JPEG Implementation Forensics Based on Eigen-Algorithms

Nicolò Bonettini, Luca Bondi, Paolo Bestagini, Stefano Tubaro
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy
nicolo.bonettini / luca.bondi / paolo.bestagini / stefano.tubaro@polimi.it

Abstract

Due to the widespread diffusion of JPEG compression standard, image forensic researchers have developed a series of techniques to reverse engineer specific JPEG-based information (e.g., the used quantization matrix, presence of double JPEG compression, JPEG grid alignment, etc.). In this paper, we focus on the detection of traces left on images by the use of different JPEG implementations (e.g., characteristic of proprietary software suites). Specifically, given a JPEG image under analysis, we propose a compact descriptor that enables to distinguish which JPEG implementation has been used. This is done considering the challenging scenario in which the same quantization matrix is used. Results show that it is possible to distinguish between two popular JPEG implementations, as well as to adapt the proposed methodology to additional JPEG-based forensic problems.

1. Introduction

Nowadays, forging a digital image in a believable manner is at everyone's hand. Anyone can use professional applications (e.g, Adobe Photoshop, etc.), online tools (e.g., Pixlr, etc.) or even smartphone apps (e.g., Google Snapseed, FaceSwap, etc.) to visually modify a photograph. Despite indicating a great technological advancement, this possibility poses a new threat every time pictures are used in a court of law, as evidence on newspapers to spread information, or for opinion formation scenarios. For this reason, multimedia forensics researchers have developed a set of tools to assess image authenticity and integrity [25, 27].

The idea behind passive image forensic tools is that non-invertible image editing operations leave peculiar footprints. These traces can be reverse engineered to detect the use of editing operations [24].

Among the many different traces that can be exploited (e.g., left by resampling [14], morphological operations [10], blurring [2], etc.), those left by JPEG compression have been deeply studied in the literature [20]. As a matter of fact, JPEG is among the most widespread image compression standards. Therefore, being able to forensically analyze JPEG images means being able to analyze the vast majority of pictures available online.

Capturing JPEG traces enables to expose different kinds of image forgeries. As an example, the authors of [11] show how to detect whether an image is uncompressed or it has been JPEG compressed at least once during its life cycle. Methods developed in [28] enable to detect the used JPEG quantization matrix in case of single or double JPEG compressions. Several algorithms have been proposed to detect double JPEG compression, either when JPEG compression grids are aligned [5, 22, 29] or misaligned [6] between the first and second compression. Other solutions are able to specifically tell how many times (up to three or four) an image has been JPEG compressed [17, 19]. Moreover, also specific kinds of convolutional neural networks have been tailored to the detection of JPEG traces [3]. Needless to say, when any of these methods is used locally on image regions, it becomes a powerful image tampering localization tool for exposing image splicings and compositions [4, 7].

In this paper, we focus on capturing traces left by different JPEG implementations in order to distinguish between images that have been compressed with different software suites, even if the very same quantization matrix has been used. As a matter of fact, the authors of [1] have recently shown that it is possible to detect whether an image has been compressed with different JPEG implementations according to the used quantization rule (i.e., flooring, ceiling or rounding). Motivated by this finding, we explore the rationale behind eigen-algorithms [16] to propose a compact descriptor that captures JPEG implementation traces. Specifically, given a JPEG image under analysis, we re-encode it using different controlled implementations of JPEG compression algorithm. We then compare the image under analysis with the re-compressed versions to expose salient differences. These differences are collected into a descriptor, that can be fed to a simple supervised classifier to detect the JPEG implementation used to originally encode the image under analysis.

The rest of the paper is structured as it follows. In Section 2, we discuss a series of forensic use cases that can benefit from the use of the proposed descriptor and classification technique. In Section 3, we provide all the details of the proposed method. In Section 4, we describe the performed experiments and discuss the achieved results. Finally, Section 5 concludes the paper.

2. Motivations

In this section, we discuss a series of use cases in which the ability of detecting traces specific of a JPEG implementation can be useful from a forensics viewpoint.

2.1. Detection of Editing Software

In most situations, tampering a digital image involves decoding a compressed image in the pixel domain, editing it with one or more software tools, and finally save the image in a compressed format. When the selected compression format is the widespread JPEG standard, different software suites may leverage proprietary solutions to achieve better JPEG compression in terms of bitrate and quality trade-off.

Being able to detect which software has been used (e.g., Photoshop, GIMP, etc.) to save an image in JPEG format can be of paramount importance from two different reasons: (i) if the image is recognized to come from an editing suite, then the image cannot be considered pristine with high probability; (ii) if the specific used software can be detected, the list of possible attackers can be narrowed down.

