POLITECNICO DI MILANO

Corso di Laurea Magistrale in Computer Science and Engineering

Dipartimento di Elettronica, Informazione e Bioingegneria

JPEG-BASED FORENSICS THROUGH CONVOLUTIONAL NEURAL NETWORKS

Image & Sound Processing Group (ISPG)

Relatore: Prof. Paolo BESTAGINI

Correlatore: Ing. Luca BONDI

Tesi di Laurea di:

Nicolò BONETTINI, matricola 835569

Anno Accademico 2016-2017

Abstract

Due to the wide diffusion of JPEG coding standard, the forensic community has devoted significant attention to the development of image authenticity detectors based on the analysis of JPEG traces. Given the trend recently gained by convolutional neural networks (CNN) in many computer vision tasks, in this thesis we propose to use CNNs for JPEG-based forensics.

Among the many JPEG-based forensic problems, one of the most studied is double JPEG (DJPEG) compression detection. Indeed, the ability of detecting whether an image has been compressed once or twice provides paramount information toward image authenticity assessment. For this reason, we focus on exploring the capability of CNNs to capture DJPEG artifacts directly from images. Results show that the proposed CNN-based detectors achieve good performance in situations considered particularly challenging in the literature: i) analysis of small sized images (i.e., 64×64); ii) analysis of the case in which the first JPEG compression has higher quality than the second one.

To further explore the flexibility of CNNs for JPEG-based forensic tasks, we also investigated two additional related problems. We considered quality factor estimation and software identification to this extent. Results show that satisfactory accuracies can be reached despite the challenging nature of these problems.

Sommario

Data la grande diffusione dello standard di codifica JPEG, negli ultimi anni la comunità forense ha dedicato una sempre crescente attenzione allo sviluppo di tecniche per l'analisi delle tracce da essa lasciate. Al contempo, si è anche assistito ad un forte slancio nell'impiego di reti neurali convolutive (CNN) in vari ambiti di computer vision. Per i suddetti motivi, in questa tesi proponiamo l'uso di CNN per problemi forensi legati alla compressione JPEG.

Fra i tanti casi, uno dei più affrontati in letteratura è quello di determinare se un'immagine sia stata sottoposta ad una doppia compressione (DJPEG). Tale informazione è infatti di grande aiuto nello stabilire l'autenticità della stessa. Lo scopo di questo lavoro è di esplorare le capacità delle CNN di catturare le tracce lasciate dalla doppia compressione partendo direttamente dall'immagine. I risultati mostrano che i modelli proposti raggiungono buone performance anche in situazioni considerate particolarmente difficili dalla letteratura: i) analisi di immagini di piccole dimensioni (i.e., 64×64); ii) analisi nel caso in cui la prima compressione avvenga ad una qualità maggiore della seconda. Per esplorare la flessibilità delle CNN in questo campo, abbiamo investigato due problemi minori. Abbiamo considerato la stima del fattore di qualità e l'identificazione del software adoperato per la compressione. I risultati mostrano come si possano raggiungere accuratezze soddisfacenti nonostante l'ardua natura di questi problemi.

Ringraziamenti

Per quanto una paginetta risicata non basti a ringraziare adeguatamente tutti quelli che se lo meritano, proveró a fare del mio meglio.

Grazie a mio padre, mia madre, mia sorella e mio cognato per avermi dato la possibilità di raggiungere questo obiettivo e per avermi sempre sostenuto in questi anni. Grazie a tutti i nonni, in particolare a Narciso che avrebbe tanto voluto esserci in questo giorno.

Grazie ai miei amici di una vita perchè mi danno motivo per tornare a casa, ricordandomi l'importanza delle radici. Fortunatamente per me, siete veramente troppi per essere menzionati singolarmente. Scrivo solo Nanni che altrimenti si offende.

Grazie ad Enrico, Pietro e Simone, la mia famiglia milanese.

Grazie a tutta la combriccola ISPG, in particolare a Sara, Silvia ed Enzo per i consigli dispensati.

Grazie a Paolo e Luca, per avermi accolto, sopportato e soprattutto insegnato tanto in questi mesi.

Contents

List of Figures

List of Tables

1.

Introduction

In the last decades, due to the wide availability of easy-to-use imaging software and inexpensive hardware devices that enable the acquisition of visual data, digital images can be readily created, stored, transmitted, modified and tampered with. Moreover, picture duplication is a quite straightforward procedure, and the storage of copies on reliable physical devices has become rather inexpensive. As a consequence, the diffusion of tampered content has become a widespread phenomenon. Not unexpectedly, with the advent of social networks and social media, a growing need to discern between original and tampered content started to interest people in everyday life.

Fortunately, when an image is edited through non-reversible operations, it is possible to partly reconstruct information about its past history for authenticity detection. Indeed, the history of an edited digital image can be described in terms of complex information processing chains, whereby each processing operator alters the underlying features of the content in a characteristic and detectable manner (e.g., compression may leave peculiar blocking artifacts). It is then possible to blindly analyze an image to search for characteristic footprints that confirms the use of specific editing operations.

Among the techniques developed by the image forensic community to

fight tampering trend exposing these footprints [5, 6], great attention has been devoted to methods analyzing JPEG traces [7, 8]. Indeed, every time an image is stored (e.g., at shooting time directly on the acquisition device, or after editing with processing tools), it is usually saved in JPEG format. Therefore, manipulated content often undergoes JPEG re-compression, and its detection provides paramount information in terms of image authenticity.

The goal of this thesis is to develop a series of JPEG traces detectors to solve forensic problems, based on the use of Convolutional Neural Networks (CNNs). As a matter of fact, Convolutional Neural Networks were employed for imaging tasks since the late 1980s [9, 10], but due to the lack of computational power were never exploited properly. Today we are experiencing a golden age for what concerns computing resources. In particular, the use of GPUs as general computation source (GP-GPUs) has proved exceptionally efficient with CNNs. The GPU's ability to process large amount of data in parallel allows fast CNN training over massive datasets in reasonable time, making Deep Learning one of the most fascinating technologies to explore in the near future. Nonetheless, CNNs have only been exploited for JPEG-based forensics in some preliminary studies [2], thus leaving room to further research on the topic.

JPEG-based forensic detectors can be roughly split into two categories: i) detectors aiming to assess the presence of JPEG compression (e.g., discriminating uncompressed from compressed images, detecting the number of subsequent compressions applied) [11, 12]; ii) methods aiming at estimating some parameters of the applied JPEG compression (e.g., estimation of the used quantization matrix) [13, 14]. Within this framework, great part of our work is specifically focused on the first class of detectors considering the problem of double JPEG compression detection (i.e., understanding whether

an image has been compressed once or twice). Then, we analyze two different cases of study related to the second class of detectors: i) software identification (i.e., which software has been used to apply JPEG compression), and; ii) quality factor estimation (i.e., which is the quality of the applied JPEG compression). For all these problems, we analyze different CNN architectures characterized by different peculiarities.

Double JPEG compression detection has received great attention in image forensics, and presence of tampering is often revealed by looking for the artifacts left by JPEG re-compression. However, depending on whether second JPEG compression grid is aligned or not with the one adopted by the first compression, different artifacts are introduced. For this reason, these two scenarios are often analyzed separately and are commonly referred to as aligned double JPEG (A-DJPEG) compression detection [3] and non aligned double JPEG (NA-DJPEG) compression detection [4, 15], respectively. Detectors proposed in this work are able to compete with state-of-the-art algorithms in the A-DJPEG and NA-DJPEG scenarios separately, and to show good results in mixed A and NA-DJPEG scenario.

In many cases, manipulation takes place on limited parts of the image only. Therefore DJPEG traces are only left on a limited number of pixels. For this reason, being able to detect DJPEG on small image patches proves paramount for localization of manipulated regions in image forgery detection problems. However, most of the techniques performing double JPEG detection in literature focus on estimating compression history of an image as a whole, whereas the localization of double compressed regions of relatively small size (i.e., possibly tampered regions) has been often overlooked and only addressed in some works [2, 3]. For this reason, all our DJPEG detectors are developed to work in the challenging scenario of small images

(i.e., 64×64 pixel).

Besides DJPEG, compression is applied by means of a software and it involves the adoption of a precise quantization table, or quality factor. Being able to identify which software performed a compression on an image, or which quality factor was used can provide additional insights during compression history investigation. We therefore examine quality factor estimation and software identification as cases of study in order to prove the flexibility and the capability of CNNs in JPEG forensic field. To the best of our knowledge, literature shows no approaches involving CNN for these two particular problems. The proposed models reach satisfactory accuracies despite the challenging nature of the problems.
The thesis is structured as follows.

In Chapter 2 we provide a background on JPEG compression and on Convolutional Neural Networks. Furthermore, we report the current state-of-the-art for JPEG forensics and for the use of CNN in JPEG forensics.

In Chapter 3 we formalize the three problems tackled in this thesis (i.e., DJPEG detection, software identification and quality factor estimation). We briefly report the motivation behind them and we define the respective hypothesis space for all of them.

In Chapter 4 we describe the systems we designed to tackle the aforementioned problems. In particular we explain all the investigated network architectures in details.

In Chapter 5 we report all the performed experiments. We focus on dataset construction, pipeline design and we give an interpretation to the obtained results.

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

2.

State of the Art

In this Chapter we introduce the main background concepts and state-of-the-art algorithms needed to understand the rest of the work. First, we provide some background explanation on how JPEG compression works. We then review state of the art techniques developed by the image forensic community to expose JPEG traces. Then we move on convolutional neural networks, the most peculiar technique we adopted in our research. Finally we give a brief overview of the state of the art involving the use of convolutional neural networks in forensics tasks.

2.1 JPEG Compression

The JPEG format is a broadly used method of lossy image compression. It achieves a good tradeoff between compression and perceptual quality loss, therefore it has emerged as a near universal image standard [16, 17]. Given a three channel color image in the Red Green Blue (RGB) color space, JPEG compression proceeds as follows. The image is first transformed into luminance/chrominance space (YC_bC_r). The two chrominance channels (C_b,C_r) can be subsampled by a factor of two relative to the luminance channel (Y).

Each channel is divided in 8×8 pixel non-overlapped blocks. These values are converted from unsigned to signed integers (e.g., from [0, 255] to [128, 127]).

For each channel c, let us denote a 8×8 block obtained from it as $\boldsymbol{f}^c$, or simply $\boldsymbol{f}$ for the sake of simplicity. Each block $\boldsymbol{f}$ is converted to a 8×8 block in frequency space $\boldsymbol{F}$, using two-dimensional discrete cosine transform (DCT) as:

$$\boldsymbol{F}_{c_1,c_2} = \sum_{i=0}^{7} \sum_{j=0}^{7} \boldsymbol{f}(i,j)\, \boldsymbol{H}_{c_1,c_2}(i,j), \tag{2.1}$$

where $\boldsymbol{H}_{c_1,c_2}$ is the base at (c_1, c_2) frequency of the 8×8 DCT, whose entries are defined as:

$$\boldsymbol{H}_{c_1,c_2}(i,j) = \alpha(c_1,c_2) \cos\left[\frac{\pi}{8}\left(i+\frac{1}{2}\right)c_1\right] \cos\left[\frac{\pi}{8}\left(j+\frac{1}{2}\right)c_2\right], \tag{2.2}$$

where $\alpha(c_1, c_2)$ is a normalization constant and $i, j, c_1, c_2 \in [0, 7]$. Depending on the specific frequency (c_1,c_2) and channel c, each DCT coefficient of $\boldsymbol{F}$ is quantized with quantization step $\boldsymbol{q}_{c_1,c_2}$, thus obtaining:

$$\hat{\boldsymbol{F}}_{c_1,c_2} = \text{round}\left(\frac{\boldsymbol{F}_{c_1,c_2}}{\boldsymbol{q}_{c_1,c_2}}\right). \tag{2.3}$$

This stage is the primary source of entropy reduction and information loss, since quantization is not invertible and it does not allow a precise reconstruction of $\boldsymbol{F}$. The primary source of variation in JPEG encoders is the choice of quantization values $\boldsymbol{q}$, Equation (2.3). Quantization is specified as a set of two 8×8 tables, one for luminance, one for chrominance. After quantization, the DCT coefficients undergo to entropy encoding (typically Huffman coding). Huffman coding is a variable-length encoding scheme that encodes frequently occurring values with shorter codes, and less frequently occurring values with longer codes. It is worth pointing out that this operation is lossless and has the only purpose to pack the quantized coefficients into a

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Figure 2.1: Example quantization tables for luminance (left) and chrominance (right) components provided in the informative sections of the standard.

bitstream. A schematic representation of the JPEG compression pipeline is shown in Figure 2.2.

Camera vendors and software engineers are free to balance quality and compression to their own tastes, since the JPEG standard does not enforce any specific quantization table $\boldsymbol{q}$ or Huffman probability tables. Nevertheless, Annex K of the standard reports a table for the luminance component and a table for the two chrominance components, as shown in Figure 2.1. These tables were obtained from a series of of psychovisual experiments and are known to offer reasonable tradeoff for natural images over a wide variety of applications and viewing conditions. Hence they have been widely accepted and over the years have become known as the "default" quantization tables.

In many applications there is a need to provide flexibility to adjust image quality by changing the overall bit rate. In practice, scaled version of the "default" quantization tables are commonly adopted to vary quality and compression performance of JPEG. This adjustment is done through the use of a *quality factor* QF for scaling all the elements of quantization tables,

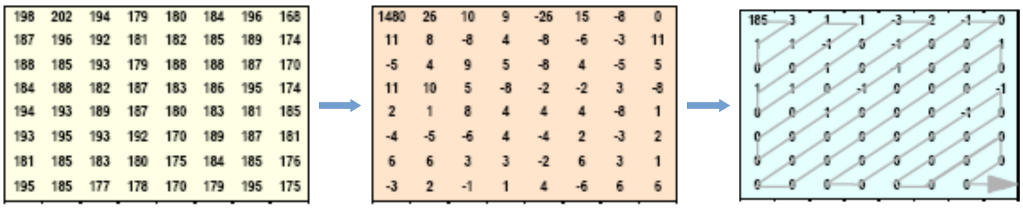


Figure 2.2: JPEG compression pipeline.

according to the rule

$$\text{scale factor} = \begin{cases} \frac{5000}{\text{QF}} & \text{for } 1 \leq \text{QF} < 50 \\ 200 - 2 \cdot \text{QF} & \text{for } 50 \leq \text{QF} \leq 99 \cdot \\ 1 & \text{for QF} = 100 \end{cases} \tag{2.4}$$

The decompression procedure is a matter of going backward in the pipeline described above. Quantization tables $\boldsymbol{q}$ are stored in the JPEG header, as well as information on the adopted entropy encoding. The DCT coefficients of each channel are first re-arranged as a 8×8 table and then multiplied by the coefficients of $\boldsymbol{q}$. As aforementioned, the rounding operation performed in Equation (2.3) is not invertible, therefore we have an unavoidable data loss in reconstructing $\boldsymbol{F}$. The last step is computing the inverse of the two-dimensional DCT, which outputs the reconstructed 8×8 blocks in (YC_bC_r) domain.

2.2 JPEG Forensics

The history of a digital image can be represented as a composition of several steps. As an example, an image can be acquired using a given camera, then resized to meet some specific requirements, edited to adjust colors or brightness, compressed to save space on disk and finally uploaded to a web

sharing platform, before some other users download it to further process it. The idea behind blind image forensics is that each non-reversible editing step applied to an image inevitably leaves peculiar footprints. By detecting these footprints it is possible to trace back image history, e.g., to possibly expose forgeries [6].

Lossy JPEG image compression is one of the most common operations performed on digital images. This is due to the convenience of handling smaller amounts of data to store and/or transmit. Indeed, most digital cameras compress each picture directly after taking a shot. Due to its lossy nature, image coding leaves characteristic footprints, which can be detected. Although revealing coding-based footprints in digital images is in itself relevant, these traces are fundamentally a powerful tool for detecting forgeries. For this reason, the forensic community has developed a series of algorithms to extract JPEG-based traces from images under different conditions. In the following, we review some of the most relevant JPEG-based forensic methods.

2.2.1 Compression Detection

Let us consider the scenario in which a digital image is available in the pixel domain in raw format, without any knowledge about prior processing. JPEG compression detection is the problem of detecting whether that image is actually uncompressed, or it has been previously compressed and which were the compression parameters being used. This is useful to avoid fake-bitrate frauds, in which someone sells an image as high-quality uncompressed one, even if the image was previously compressed.

The underlying idea of forensic methods coping with the problem of JPEG compression detection is that the block-based image coding leaves characteristics compression traces both in the pixel and in the transform domain [6].

In pixel domain, block based image coding scheme introduces blockiness. Several proposed methods aim to estimate this blockiness. Authors in [13, 11] propose an algorithm based on the idea that if the image has not been compressed, the pixel differences across 8×8 block boundaries should be similar to those within blocks. Then, it is possible to build two functions, Z' and Z'', taking into account inter and intrablock pixel differences. The presence of prior compression is deduced if the energy of the difference between the histograms of Z' and Z'' is higher than a threshold. In [12], the authors model a blocky image as a nonblocky image interfered with a pure blocky signal. Then, the estimation of blockiness in a blind way is turned into the problem of evaluating the power of the blocky signal without accessing the original image. In order to achieve this goal, the absolute value of the gradient between each column or row of the image is computed separately. The power of the blocky signal can be estimated in order to reveal its presence.

