Noise-based anomaly detection for image forgery localization

Tesi di Laurea Magistrale in
Music and Acoustic Engineering - Ingegneria Musicale e Acustica

Manuel Alejandro Jaramillo Rodríguez, 10821584

Advisor:
Prof. Paolo Bestagini

Co-advisors:
Sara Mandelli

Academic year:
2022-2023

Abstract: Nowadays, capturing, editing and sharing pictures has become a common practice on our daily routines. With the appearance of new and advanced technologies, we are now capable of manipulating images by simply using our smartphones. These technologies evolve so rapidly that traditional forgery detection and localization methods have become obsolete. Most of the time, these tools are used for innocent matters, but unfortunately, they are often used for malicious purposes such as political manipulation, fake publicity, extortion or even identity theft, becoming a potential risk not only to individuals but also to society. For these reasons, there is a constant interest in the scientific community of creating new powerful tampering localization methods capable of detecting manipulations done on images. These methods usually rely on exploiting specific characteristics of the cameras, such as the Color Filter Array (CFA) or the Camera Response Function (CRF). One of the most exploited of these features is probably the Photo-Response Non-Uniformity Noise (PRNU), a device-dependent noise residual. Some of the most successful methods for forgery localization are based on the idea of extracting the PRNU from the images and using it for forensic purposes. The same idea can be extrapolated to a camera-model level, stating that each camera-model leaves its own fingerprint on the captured images. Our purpose is to construct a method capable of extracting this fingerprint from images, and then employ it as a forgery localization tool. To achieve this goal, we rely on two different types of denoisers, one based on Convolutional Neural Network (CNN) architecture, and one based on Transformer architecture. Then, we use a set of activation maps extracted from these networks in order to perform an anomaly detection task to construct a heatmap, exhibiting potential traces of tampering on the input image. Results show that our proposed method outperforms state-of-the-art techniques in most of the cases.

Key-words: Image forgery localization, deep learning, anomaly detection, multimedia forensics.

1. Introduction

We live in an epoch where multimedia acquisition devices such as cameras and smartphones have become very accessible tools of common use in our day life. Constantly, we are sharing and consuming images, videos and recordings through internet platforms such as social media. With this growth on the availability of capturing devices, also a huge amount of digital processing techniques have appeared. Some of these techniques allow us to completely alter the content of the media, giving us a powerful tool for enhancing and personalizing our pictures, videos and audios, but also, opening a door for using them with malicious purposes such as fake publicity or impersonation [17, 24]. For instance, online tools like DALL·E [30] and deepfakesweb.com [13] let the users insert fake faces on images and videos. For these reasons, multimedia manipulations have become a potential risks in different scenarios of our society.

In this context, multimedia forensics have become of great interest in the scientific community. In the last decade, different approaches have been proposed to deal with local manipulations of multimedia contents, specializing in video [7, 31], image [9, 11, 16, 42] and audio forensics [21, 38]. In particular, for image forensics, these techniques usually rely on the fact that different kind of traces are implanted on images at some stage of their acquisition, such as the image reconstruction from the Color Filter Array (CFA) [18], image compressions [2], Camera Response Function (CRF) [5], among others. These traces vary from device to device, leaving a unique trace on the acquired image.

Focusing on image forensics, one of the most used types of image local tampering is splicing, which consist on cutting a pixel region from an image and inserting it into another image. Sometimes, post-processing steps are done in order to hide the manipulation and make it more difficult to detect. Another common manipulation technique is based on generative models. These models have evolved so rapidly that it has become almost impossible to the human eye to tell whether an image has been manipulated or not. One of the most famous tools of the last years is DALL·E. [30], a model that is capable of modifying images by selecting a region and giving a text input with the instructions for the modifications.

Some of the most successful techniques for spotting local image manipulations are data-driven statistical methods [6, 9, 11, 43]. Data-driven approaches harness the power of deep learning and statistical analysis to uncover patterns and anomalies within images that may indicate that some kind of tampering has been made. Leveraging on large datasets, these methods employ sophisticated algorithms to automatically learn and detect complex characteristics of manipulated images at a pixel level. In particular, some of these models specialize on extracting the Photo-Response Non-Uniformity Noise (PRNU) [8, 10]. The PRNU is a unique noise-like pattern that is inherent in the imaging sensors of digital cameras, meaning that each specific device stamps a unique trace on its captured images [25]. This pattern arises due to slight variations in the individual sensitivity of pixels in the sensor. These variations can be attributed to manufacturing imperfections and other factors.