2.2. Detection of Device Manufacturer

As for different image editing suites, also different camera manufacturers make use of different JPEG implementations [1]. Even though many robust state-of-the-art detectors are already tailored to camera model attribution problem [15, 9, 8], capturing and analyzing JPEG traces could be helpful to narrow down the camera model search. As a matter of fact, it could be possible to first use traces left by JPEG compression to attribute images to the manufacturer of the camera used to shot it, and then apply a tailored camera model solution to detect the specific model as a refinement step.

2.3. Detection of JPEG Anti-Forensics

Forging an image leaving absolutely no traces is not an easy task. Indeed, an expert attacker has to cope with at least two different kinds of traces he might leave: (i) traces left in the pixel domain by processing operations (e.g., multiple compressions, resizing, noise addition, copy-move, etc.) that can be captured by one or more of the many passive image forensic tools proposed in the literature [25, 24, 27]; (ii) traces left in headers and metadata (e.g., missing fields with respect to original images coming from cameras, name of the used image processing suite, incoherent geo-tagging information, etc.) that can be easily detected by simple header inspection.

One of the most trivial mistakes an attacker might fall into is to edit a JPEG photograph coming from a specific camera model, then save it using one of the default JPEG quantization matrices provided by the used editing software. As a matter of fact, different camera models make use of different custom JPEG quantization matrix. The quantization matrix is stored into each JPEG file as it is needed at the decoding step. If an analyst detects that the quantization matrix stored in the JPEG header is not compatible with the ones used by the camera manufacturer, then the forgery is easily spotted.

To overcome this issue, an expert attacker is left with two options: (i) save the edited photograph with any quantization matrix from the considered manufacturer; (ii) save the edited photograph using the very same quantization matrix of the original picture. However, the first option may leave traces typical of double JPEG compression [23, 6]. Therefore, the second option is the safest solution for the attacker. However, as the attacker does not have access to the manufacturer JPEG implementation, the original JPEG photograph and the edited JPEG pictures will still show different JPEG-traces despite the use of the same quantization matrix.

In this paper, we show that the proposed descriptor allows to capture traces of this specific anti-forensic operation, hardly detectable otherwise. In other words, we are able to detect whether an image has been compressed with one (e.g., camera) or another (e.g., editing software) JPEG implementation, regardless of the used quantization matrix.

3. Proposed Method

In this section we report how to extract the proposed JPEG-based descriptor $\mathbf{f}$ from a JPEG image $\mathbf{I}$, and how to possibly use it for classification tasks based on JPEG image history.

3.1. Descriptor Extraction

Let us consider an image in the pixel domain $\mathbf{I}$. For the sake of notation simplicity and without loss of generality, let us consider $\mathbf{I}$ being a grayscale image. Let us denote as $\tilde{\mathbf{I}}$ the JPEG encoded version of $\mathbf{I}$ in the discrete cosine transform (DCT) domain. In a nutshell, $\tilde{\mathbf{I}}$ is obtained by: (i) splitting $\mathbf{I}$ into non-overlapping 8×8 pixel blocks; (ii) applying JPEG-defined DCT transform to each block; (iii) quantizing each DCT coefficient according to a given quantization matrix and a quantization rule (e.g., rounding, ceiling, etc.); (iv) tiling all quantized DCT blocks back together to obtain a matrix $\tilde{\mathbf{I}}$ having the same size of $\mathbf{I}$. Notice that, from an implementational view-point, if an image is already available in compressed JPEG format, its DCT representation $\tilde{\mathbf{I}}$

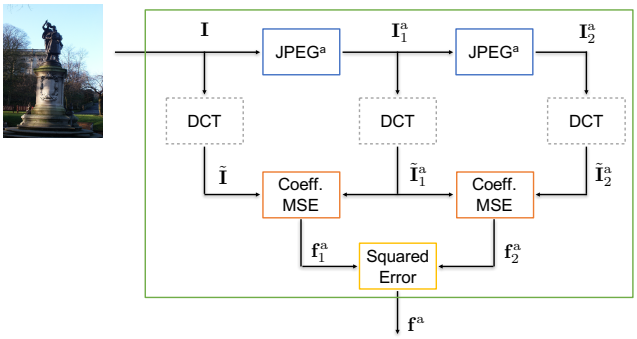

Figure 1: Feature extraction pipeline using a single JPEG implementation (i.e., JPEG$^{\text{a}}$) as eigen-algorithm.

is already available in the bitstream, thus there is no need to re-compute $\tilde{\mathbf{I}}$ through DCT transform and quantization.

