, omitting further compressions.

Table 5.6: Table that collects the classification accuracies of Late Fusion 3 and mono-modal methods as a function of training/testing sets; for Late Fusion 3 we have an EfficientNetB0 + EfficientNetB0₁₉₂, for audio patches we have a EfficientNetB0₁₉₂. We indicate in bold the numbers analyzed in the text.

-	Late Fusion 3			Visual Patches			Audio Patches		
Testing Set → Training Set ↓	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube
Original	0.9900	0.4544	0.8389	0.8202	0.3579	0.4869	0.9518	0.3307	0.7729
WhatsApp	0.5703	0.9163	0.5602	0.5599	0.6739	0.5158	0.3418	0.8583	0.3706
YouTube	0.9172	0.4957	0.9519	0.7590	0.5531	0.7404	0.8490	0.3902	0.9398

As expected, the network model obtained works worse in cross tests with WhatsApp data than Late Fusion 1 and Late Fusion 2. This derives from the fact that this multi-input network is composed of two networks that are able to very well approximate the function that maps the training data to their labels; this is certainly good if this network model should work only with data of the same category, but it is not very suitable for more heterogeneous data. Table 5.6 shows a very large capacity of the network to classify homogeneous data with extremely high accuracy values, up to 99%.

5.3.4 Early Fusion

As for Late Fusion, we focus on trying to improve the cases where mono-modal methods performed worse (i.e., cross-tests with WhatsApp data). In order to improve these results, we train two multi-input networks using visual and audio patches from original videos, videos compressed by the WhatsApp algorithm and videos compressed by the YouTube algorithm. The first one is composed of the EfficientNetB0 and the VGGish, the second one of the EfficientNetB0 and the EfficientNetB0₆₄. We call the first configuration Early Fusion 1 and the second Early Fusion 2.

Table 5.7: Table that collects the classification accuracies of Early Fusion 1 and mono-modal methods as a function of training/testing sets; for Early Fusion 1 we have an EfficientNetB0 + VGGish, for audio patches we have a VGGish. We indicate in bold the numbers analyzed in the text.

-	Early Fusion 1			Visual Patches			Audio Patches		
Testing Set → Training Set ↓	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube
Original	0.8210	0.6879	0.7784	0.8202	0.3579	0.4869	0.6578	0.5304	0.6654
WhatsApp	0.5810	0.7519	0.5766	0.5599	0.6739	0.5158	0.5028	0.6757	0.5245
YouTube	0.7548	0.6212	0.7590	0.7271	0.5531	0.7404	0.6954	0.5924	0.7010

Table 5.8: Table that collects the classification accuracies of Early Fusion 2 and mono-modal methods as a function of training/testing sets; for Early Fusion 2 we have an EfficientNetB0 + EfficientNetB0₆₄, for audio patches we have an EfficientNetB0₆₄. We indicate in bold the numbers analyzed in the text.

-	Early Fusion 2			Visual Patches			Audio Patches		
Testing Set → Training Set ↓	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube
Original	0.8396	0.6120	0.7956	0.8202	0.3579	0.4869	0.7758	0.6214	0.7847
WhatsApp	0.5930	0.8076	0.5873	0.5599	0.6739	0.5158	0.5730	0.7408	0.5857
YouTube	0.8071	0.6903	0.8090	0.7271	0.5531	0.7404	0.7203	0.5823	0.7404

In Tables 5.7 and 5.8 we note how Early Fusion 1 and Early Fusion 2 perform better than their mono-modal counterparts. The only result in which the mono-modal has a slightly higher accuracy is in the original/WhatsApp cross test with respect to Early Fusion 2, as reported by the bold numbers in Table 5.8. Regarding Early Fusion 1, we can record a fairly significant increase in accuracy in the cross test with original data as training set and WhatsApp data as testing set (as reported by the bold number in Table 5.7). This cross test is important because it can be traced back to different realistic scenarios in which we have to classify a data compressed through internet services (e.g., social media, upload sites, etc.).