In transform domain, block based image coding schemes modify the histogram of transformed coefficients. Several proposed methods analyze the shape of the histogram in order to detect traces of JPEG compression. In [14], the authors derive a method based on the observation that in a JPEG-compressed image, the integral of the DCT coefficients histogram in the range $(1, +1)$ is greater than the integral in the range $(-2, -1] \cup [+1, +2)$, with quantization steps that are equal to or larger than 2. By examining, as feature, the ratio between the first and the second integral, it is possible to verify that its value, in case of JPEG-compressed image, will be close to zero, and it would be much smaller than that of the corresponding uncompressed one. So, JPEG compression is detected when the ratio is smaller than a given threshold.

Once verified the presence of JPEG compression, another interesting task

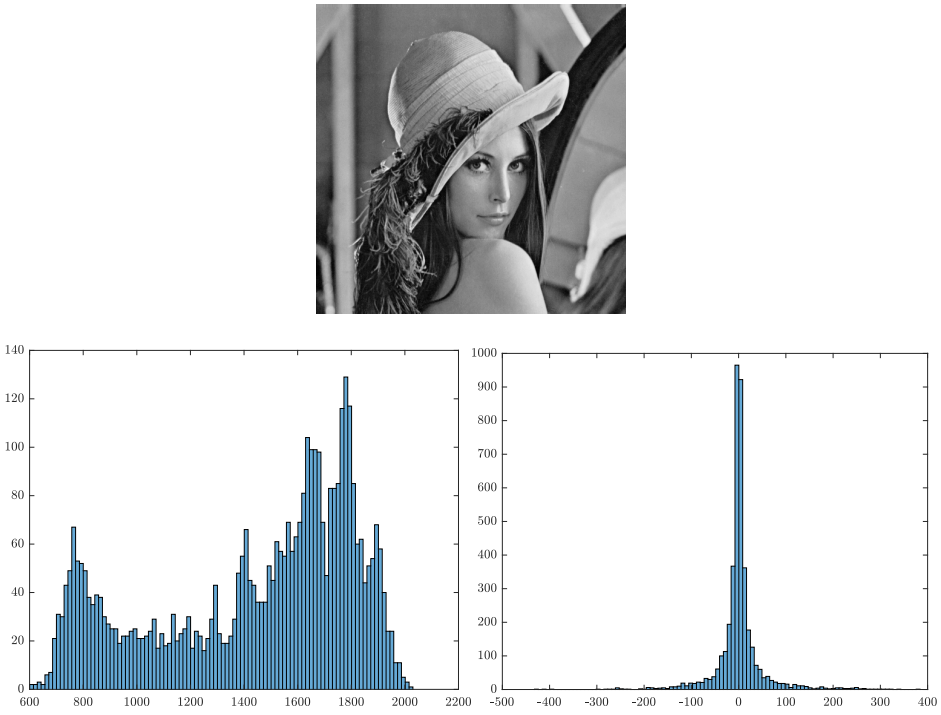


Figure 2.3: Histograms of DC coefficients (left) and $AC_{0,1}$ coefficients (right) of the top image.

is the estimation of quantization step QF (or the whole quantization table $\boldsymbol{q}$). Most of the methods proposed in literature exploit the fact that the histogram of the DCT coefficients has a comb-like shape, and the spacing between consecutive peaks is related to the adopted quantization step size. Furthermore, it is well-known that the histogram envelope of DC coefficients (i.e., the first DCT coefficient $\hat{\boldsymbol{F}}_{0,0}$, Equation (2.3)) can be approximated by means of a Gaussian or uniform distribution, whereas the histogram envelope of each AC coefficient (i.e., the other 63 DCT coefficients) can be approximated by means of a Laplacian distribution, as shown in Figure 2.3

In particular, the method proposed in [13, 11] leverages this information and estimates the quality factor through a Maximum Likelihood approach. In [14], the authors propose a scheme to identify whether a image has been

previously JPEG compressed, estimate the quantization step and detect the whole quantization table. The key idea is that when a JPEG image is reconstructed in the pixel domain, pixel values are rounded to integers. As a consequence, the histograms of DCT values computed from decoded pixel values are not exactly comb-shaped, but they are blurred with respect to the ones computed directly after the quantization. In this way, is possible to compute the quantization step for each DCT frequency by looking at peak distances in these rounded histograms.

2.2.2 Double Compression Detection

The aforementioned solutions aim at detecting whether an image is uncompressed, or has been JPEG compressed once. However, images are often JPEG compressed once at photo inception directly by the acquisition device. When they are edited with whathever software suite (e.g., PhotoShop, GIMP, etc.) and saved, they often undergo a second compression. Being able to estimate whether an image has been JPEG compressed once or twice proves then paramount as forgery indicator. Specifically, we refer to this problem as double JPEG (DJPEG) compression detection.

We can distinguish between two separate kinds of double JPEG compression. If no operations are applied between the two compression steps, 8×8 JPEG blocks of the first and second compressions are perfectly aligned, thus we speak of A-DJPEG compression. Conversely, when the second compression 8×8 grid is shifted with respect the previous one (e.g., due to cropping between first and second compression or to a cut and paste operation), we have a non aligned (NA-DJPEG) compression.

Concerning A-DJPEG detection, a promising method is proposed in [18]. It is proposed to detect the presence of double-aligned JPEG compression

by observing that consecutive quantizations introduce periodic artifacts into the histogram of DCT coefficients. These periodic artifacts are visible in the Fourier domain as strong peaks in medium and high frequencies and are defined as double quantization (DQ) effect. These peaks in the histogram assume different configurations according to the relationship between the quantization steps of the first and of the second compression. Given a JPEG file with quality factor QF2, to decide if the file was previously JPEG compressed with a different quality factor QF1, the approach proposed in [18] works as follows: as the first step, the histograms of absolute values of all analyzed DCT coefficients are computed from the image under investigation I. The image is then cropped (in order to disrupt the structure of JPEG blocks) and compressed with a set of candidate quantization tables representing various quality factors. The cropped and compressed images are then recompressed using QF2; finally, the histograms of absolute values of DCT coefficients from the double-compressed cropped images are computed. The estimator chooses the quantization table such that the resulting histogram is as close as possible to that obtained from the image I.

Detecting NA-DJPEG is indeed very interesting in all those cases in which a portion of an image could be copy-pasted into another image. In such a scenario, there is a high probability that the pasted portion does not match the existing JPEG grid (63/64 if we consider all the possible shift among the 8×8 grid). In [4, 15], authors propose a threshold method to detect and even estimate the region affected by not aligned double compression. The main idea behind the method detecting NA-DJPG compression by measuring how DCT coefficients cluster around a given lattice (defined by the JPEG quantization table) for any possible grid shift. When NA-DJPG is detected, the parameters of the lattice also give the primary quantization table.

2.2.3 Multiple Compression Detection

JPEG multiple compression detection is the problem of detecting whether a JPEG image has been compressed more than twice. Some of the proposed solutions analyze the statistics of DCT coefficients. Although the problem of multiple compression detection is relatively new in multimedia forensics, several works on double compression detection have been presented in literature [19, 20]. The idea is to exploit DCT coefficients histogram to obtain a descriptor based on the First Digits law (Benford's law). Descriptors are then used to train a classifier, typically a SVM, to distinguish between single compression and multiple compressions.

2.2.4 Forgery Localization

Working on detection with small portion of the original images can be very useful for localization purposes. Localization aims at discover which portion of the image has been tampered with, i.e., precisely identify where the forgery is within the image area. Once built a working methodology for detection on small blocks, the underlying idea is to divide the investigated image into smaller patches and to carry out a probability scoring for each patch [21, 22]. The aggregation of the various probabilities is used to produce a heatmap representing the likelihood of each region to be tampered with. Figure 2.4 shows the result of the method depicted in [21]. The outcome of the algorithm can be further refined by considering a color segmentation of the original image and its overlapping with the tampered region, Figure 2.5. In this way the analysis of the tampered area is restricted to a meaningful region, instead of merely indicate the approximated tampered parts.

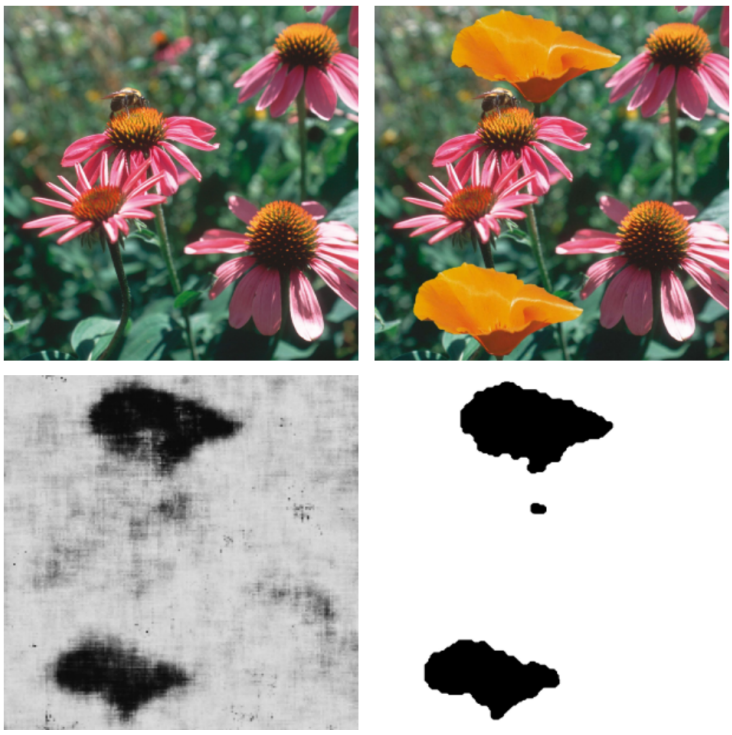


Figure 2.4: Example of blockwise tamper detection. Top-left: original image; top-right: tampered image; bottom-left: tampering map; bottom-right: tampered region identified by the blockwise detector.

Figure 2.5: Example of regionwise tamper detection. Left: output of the segmentation algorithm; right: output of the tamper detector.

2.3 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a special type of multi-layer neural network used in deep learning. They first appeared in the late 1980's with handwritten zip code recognition [9], but they recently gained significant attention in the computer vision and machine learning communities due to the spreading adoption of GPUs as general-purpose sources of computation over the last few years. CNN are indeed very expensive in terms of computational power as they exploit very high dimensionality in data to adaptively learn classification features.

Every CNN employs a set of common layers, whose typical concatenation is depicted in Figure 2.6:

- *Convolution*: each convolution layer is a bank of filters h. Given an input signal x, the output of each filter is the valid part of the linear convolution computed with stride S (i.e., $y = \text{conv}_S(x, h)$) and it is known as *feature map*. The output of the layer is obtained stacking all the feature maps obtained through different filters h.

- *Pooling*: this layer downsamples the input x by sliding a small window over it. A common choice is to keep the maximum value for each window position (max-pooling). Following the same idea, also min-pooling and average-pooling layers can be constructed.

- *Activation*: acts in a non-linear fashion on the input values. Rectified Linear Unit (ReLU) applies the rectification function $\max(0, x)$ to the input x, thus truncating negative values to zero [23]. This is one of the possible way to add non-linear behavior to the network in addition to sigmoids and hyperbolic tangents among others.

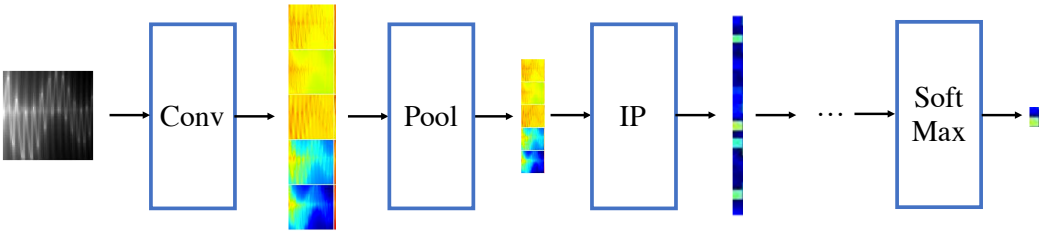


Figure 2.6: Example of simple CNN architecture composed by some of the principally used layers. A convolution layer (Conv) processes the input image with a series of linear filters. A pooling layer (Pool) downsamples each filtered image. Inner product (IP) linearly combines all its input data before applying a non-linear transformation. After a few other layers, SoftMax normalizes its input to sum to 1.

These three layers represent the convolutional part of a CNN which have the purpose of focusing on important patterns in data. It is not uncommon to come across a chain of several convolution→ pooling→ activation layers, in order to reduce the dimensionality of the input, Figure 2.7.

The following layers are usually employed atop of the convolutional part:

- *Inner Product*: this is a fully-connected layer that performs a set of linear combinations of all samples of the input x. Typically inner product layers also apply some non-linearity at the end (e.g., ReLU).

- *SoftMax*: this normalizes the input values in the range $[0, 1]$ and guarantees that they sum up to one. This is particularly useful at the end of the network in order to interpret its outputs as probability values.

CNN are known to deliver good results when working with images [10]. Firstly, they have built-in invariance with respect to translation and local distortion of the inputs. Before being sent to the fixed-sized input level of a neural network, images (like generic 2D or 1D signals) must be size-normalized and centered in the input field. This cause variations in the posi-

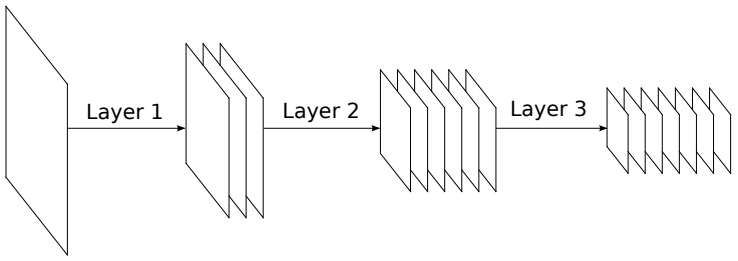


Figure 2.7: Qualitative representation of feature maps transformation.

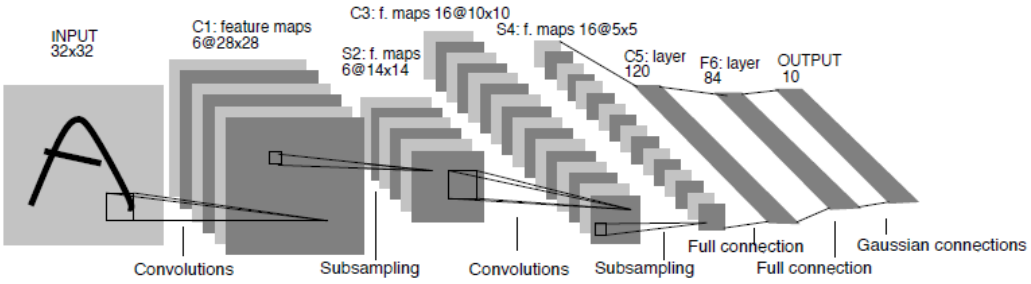


Figure 2.8: Convolutional architecture for characters recognition proposed in [1].

tion of distinctive features in input objects, complicating a lot the learning of a generic fully-connected network which would require the replication of input data to cover every possible variation in space. CNN are employed since shift invariance is automatically obtained by forcing the replication of weight configurations across space. Secondly, in fully connected architectures, the topology of the input is completely ignored. Images have a strong 2D local structure: spatially nearby pixels are in fact highly correlated. CNN force the extraction of local features by restricting the receptive fields of hidden units to be local, and focus on recognition of spatial objects only in a second time.

CNN combine three architectural ideas to ensure some degree of shift and distortion invariance: local receptive fields, shared weights, and, generally, spatial subsampling.

A very well-known and studied network for character recognition [1] is shown in Figure 2.8. The input image has been approximately size-normalized and centered during preprocessing. Each unit of a layer receives inputs from a set of units located in a small neighborhood in the previous layer. With local receptive fields, neurons can extract elementary visual features such as oriented edges, end-points, corners, and having this features combined in the higher levels. Another interesting characteristic of CNN is that elementary feature detectors that are useful on one part of the image are likely to be useful across the entire image. This knowledge can be applied by forcing a set of units whose receptive fields are located at different places on the image to have identical weight vectors. The outputs of such a set of neurons constitute a *feature map*. At each position, different types of units in different feature maps compute different types of features. A sequential implementation of this, for each feature map, would be to scan the input image with a single neuron that has a local receptive field and to store the states of this neuron at corresponding locations in the feature map. This operation is equivalent to a convolution with a small-size kernel, followed by a squashing function. The process can be performed in parallel by implementing the feature map as a plane of neurons that share a single weight vector. Units in a feature map are constrained to perform the same operation on different parts of the image. A convolutional layer is usually composed of several feature maps (with different weight vectors), so that multiple features can be extracted at each location.