As JPEG implementations may differ (e.g., due to different quantization rules [1], or even more trivially due to the use of different quantization matrices), we can exploit the eigen-algorithm [16] idea to capture JPEG-based traces. This means, we can re-compress image $\mathbf{I}$ using different JPEG implementations, and check the differences introduced by each implementation with respect to the original image. Deviations and perturbations introduced by different implementations can characterize the implementation itself. Indeed, also works aiming at detecting double compression with the same quantization matrix leverage DCT residual statistical changes after multiple JPEG compressions [13]. In the following, we report how to capture these deviations by means of a compact feature vector using a single, and then multiple eigen-algorithms.

3.1.1 Single Eigen-Algorithm

To extract a compact feature vector capturing traces left by a single JPEG implementation, we proceed according to the pipeline shown in Fig. 1. Let us consider we have access to a specific JPEG implementation denoted as JPEG$^{\text{a}}$. By applying JPEG$^{\text{a}}$ compression to $\mathbf{I}$ once, we obtain the decoded image $\mathbf{I}_1^{\text{a}}$, and the respective DCT representation $\tilde{\mathbf{I}}_1^{\text{a}}$. By applying JPEG$^{\text{a}}$ compression to $\mathbf{I}$ twice, we obtain the decoded image $\mathbf{I}_2^{\text{a}}$, and the respective DCT representation $\tilde{\mathbf{I}}_2^{\text{a}}$.

We first compare $\tilde{\mathbf{I}}$ and $\tilde{\mathbf{I}}_1^{\text{a}}$ by computing the mean squared error (MSE) for each of the 64 DCT coefficients, thus obtaining the 64-element feature vector $\mathbf{f}_1^{\text{a}}$, whose i-th element is defined as

$$\mathbf{f}_1^{\text{a}}(i) = \frac{1}{B}\sum_{b=1}^{B}\left|\tilde{\mathbf{I}}(i,b) - \tilde{\mathbf{I}}_1^{\text{a}}(i,b)\right|^2,\ i \in [1,64], \quad (1)$$

where i is the index of a DCT coefficient, B is the number of 8×8 blocks the image is split into during JPEG compression and b is the index of an 8×8 block.

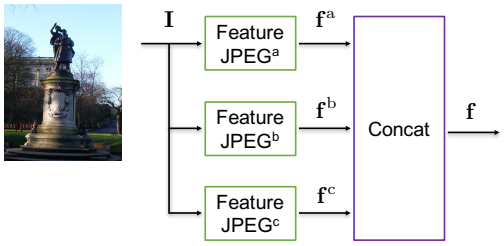

Figure 2: Feature extraction pipeline using multiple JPEG implementations as eigen-algorithms.

Then, we compare $\tilde{\mathbf{I}}_1^{\text{a}}$ and $\tilde{\mathbf{I}}_2^{\text{a}}$ by computing the MSE for each of the 64 DCT coefficients in the same way, thus obtaining the 64-element feature vector $\mathbf{f}_2^{\text{a}}$. Finally, we obtain the final descriptor by computing the element-wise squared error between $\mathbf{f}_1^{\text{a}}$ and $\mathbf{f}_2^{\text{a}}$, thus obtaining $\mathbf{f}^{\text{a}}$, whose i-th element is defined as

$$\mathbf{f}^{\text{a}}(i) = \left|\mathbf{f}_1^{\text{a}}(i) - \mathbf{f}_2^{\text{a}}(i)\right|^2,\ i \in [1,64]. \quad (2)$$

Also $\mathbf{f}^{\text{a}}$ is a 64-element feature vector.

3.1.2 Multiple Eigen-Algorithms

When multiple JPEG implementations are available, we can compute different feature vectors and concatenate them according to the pipeline reported in Fig. 2. In this paper, we consider three different JPEG implementations: (i) JPEG$^{\text{a}}$, which uses rounding quantization rule and gives rise to feature vector $\mathbf{f}^{\text{a}}$; (ii) JPEG$^{\text{b}}$, which uses flooring quantization rule and gives rise to feature vector $\mathbf{f}^{\text{b}}$; (iii) JPEG$^{\text{c}}$, which uses ceiling quantization rule and gives rise to feature vector $\mathbf{f}^{\text{c}}$. The three feature vectors obtained with the different implementations are concatenated in a single vector as

$$\mathbf{f} = \left[\mathbf{f}^{\text{a}},\ \mathbf{f}^{\text{b}},\ \mathbf{f}^{\text{c}}\right]. \quad (3)$$

The overall feature vector $\mathbf{f}$ is composed by $64 \times 3 = 192$ elements, and can be used for JPEG-based classification tasks.