To conclude, we train the multi-input network composed by the EfficientNetB0 and the EfficientNetB0₁₉₂. We call this configuration Early Fusion 3.

Table 5.9: Table that collects the classification accuracies of Early Fusion 3 and mono-modal methods as a function of training/testing sets; for Early Fusion 3 we have an EfficientNetB0 + EfficientNetB0₁₉₂, for audio patches we have a EfficientNetB0₁₉₂.

-	Early Fusion 3			Visual Patches			Audio Patches		
Testing Set → Training Set ↓	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube
Original	0.9598	0.1795	0.7968	0.8202	0.3579	0.4869	0.9518	0.3307	0.7729
WhatsApp	0.5091	0.9120	0.4954	0.5599	0.6739	0.5158	0.3418	0.8583	0.3706
YouTube	0.8731	0.4146	0.9513	0.7590	0.5531	0.7404	0.8490	0.3902	0.9398

As expected, the network model obtained works very badly in cross tests with WhatsApp data. In particular, we note how the accuracy value in the cross test with original data as training set and WhatsApp data as testing set is the lowest compared to the previous methods. This derives from the fact that this multi-input network is composed of two networks that are able to very well approximate the function that maps the training data to their labels; this is certainly good if this network model should work only with data of the same category, but it is not very suitable for more heterogeneous data. Table 5.9 shows a very large capacity of the network to classify homogeneous data with extremely high accuracy values.

We merge in Tables 5.10 and 5.11 the classification accuracies of the multi-modal methods as a function of the used dataset, in order to have a global overview.

Table 5.10: Table that collects the classification accuracies of Late Fusion 1, Late Fusion 2 and Late Fusion 3 as a function of training/testing sets; for Late Fusion 1 we have an EfficientNetB0 + VGGish, for Late Fusion 2 we have an EfficientNetB0 + EfficientNetB0₆₄, for Late Fusion 3 we have an EfficientNetB0 + EfficientNetB0₁₉₂. We indicate in bold the numbers analyzed in the text.

-	Late Fusion 1			Late Fusion 2			Late Fusion 3		
Testing Set → Training Set ↓	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube
Original	0.9039	0.5960	0.7069	0.8945	0.6020	0.8039	0.9900	0.4544	0.8389
WhatsApp	0.6413	0.7610	0.6368	0.6262	0.8198	0.6208	0.5703	0.9163	0.5602
YouTube	0.8163	0.6595	0.8274	0.8321	0.6976	0.8390	0.9172	0.4957	0.9519

Table 5.11: Table that collects the classification accuracies of Early Fusion 1, Early Fusion 2 and Early Fusion 3 as a function of training/testing sets; for Early Fusion 1 we have an EfficientNetB0 + VGGish, for Early Fusion 2 we have an EfficientNetB0 + EfficientNetB0₆₄, for Early Fusion 3 we have an EfficientNetB0 + EfficientNetB0₁₉₂. We indicate in bold the numbers analyzed in the text.

-	Early Fusion 1			Early Fusion 2			Early Fusion 3		
Testing Set → Training Set ↓	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube	Original	WhatsApp	YouTube
Original	0.8210	0.6879	0.7784	0.8396	0.6120	0.7956	0.9598	0.1795	0.7968
WhatsApp	0.5810	0.7519	0.5766	0.5930	0.8076	0.5873	0.5091	0.9120	0.4954
YouTube	0.7548	0.6212	0.7590	0.8071	0.6903	0.8090	0.8731	0.4146	0.9513

In general, multi-modal methods prove to be more effective than mono-modal methods, showing higher accuracy values. Starting from the analysis of the non-cross tests, Late Fusion 3 is the configuration that achieves the highest accuracy, up to 99%.

Late Fusion 3 shows the highest accuracies even in cross tests including original and YouTube data; this is probably due to the fact that YouTube’s compression is not high enough to interfere with classification. With more massive compressions, as in the case of WhatsApp, this configuration presents some problems. Early Fusion 3 also has very high accuracy values, but still lower than all those of Late Fusion 3.