More specifically, the analytical expression of the convolution within the CNN architecture is given in:

$$\boldsymbol{h}_j^{(n)} = \sum_{k=1}^{K} \boldsymbol{h}_k^{(n-1)} * \boldsymbol{w}_{kj}^{(n)} + b_j^{(n)}, \tag{2.5}$$

where $\boldsymbol{h}_j^{(n)}$ is the j^{th} feature map output in the hidden layer $h^{(n)}$, $\boldsymbol{h}_k^{(n-1)}$ is the $k^{(th)}$ channel in the hidden layer $h^{(n-1)}$, $\boldsymbol{w}_{kj}^{(n)}$ is the $k^{(th)}$ channel in the j^{th} filter in $h^{(n)}$ and $b_j^{(n)}$ is its corresponding bias term.

The filter coefficients are generally initialized with random numbers, and are learned during the training via a process known as *backpropagation* [9]. This iterative algorithm alternates between feedforward and backpropagation passes of the data, and its ultimate goal is to minimize the average loss between the actual labels (the ground truth) and the network outputs:

$$L = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{c} y_i^{*(k)} \log y_i^{(k)}, \tag{2.6}$$

where $y_i^{*(k)}$ and $y_i^{(k)}$ are respectively the true label and the network output of the $i^{(th)}$ image at the $k^{(th)}$ class with m training images and c neurons in the output layer.

The iterative update rule for the kernel coefficients $\boldsymbol{w}_{ij}^{(n)}$ in CNN during the backpropagation pass is given by:

$$\begin{aligned} \boldsymbol{w}_{ij}^{(n)} &= \boldsymbol{w}_{ij}^{(n)} + \Delta \boldsymbol{w}_{ij}^{(n)} \\ \Delta \boldsymbol{w}_{ij}^{(n)} &= m \cdot \Delta \boldsymbol{w}_{ij}^{(n)} - d \cdot \epsilon \cdot \boldsymbol{w}_{ij}^{(n)} - \epsilon \cdot \frac{\partial E}{\partial \boldsymbol{w}_{ij}^{(n)}}, \end{aligned} \tag{2.7}$$

where $\boldsymbol{w}_{ij}^{(n)}$ represents the $i^{(th)}$ channel from the $j^{(th)}$ kernel matrix in the hidden layer $h^{(n)}$ that convolves with the $i^{(th)}$ channel in the previous feature maps denoted by $h_i^{(n-1)}$, $\Delta \boldsymbol{w}_{ij}^{(n)}$ denotes the gradient of $\boldsymbol{w}_{ij}^{(n)}$ and ϵ is the *learning rate*. The letters m and d are respectively the *momentum* and the *weight decay*, used for fast convergence as explained in [24]. The bias term $b_j^{(n)}$ in (2.5) is updated using the same equation presented in (2.7).

2.4 Convolutional Neural Networks in Image Forensics

As mentioned in the previous section, CNNs have been successfully used in recent years for many image recognition and classification tasks [1]. However, only recently, some works have started to explore CNNs for multimedia forensic applications.

One of the first works using CNNs for multimedia forensics is [25]. In this paper, the authors developed a detector for median-filtered images, whose capability of working on small 64×64 patches enabled its use also for tampering localization. In developing this algorithm, authors showed the importance of applying a pre-processing filtering step to images, in order to better expose forgery traces in a residual domain. The used CNN architecture is composed by only four convolutional layers and fewer inner-products, nonetheless providing very accurate results.

The importance of working with high-pass versions of the image under analysis for forensic works was also remarked in [26]. In this paper, the authors use a CNN to build a detection model for image manipulation. The key idea is to learn to discriminate between a set of known manipulation (i.e., median filtering, gaussian blurring, addition of white gaussian noise, resizing) performed on the image. During the training phase, they constrain the weight of the first convolutional level to evolve towards a designed high-pass filter, instead of following the usual gradient descent update. In this way they manage to exclude the scene (image content) from the learning procedure, focusing only on manipulation features. With this method, they do not require to know the characteristic traces of each transformation, and they reach very satisfactory results.

A forensic task that has been better investigated with CNNs is camera

model identification. This is detecting the camera model used to shoot a picture from the analysis of the picture itself. To this purpose, authors of [27] made use of a three convolutional layers network to detect the camera model used to shot a picture. In [28], the same goal was achieved using four convolutional layers, also showing the capability of CNNs to generalize to camera models never used for training. Finally, in [29], the authors investigated the possibility of using up to ten convolutional layers, concluding that no additional benefit was provided by going that deep. It is worth noting that [28, 29] perform well against the previous state-of-the-art ones, especially when results are compared on small blocks extracted from images (i.e., 64×64 pixel). The main advantage of the proposed algorithms is that they work on raw data and they do not rely on any analytical modeling, which can be prone to errors due to simplistic assumptions or model simplifications.

To the best of our knowledge, the only work based on CNNs for double JPEG detection is [2]. However, in this work, the authors feed the CNN with hand-crafted features (i.e., DCT coefficients histograms) rather than letting the network learn directly from data. Specifically, the authors exploit hierarchical feature learning of CNNs to detect double JPEG compression. Given an input image, they extract a feature vector based on some considerations on DCT histograms and feed it to a CNN. In particular, they observe that double JPEG compression considerably changes the shape of the distribution of DCT coefficients at given frequency, exhibiting characteristics peaks and valleys, Figure 2.9.

They consider only Y component of (YC_bC_r) and for simplicity they drop the DC component since it has a different distribution histogram with respect to the AC components' ones. Furthermore, they reduce the dimensionality of the input by considering only the 2nd-10th DCT coefficients in zigzag order

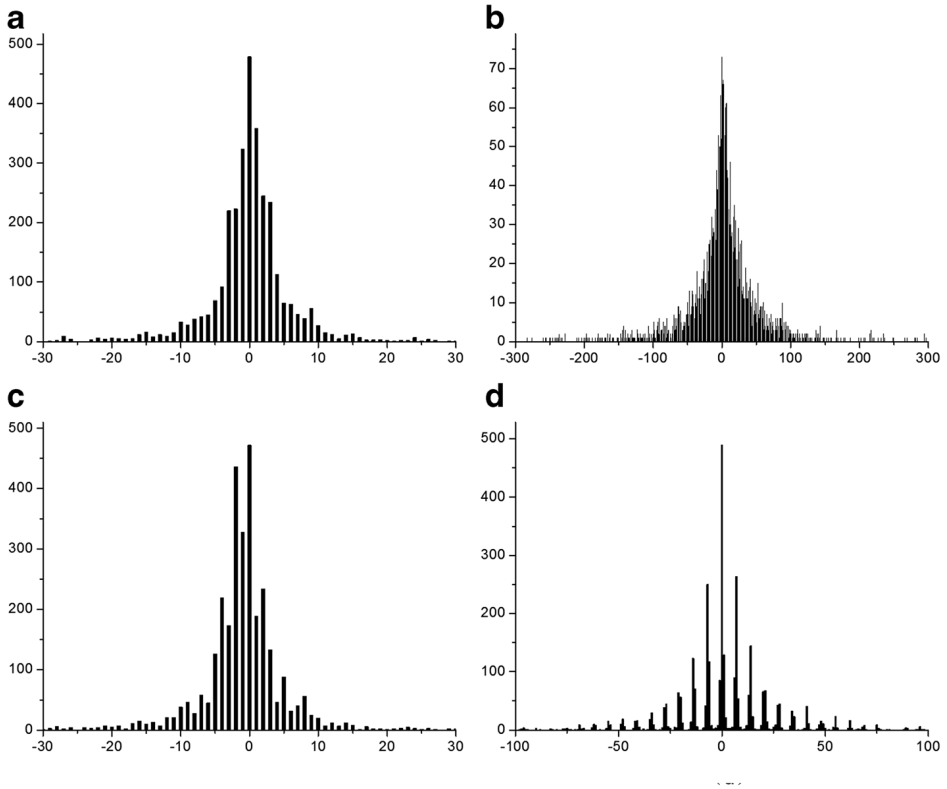


Figure 2.9: DCT coefficient histograms corresponding to the (0,1) position. **a**, **b** *DCT coefficient histograms of a single-compressed image with* **a** *QF1 = 60 and* **b** *QF1 = 90.* **c**, **d** *DCT coefficient histograms of the same double-compressed image with* **c** *QF1 = 90, QF2 = 60, and* **d** *QF1 = 60, QF2 = 90.*

and using only the values corresponding to $\{-5, -4, ..., 4, 5\}$ as representative of the whole histogram. In details: let B denote a block with a size of $W \times W$, and let $hi(u)$ denote the histogram of DCT coefficients with the value u at the ith frequency in zigzag order in B. Then, the feature set consists of the following values:

$$XB = \{hi(-5), hi(-4), hi(-3), hi(-2), hi(-1), hi(0), \\ hi(1), hi(2), hi(3), hi(4), hi(5) \,|\, i \in 2, 3, ..., 9, 10\}, \quad (2.8)$$

in this way they obtain a 9×11 feature vector for each block. The architecture

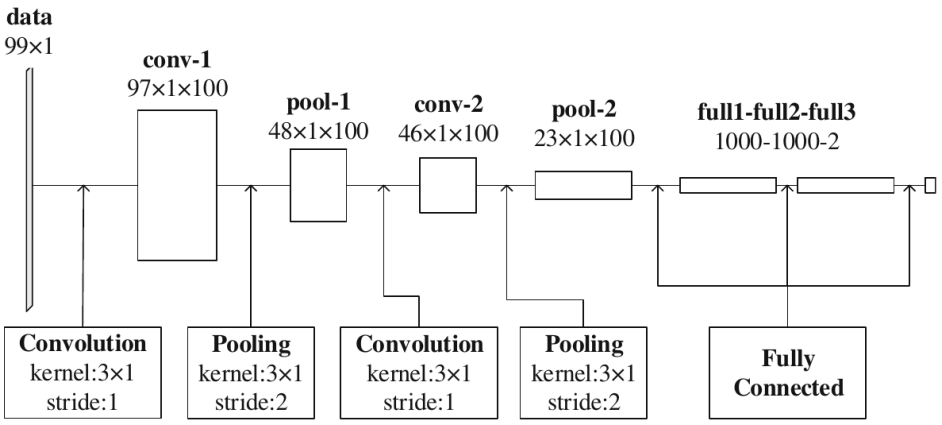


Figure 2.10: CNN architecture used in [2].

of their network is shown in Figure 2.10.

It contains two convolutional connections followed by two pooling connections and three full connections. The size of the input data is 9×11, and the output is a distribution of two classes (single or double compressed). For the convolutional connections, they set the kernel size $m \times n$ to 3×1, the number of kernels k to 100, and the stride s to 1. Here, we consider the first convolutional layer as an example: the size of the input data is 99×1, and the first convolutional layer convolves these data with 100 3×1 kernels, with a stride (step size) of 1. The size of the output is $97 \times 1 \times 100$, which means that the number of feature maps is 100 and the output feature maps have dimensions of 97×1. For the pooling connections, they set the pooling size $m \times n$ to 3×1, and the pooling stride s to 2 with "max pooling" as pooling function. Each full connection has 1000 neurons, and the output of the last one is sent to a two-way softmax connection, which produces the probability that each sample should be classified into each class. The accuracy of this method, for different QF2 averaged over QF1 $\in \{60, 70, 80, 90, 95\}$ and different block sizes W is shown in Figure 2.11. Notice that the performance rapidly degrade as image size decreases. As a matter of fact, double JPEG

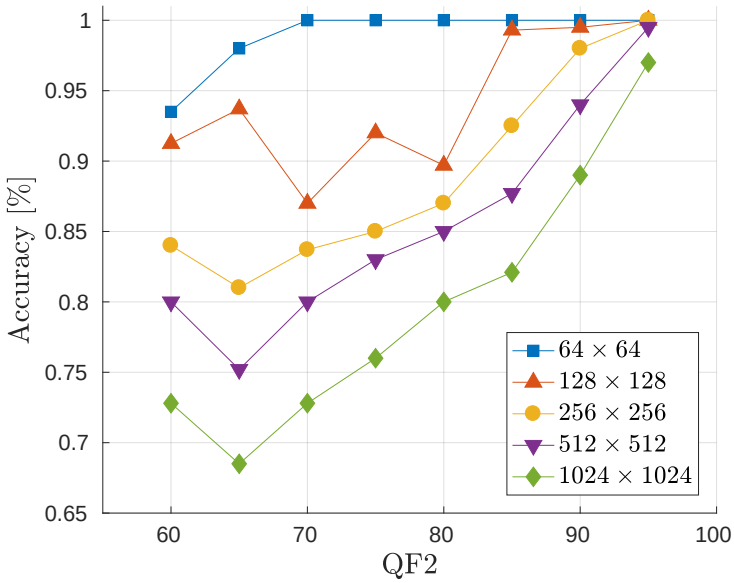


Figure 2.11: Accuracy of [2] for different QF2 and different block sizes W.

compression detection on small images (e.g., 64×64) is still an open issue in the literature.

Based on the aforementioned works, it is worth noting how CNNs for multimedia forensics typically necessitate of just a few layers to achieve promising results, being much less deeper than CNNs used for artificial intelligent and computer vision applications. This is probably due to the fact that the considered forensic classification tasks can be efficiently solved also using simple statistical model. This means that they do not need a very high abstraction capability, which proves essential for complex tasks like object recognition, in which the underlying model (i.e., human brain behavior) is particularly hard to describe.

3.

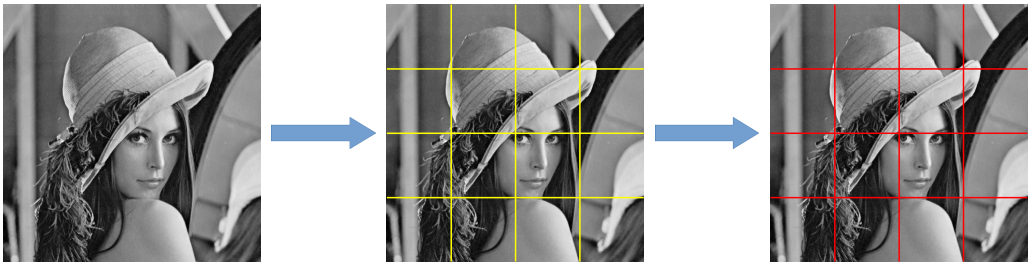
Problem formulation

In this Chapter we first introduce the problem of double JPEG compression detection, which is the main focus of our research. Than, we introduce the problem of JPEG quality factor estimation and software suite identification. These are two additional JPEG-based forensic problems we analyze as peculiar cases of study.

3.1 Double JPEG detection

JPEG compression is a destructive transformation, as Equation (2.3) is not invertible. As a consequence, a JPEG image subject to a second compression will show some characteristic features that can be detected. The pipeline of a double compression is the following. Image I_0 is compressed with QF = QF1 (Chapter 2.1), obtaining a JPEG image I_1. Then, I_1 is decompressed obtaining $\tilde{I}_1$ which is compressed with QF = QF2 obtaining I_2. If no operations are applied between the two compression steps, 8×8 JPEG blocks of the first and second compressions are perfectly aligned, thus we speak of A-DJPEG compression, Figure 3.1. Conversely, when the second compression 8×8 grid is shifted with respect the previous one (e.g., due to cropping between

first and second compression or to a cut and paste operation), we have a NA-DJPEG compression, Figure 3.2.



I_0 I_1 I_2

Figure 3.1: A-DJPEG example: Image I_0 is first compressed with the block grid shown in yellow, obtaining image I_1. Then I_1 is again compressed with the red block grid, aligned with the one of the first compression, obtaining I_2.

I_0 I_1 I_2

Figure 3.2: NA-DJPEG example: Image I_0 is first compressed with the block grid shown in yellow, obtaining image I_1. Then I_1 is again compressed with the red block grid, misaligned with the one of the first compression, obtaining I_2.