3.2. Classification

In principle, as $\mathbf{f}$ is a feature vector containing information about JPEG-based image deviations, it can be fed to any kind of supervised or unsupervised learning algorithm. As in this paper we are more interested in the amount of JPEG-based information that the proposed feature is able to capture, rather than in developing a powerful classifier, we make use of a simple classification technique, i.e., a random forest classifier.

For all proposed tasks, we train a random forest classifier without further optimization using feature vectors extracted from a set of training images. When a new image has to be classified, we simply extract the feature vector $\mathbf{f}$ from

the image then feed $\mathbf{f}$ to the trained random forest classifier. This predicts the likelihood of the image to belong to any class. The class with highest likelihood is selected as candidate solution. Depending on the application, it is possible to set a custom threshold and only consider results for images with an estimated likelihood higher than the threshold.

4. Experiments and Results

In this section we report all the performed experiments and results, separately considering each different use case. A reference implementation of the presented features extractor is available online[1]. All the experiments are carried out on a workstation equipped with Ubuntu 16.04, Python 3.6, PIL 5.2.0, and libjpeg 9.2.0. JPEG compression with Adobe Photoshop CC 2017 is performed on macOS Sierra. It is worth noting that the forensics analyst doesn't need to know which JPEG implementations are used to compress the images.

As all the experiments involve the use of a supervised classifier (i.e., a random forest), we always proceed in the following way: (i) given a dataset, we randomly split its samples into 50% training and 50% validation; (ii) the random forest is trained using default parameters[2] on the training set; (iii) results are reported on the test set. As no hyper-parameters tuning is performed, there is no need for a validation set of data. This is done on purpose in order to evaluate the proposed feature discrimination capability with a simple classifier, rather than our ability of fully optimizing a machine-learning technique.

4.1. Photoshop vs. PIL (Single Compression)

The first experiment we performed is a two-class classification problem: to detect whether a single compressed JPEG image has been saved using Adobe Photoshop CC 2017 (hereinafter Photoshop) or the Python Imaging Library (PIL), considering that the same quantization matrix has been used.

To this purpose, we built a dataset starting from the 1 338 uncompressed color images at 512×384 pixel resolution of the UCID dataset [26]. Notice that considering low resolution images makes the problem more challenging. Indeed, less pixels lead to less reliable statistics, thus feature vectors.

We first JPEG compressed each uncompressed image using Photoshop at different JPEG quality levels, thus obtaining four distinct datasets: $\mathcal{D}^1_{\text{PS}}$, $\mathcal{D}^3_{\text{PS}}$, $\mathcal{D}^5_{\text{PS}}$ and $\mathcal{D}^7_{\text{PS}}$ consisting of 1 338 single compressed images at quality 1, 3, 5 and 7, respectively[3]. Then, we compressed each UCID uncompressed image with PIL, forcing the use of the respective JPEG quantization matrix used by Photoshop, thus obtaining datasets $\mathcal{D}^1_{\text{PIL}}$, $\mathcal{D}^3_{\text{PIL}}$, $\mathcal{D}^5_{\text{PIL}}$ and $\mathcal{D}^7_{\text{PIL}}$. With this setup, we extracted the proposed feature vector from the luminance component of every picture, and trained a different classifier for each dataset pair (i.e., $\mathcal{D}^1_{\text{PS}}$ vs. $\mathcal{D}^1_{\text{PIL}}$, $\mathcal{D}^3_{\text{PS}}$ vs. $\mathcal{D}^3_{\text{PIL}}$, etc.). Notice that, for these experiments, an image in $\mathcal{D}^1_{\text{PS}}$ and the relative image in $\mathcal{D}^1_{\text{PIL}}$ are single JPEG compressed with the very same quantization matrix. Their differences are only due to the used software, i.e., JPEG implementation.

(a) Original (b) Low quality (c) High quality

Figure 3: Example of different Photoshop quality factors.