One of the state-of-the-art methods for image forgery localization is the so called Noiseprint [11]. This data driven method extrapolates the PRNU to a camera-model level, in the sense that, as the PRNU is a unique trace for each single device, the Noiseprint is a unique trace associated to each camera-model. In fact, each camera model possesses its own specifications, including the lenses, sensors, the CFA, etc. All of them are characterized by specific features; for example, demosaicing artifacts may emerge during the reconstruction of the image from the CFA. Additionally, diverse digital processing stages may be incorporated depending on the manufacturer and specific to each camera model, like white balancing or color correction. It is expected that localized manipulations will have an effect on the Noiseprint, leaving local traces on it. By extracting the Noiseprint from an image, the authors of [11] end up with an efficient forgery localization method.

In this work, we aim to construct a robust algorithm capable of detecting and localizing image manipulations. For this purpose, we take inspiration from the idea of the Noiseprint. To do so, we start by training a Convolutional Neural Network (CNN)-based denoiser capable of extracting a camera-model fingerprint from images. Then, we exploit different activation maps of the denoiser as abstract low-level noise-related features for an Anomaly Detection (AD) algorithm. This algorithm makes use of the extracted features to generate a heatmap, i.e, a one channel image that quantizes the dissimilarity between the input image and a set of normal images. By doing so, we are able to detect any present local variations on the extracted noise fingerprint, exhibiting possible tampered pixels inside the images.

To validate our results, apart from a public dataset [12] based on splicing, we construct two more datasets based on generative Artificial Intelligence (AI) models by applying realistic local manipulations to natural images. We proceed by evaluating different possible configurations of our algorithm in order to discover the optimal one. Our results show remarkable performance in the proposed datasets, outperforming in most of the cases the state-of-the-art.

The next sections are organized as follows. In Section 2 we introduce some useful deep learning tools employed throughout the development of our method. In Section 3 we formulate the image forgery localization task. In Section 4 we make a deep description of our proposed method, emphasizing on the training procedure for each of the stages and showing a complete pipeline of the application. In Section 5 we describe the datasets used for training and testing, as well as the hyperparameters and specific training choices. In Section 6 we analyze all the experiments done and make a comparison with state-of-the-art methods. Finally, in Section 7 we discuss the final considerations on our work.

2. Background

In this section, we provide the reader with a general introduction to the different tools used during the development of this work. First, we introduce the denoising neural networks that we employ in our work. Then, we describe the Noiseprint technique, which we take as inspiration for our forensics detector. Eventually, we provide details on the method used for detecting image local anomalies.

2.1. Denoising Neural Networks

Denoising neural networks are deep learning architectures specifically designed for image denoising. The purpose of an image denoiser is to take a noisy input $\mathbf{y} = \mathbf{x} + \mathbf{v}$, where $\mathbf{x}$ is the restored image and $\mathbf{v}$ is the noise, and apply a transformation $F(\mathbf{y})$ such that $F(\mathbf{y}) \approx \mathbf{x}$. In the specific case of a denoising architecture, the operator $F(\cdot)$ is applied by the network to the input noisy image.

In this work, we consider two kinds of denoising architectures. The first one is a Convolutional Neural Network (CNN), namely the Denoising CNN (DnCNN) proposed in [40], and the other is a Vision Transformer (ViT), specifically the Efficient Transformer for High-resolution Image Restoration (Restormer) presented in [39]. In the next lines, we provide more details on each of the two deep learning architectures.

2.1.1 DnCNN

Image denoising is one of the tasks that can be done by means of CNNs, and it has been widely studied over the years [19, 41]. For image forensic purposes, Denoising CNN (DnCNN)[40] has been successfully exploited several times [3, 4, 11]. Its primary advantage lies in its fully convolutional structure. Indeed, in fully convolutional networks, the input resolution is not restricted, i.e., after the network has been trained, it can be fed with images of any size.

The training is based on residual mapping, this means that a network R is trained to learn a mapping such that $R(\mathbf{y}) = \mathbf{v}$. By doing this, we can obtain the denoised image simply by $\mathbf{x} = \mathbf{y} - R(\mathbf{y})$. DnCNN is trained by minimizing the mean squared error between the desired residual images and the estimated ones.

Regarding the network's architecture, it starts with a Conv+ReLu layer, which takes the input image and applies 64 filters of size $3 \times 3 \times c$ where c is the number of channels and a ReLu activation function, in order to generate 64 features maps, then a cascade of $(D - 2)$ Conv+BN+ReLu layers (where BN stands for batch normalization) is applied, where D is the total depth of the network, finally, a last convolution with a unique filter of size $3 \times 3 \times 64$ is applied in order to reconstruct the output.