Our goal is to build a detector which is able to classify between single compressed and double compressed images. In other words, let us define:

$$\begin{cases} \mathcal{H}_0, & \text{single compressed} \\ \mathcal{H}_1, & \text{double compressed} \end{cases},$$

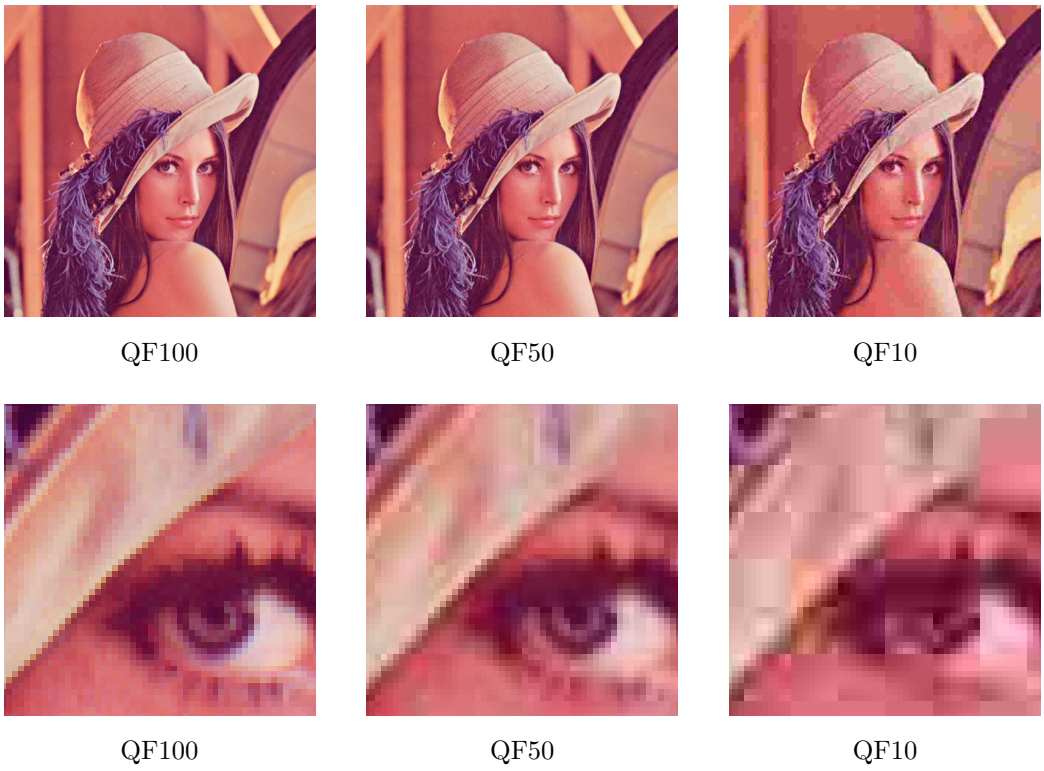


Figure 3.3: JPEG compression image degradation due to different quality factors compression. Top: example images compressed with various QFs. Bottom: magnified detail of the respective top image. Blocky artifacts become particularly evident as QF lowers.

corresponding to the hypothesis of single compressed image and to the hypothesis of image compressed twice, respectively.

Given a $B \times B$ pixel image I, we want to detect whether $\mathcal{H}_0$ or $\mathcal{H}_1$ is verified, considering: i) only A-DJPEG case; ii) only NA-DJPEG; iii) both A-DJPEG and NA-DJPEG cases.

3.2 Quality factor estimation

The outcome of JPEG compression heavily depends on the adopted quality factor QF (Chapter 2.1). Clearly, using higher quality factors leads to lower image degradation whilst using lower quality factors leads to a greater compression in spite of a greater degradation. The effect of using different quality factors is depicted in Figure 3.3.

Our goal is to build a detector which is able to estimate the adopted quality factor $\overline{QF}$ among a set of possible quality factors $\mathcal{Q} = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. For this task, let us define:

$$\begin{cases} \mathcal{H}_0, & \text{QF10} \\ \mathcal{H}_1, & \text{QF20} \\ \mathcal{H}_2, & \text{QF30} \\ \mathcal{H}_3, & \text{QF40} \\ \mathcal{H}_4, & \text{QF50} \\ \mathcal{H}_5, & \text{QF60} \\ \mathcal{H}_6, & \text{QF70} \\ \mathcal{H}_7, & \text{QF80} \\ \mathcal{H}_8, & \text{QF90} \\ \mathcal{H}_9, & \text{QF100} \end{cases},$$

each hypothesis corresponding to the respective quality factor of $\mathcal{Q}$. Given an $B \times B$ pixel image I, compressed with quality factor $\overline{QF} \in \mathcal{Q}$ we want to detect $\overline{QF}$.

3.3 Software identification

JPEG compression has indeed to be done by means of a software. The underlying idea of this experiment is that the employed software could leave some characteristic traces on the bitmap of the compressed image. We take in consideration the two most used software products in image editing world: *Photoshop* and *GIMP*. The former is a proprietary software produced by Adobe, the latter is an open source alternative.

Our goal is to build a detector which is able to classify between Photoshop and GIMP compression. According to the file size of some test images compressed with both the products, we set 4 compression levels in order to have a common reference between the two of them. These compression levels are: *high*, *mid-high*, *mid-low* and *low*, loosely representing QF $= \{95, 70, 30, 15\}$. Figure 3.4 shows the differences between the compressed JPEG version of the two software products. As we can see, discerning can be very challenging, especially on higher compression levels. Let us define:

$$\begin{cases} \mathcal{H}_0, & \text{Photoshop} \\ \mathcal{H}_1, & \text{GIMP} \end{cases},$$

corresponding to Photoshop compression and GIMP compression hypothesis respectively. Given a $B \times B$ pixel image I, we want to detect whether $\mathcal{H}_0$ or $\mathcal{H}_1$ is verified considering the four possible scenarios given by the four defined compression levels.

Photoshop high | Photoshop mid-high | Photoshop mid-low | Photoshop low

GIMP high | GIMP mid-high | GIMP mid-low | GIMP low

Figure 3.4: Different JPEG compression outcomes from different software products. Top: various Photoshop JPEG compression levels. Bottom: various GIMP JPEG compression levels.

4.

Forensics detection system

The problems described in the previous chapter (i.e., DJPEG detection, QF estimation and software recognition) share two common aspects: i) their solution is based on detection of JPEG footprints; ii) they can all be cast as supervised classification problems. In this chapter we first describe the pipeline common to them and then we examine in details the proposed networks.

4.1 Pipeline overview

Given a input image I, our goal is to classify it between M classes, depending on the specific considered problem. Therefore, we make a set of hypoteses $\mathcal{H}_m$, $m \in [0, M]$. The proposed pipeline, depicted in Figure 4.1, is composed by two steps: training and testing. During training, a database of labeled images is used to learn CNN parameters for the selected architecture. The CNN is fed with N pairs $\{I_n, l_n\}$, $n \in [1, N]$, where $l_n = m$ if image I_n verifies $\mathcal{H}_m$. After training, the CNN outputs the learned model $\mathcal{M}$ containing all CNN parameters (e.g., filters, fully connected weights, etc.). When an image I is under analysis, it is fed to the trained CNN. The output of the network

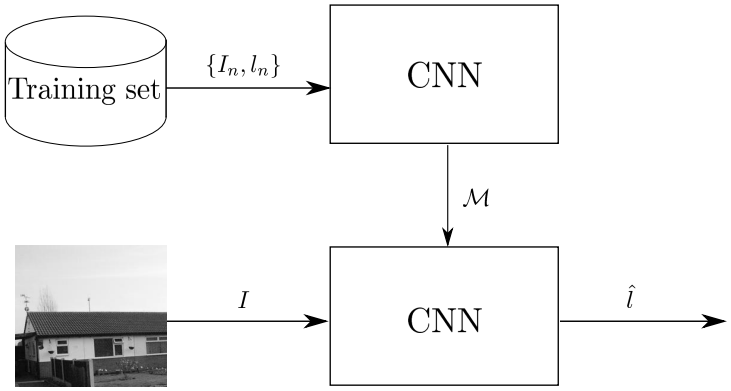


Figure 4.1: CNN training (top) is performed using images I_n labeled with l_n. The CNN model $\mathcal{M}$ is then used for testing (bottom) a new image I and obtain the candidate label $\hat{l}$.

will be a set of M probabilities, one for each hypotesis $\mathcal{H}_m$, for the image I to verify that specific hypotesis. A label $\hat{l}$ is then assigned to I by simply considering the maximum among these probabilities.

In particular, we would like to stress that it is convenient for us to work with small images, i.e., small patches of size $B \times B$ obtained by cropping an image I. We impose such a condition for different reasons: i) reducing the computational burden for processing a single image; ii) enlarging the amount of available images for a better training of the network; iii) working with small patches paves the way to tampering localization, a task on which we would like to focus in future. For these reasons, when we refer to image I, from now on we mean a small patch extracted from an actual image.

4.2 CNN architectures

In this section we report detailed information about all the CNN architectures applied in this work to solve the aforementioned tasks.

4.2.1 Classic architectures

Following recent literature results in image forensics we selected as starting point of our investigation the well known LeNet architecture for handwritten digits recognition.

LeNet is the network proposed in [1] for handwritten digits recognition. It has somehow become the standard baseline for image classification tasks. We therefore decided to test it to have a reliable yardstick for results on the other investigated networks.

The architecture is depicted in Table 4.1. Given an input image of size B×B×1 (i.e., a grayscale image), two convolutional layers (i.e., Conv-1, Conv-2) apply stride 1 convolution with a 5×5 kernel. Conv-1 uses 20 filters, whereas Conv-2 uses 50 filters. Each one of them is followed by a max pooling layer (i.e., Pool-1, Pool-2) with stride 2 and a 2×2 kernel. The first inner product layer (i.e., IP-1) reduces its input to 500 neurons and a ReLU non-linearity is performed on top of it. Finally, the last fully connected layer (i.e., IP-2) reduces its input to M neurons (one per class) and SoftMax layer normalize them to probability values.

Although the results yielded by the use of LeNet are often satisfactory, we can fall in some situations in which is preferable to "going deeper" [30] in the network by extending the chain of convolutional layers. The advantage is that we shrink the number of free parameters in the model, since we progressively reduce the feature maps dimension. To this purpose, we extended LeNet to two additional architectures. The first one has 3 convolutional→pooling layers. The second one has 6 convolutional→pooling layers. We refer to them as "LeNet-3" and "LeNet-6" respectively and we report Table 4.2 and Table 4.3 with the specifics of each layer.

Layer	**Kernel size**	**Stride**	**Num. filters**	**Output Size**
Conv-1	5×5	1	30	B-4 × B-4 × 30
Pool-1	2×2	2	-	$B/2$-2 × $B/2$-2 × 30
Conv-2	5×5	1	30	$B/2$-6 × $B/2$-6 × 30
Pool-2	2×2	2	-	$B/4$-3 × $B/4$-3 × 30
IP-1	-	-	500	500
ReLU-1	-	-	-	500
IP-2	-	-	M	M
SoftMax	-	-	-	M

Table 4.1: LeNet architecture parameters. Output of each layer is reported as function of the input image size $B \times B \times 1$.

4.2.2 DCT driven architecture

The literature has shown how working in DCT domain might be profitable in various aspects of JPEG detection, as mentioned in Chapter 2.2.1. A valid approach would be apply DCT transform to images and then feed them to the network. Starting from this idea and knowing that DCT can be implemented as a linear filtering operation, we ended up to a solution that exploits the learning capability of the network to explore a DCT-like domain. The architecture is depicted in Table 4.4.

The first convolutional layer (i.e., DCT) is initialized with the weights shown in Figure 4.2, which are the 8×8 DCT basis defined as

$$\boldsymbol{H}_{c_1,c_2}(i,j) = \alpha(c_1,c_2)\cos\left[\frac{\pi}{8}\left(i+\frac{1}{2}\right)c_1\right]\cos\left[\frac{\pi}{8}\left(j+\frac{1}{2}\right)c_2\right],$$

where $c_1 \in [0,7]$ and $c_2 \in [0,7]$ define horizontal and vertical frequencies of a DCT base (i.e., which filter), $i \in [0,7]$ and $j \in [0,7]$ are sample positions

Layer	**Kernel size**	**Stride**	**Num. filters**	**Output Size**
Conv-1	3×3	1	20	B-2 × B-2 × 20
Pool-1	2×2	2	-	$B/2$-1 × $B/2$-1 × 20
Conv-2	3×3	1	50	$B/2$-3 × $B/2$-3 × 50
Pool-2	2×2	2	-	$B/4$-1 × $B/4$-1 × 50
Conv-3	3×3	1	80	$B/4$-3 × $B/4$-3 × 80
Pool-3	2×2	2	-	$B/8$-3 × $B/8$-3 × 80
IP-1	-	-	500	500
ReLU-1	-	-	-	500
IP-2	-	-	M	M
SoftMax	-	-	-	M

Table 4.2: LeNet-3 architecture parameters. Output of each layer is reported as function of the input image size $B \times B \times 1$.

of each filter, and α is a scale normalization term. Given an input image of size B×B×1, the stride 8 convolution with this particular set of filters is equivalent to compute the 8×8 block DCT on the whole image and to map frequence c_1, c_2 of each block to a single feature map on the third dimension of the output, shaped $\lceil \frac{B-8}{8} \rceil + 1 \times \lceil \frac{B-8}{8} \rceil + 1 \times 64$. We set the first filter (i.e., the DC component) to 0 in order to ease the computation. In fact, DC terms are generally on a different magnitude order with respect to the AC terms, hence keeping the DC would have led to non-convergence problems. Two convolutional layers (i.e., Conv-1, Conv-2) apply stride 1 convolution with a 5×5 kernel, the first one with 20 filters, the second one with 50 filters. Both of them are followed by a max pooling layer (i.e., Pool-1, Pool-2) with stride 2 and a 2×2 kernel. The first inner product layer (i.e., IP-1) reduces its input to 500 neurons and a ReLU non-linearity is performed on top of

Layer	**Kernel size**	**Stride**	**Num. filters**	**Output Size**
Conv-1	5×5	1	20	B-4 × B-4 × 20
Pool-1	2×2	2	-	$B/2$-2 × $B/2$-2 × 20
Conv-2	5×5	1	50	$B/2$-6 × $B/2$-6 × 50
Pool-2	2×2	2	-	$B/4$-3 × $B/4$-3 × 50
Conv-3	3×3	1	80	$B/4$-5 × $B/4$-5 × 80
Pool-3	2×2	2	-	$B/8$-2 × $B/8$-2 × 80
Conv-4	3×3	1	100	$B/8$-4 × $B/8$-4 × 100
Pool-4	2×2	2	-	$B/16$-2 × $B/16$-2 × 100
Conv-5	2×2	1	120	$B/16$-3 × $B/16$-3 × 20
Pool-5	2×2	2	-	$B/32$-1 × $B/32$-1 × 120
Conv-6	2×2	1	140	$B/32$-2 × $B/32$-2 × 140
Pool-6	2×2	2	-	$B/64$-1 × $B/64$-1 × 140
IP-1	-	-	250	250
ReLU-1	-	-	-	250
IP-2	-	-	M	M
SoftMax	-	-	-	M

Table 4.3: LeNet-6 architecture parameters. Output of each layer is reported as function of the input image size $B \times B \times 1$.

it. Finally, the last fully connected layer (i.e., IP-2) reduces its input to M neurons (one per class) and SoftMax layer normalize them to probability values.

4.2.3 High pass filters driven architecture

As stated in [3, 31], high-pass filtering is a valid preprocessing in order to eliminate semantical content from images and focus on the traces left by JPEG compression (or even other forensic traces). Therefore, we decided to

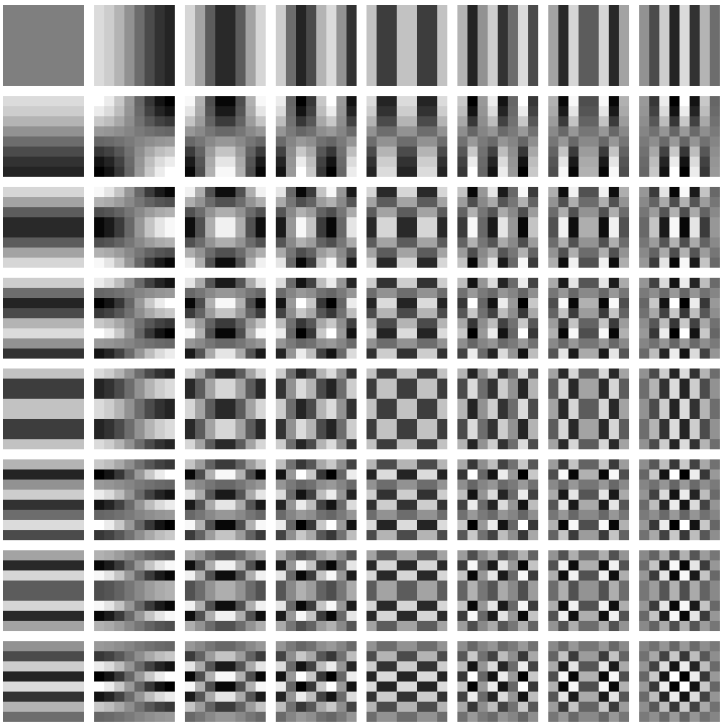


Figure 4.2: 2D DCT filters adopted in the first convolutional layer of the network.

exploit this idea embedding a high-pass filtering layer to the network, rather than performing any preprocessing on raw images. We insert a handcrafted convolutional layer that computes the high-pass filtering starting with the weights used in [3] and possibly improving them.