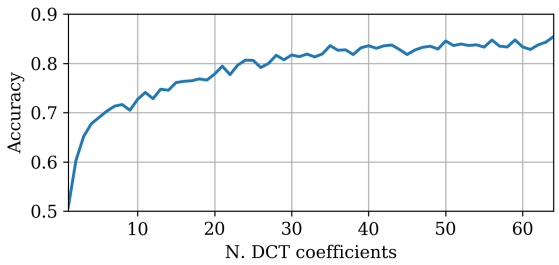

Figure 4: Accuracy for the case $\mathcal{D}^1_{\text{PS}}$ vs. $\mathcal{D}^1_{\text{PIL}}$ while increasing the amount of used DCT coefficients in zig-zag order.

Fig. 3 shows an example of original UCID image, the low quality Photoshop version from $\mathcal{D}^1_{\text{PS}}$, and the higher quality Photoshop version from $\mathcal{D}^7_{\text{PS}}$. It is possible to notice that images are not strongly visually degraded by JPEG artifacts.

Fig. 4 shows the effect on accuracy of using a different amount of DCT coefficients to build the proposed feature vector, considering the scenario $\mathcal{D}^1_{\text{PS}}$ vs. $\mathcal{D}^1_{\text{PIL}}$. It is possible to notice that, by increasingly selecting a greater amount of DCT coefficients read in zig-zag mode, accuracy increases. This is expected, as the more the coefficients, the better the captured JPEG deviations. This validates the idea of using all 64 JPEG DCT coefficients.

Table 1a reports results in terms of true positive rate (i.e., Photoshop images correctly detected), true negative rate (PIL images correctly detected) and accuracy for each dataset pair. It is possible to notice that, the lower the JPEG quality, the better the results. Indeed, accuracy for JPEG low quality images is higher than 86%, and it drops to 75% when high quality images are considered. This behavior is not surprising, as it is reasonable to assume that low quality

[1] https://github.com/polimi-ispl/jpeg-eigen

[2] According to scikit-learn implementation [21].

[3] Photoshop JPEG quality ranges from 0 (very low) to 12 (very high).

Table 1: Photoshop vs. PIL detection in case of single and double compression. TPR and TNR are the fraction of correctly detected Photoshop and PIL images, respectively.

(a) Single Compression

Task	TPR	TNR	Accuracy
$\mathcal{D}^1_{\text{PS}}$ vs. $\mathcal{D}^1_{\text{PIL}}$	0.89	0.82	0.86
$\mathcal{D}^3_{\text{PS}}$ vs. $\mathcal{D}^3_{\text{PIL}}$	0.82	0.79	0.81
$\mathcal{D}^5_{\text{PS}}$ vs. $\mathcal{D}^5_{\text{PIL}}$	0.80	0.74	0.77
$\mathcal{D}^7_{\text{PS}}$ vs. $\mathcal{D}^7_{\text{PIL}}$	0.77	0.73	0.75

(b) Double Compression

Task	TPR	TNR	Accuracy
$\ddot{\mathcal{D}}^1_{\text{PS}}$ vs. $\ddot{\mathcal{D}}^1_{\text{PIL}}$	0.95	0.83	0.89
$\ddot{\mathcal{D}}^3_{\text{PS}}$ vs. $\ddot{\mathcal{D}}^3_{\text{PIL}}$	0.83	0.70	0.77
$\ddot{\mathcal{D}}^5_{\text{PS}}$ vs. $\ddot{\mathcal{D}}^5_{\text{PIL}}$	0.75	0.63	0.69
$\ddot{\mathcal{D}}^7_{\text{PS}}$ vs. $\ddot{\mathcal{D}}^7_{\text{PIL}}$	0.72	0.57	0.65

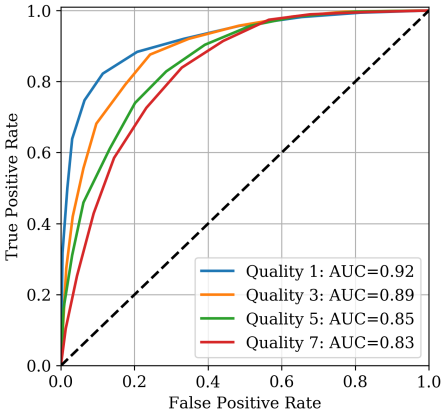

Figure 5: ROC curves showing the different possible working points for each dataset pair according to the used JPEG quality in case of single compression.

factors lead to more pronounced artifacts.

Finally, Fig. 5 shows the receiver operating characteristic (ROC) curve for each dataset pair obtained thresholding the soft output of each random forest classifier. Indeed, for a two-class problem, the output of the classifier can be interpreted as the likelihood of an image to belong to a single class. It is possible to notice that by properly selecting this threshold, it is possible to enforce a specific working condition in terms of true and false positive rates.