The other configurations never reach the accuracy values of Late Fusion 3 and Early Fusion 3, but prove to be more efficient in dealing with data with significant compression. In particular, Late Fusion 1 has the best results in WhatsApp/original and WhatsApp/YouTube cross tests, Late Fusion 2 has the best result in YouTube/WhatsApp cross test and Early Fusion 1 has the best result in original/WhatsApp cross test. Therefore, there is not one configuration that is clearly better in cross test including WhatsApp data; it depends on the case of our interest. We emphasize the original/WhatsApp cross test improved by Early Fusion 1, because it can be traced back to different realistic scenarios (as mentioned in Section 5.3.4).

5.4 Conclusive Remarks

In this chapter we have evaluated the proposed methodology through simulations and experiments. We started by analyzing the results obtained by mono-modal methods and then moving on to those obtained by multi-modal methods. For each method we have highlighted its

strengths and weaknesses, deducing which could be the best based on a proposed scenario. Multi-modal methods performed better than mono-modal ones, constituting a great improvement in both non-cross and cross tests. Among the multi-modal methods, however, there is not one that is better than all the others; it all depends on the case of our interest.

6

Conclusions and Future Works

This thesis proposes a methodology for camera model identification related to digital video sequences. The aim is to determine the smartphone model used to acquire digital video sequences by exploiting visual and audio information from the videos themselves.

The devised methodology is based on convolutional neural networks (CNNs) capable of classifying videos by extracting suitable features from their visual and audio content. Given a video, as visual content to provide to the CNNs we use patches cropped from its video frames, while as audio content we use patches cropped from the log-mel spectrogram of its audio track. As for the networks, we use the EfficientNetB0 to classify both the visual and audio patches, while the VGGish only to classify the audio patches. The networks are appropriately modified in order to match the shape of input data and the number of smartphone models to be classified.

We propose two multi-modal camera model identification approaches: in the first one, we compare the scores individually obtained from two trained mono-modal CNNs (one only with visual patches, the other only with audio patches); in the second one, we build multi-input networks and train them with visual/audio patch pairs. For completeness, before proceeding with the resolution of the multi-modal model attribution, we investigate the mono-modal model attribution using CNNs trained only with video patches or audio patches.

The dataset from which we take videos is the Vision dataset, a recent image and video dataset, purposely designed for multimedia forensics investigations. The videos on which we experience are not only the original ones; we also use those compressed by the WhatsApp and YouTube algorithms and those further compressed in other ways (e.g., H.264 or H.265 codecs) so as to obtain many results with different configurations.

Based on the results obtained, we find that multi-modal methods are more effective than mono-modal methods and, depending on our needs, the first multi-modal method could be better or worse than the second one. We achieve very good results in non-cross tests (i.e., when training and testing sets are selected from the same dataset), reaching accuracy values up to 99% in the first multi-modal method. In cross tests (i.e., when training and testing sets belong to different datasets) the achieved accuracy depends on the kind of datasets involved and the applied compression factors.

The results obtained with the original and YouTube data are very good, both in the first and in the second multi-modal methods. This is probably due to the fact that YouTube's compression is not high enough to interfere with classification. With more massive compressions, as in the case of WhatsApp, classification becomes more challenging.

Thanks to the realization of different configurations of the two multi-modal methods, we can visualize which one gives better results according to our needs. In non-cross tests and cross tests that include only original and YouTube data, the configuration that gives the best results belongs to the first multi-modal method. It compares the scores obtained from an EfficientNetB0 trained with visual patches and an EfficientNetB0 trained with audio patches. In cross test that include WhatsApp data, however, we do not have a configuration that gives better results in all cases; we therefore prefer a configuration based on the specific result of our interest. For example, if we are interested in training a model with original data in order to classify WhatsApp data, the configuration that suits us belongs to the second multi-modal method and is based on a multi-input network that combines an EfficientNetB0 for visual patches and a VGGish for audio patches.