The network architecture is depicted in Table 4.5. Given an input image of size $B \times B \times 1$, the first convolutional layer (i.e., HP) performs high-pass filtering using 26 filters. A subset of adopted values used for weights initialization is shown in Figure 4.3. Two convolutional layers (i.e., Conv-1, Conv-2) apply stride 1 convolution with a 5×5 kernel, the first one with 20 filters, the second one with 50 filters. Both of them are followed by a max pooling layer (i.e., Pool-1, Pool-2) with stride 2 and a 2×2 kernel. The first inner product layer (i.e., IP-1) reduces its input to 500 neurons and a ReLU

Layer	Kernel size	Stride	Num. filters	Output Size
DCT	8×8	8	64	$\lceil\frac{B-8}{8}\rceil+1 \times \lceil\frac{B-8}{8}\rceil+1 \times 26$
Conv-1	5×5	1	20	$\lceil\frac{B-8}{8}\rceil-3 \times \lceil\frac{B-8}{8}\rceil-3 \times 20$
Pool-1	2×2	2	-	$\lceil\frac{B-8}{16}\rceil-2 \times \lceil\frac{B-8}{16}\rceil-2 \times 20$
Conv-2	5×5	1	50	$\lceil\frac{B-8}{16}\rceil-6 \times \lceil\frac{B-8}{16}\rceil-6 \times 50$
Pool-2	2×2	2	-	$\lceil\frac{B-8}{32}\rceil-3 \times \lceil\frac{B-8}{32}\rceil-3 \times 50$
IP-1	-	-	250	250
ReLU-1	-	-	-	250
IP-2	-	-	M	M
SoftMax	-	-	-	M

Table 4.4: DCT driven architecture parameters. Output of each layer is reported as function of the input image size $B \times B \times 1$.

non-linearity is performed on top of it. Finally, the last fully connected layer (i.e., IP-2) reduces its input to M neurons (one per class) and SoftMax layer normalize them to probability values.

4.2.4 Noise driven architecture

The HP-driven architecture just described can be seen as a layer of high-pass filters that pre-process an input image I enhancing noisy components, followed by the basic LeNet architecture. The use of noise enhancement techniques to analyze fine image details related to the past processing history of an image is widely adopted in the forensic literature. Specifically, it is well known that characteristic artifacts can be extracted using non-linear noise enhancement techniques, e.g., for camera device identification [32]. In order to gain a deeper insight on the importance of using a non-linear pre-processing operator, we also propose to apply a non-linear pre-processing step to image I, and feed the resulting image residual to LeNet. This solution mimic

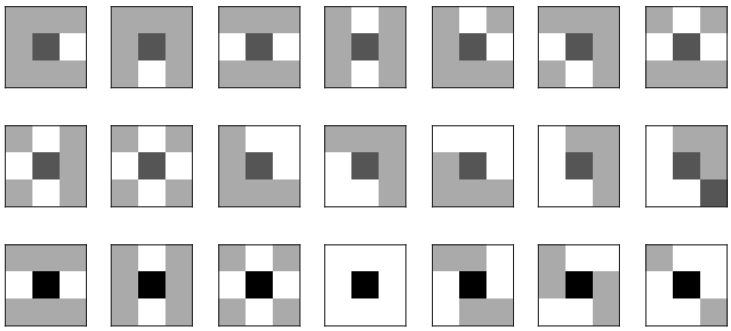


Figure 4.3: Subset of filters used in [3]. Values $\in \{-2, -1, 0, +1\}$ *from the darkest to the brightest.*

the behavior of the HP-driven approach, substituting high-pass filters with a fixed non-linear hih-pass-like operation. Specifically, the pre-processing operation we consider is

$$\tilde{I} = I - \mathcal{F}(I) \quad , \tag{4.1}$$

where $\tilde{I}$ is the image residual fed to LeNet, and $\mathcal{F}(\cdot)$ is the non-linear denoising operator described in [33], widely used in forensics for its good capability of separating image content from noise [32].

4.2.5 Histogram based architecture

In [2], histogram of DCT coefficients are successfully employed as feature vector. They are extracted from the images and fed to a CNN (Chapter 2.4) for further feature transformations and final classification.

Despite our approach is similar to the one proposed in [2], we would like to stress that: i) we do not make use of DCT coefficients extracted from JPEG bitstream, rather we compute DCT with a CNN layer enabling us to work with decompressed images (i.e., our method still works if double JPEG images are stored in bitmap or PNG format); ii) we exploit a 2D-convolutional CNN, rather than a 1D one, thus capturing possible correlation among DCT

Layer	**Kernel size**	**Stride**	**Num. filters**	**Output Size**
HP	3×3	1	26	B-2 × B-2 × 26
Conv-1	5×5	1	20	B-6 × B-6 × 20
Pool-1	2×2	2	-	$B/2$-3 × $B/2$-3 × 20
Conv-2	5×5	1	50	$B/2$-7 × $B/2$-7 × 50
Pool-2	2×2	2	-	$B/4$-3 × $B/4$-3 × 50
IP-1	-	-	500	500
ReLU-1	-	-	-	500
IP-2	-	-	M	M
SoftMax	-	-	-	M

Table 4.5: High-pass filters driven architecture parameters. Output of each layer is reported as function of the input image size $B \times B \times 1$.

coefficient histograms; iii) our solution embeds histogram computation as part of the CNN; iv) by embedding histogram computation in the CNN, we are able to also optimize the choice of quantization bins, rather than fixing it manually as in any hand-crafted approach. Indeed, the used set of bins become a parameter that the CNN can learn during training.

The proposed architecture can be thought as split into two parts as show in Figure 4.4: i) the former computes DCT coefficients histograms; ii) the latter, fed with these histograms, is the CNN described in Table 4.1. For the first part, the first step consists in obtaining the 2D DCT representation of each 8×8 image block. To this purpose, let us define $\boldsymbol{D}_{c_1,c_2}$ as the $\frac{B}{8} \times \frac{B}{8}$ matrix containing the DCT coefficients at frequency (c_1, c_2) for each 8×8 image block. This can be computed with a convolutional layer as:

$$\boldsymbol{D}_{c_1,c_2} = \text{conv}_8(I, \boldsymbol{H}_{c_1,c_2}), \tag{4.2}$$

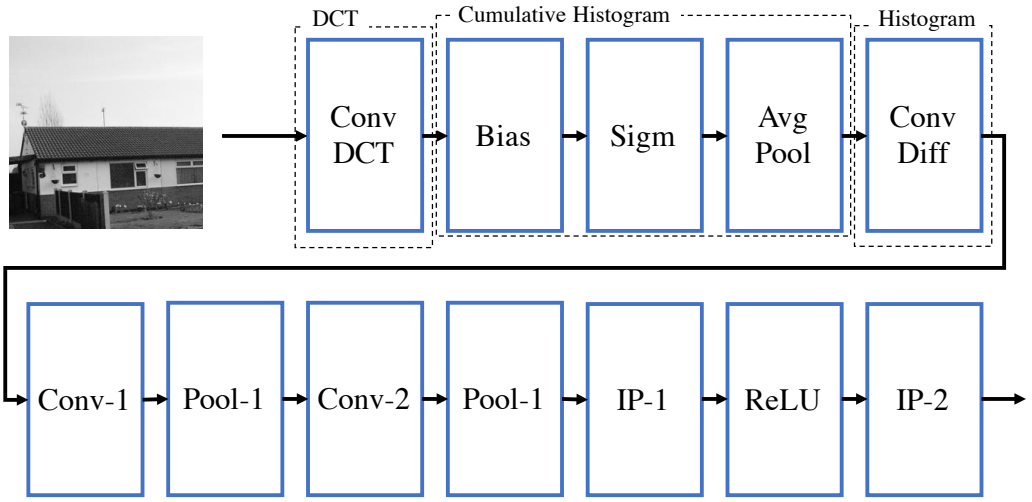


Figure 4.4: Pipeline of the CNN layers used by the third proposed method. On the top, the part devoted to DCT histogram computation. On the bottom, the CNN described in Table 4.1.

where $\text{conv}_8(\cdot,\cdot)$ computes the valid part of the 2D linear convolution using stride 8, and $\boldsymbol{H}_{c_1,c_2}$ is the base at (c_1, c_2) frequency of the 8×8 DCT, whose entries are defined as:

$$\boldsymbol{H}_{c_1,c_2}(i,j) = \alpha(c_1,c_2) \cos\left[\frac{\pi}{8}\left(i+\frac{1}{2}\right)c_1\right] \cos\left[\frac{\pi}{8}\left(j+\frac{1}{2}\right)c_2\right], \qquad (4.3)$$

where $\alpha(c_1, c_2)$ is a normalization constant and $i, j, c_1, c_2 \in [0, 7]$. An example of $\boldsymbol{D}_{c_1,c_2}$ is reported in Figure 4.5. In order reduce computational burden, we decided to use only 9 DCT frequencies (i.e., the first 10 coefficients in zigzag order, DC excluded). At this point we want to compute the histogram for each frequency (c_1, c_2). To do so by means of common CNN layers, we first compute the cumulative histogram then we differentiate it. Specifically, to count the average number $B_{c_1,c_2}(b)$ of values in $\boldsymbol{H}_{c_1,c_2}$ that are greater than a constant b, we resort to a series of bias, sigmoid and average-pooling layers,

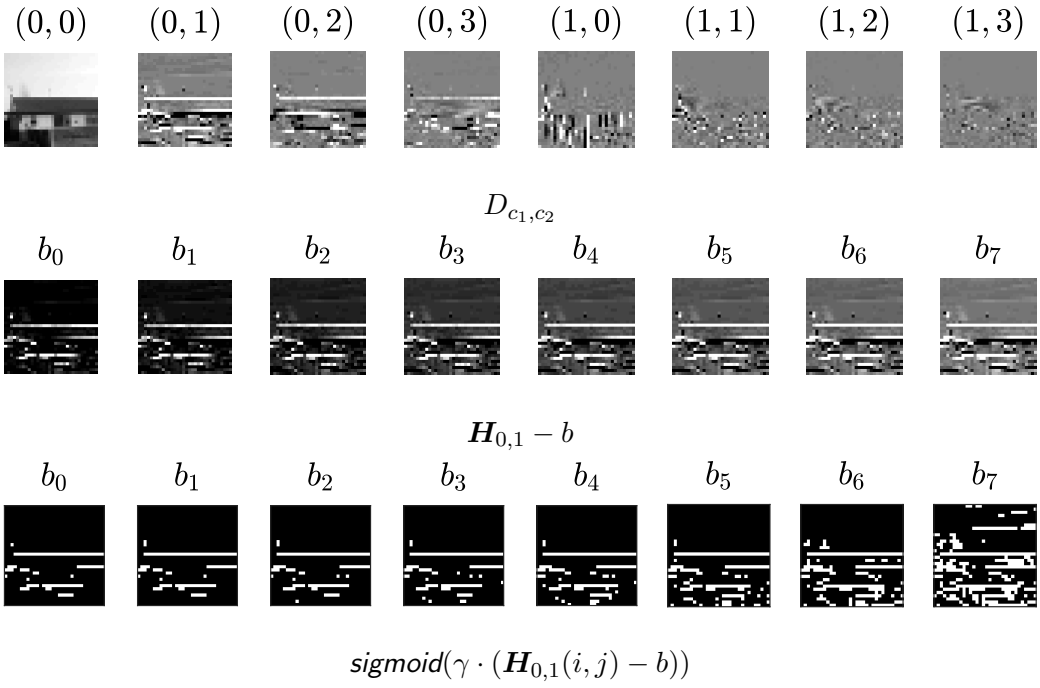


Figure 4.5: Outputs of CNN layers devoted to histogram computation: (a) output of the DCT layer D_{c_1,c_2} for nine different pairs (c_1, c_2); (b) output of the bias layer $\boldsymbol{H}_{c_1,c_2} - b$ for $(c_1, c_2) = (0, 1)$ and different b values; (c) output of sigmoid layer sigmoid$(\gamma \cdot (\boldsymbol{H}_{c_1,c_2}(i,j) - b))$ for $(c_1, c_2) = (0, 1)$ and different b values.

obtaining:

$$B_{c_1,c_2}(b) = \frac{B^2}{64} \sum_{i,j \in [0,7]} \text{sigmoid}\left[\gamma \cdot (\boldsymbol{H}_{c_1,c_2}(i,j) - b)\right], \tag{4.4}$$

where the bias b is a constant value identifying a histogram bin boundary, γ is a gain (i.e., 10^6 in our experiments) used to expand the dynamic of $\boldsymbol{H}_{c_1,c_2}(i,j) - b$ (i.e., to obtain very high values for $\boldsymbol{H}_{c_1,c_2}(i,j) > b$ and very low values for $\boldsymbol{H}_{c_1,c_2}(i,j) < b$), sigmoid$(\cdot)$ turns very low and very high input values into $\{0, 1\}$ respectively, and the average-pooling layer performs the sum and normalization for $B^2/64$. In other words, $B_{c_1,c_2}(b)$ is the b-th cumulative histogram bin for DCT coefficient (c_1, c_2). Examples of these

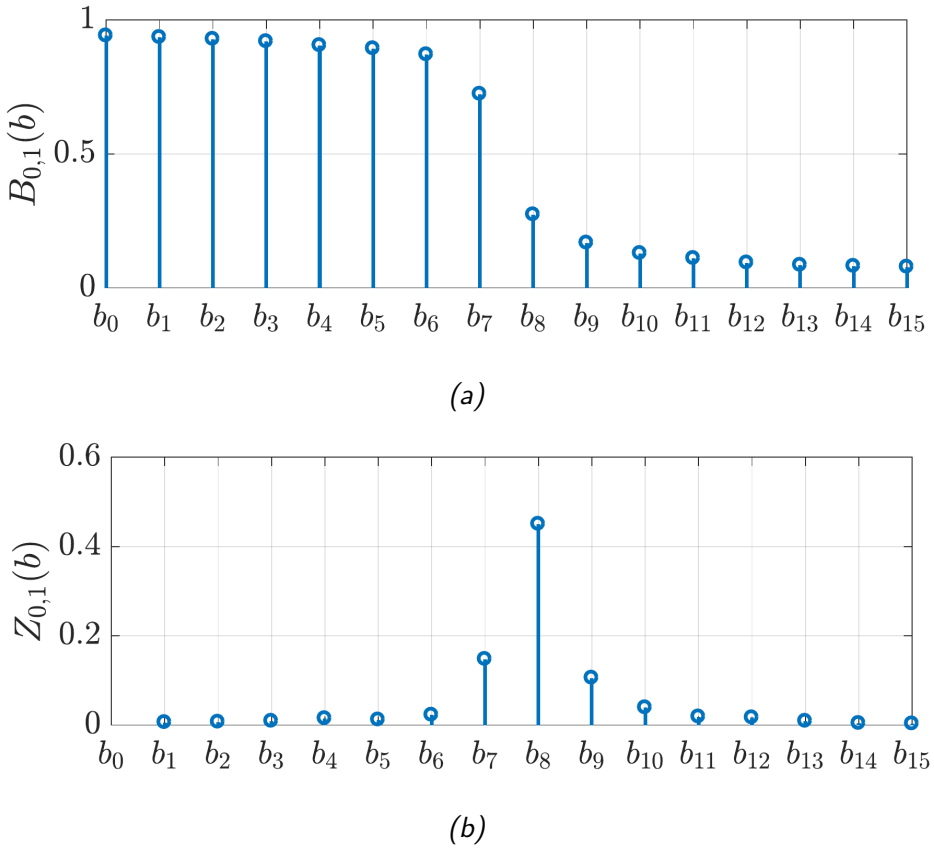


Figure 4.6: Example of (a) cumulative histogram B_{c_1,c_2} and (b) histogram Z_{c_1,c_2}, for $(c_1, c_2) = (0, 1)$ and $b \in [b_0, b_{15}]$. These are the output of average pooling and derivative convolutional layer fixing $(c_1, c_2) = (0, 1)$, respectively.

signals are depicted in Figure 4.5.

The histogram for each (c_1, c_2) can be obtained from the cumulative histogram using a convolutional layer that computes:

$$Z_{c_1,c_2}(b) = \text{conv}_1(B_{c_1,c_2}, [1, -1]), \tag{4.5}$$

namely the 1D convolution with the filter $[1, -1]$ acting as differentiator in the b-th direction. Differently from [2], we do not assume to already have access to quantized DCT coefficients. Therefore, the set of b values used to construct histograms is not known and must be sought. An example of

obtained histogram and its cumulative version is reported in Figure 4.6.

Once all histograms Z_{c_1,c_2} for all considered DCT frequency pairs (c_1, c_2) have been computed in parallel by the CNN, they are concatenated into a 2D matrix, where each row represents a histogram bin b, and each column represents a frequency pair (c_1, c_2). This matrix (i.e., the output of ConvDiff layer of Figure 4.4) can be considered as an image, fed as input to the CNN pipeline defined in Table 4.1.

5.

Experimental results

In this Chapter we describe the setup of each carried experiment and we analyze the results of our methodology with respect to the state of the art. All the employed datasets are built from images belonging either to the *RAISE* dataset [34] or to the *Dresden Image Database* dataset [35]. Both of them are large collections of natural uncompressed images (in *TIFF* or *PNG* format) taken with different instances of different camera models from different manufactorers. We choose these two particular datasets because they are largely employed in forensic field research e.g., [25, 28, 29, 2]. From now on, we will refer to them simply as the *RAISE* and the *Dresden*.