4.2. Photoshop vs. PIL (Double Compression)

The second experiment we performed is a more challenging version of the first one: to detect whether an image has been originally JPEG compressed using Photoshop or PIL, given that afterward it is re-compressed with the same quantization matrix using PIL.

For this experiment, we took all previously built datasets,

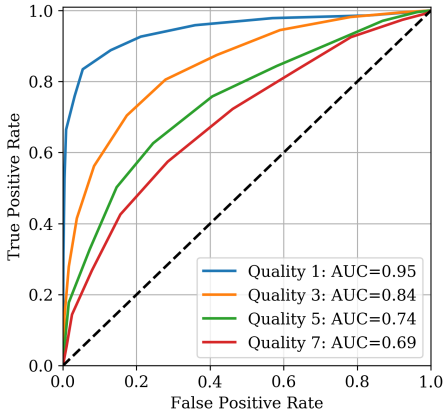

Figure 6: ROC curves showing the different possible working points for each dataset pair according to the used JPEG quality in case of double compression.

and re-compressed each image using PIL and the same quantization matrix used for the first compression. We therefore generated datasets $\ddot{\mathcal{D}}^1_{\text{PS}}$, $\ddot{\mathcal{D}}^3_{\text{PS}}$, $\ddot{\mathcal{D}}^5_{\text{PS}}$, $\ddot{\mathcal{D}}^7_{\text{PS}}$ (i.e., first compression with Photoshop and second with PIL), and $\ddot{\mathcal{D}}^1_{\text{PIL}}$, $\ddot{\mathcal{D}}^3_{\text{PIL}}$, $\ddot{\mathcal{D}}^5_{\text{PIL}}$, $\ddot{\mathcal{D}}^7_{\text{PIL}}$ (i.e., both compressions with PIL). Notice that, each image in $\ddot{\mathcal{D}}^1_{\text{PS}}$ has been double compressed with the same quantization matrix, and the relative image in $\ddot{\mathcal{D}}^1_{\text{PIL}}$ underwent the same processing. The only difference is the software used for the first compression step (i.e., Photoshop or PIL). Also in this scenario, we analyzed each dataset pair separately according to their quality factors.

Table 1b reports the true positive rate (i.e., Photoshop images correctly detected), true negative rate (PIL images correctly detected) and accuracy for each dataset pair. It is possible to notice also this time that the accuracy increases for low JPEG qualities. Indeed, accuracy ranges from 89% for low quality images, to 65% for higher quality pictures. However, given the challenging task, we can conclude that the proposed feature vector is still a viable solution toward capturing JPEG implementation traces.

Finally, Fig. 6 reports the ROC curves obtained thresholding random forest outputs for each dataset pair. It is possible to notice that, in case of double compression, low JPEG quality factors lead to better results compared to single JPEG compression scenario. Conversely, if two JPEG compressions at high quality are applied, traces of the used software are hindered.

4.3. Device Manufacturer

The third experiment we performed can be considered a multi-class classification problem: to detect the manufacturer of the camera used to shot a JPEG image within a closed set of possible known vendors.

To this purpose, we selected all images from Dresden Image Dataset [12]. This dataset is composed by more than $16\,000$ images belonging to almost 30 camera models from

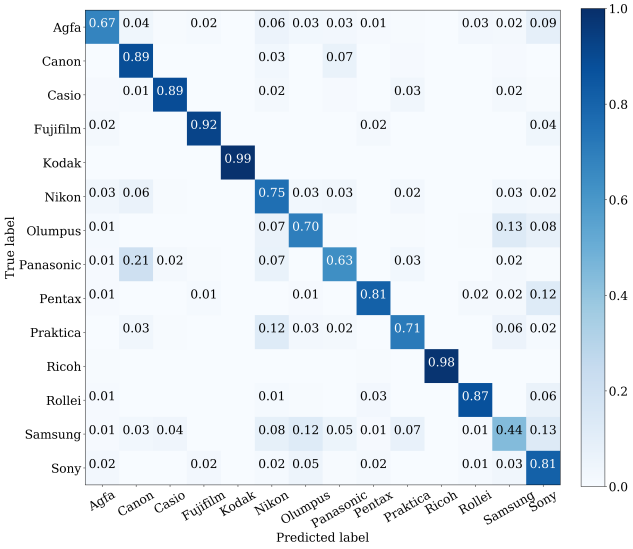

Figure 7: Camera manufacturer detection confusion matrix. Entries smaller than 1% are not reported.

14 different camera manufacturers.