The results achieved with our experimental campaign highlight a series of future challenges and improvements that can be tackled as future work.

Generalization The tests have brought us good results, but we have not been able to obtain an overall model that is better than all the

others in the analysis. This leads us, in a realistic scenario, to change configuration according to the classification we want to do. Instead, the ideal would be to build a single general model that is suitable for any circumstance.

Challenging scenarios We obtained the worst results in cross tests where we consider a training set composed of more compressed data than those of the testing set, or vice versa (e.g., training set: original; testing set: WhatsApp). A possible improvement consists in the construction of a model that is able to classify data belonging to these tests well, with high accuracy values.

Different approaches A possible different approach concerns the type of frames extracted from the video sequences. In our analysis, we extracted frames equally spaced over time and distributed over the entire duration of the video considered; regardless of the frame type. We could instead work using only I-frames, or P-frames, or P-frames and B-frames together etc. This is because the compression could affect only some types of frames considerably, intervening less on the others. This could help us in creating a model that is more stable and less sensitive to data compression.

Bibliography

- [1] F. Bre, J. Gimenez, and V. Fachinotti, “Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks,” *Energy and Buildings*, vol. 158, 11 2017.
- [2] S. Bhattarai, “What is gradient descent in Machine Learning?.” <https://saugatbhattarai.com/np/what-is-gradient-descent-in-machine-learning/>, 2018. Accessed: 2021-04-07.
- [3] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks the ELI5 way.” <https://tinyurl.com/ypz3henb>, 2018. Accessed: 2021-04-07.
- [4] V. Tabora, “Photo Sensors In Digital Cameras.” <https://medium.com/hd-pro/photo-sensors-in-digital-cameras-94fb26203da1>, 2019. Accessed: 2021-04-07.
- [5] C. M. Burnett, “Profile/cross-section of a Bayer filter.” https://commons.wikimedia.org/wiki/File:Bayer_pattern_on_sensor_profile.svg, 2006. Accessed: 2021-04-07.
- [6] S. Sahoo, “Residual blocks Building blocks of ResNet.” <https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec>, 2018. Accessed: 2021-04-07.
- [7] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
- [8] D. Shullani, M. Fontani, M. Iuliani, O. A. Shaya, and A. Piva, “VISION: a video and image dataset for source identification,” *EURASIP J. Inf. Secur.*, vol. 2017, p. 15, 2017.

-
- [9] B. C. Hosler, O. Mayer, B. Bayar, X. Zhao, C. Chen, J. A. Shackelford, and M. C. Stamm, "A video camera model identification system using deep learning and fusion," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019, Brighton, United Kingdom, May 12-17, 2019*, pp. 8271–8275, IEEE, 2019.
- [10] M. Kirchner, "Efficient estimation of CFA pattern configuration in digital camera images," in *Media Forensics and Security II, part of the IS&T-SPIE Electronic Imaging Symposium, San Jose, CA, USA, January 18-20, 2010, Proceedings* (N. D. Memon, J. Dittmann, A. M. Alattar, and E. J. Delp, eds.), vol. 7541 of *SPIE Proceedings*, p. 754111, SPIE, 2010.
- [11] B. K. Gunturk, J. W. Glotzbach, Y. Altunbasak, R. W. Schafer, and R. M. Mersereau, "Demosaicking: color filter array interpolation," *IEEE Signal Process. Mag.*, vol. 22, no. 1, pp. 44–54, 2005.
- [12] A. Fischer and T. Gloe, "Forensic analysis of interdependencies between vignetting and radial lens distortion," in *Media Watermarking, Security, and Forensics 2013, Burlingame, CA, USA, February 5-7, 2013, Proceedings* (A. M. Alattar, N. D. Memon, and C. Heitzenrater, eds.), vol. 8665 of *SPIE Proceedings*, p. 86650D, SPIE, 2013.
- [13] G. Xu and Y. Shi, "Camera model identification using local binary patterns," in *Proceedings of the 2012 IEEE International Conference on Multimedia and Expo, ICME 2012, Melbourne, Australia, July 9-13, 2012*, pp. 392–397, IEEE Computer Society, 2012.
- [14] A. Tuama, F. Comby, and M. Chaumont, "Camera model identification with the use of deep convolutional neural networks," in *IEEE International Workshop on Information Forensics and Security, WIFS 2016, Abu Dhabi, United Arab Emirates, December 4-7, 2016*, pp. 1–6, IEEE, 2016.
- [15] B. Bayar and M. C. Stamm, "Augmented convolutional feature maps for robust cnn-based camera model identification," in *2017 IEEE International Conference on Image Processing, ICIP 2017, Beijing, China, September 17-20, 2017*, pp. 4098–4102, IEEE, 2017.
- [16] M. Kirchner and T. Gloe, "Forensic camera model identification," in *Handbook of Digital Forensics of Multimedia Data and Devices*, p. 329374, Wiley-IEEE Press, 2015.