All the experiments have been run exploiting Caffe framework [36] on a workstation equipped with an Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz with 64GB of RAM and one NVIDIA Titan-X GPU.

5.1 Double JPEG detection

We put a considerable part of our effort in investigating the traces left by double JPEG compression. Here we present our experimental methodology after problem formalization of Chapter 3.1.

5.1.1 Dataset construction

In order to thoroughly validate the proposed solutions, we generated a set of training and test datasets of single and double compressed images at different resolutions and with different quality factors, for a total amount of more than 2 500 000 images.

All datasets were built starting from images of RAISE database. Images have been first converted to grayscale, then randomly cropped in order to obtain smaller resolution images used in our tests. Attention were paid to split into only one set (training or validation) all cropped portions coming from a same original image. All sets are balanced, i.e., they contain the same number of single and double JPEG images.

Training sets have been created in the following cases: i) $B = \{64, 256\}$; ii) aligned and non-aligned DJPEG. For each scenario, the image set was built as it follows: for the first class ($\mathcal{H}_0$), images of size $B \times B$ were single compressed with quality factor QF2; for the second class ($\mathcal{H}_1$), double compressed images were built by coding $B \times B$ images first with various QF1 and then with QF2. For a meaningful analysis, we took QF = QF2 as done in [2].

To build double compressed images for the non-aligned case, we started from images of size $B' \times B'$ with $B' \geq B + 7$. Then, after the first compression with QF1, images were shifted by a random quantity (r, c), $0 < r < 7$, $0 < c < 7$, and cropped to the size $B \times B$, before being compressed again with QF2, thus simulating grid misalignement. In all our experiments, we considered two possible values for QF2, that is 75 and 85, whereas QF1 $\in \{50, 60, 70, 80, 90\}$. Table 5.1 reports the breakdown of all these training datasets. We denote with $\bar{\mathcal{D}}$ datasets for the aligned DJPEG case and with $\hat{\mathcal{D}}$ datasets for non-aligned JPEG scenario. Superscripts indicate

the adopted QF2 (i.e., 75 or 85), whereas subscripts indicate image size (i.e., $B = 64$ or 256).

Validation datasets have been created to evaluate: i) detection accuracy under normal working conditions, i.e., the ability of classifying test images built under the same conditions of training, and also; ii) generalization capability, that is, the ability of classifying images even when they are not perfectly compliant with the used training set. To this purpose, we generated different sets of double JPEG images with many different $(\mathrm{QF1}, \mathrm{QF2})$ pairs and single JPEG images with the corresponding QF2. Specifically, in addition to the same pairs used for training, we considered some new pairs where QF1 or QF2 deviates from the values used for training. Each set contains 3 000 single compressed images and 3 000 double compressed ones. As for training, validation sets were built for the case $B = \{64, 256\}$, with either aligned or non-aligned DJPEG.

As commonly done to evaluate the performance with data-driven approaches, detection accuracy is measured over the same $(\mathrm{QF1}, \mathrm{QF2})$ pairs used for training. Then, to test their generalization capability, we also measured the performance of the detectors with respect to $(\mathrm{QF1}, \mathrm{QF2})$ pairs never used for training.

5.1.2 Evaluation Methodology

In order to fairly evaluate all CNN-based considered approaches, we devised a common training-validation strategy. All CNNs have been trained using stochastic gradient descent (SGD) algorithm with batch size (i.e., number of images used for each SGD iteration) set to 128. Momentum was set to 0.9. Learning rate was set to 0.01 for 64×64 images and 0.001 for 256×256 images, and was progressively decreased with exponential decay at each iteration.

Dataset	***I* Size ($B \times B$)**	**QF1**	**QF2**	**Alignement**	**N. Train**
$\bar{\mathcal{D}}_{256}^{(85)}$	256×256	50,60,70,80,90	85	A	280 000
$\hat{\mathcal{D}}_{256}^{(85)}$	-	-	-	NA	-
$\bar{\mathcal{D}}_{256}^{(75)}$	-	-	75	A	-
$\hat{\mathcal{D}}_{256}^{(75)}$	-	-	-	NA	-
$\bar{\mathcal{D}}_{64}^{(85)}$	64×64	50,60,70,80,90	85	A	300 000
$\hat{\mathcal{D}}_{64}^{(85)}$	-	-	-	NA	-
$\bar{\mathcal{D}}_{64}^{(75)}$	-	-	75	A	-
$\hat{\mathcal{D}}_{64}^{(75)}$	-	-	-	NA	-
				Tot Images	2 320 000

Table 5.1: Datasets used for training. All datasets are balanced in both classes and QF pairs.

The maximum amount of epochs (i.e., number of times the CNN sees all training data) was set to 30 to ensure network convergence. Initialization of CNN parameters have been performed using the method devised in [37]. As best CNN trained model, we always selected the one at the epoch with minimum validation loss in order to avoid overfitting.

The results are provided in terms of accuracy, namely the percentage of correctly classified single and double JPEG images in the validation dataset. We use notation $\mathcal{C}_{\text{pix}}$ to refer to the CNN-based detector in the pixel domain (Chapter 4.2.1), $\mathcal{C}_{\text{noise}}$ for the one in the noise domain (Chapter 4.2.4), and $\mathcal{C}_{\text{hist}}$ for the case of CNN embedding DCT histogram computation (Chapter 4.2.5). Concerning parameters of the latter, we only made use of the first 9 DCT frequencies after DC in zig-zag order for fair evaluation with [2]. Histograms have been computed using 61 integer bins initialized with $b \in [-30, 30]$. We avoid reporting results for CNN architectures providing less accurate results, leaving them to the next applications for which they

proved more robust.

5.1.3 Results

In this section we evaluate the performance of the proposed detectors relying on $\mathcal{C}_{\text{pix}}$, $\mathcal{C}_{\text{noise}}$ and $\mathcal{C}_{\text{hist}}$, and we compare them with the state-of-the-art. We first focus on the classification in the aligned double JPEG compression scenario, then we move to the case of non-aligned double JPEG compression. Finally, we provide some results in the mixed scenario of aligned and non-aligned double compression.

Aligned Double JPEG

It is well known that the performance of supervised machine learning techniques strongly depends on the amount of data used for training. In order to assess the dependency between number of images used for training and detection accuracy in our case, Figure 5.1 shows the results achieved with $\mathcal{C}_{\text{pix}}$ in the most difficult scenario with small patches ($B = 64$) and strong second quantization (QF = 75). To get the plot, the network is trained on different percentages of training images from $\bar{\mathcal{D}}_{64}^{(75)}$. We see that, when 10% of the dataset is used for training, accuracy is below 0.75. However, when more than 70% of training data is used, accuracy saturates around 0.82. Therefore, using the whole training dataset, we are sure that we are not experiencing losses due to insufficient amount of training data.[1]

To assess the performance of the proposed approaches for aligned double JPEG detection, we compare them to the state-of-the-art technique in [2]. We selected [2] as a baseline for two reasons: i) it is shown to outperform

[1]It is worth pointing that the other proposed solutions, i.e., $\mathcal{C}_{\text{noise}}$ and $\mathcal{C}_{\text{hist}}$, usually need less training images to converge.

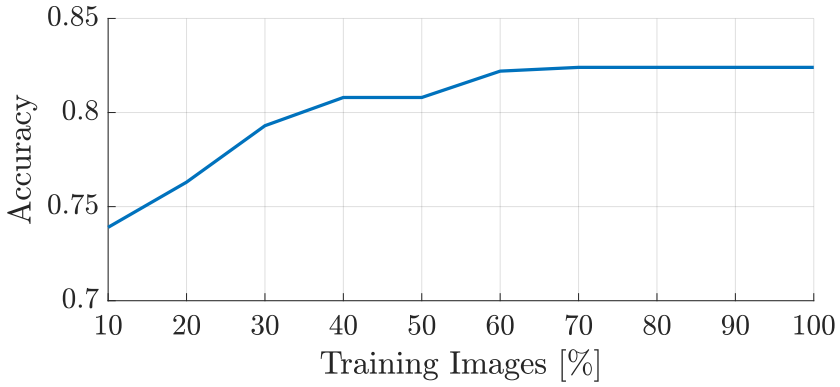


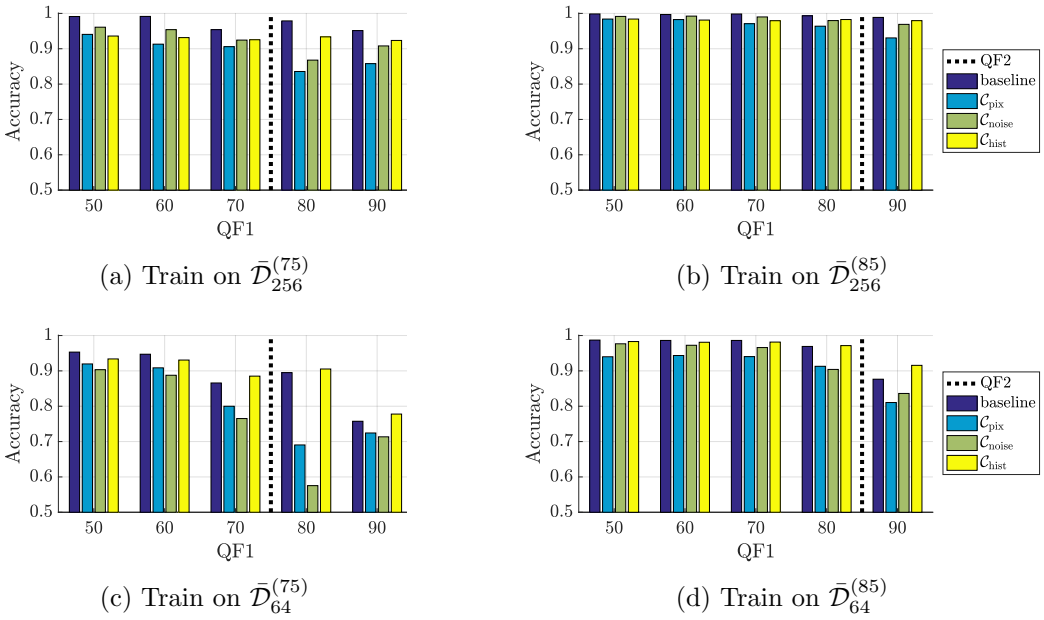
Figure 5.1: Impact of training set size on DJPEG detection accuracy using $\mathcal{C}_{pix}$.

other existing state-of-the-art detectors, e.g., [38, 39, 40, 41]; ii) to the best of our knowledge, it is the only double JPEG compression detection method based on CNNs, thus being a natural yardstick for our methods.

Figure 5.2 reports results obtained training all proposed CNNs in the various cases, i.e., on the datasets $\bar{\mathcal{D}}_{256}^{(75)}$, $\bar{\mathcal{D}}_{256}^{(85)}$, $\bar{\mathcal{D}}_{64}^{(75)}$ and $\bar{\mathcal{D}}_{64}^{(85)}$.

Results for $B = 256$ show that the proposed solutions achieve slightly lower performance with respect to the baseline [2]. This is due to the fact that hand-crafted features exploited in [2] are very distinctive, especially when large images are concerned.

However, things are different in case $B = 64$. Quite expectedly, all algorithms suffer when QF2 $\cong$ QF1 and QF2 $<$ QF1, as a stronger second compression tends to mask artifacts left by the first one; however, on 64×64 patches, $\mathcal{C}_{\text{hist}}$ is the one with the best performance and always outperforms state-of-the-art. Concerning the proposed methods, $\mathcal{C}_{\text{hist}}$ always outperforms $\mathcal{C}_{\text{pix}}$ and $\mathcal{C}_{\text{noise}}$. This is also expected, as aligned DJPEG traces are better exposed in the DCT domain, rather than the pixel domain. Nonetheless, a part when QF1 and QF2 are very close, also $\mathcal{C}_{\text{pix}}$ and $\mathcal{C}_{\text{noise}}$ allow to achieve accuracy greater than 0.70 on small images.



(a) Train on $\bar{\mathcal{D}}_{256}^{(75)}$ (b) Train on $\bar{\mathcal{D}}_{256}^{(85)}$

(c) Train on $\bar{\mathcal{D}}_{64}^{(75)}$ (d) Train on $\bar{\mathcal{D}}_{64}^{(85)}$

Figure 5.2: Aligned DJPEG compression detection accuracy against baseline [2]. Dashed black line indicates the considered QF2.

Regarding generalization capability, Figure 5.3 shows the accuracy achieved by all CNNs trained on the most difficult scenario with QF = 75 and small images ($B = 64$). The methods based on DCT histograms (i.e., $\mathcal{C}_{\text{hist}}$ and baseline [2]) suffer to recognize aligned DJPEG for values of QF1 different from those used during training when they are close to QF2, and completely fail when these QF1s are larger than QF2. Contrarily, the methods relying on pixel analysis (i.e., $\mathcal{C}_{\text{pix}}$ and $\mathcal{C}_{\text{noise}}$) show greater robustness to changes in (QF1, QF2).

To further explore this fact, Table 5.2(a) shows the behavior of $\mathcal{C}_{\text{noise}}$ trained on $\bar{\mathcal{D}}_{64}^{(75)}$ and $\bar{\mathcal{D}}_{256}^{(75)}$ and tested on images with several different (QF1, QF2) pairs. (similar results hold for $\mathcal{C}_{\text{pix}}$). Similarly, Table 5.2(b) reports the accuracy results with $\mathcal{C}_{\text{noise}}$ trained on $\bar{\mathcal{D}}_{64}^{(85)}$ and $\bar{\mathcal{D}}_{256}^{(85)}$.

We notice that, by varying QF1, results are perfectly in line with those

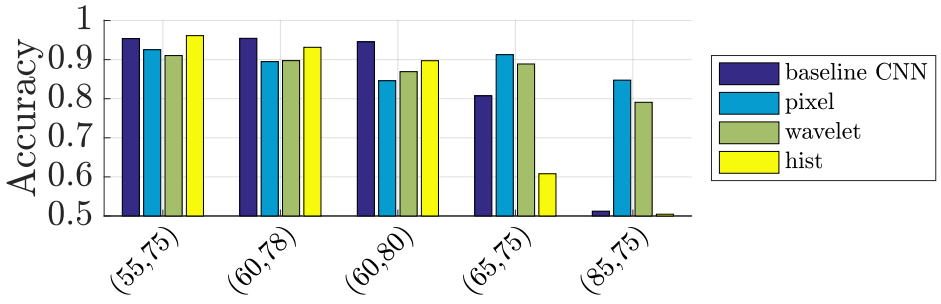


Figure 5.3: Sensitivity analysis for aligned DJPEG compression detection when QF2 = 75. *Image size is* 64×64.

achieved with matched QF pairs. Good results are also obtained with different QF2s, a part for the case of much higher QF2 and QF2 $>$ QF1, and especially QF2 $= 90$. This behavior is not surprising, since compression with high QF2 leaves few traces on images compressed at lower quality, hence detecting a DJPEG compression in these cases is hard even when such examples are included in the training set. Therefore, although on one side CNNs based on a strong hand-crafted modeling assumption (as baseline [2] and $\mathcal{C}_{\text{hist}}$) allow to achieve better accuracies, the ones based on the analysis of the pixel image (i.e., $\mathcal{C}_{\text{pix}}$ and $\mathcal{C}_{\text{noise}}$) prove to be more robust to perturbations of QF1 and QF2 with respect to the values used for training, which is paramount every time the algorithm works in the wild.

Non-aligned Double JPEG

When DJPEG compression occurs with misalignment between the grids, detectors in the previous section trained on aligned data do not work anymore, getting an accuracy which is around 0.5. To evaluate the performance of our method for NA-DJPEG detection, we re-trained the detectors in the misaligned case. In this case, not surprisingly, the algorithm in [2] does not work. Indeed, the features extracted by this method, i.e., the DCT his-

Testing (QF1, QF2)	$B = 64$	$B = 256$
(55, **75**)	0.925	0.982
(65, **75**)	0.880	0.981
(85, **75**)	0.820	0.952
(**60**, 78)	0.900	0.917
(**70**, 78)	0.810	0.907
(**60**, 80)	0.860	0.810
(**70**, 80)	0.790	0.800

(a) Train on $\bar{\mathcal{D}}_B^{(75)}$, $B \in \{64, 256\}$.