Fig. 7 shows the confusion matrix reporting all misclassification errors for the 14 considered camera brands. The overall accuracy in this case is slightly higher than 79%.

Considering this specific experiment, we believe it is important to make an additional consideration. Many state-of-the-art solutions already exist to solve camera attribution problems in all their shapes (e.g., detecting the camera brand, model, instance, etc.) with high accuracy results. However, these methods do not simply rely on JPEG information. Conversely they exploit many kinds of different camera-related traces (e.g., sensor pattern noise, color filter array interpolations, lens aberrations, etc.). Therefore, we still find interesting to show how it is possible to solve camera brand detection problem, just relying on JPEG traces, even with reduced accuracy. Indeed, we believe that the proposed feature could be used as additional input to help other camera model detection algorithm in the literature.

4.4. JPEG Anti-Forensics

The last experiment we performed consists in solving the following two-class problem related to the JPEG anti-forensic scenario: to detect whether a JPEG image is a pristine one coming directly from a camera, or if it has been edited and re-compressed with the same JPEG quantization matrix.

To this purpose, we utilized a new, state-of-the-art dataset from NIST called Nimble Challenge 2017 (NC2017) for image manipulation detection [18]. This dataset consists of 3 415 JPEG compressed images split into two classes: 916 images are pristine, as generated by a camera firmware, and 2 499 images are forged. The latter have been JPEG re-compressed with the aforementioned anti-forensic technique, thus a forged image file has the same quantization matrix as the pristine image file that serves as background. Notice that, as camera models JPEG implementations are not available, two important differences can be observed comparing images belonging to the two classes: (i) forged images are at least double JPEG compressed (if not multiple JPEG compressed); (ii) the JPEG implementation used for the final anti-forensics JPEG compression is most likely different than any JPEG implementation used for pristine images. Fig.8 shows the ROC curve obtained by comparing the random forest output with different thresholds for the anti-forensic detection task. This shows that it is possible to distinguish pristine images from those edited and re-saved with the JPEG anti-forensic technique with a value of area under the curve of 0.82.

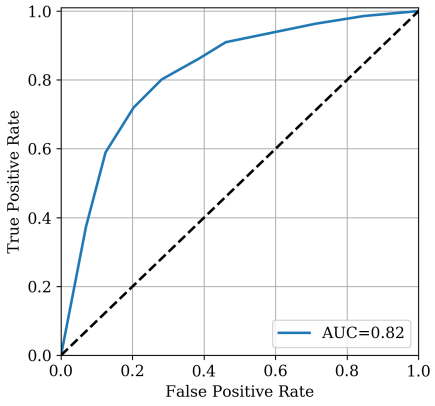

Figure 8: ROC curve obtained on NC2017 dataset for the JPEG antiforensic detection task.

5. Conclusions

In this paper, we presented a novel descriptor based on the eigen-algorithms idea to capture traces left by different JPEG implementations. The rationale behind the proposed method is that it is possible to analyze a JPEG image under analysis by re-compressing it multiple times with a set of available JPEG implementations. Deviations introduced by eigen-algorithms enable to characterize unknown JPEG implementations, thus allowing to detect the implementation used to compress an image under analysis.

The proposed method has been tested considering different challenging use cases. These range from software detection (i.e., Photoshop vs. PIL), to detection of a JPEG anti-forensic technique. Results show that the proposed solution achieves high accuracy on several different tasks, and that it benefits from low quality JPEG compression factors. Moreover, the proposed feature vector can be interestingly used to cope with camera brand attribution as well.

Future work will be devoted to the local application of the proposed technique in order to detect local editing operations such as splicings and copy-paste.

References

[1] S. Agarwal and H. Farid. Photo forensics from JPEG dimples. In *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2017. 1, 2, 3

[2] K. Bahrami, A. C. Kot, L. Li, and H. Li. Blurred image splicing localization by exposing blur type inconsistency. *IEEE Transactions on Information Forensics and Security (TIFS)*, 10:999–1009, 2015. 1

[3] M. Barni, L. Bondi, N. Bonettini, P. Bestagini, A. Costanzo, M. Maggini, B. Tondi, and S. Tubaro. Aligned and non-aligned double JPEG detection using convolutional neural networks. *Journal of Visual Communication and Image Representation (JVCI)*, 49:153–163, 2017. 1

[4] M. Barni, A. Costanzo, and L. Sabatini. Identification of cut & paste tampering by means of double-JPEG detection and image segmentation. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010. 1