-
- [17] IBM Cloud Education, “Neural networks.” <https://www.ibm.com/cloud/learn/neural-networks>, 2020. Accessed: 2021-04-07.
- [18] K. Gurney, *An Introduction to Neural Networks*. An Introduction to Neural Networks, Taylor & Francis, 1997.
- [19] E. Stevens, L. Antiga, and T. Viehmann, *Deep Learning with PyTorch*. Manning Publications, 2020.
- [20] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, “A Convolutional Neural Network for Modelling Sentences,” *CoRR*, vol. abs/1404.2188, 2014.
- [21] R. Ramanath, W. E. Snyder, Y. Yoo, and M. S. Drew, “Color image processing pipeline,” *IEEE Signal Process. Mag.*, vol. 22, no. 1, pp. 34–43, 2005.
- [22] P. Pedersen, “The Mel Scale,” *Journal of Music Theory*, vol. 9, no. 2, pp. 295–308, 1965.
- [23] L. Roberts, “Understanding the Mel Spectrogram.” <https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53>, 2020. Accessed: 2021-04-07.
- [24] A. M. Rafi, T. I. Tonmoy, U. Kamal, Q. J. Wu, and M. K. Hasan, “RemNet: remnant convolutional neural network for camera model identification,” *Neural Computing and Applications*, pp. 1–16, 2020.
- [25] H. Yao, T. Qiao, M. Xu, and N. Zheng, “Robust Multi-Classifer for Camera Model Identification Based on Convolution Neural Network,” *IEEE Access*, vol. 6, pp. 24973–24982, 2018.
- [26] A. Swaminathan, M. Wu, and K. J. R. Liu, “Component forensics,” *IEEE Signal Process. Mag.*, vol. 26, no. 2, pp. 38–48, 2009.
- [27] C.-H. Choi, J.-H. Choi, and H.-K. Lee, “Cfa pattern identification of digital cameras using intermediate value counting,” in *Proceedings of the thirteenth ACM multimedia workshop on Multimedia and security*, pp. 21–26, 2011.
- [28] D. Menon and G. Calvagno, “Color image demosaicking: An overview,” *Signal Process. Image Commun.*, vol. 26, no. 8-9, pp. 518–533, 2011.