Testing (QF1, QF2)	$B = 64$	$B = 256$
(55, **85**)	0.963	0.994
(65, **85**)	0.960	0.993
(75, **85**)	0.923	0.978
(**70**, 88)	0.860	0.914
(**80**, 88)	0.640	0.656
(**70**, 90)	0.718	0.687
(**80**, 90)	0.500	0.510

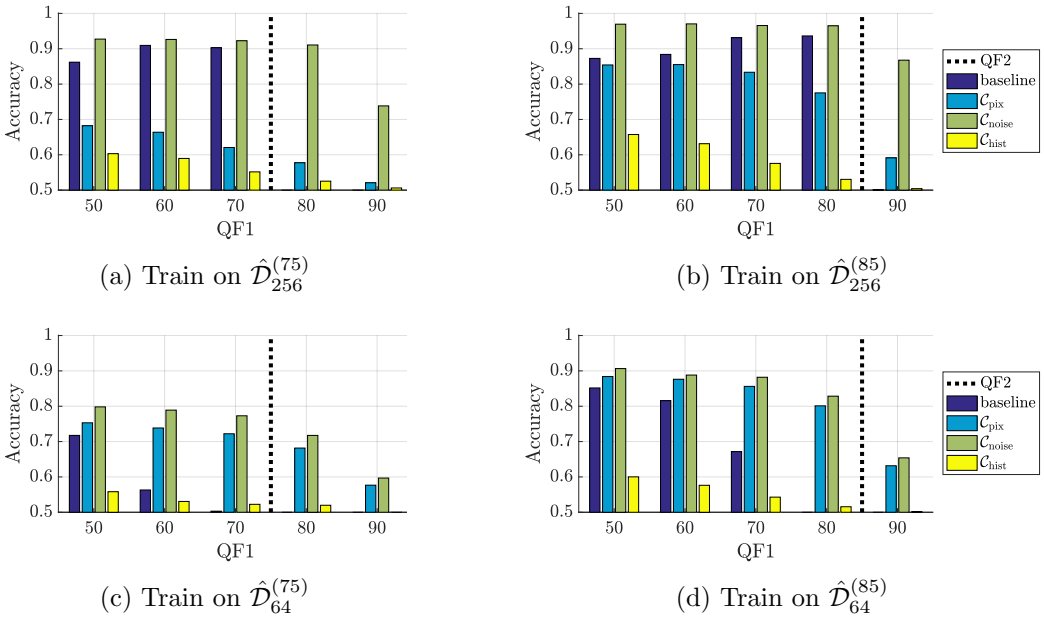
(b) Train on $\bar{\mathcal{D}}_B^{(85)}$, $B \in \{64, 256\}$.

Table 5.2: Sensitivity of $\mathcal{C}_{noise}$ to variations of QF1 and QF2 for aligned DJPEG detection. For any pair, only one between QF1 and QF2 is common to images used in the training set (reported in **bold***).*

tograms, are particularly distinctive only when the second compression is aligned with the first one (the typical peak and gap artifacts shows up in the DCT histograms). Therefore, we selected the well-known algorithm for NA-DJPEG detection proposed in [4] as baseline in this case.

Figure 5.4 shows the performance of all proposed techniques and the baseline [4] for QF2 $= \{75, 85\}$ with image size 64×64 and 256×256. It is known that the method in [4] does not work when QF1 $>$ QF2. Besides, the accuracy significantly drops for small images, especially in the case QF1 $\simeq$ QF2. Concerning our methods, not surprisingly, our solution $\mathcal{C}_{\text{hist}}$ shows poor performance. Indeed, similarly to [2], the traces in the DCT domain that $\mathcal{C}_{\text{hist}}$ looks at are weak in the non-aligned case.

On the other hand, CNNs designed to work in the pixel domain, and especially $\mathcal{C}_{\text{noise}}$, show good detection performance even for small images (i.e., 64×64). From these results, we see that the detector based on $\mathcal{C}_{\text{noise}}$ always outperforms state-of-the-art.



(a) Train on $\hat{\mathcal{D}}_{256}^{(75)}$ (b) Train on $\hat{\mathcal{D}}_{256}^{(85)}$

(c) Train on $\hat{\mathcal{D}}_{64}^{(75)}$ (d) Train on $\hat{\mathcal{D}}_{64}^{(85)}$

Figure 5.4: Non-aligned DJPEG compression detection accuracy against baseline [4]. Dashed black line indicates the considered QF2.

It is worth observing that the performance of $\mathcal{C}_{\text{pix}}$ are very poor when QF2 $= 75$ and images are large ($B = 256$). A possible motivation can be the following. When larger images are considered, it is more difficult for the CNN to automatically get rid of the content (in fact, $\mathcal{C}_{\text{noise}}$ always outperform $\mathcal{C}_{\text{pix}}$ when $B = 256$, even in the aligned case) and this is even more difficult when the second quantization is coarser (e.g., with QF2 $= 75$) and traces are more complex, as it is in the NA-DJPEG scenario (traces also vary with the specific misalignment between the grids). Actually, this is the case in which a deeper network would probably provide some benefits. However, since we are more interested in the performance with smaller images, we do not further investigate this case here.

Concerning network sensitivity to QF pairs different from those in the training set, Table 5.3 shows the results obtained with our best method $\mathcal{C}_{\text{noise}}$

Testing (QF1, QF2)	$B = 64$	$B = 256$
(55, **75**)	0.816	0.876
(65, **75**)	0.805	0.866
(75, **75**)	0.764	0.842
(85, **75**)	0.674	0.776
(**60**, 78)	0.777	0.845
(**70**, 78)	0.765	0.830
(**60**, 80)	0.723	0.794
(**70**, 80)	0.720	0.790

(a) Train on $\hat{\mathcal{D}}_B^{(75)}$, $B \in \{64, 256\}$.

Testing (QF1, QF2)	$B = 64$	$B = 256$
(55, **85**)	0.897	0.972
(65, **85**)	0.878	0.972
(75, **85**)	0.865	0.961
(85, **85**)	0.793	0.954
(**70**, 88)	0.751	0.786
(**80**, 88)	0.738	0.785
(**70**, 90)	0.650	0.610
(**80**, 90)	0.634	0.600

(b) Train on $\hat{\mathcal{D}}_B^{(85)}$, $B \in \{64, 256\}$.

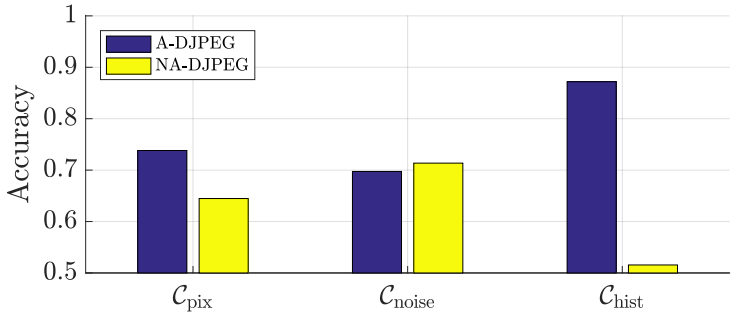
Table 5.3: Sensitivity of $\mathcal{C}_{noise}$ to variations of QF1 and QF2 for non-aligned DJPEG detection. Test and training images have only QF1 or QF2 in common (reported in **bold***).*

for both QF1 = 75 and 85, and image sizes. As for the aligned scenario, $\mathcal{C}_{\text{noise}}$ enables good detection accuracy, the only critical cases being those with much higher QF2.

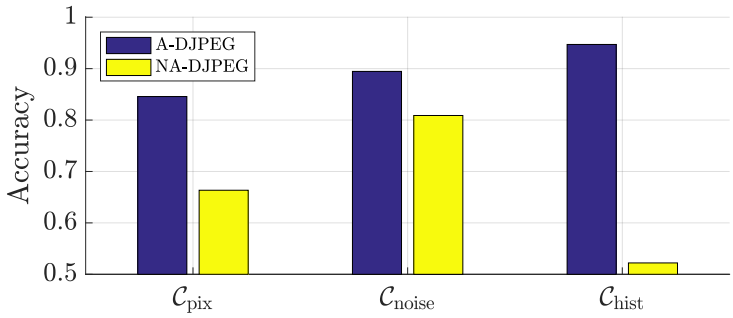
It is interesting to notice that $\mathcal{C}_{\text{noise}}$ is able to detect non-aligned DJPEG compression with good accuracy also in the very challenging scenario in which QF1 = QF2.

Aligned and Misaligned Double JPEG

Since it is usually not known a-priori whether double compression is aligned or not, it is relevant to be able to detect both A-DJPEG and NA-DJPEG with the same system. To this purpose, we trained the proposed architectures on a dataset obtained by the union of the one used for A-DJPEG, namely $\bar{\mathcal{D}}$, and the one used for NA-DJPEG, namely $\hat{\mathcal{D}}$. For the experiments of these section, we considered the most challenging scenario with small images



(a) Train on $(\bar{\mathcal{D}}_{64}^{(75)} \cup \hat{\mathcal{D}}_{64}^{(75)})$



(b) Train on $(\bar{\mathcal{D}}_{64}^{(85)} \cup \hat{\mathcal{D}}_{64}^{(85)})$

Figure 5.5: DJPEG compression detection accuracy tested separately on aligned and misaligned cases, when training is performed on a mixed dataset. Image size is 64 and QF2 $= \{75, 85\}$.

($B = 64$). Figure 5.5 shows the performance of the CNN-based detectors in terms of average accuracy computed separately on A-DJPEG and NA-DJPEG images. The average is taken over all the QF pairs used for training. As expected from the previous analysis, $\mathcal{C}_{\text{hist}}$ tends to learn characteristics of aligned DJPEG but fails in non-aligned case. Conversely, $\mathcal{C}_{\text{noise}}$ is the most stable solution being able to detect with almost the same accuracy both A-DJPEG and NA-DJPEG images. The $\mathcal{C}_{\text{pix}}$ approach still proves to be robust, but provides average accuracy lower than $\mathcal{C}_{\text{noise}}$.

Driven by the accurate performance of $\mathcal{C}_{\text{hist}}$ on A-DJPEG compression, we also investigated an alternative solution according to which the detection

for the mixed case is obtained by fusing the outputs of our best CNN-based detectors for the aligned and non-aligned case, through the use of a binary classifier. Specifically, we considered the output provided by $\mathcal{C}_{\text{hist}}$ trained on A-DJPEG images, and the output of $\mathcal{C}_{\text{noise}}$ trained in the NA-DJPEG case, as feature vector. By feeding this feature vector to a binary classifier (i.e., a random forest in our case), it is possible to further increase the final accuracy in the mixed case by up to 2%. However, other solutions and fusing strategies might be investigated. We leave a thorough investigation of this case to future studies.

5.2 Quality factor estimation

We report here the set of experiments we performed for quality factor estimation task (Chapter 3.2). We describe the dataset construction procedure, the evaluation methodology and then we move to results.

5.2.1 Dataset construction

For this task we considered a subset of RAISE dataset. We converted TIFF images in grayscale and then we compressed each of them with $\text{QF} = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. We extracted 100 patch of size 256×256 from each image, reaching the final dataset size of 284 000 images. The patches were selected as the top-100 scoring according to a sorting function that assign a score between 0 and 1 to them. Score tends to 1 as pixels intensity is not saturated and high texture occurs. We followed a common train-validation split policy, adopting 70% of the images for training set and 30% for validation set. The 10 classes were balanced i.e., each class representing 1/10 of the total amount of the images in both the datasets. During

splitting, we paid particular attention to put cropped patches belonging to the same original image only in one set (train or validation).

We built validation set in order to assess a good point in which stop the training and avoid overfitting over the training set. We carried out an accuracy measure over the validation set, but we also built a test set for the sake of completeness. The test set is made of nearly 150 000 images from the Dresden dataset, following the same aforementioned pipeline for compression and patch extraction.

5.2.2 Evaluation methodology

In order to fairly evaluate our CNN-based approach, we devised a common training-validation strategy. We resorted to LeNet-6 (Table 4.3) trained using stochastic gradient descent (SGD) algorithm with batch size (i.e., number of images used for SGD iteration) set to 64. Momentum was set to 0.9. Learning rate was set to 0.001 and was progressively decreased with a step down policy by a factor of 10 every 10 epochs. The maximum amount of epochs (i.e., the number of times the CNN sees all training data) was set to 30 to ensure network convergence. Initialization of CNN parameters has been performed using the method devised in [37].

The results are provided in terms of accuracy, namely the percentage of correctly classified images in the validation dataset.

5.2.3 Results

Figure 5.6 shows the train and validation loss and the validation accuracy over the training epochs. This particular chart gives us insights on how the learning capability of the network is exploited over time, i.e, how "fast" or "slow" the network is learning and if it is learning the correct parameters.

	QF10	**QF20**	**QF30**	**QF40**	**QF50**	**QF60**	**QF70**	**QF80**	**QF90**	**QF100**	**Per-class accuracy**
QF10	14791	104	3	2	1	0	0	9	0	0	99.2%
QF20	61	14568	272	9	0	0	0	0	0	0	97.71%
QF30	10	471	13939	474	12	2	0	2	0	0	93.49%
QF40	5	34	925	13429	498	14	5	0	0	0	90.07%
QF50	2	19	69	1054	12741	943	82	0	0	0	85.45%
QF60	3	5	22	84	1406	12571	714	2	99	4	84.31%
QF70	7	6	27	70	299	208	13060	2	1154	77	87.59%
QF80	6	4	2	3	5	36	128	11261	322	3143	75.53%
QF90	6	5	19	29	55	126	4839	29	8164	1638	54.76%
QF100	6	6	12	6	21	50	545	606	2155	11503	77.15%

Table 5.4: Confusion matrix over the test dataset.

Blue line represents the value of the training loss function, therefore we want it to decrease as fast as possible. We want a similar behavior for the green line as well, representing the validation loss function. A separation between the two of them (i.e, the training loss decreases while the validation loss increases) is the first symptom of overfitting over the training data, a situation we want to avoid. Orange line represents the validation accuracy.

Table 5.4 shows the confusion matrix for the various classes. Per-class accuracy decreases as the quality factor increases. This trend is justified by the fact that a low quality factor implies a stronger degradation of the image, leaving palpable traces. We notice a peak in accuracy for QF100. This can be explained by the fact that traces left by the compression with such a quality factor are negligible. This is somehow a characteristic feature for the network, therefore it is able to classify QF100 with a improved accuracy with respect to the trend. Anyway, the average accuracy is more than satisfying, having:

- Top-1 accuracy: **84.53%**
- Top-5 accuracy: **99.68%**

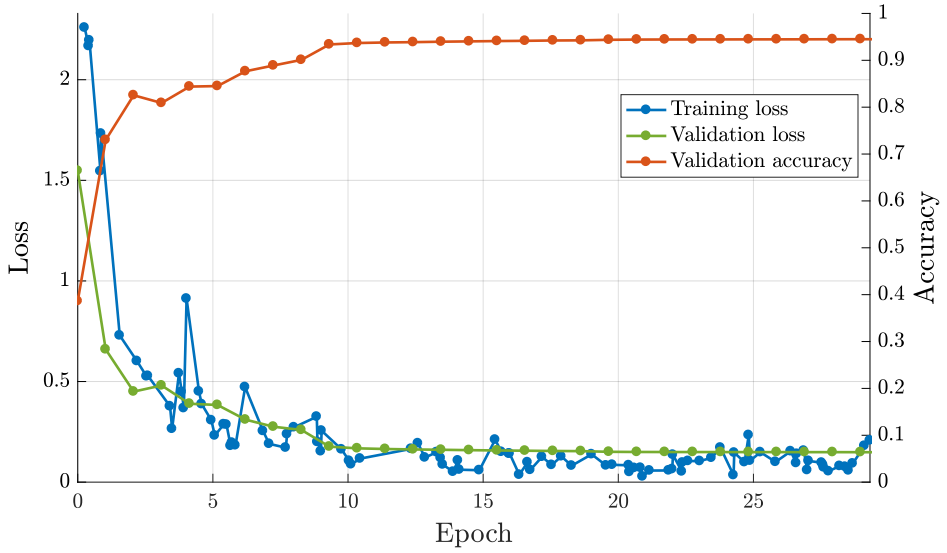


Figure 5.6: Train and validation loss, validation accuracy curves over training epochs for quality factor estimation task.

5.3 Software identification

We report here the set of experiments we performed for software identification task (Chapter 3.2). We describe the dataset construction procedure, the evaluation methodology and then we move to results.