[5] T. Bianchi and A. Piva. Detection of non-aligned double JPEG compression with estimation of primary compression parameters. In *IEEE International Conference on Image Processing (ICIP)*, 2011. 1

[6] T. Bianchi and A. Piva. Detection of nonaligned double JPEG compression based on integer periodicity maps. *IEEE Transactions on Information Forensics and Security (TIFS)*, 7:842–848, 2012. 1, 2

[7] T. Bianchi and A. Piva. Image forgery localization via block-grained analysis of jpeg artifacts. *IEEE Transactions on Information Forensics and Security (TIFS)*, 7:1003–1017, 2012. 1

[8] L. Bondi, L. Baroffio, D. Güera, P. Bestagini, E. J. Delp, and S. Tubaro. First steps toward camera model identification with convolutional neural networks. *IEEE Signal Processing Letters (SPL)*, 24:259–263, March 2017. 2

[9] C. Chen and M. C. Stamm. Camera model identification framework using an ensemble of demosaicing features. *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2015. 2

[10] F. De Natale and G. Boato. Detecting morphological filtering of binary images. *IEEE Transactions on Information Forensics and Security (TIFS)*, 12:1207–1217, 2017. 1

[11] Z. Fan and R. L. de Queiroz. Identification of bitmap compression history: JPEG detection and quantizer estimation. *IEEE Transactions on Image Processing (TIP)*, 12:230–235, 2003. 1

[12] T. Gloe and R. Böhme. The Dresden image database for benchmarking digital image forensics. *Journal of Digital Forensic Practice*, 3:150–159, 2010. 5

[13] F. Huang, J. Huang, and Y. Q. Shi. Detecting double JPEG compression with the same quantization matrix. *IEEE Transactions on Information Forensics and Security (TIFS)*, 5:848–856, 2010. 3

[14] M. Kirchner. Fast and reliable resampling detection by spectral analysis of fixed linear predictor residue. In *ACM workshop on Multimedia and security (MM&Sec)*, 2008. 1

[15] M. Kirchner and T. Gloe. Forensic camera model identification. In *Handbook of Digital Forensics of Multimedia Data and Devices*, pages 329–374. John Wiley & Sons, Ltd, Chichester, UK, 2015. 2

[16] S. Milani, P. Bestagini, M. Tagliasacchi, and S. Tubaro. Demosaicing strategy identification via eigenalgorithms. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014. 1, 3

[17] S. Milani, M. Tagliasacchi, and S. Tubaro. Discriminating multiple JPEG compressions using first digit features. *APSIPA Transactions on Signal and Information Processing*, 3:1–11, 2014. 1

[18] NIST. Nimble Challenge 2017 Dataset (NC2017) for image manipulation detection. Available at: https://www.nist.gov/itl/iad/mig/media-forensics-challenge. 6

[19] C. Pasquini, G. Boato, and F. Pérez-González. Multiple JPEG compression detection by means of Benford-Fourier coefficients. In *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2014. 1

[20] C. Pasquini, G. Boato, and F. Pérez-González. Statistical detection of JPEG traces in digital images in uncompressed formats. *IEEE Transactions on Information Forensics and Security (TIFS)*, 12:2890–2905, 2017. 1

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 4

[22] T. Pevny and J. Fridrich. Detection of double-compression in JPEG images for applications in steganography. *IEEE Transactions on Information Forensics and Security (TIFS)*, 3:247–258, 2008. 1

[23] T. Pevny and J. Fridrich. Detection of Double-Compression in JPEG Images for Applications in Steganography. *IEEE Transactions on Information Forensics and Security (TIFS)*, 3:247–258, 2008. 2

[24] A. Piva. An overview on image forensics. *ISRN Signal Processing*, 2013. 1, 2

[25] 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*, 43:1–42, 2011. 1, 2

[26] G. Schaefer and M. Stich. UCID - an uncompressed colour image database. In *SPIE Storage and Retrieval Methods and Applications for Multimedia*, 2004. 4

[27] M. C. Stamm, Min Wu, and K. J. R. Liu. Information forensics: An overview of the first decade. *IEEE Access*, 1:167–200, 2013. 1, 2

[28] T. Thai, R. Cogranne, F. Retraint, and T. Doan. JPEG quantization step estimation and its applications to digital image forensics. *IEEE Transactions on Information Forensics and Security (TIFS)*, 12:123–133, 2017. 1

[29] Q. Wang and R. Zhang. Double JPEG compression forensics based on a convolutional neural network. *EURASIP Journal on Information Security*, 2016:1–23, 2016. 1