- [29] J. Yu, S. Craver, and E. Li, "Toward the identification of DSLR lenses by chromatic aberration," in *Media Forensics and Security III, San Francisco Airport, CA, USA, January 24-26, 2011, Proceedings* (N. D. Memon, J. Dittmann, A. M. Alattar, and E. J. D. III, eds.), vol. 7880 of *SPIE Proceedings*, p. 788010, SPIE, 2011.
- [30] A. E. Dirik, H. T. Sencar, and N. D. Memon, "Digital single lens reflex camera identification from traces of sensor dust," *IEEE Trans. Inf. Forensics Secur.*, vol. 3, no. 3, pp. 539–552, 2008.
- [31] Z. Deng, A. Gijsenij, and J. Zhang, "Source camera identification using auto-white balance approximation," in *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011* (D. N. Metaxas, L. Quan, A. Sanfeliu, and L. V. Gool, eds.), pp. 57–64, IEEE Computer Society, 2011.
- [32] T. H. Thai, F. Reirant, and R. Cogranne, "Camera model identification based on the generalized noise model in natural images," *Digit. Signal Process.*, vol. 48, pp. 285–297, 2016.
- [33] R. Caldelli, I. Amerini, and F. Picchioni, "A dft-based analysis to discern between camera and scanned images," *Int. J. Digit. Crime Forensics*, vol. 2, no. 1, pp. 21–29, 2010.
- [34] O. Mayer and M. C. Stamm, "Learned forensic source similarity for unknown camera models," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*, pp. 2012–2016, IEEE, 2018.
- [35] Y. Chen, Y. Huang, and X. Ding, "Camera model identification with residual neural network," in *2017 IEEE International Conference on Image Processing, ICIP 2017, Beijing, China, September 17-20, 2017*, pp. 4337–4341, IEEE, 2017.
- [36] Q. Shu, J. Ni, and H. Xie, "Decoupling-gan for camera model identification of JPEG compressed images," in *IEEE International Conference on Multimedia and Expo, ICME 2020, London, UK, July 6-10, 2020*, pp. 1–6, IEEE, 2020.
- [37] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June*

- 2019, Long Beach, California, USA (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114, PMLR, 2019.
- [38] P.-L. Pröve, “MobileNetV2: Inverted Residuals and Linear Bottlenecks.” <https://tinyurl.com/be6fmfr3>, 2018. Accessed: 2021-04-07.
- [39] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. J. Weiss, and K. Wilson, “Cnn architectures for large-scale audio classification,” in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 131–135, 2017.
- [40] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *arXiv e-prints*, p. arXiv:1409.1556, Sept. 2014.
- [41] L. Shi, K. Du, C. Zhang, H. Ma, and W. Yan, “Lung Sound Recognition Algorithm based on VGGish-BiGRU,” *IEEE Access*, vol. 7, pp. 139438–139449, 2019.
- [42] T. Chen and U. Gupta, “Attention-based convolutional neural network for audio event classification with feature transfer learning,” 2018.
- [43] D. Kim, “Normalization methods for input and output vectors in backpropagation neural networks,” *International Journal of Computer Mathematics*, vol. 71, no. 2, pp. 161–171, 1999.
- [44] S. Mandelli, N. Bonettini, P. Bestagini, and S. Tubaro, “Training cnns in presence of JPEG compression: Multimedia forensics vs computer vision,” *CoRR*, vol. abs/2009.12088, 2020.
- [45] D. Cozzolino and L. Verdoliva, “Noiseprint: A cnn-based camera model fingerprint,” *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 144–159, 2020.
- [46] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still image data compression standard*. Springer Science & Business Media, 1992.
- [47] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*

- (*CVPR 2009*), 20-25 June 2009, Miami, Florida, USA, pp. 248–255, IEEE Computer Society, 2009.
- [48] J. F. Gemmeke, D. P. W. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, “Audio set: An ontology and human-labeled dataset for audio events,” in *Proc. IEEE ICASSP 2017*, (New Orleans, LA), 2017.
- [49] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, “Albumentations: Fast and flexible image augmentations,” *Inf.*, vol. 11, no. 2, p. 125, 2020.
- [50] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox, and R. Garnett, eds.), pp. 8024–8035, 2019.
- [51] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the H.264/AVC video coding standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.
- [52] P. Yakaiah and C. Sheka, “OVERVIEW OF H. 265,” *Technology*, vol. 8, no. 8, pp. 48–54, 2017.
- [53] S. Tomar, “Converting video formats with FFmpeg,” *Linux J.*, vol. 2006, p. 10, June 2006.
- [54] J. Sterne, *MP3: The meaning of a format*. Duke University Press, 2012.