5.3.1 Dataset construction

For this task we considered a subset of the Dresden JPEG dataset. In order to apply the smallest possible transformation by means of a software, we limited to open each JPEG image separately with Photoshop and GIMP and to save it according to some fixed compression levels. These levels are motivated by the fact that the two products have different scales for defining the compression level. In particular, Photoshop scale goes from 0 to 12, whereas GIMP scale follows the "standard" QF scale (Chapter 2.1) going

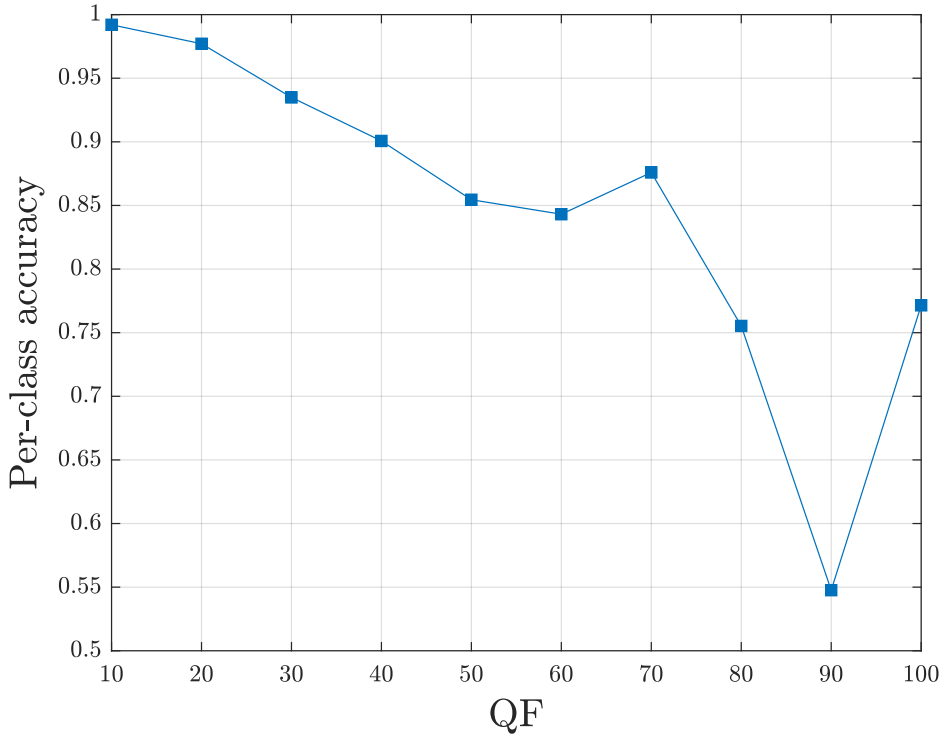


Figure 5.7: Per-class accuracy over the test dataset for quality factor estimation task.

from 0 to 100. The levels definition aside their software counterparts is depicted in Table 5.5. For this experiment we extracted patches of size 32×32 from the images.

5.3.2 Evaluation methodology

In order to fairly evaluate our CNN-based approach, we devised a common training-validation strategy. We resorted to LeNet-3 (Table 4.2) trained using Nesterov's accelerated gradient (NAG) algorithm [42] with batch size (i.e., number of images used for NAG iteration) set to 128. Momentum was set to 0.9. Learning rate was set to 0.1 and was progressively decreased with a step down policy by a factor of 10 every 10 epochs. The maximum amount of epochs (i.e., the number of times the CNN sees all training data) was set

Photoshop	GIMP	Compression level
0	16	low
3	33	mid-low
7	69	mid-high
10	94	high

Table 5.5: Definition of adopted compression levels with their software counterparts.

Compression level	Patch accuracy	Aggregate accuracy
low	0.933	0.998
mid-low	0.925	0.998
mid-high	0.918	0.994
high	0.794	0.967

Table 5.6: Accuracies for software identification task considering various compression levels.

to 30 to ensure network convergence. Initialization of CNN parameters has been performed using the method devised in [37].

The results are provided in terms of accuracy, namely the percentage of correctly classified images in the validation dataset.

5.3.3 Results

Figure 5.8 reports the train and validation loss and the validation accuracy over the training epochs for all the considered compression levels. Intuitively, the validation accuracy increases as the quality factor decreases. We explain such a behavior as a stronger degradation of the original image, hence a conspicuous characteristic trace of the particular software. Table 5.6 reports the accuracies for the various compression levels. The first column shows the

validation patch accuracy, i.e., the accuracy on single patch classification. The second column shows the validation aggregate accuracy, i.e, the result of the majority voting of all the patches belonging to the same image. We remark that in this experiment we are considering patches of size 32×32 pixel, considerably small with respect to the size of the original images. We reached high accuracies, especially with aggregation. This bodes well for localization tasks we want to explore in future.

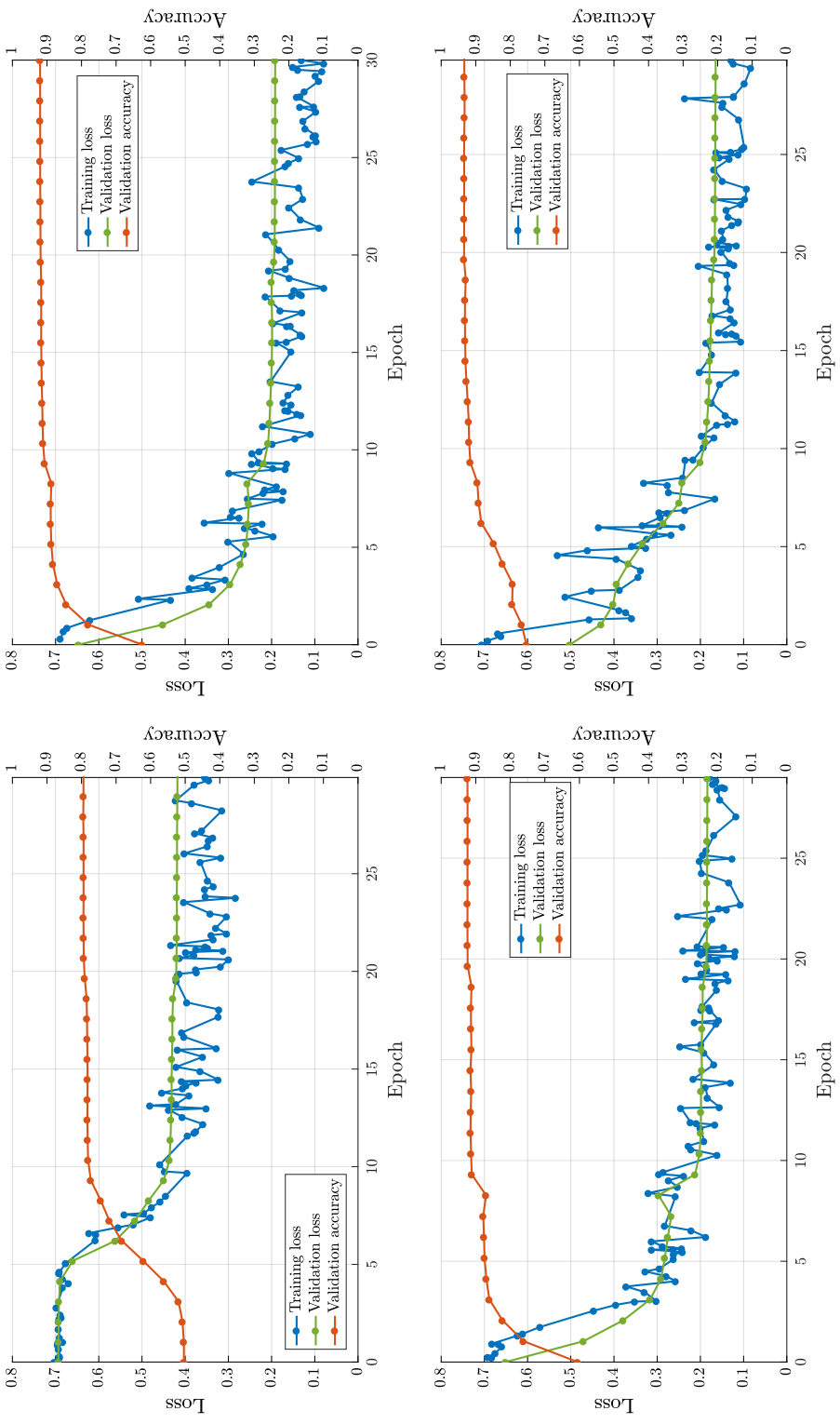


Figure 5.8: Train and validation loss, validation accuracy curves over training epochs for software identification task. Top-left: high; top-right: mid-high; bottom-left: mid-low; bottom-right: low

6.

Conclusions and future work

The ability of detecting the use of JPEG compression on images under analysis is considered of paramount importance in the forensic literature. For this reason, in this thesis we explored the use of Convolutional Neural Networks for various JPEG forensics tasks and scenarios.

We put most of our effort in investigation of double JPEG (DJPEG) compression detection, i.e., estimating whether a picture has been compressed once or twice. We extensively tested both the hypothesis of aligned (A-DJPEG) and non-aligned (NA-DJPEG) second compression. We considered "difficult" quality factors, i.e., those on which state-of-the-art methods fail or show poor results. Specifically, three different solutions were investigated: in one of them, the CNN is based on hand-crafted features extracted from the images; in the other two, the CNN is trained directly with images and their noise residuals, then features are self-learned by the CNN itself. Results show that CNN based on hand-crafted features allow to achieve better accuracies in the case of A-DJPEG. For the NA-DJPEG instead, the CNN based on self-learned features applied to the image noise residuals is shown to outperform the state-of-the-art in every tested scenario. Good performance are achieved even in the difficult cases in which the second quality factor is larger

than the first one and over small images. Besides, CNN based on self-learned features prove very robust to deviations between training and test conditions. Additionally, some preliminary experiments show the proposed CNN-based methods can also be successfully applied to simultaneously detect an aligned or non-aligned DJPEG compression.

In addition to DJPEG detection, we also investigated two different tasks as cases of study in order to proof the flexibility of convolutional neural networks for JPEG-based forensic works. To this purpose, we analyzed the problem of JPEG quality factor estimation, which was never tried before by means of CNNs to the best of our knowledge. Our detector was able to reach high accuracies over test dataset, especially with lower quality factors. Then, we focused on detection of the software suite used to JPEG compress images. Software identification can be considered a very tough task, as it is not trivial to establish a common baseline between the investigated software products. Plus, the traces they leave might be negligible at first glance. We remark that for this specific case we designed our pipeline to work with very small patches. Nonetheless, our detector hit very good accuracies considering the single patch and excellent ones considering the whole image.

Considering the outcome of the performed research on DJPEG detection, future work will be devoted to the study of fusion techniques to make the most out of each network in a mixed aligned and non-aligned DJPEG case. Moreover, as we decided to work with patches instead of whole images still obtaining very satisfactory results, the development of tampering localization techniques based on our detector is a natural follow up. Indeed, we intent to develop a proper aggregation policy over the entire image area once detection result for each small patch is obtained.

In this thesis we focused on double JPEG compression as simplification

of multiple JPEG compression. Considering $N > 2$ compressions paves the way to a series of exciting problems, such as knowing how many times an image was compressed and which quality factors were adopted. We could investigate the history of an image posted on a social network in first place and then reposted several times on others. We could determine the phylogeny between different versions of the same image in order to establish which one was the original content.

In addition to double JPEG detection, also results obtained through the two considered cases of study pave the way to interesting future work. In this preliminary work, we only considered two software products, which is a very small subset of all the available image editing products indeed. Extending our two-class classification to N-class in order to perform an automatic estimation of the editing tool would definitely be more comprehensive. Moreover, with the same rationale, we can study traces left by compression applied by different multimedia sharing platforms, thus aiming to estimate the website an image is coming from.

Bibliography

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] Q. Wang and R. Zhang, "Double JPEG compression forensics based on a convolutional neural network," *EURASIP Journal on Information Security*, vol. 2016, no. 1, pp. 23, 2016.

[3] J. Fridrich and J. Kodovsky, "Rich models for steganalysis of digital images," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 3, pp. 868–882, June 2012.

[4] T. Bianchi and A. Piva, "Detection of nonaligned double JPEG compression based on integer periodicity maps," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 7, pp. 842–848, 2012.

[5] A. Rocha, W. Scheirer, T. Boult, and S. Goldenstein, "Vision of the unseen: Current trends and challenges in digital image and video forensics," *ACM Computing Surveys*, vol. 43, pp. 1–42, 2011.

[6] A. Piva, "An overview on image forensics," *ISRN Signal Processing*, vol. 2013, pp. 22, 2013.

[7] A. C. Popescu and H. Farid, "Statistical tools for digital forensics," in *International Conference on Information Hiding*, 2004.

[8] C. Chen, Y. Q. Shi, and W. Su, "A machine learning based scheme for double JPEG compression detection," in *IEEE International Conference on Pattern Recognition (ICPR)*, 2008.

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec 1989.

[10] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time-series," in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed., pp. 255–258. MIT Press, Cambridge, MA, 1995.

[11] Z. Fan and R. L. de Queiroz, "Identification of bitmap compression history: JPEG detection and quantizer estimation," *IEEE Transactions on Image Processing*, vol. 12, no. 2, pp. 230–235, Feb 2003.

[12] Z. Wang, A. C. Bovik, and B. L. Evan, "Blind measurement of blocking artifacts in images," in *Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101)*, 2000, vol. 3, pp. 981–984 vol.3.

[13] Z. Fan and R. de Queiroz, "Maximum likelihood estimation of JPEG quantization table in the identification of bitmap compression history," in *IEEE International Conference on Image Processing (ICIP)*, 2000, vol. 1, pp. 948–951 vol.1.

[14] W. Luo, J. Huang, and G. Qiu, "JPEG error analysis and its applications to digital image forensics," *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 3, pp. 480–491, Sept 2010.

[15] T. Bianchi and A. Piva, "Detection of non-aligned double JPEG compression with estimation of primary compression parameters," in *2011 18th IEEE International Conference on Image Processing*, Sept 2011, pp. 1929–1932.

[16] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[17] D. S. Taubman and M. W. Marcellin, *JPEG 2000: Image Compression Fundamentals, Standards and Practice*, Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[18] J. Lukas and J. Fridrich, "Estimation of Primary Quantization Matrix in Double Compressed JPEG Images," *Digital Forensic Research Workshop*, 2003.

[19] S. Milani, M. Tagliasacchi, and S. Tubaro, "Discriminating multiple jpeg compression using first digit features," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2012, pp. 2253–2256.

[20] S. Milani, M. Tagliasacchi, and S. Tubaro, "Discriminating multiple JPEG compressions using first digit features," *APSIPA Transactions on Signal and Information Processing*, vol. 3, 2014.

[21] M. Barni, A. Costanzo, and L. Sabatini, "Identification of cut & paste tampering by means of double-JPEG detection and image segmentation," *ISCAS 2010 - 2010 IEEE International Symposium on Circuits*

and Systems: Nano-Bio Circuit Fabrics and Systems, vol. 8, pp. 1687–1690, 2010.

[22] T. Bianchi and A. Piva, "Image forgery localization via block-grained analysis of JPEG artifacts," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 3, pp. 1003–1017, 2012.

[23] V. Nair and G. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *International Conference on Machine Learning (ICML)*, 2010.

[24] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, *Efficient BackProp*, pp. 9–50, Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[25] C. Jiansheng, K. Xiangui, L. Ye, and Z. J. Wang, "Median Filtering Forensics Based on Convolutional Neural Networks," *IEEE Signal Processing Letters*, vol. 22, no. 11, pp. 1849–1853, 2015.

[26] B. Bayar and M. C. Stamm, "A Deep Learning Approach to Universal Image Manipulation Detection Using a New Convolutional Layer," *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*, pp. 5–10, 2016.

[27] 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.

[28] L. Bondi, L. Baroffio, D. Guera, P. Bestagini, E. J. Delp, and S. Tubaro, "First Steps Toward Camera Model Identification With Convolutional Neural Networks," *IEEE Signal Processing Letters*, vol. 24, pp. 259–263, 2017.

[29] L. Bondi, D. Guera, L. Baroffio, P. Bestagini, E. J. Delp, and S. Tubaro, "A preliminary study on convolutional neural networks for camera model identification," in *IS&T International Symposium on Electronic Imaging: Media Watermarking, Security, and Forensics*, 2017.

[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1–9.

[31] D. Cozzolino, D. Gragnaniello, and L. Verdoliva, "Image forgery localization through the fusion of camera-based, feature-based and pixel-based techniques," *Proceedings of the IEEE International Conference on Image Processing*, pp. 5302–5306, October 2014, Paris, France.

[32] J. Lukáš, J. Fridrich, and M. Goljan, "Digital camera identification from sensor pattern noise," *IEEE Transactions on Information Forensics and Security*, vol. 1, no. 2, pp. 205–214, June 2006.

[33] M. K. Mihcak, I. Kozintsev, and K. Ramchandran, "Spatially adaptive statistical modeling of wavelet image coefficients and its application to denoising," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Mar 1999, vol. 6, pp. 3253–3256.

[34] D.-T. Dang-Nguyen, C. Pasquini, V. Conotter, and G. Boato, "RAISE: A raw images dataset for digital image forensics," in *ACM Multimedia Systems Conference*, 2015.

[35] T. Gloe and R. Bhme, "The 'Dresden Image Database' for benchmarking digital image forensics," in *Proceedings of the 25th Symposium On Applied Computing (ACM SAC 2010)*, 2010, vol. 2, pp. 1585–1591.

[36] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv:1408.5093*, 2014.

[37] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

[38] T. Bianchi, A. De Rosa, and A. Piva, "Improved DCT coefficient analysis for forgery localization in JPEG images," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011.

[39] I. Amerini, R. Becarelli, R. Caldelli, and A. Del Mastio, "Splicing forgeries localization through the use of first digit features," in *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2014.

[40] B. Li, Y. Shi, and J. Huang, "Detecting doubly compressed JPEG images by using mode based first digit features," in *IEEE Workshop on Multimedia Signal Processing (MMSP)*, 2008.

[41] T. Pevny and J. Fridrich, "Detection of double-compression in JPEG images for applications in steganography," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 3, pp. 247–258, 2008.

[42] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. 2013, ICML'13, pp. III–1139–III–1147, JMLR.org.