Figure 1 shows a diagram of the network's architecture.

2.1.2 Restormer

Efficient Transformer for High-resolution Image Restoration (Restormer) [39] is a Vision Transformer (ViT) architecture used for general image restoration purposes, including denoising. Transformer architectures were originally designed for Natural Language Processing (NLP). Introduced by Vaswani et al. in 2017 [36], transformers are based on the attention mechanism, which allows the network to process sequential data while capturing dependencies and relations across the input.

Transformers consist of an encoder-decoder structure, where the encoder processes the input sequence by looking at all its positions simultaneously and encoding a context, while the decoder generates the output by

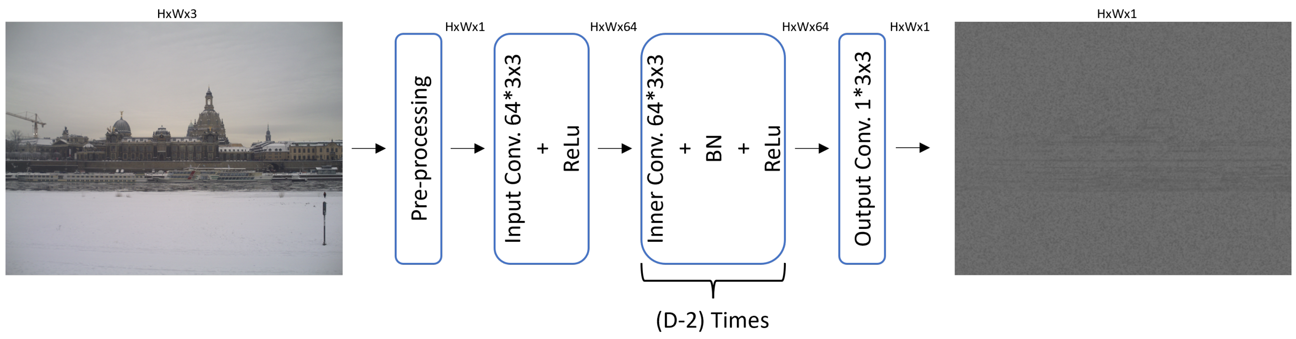


Figure 1: The architecture of DnCNN for denoising images. Notice that since DnCNN is trained as a residual mapper, the output of the networks is directly the noise.

attending to relevant parts of the input and previous generated encoding contexts. This has been adapted to image processing by introducing ViT. ViT works by decomposing images into sequences of patches [22, 29], in this way, the attention mechanism is used to learn complex spatial relationships between those patches.

In particular, Restormer introduces a modification in the self-attention mechanism. This adjustment involves applying the Self-attention Mechanism (SA) across the channels dimension instead of the spatial dimensions. This modification notably reduces computational complexity, making it quadratic only in the number of channels and linear in the spatial size. As a result, Restormer offers improved efficiency in high-resolution image processing.

Restormer is formed by a four-level encoder-decoder structure, at the same time, each of those levels consists of two transformer blocks (one for encoding and one for decoding). Each block is conformed by two modules, the first one called multi-Dconv head transposed attention (MDTA), is in charge of applying the attention mechanism across channels. The second module, called Gated-Dconv feed-forward network (GDFN), performs a controlled feature transformation. This is done by a gating mechanism that is trained to allow only useful features to propagate.

Before passing the image $\mathbf{I} \in c \times H \times W$ to the actual transformer architecture, Restormer maps it to a set of low-level feature embeddings $\mathbf{F} \in C \times H \times W$ by means of a convolutional layer. The encoder-decoder structure works as follows: At each encoder level, the image size is downsampled by a factor of 2, but at the same time the original number of feature maps C is increased by 2 in order to maintain efficiency, on the other hand, the decoder does the opposite, increasing the spatial dimensions by 2 at every level.

An important characteristic of the architecture is that same level encoder-decoder blocks are connected to each other by means of skip connections, due to these skip connections, after the concatenation is done the number of channels is conserved while the spatial dimensions have been upsampled by a factor of 2, so in order to gradually reduce the number of channels, a 1×1 convolution layer is applied. The next step after the original spatial dimensions $H \times W$ have been recovered is to apply a last transformer block to the last set of feature embeddings. Finally, a convolutional layer with a unique filter is used, obtaining the residual image $R \in 1 \times H \times W$.

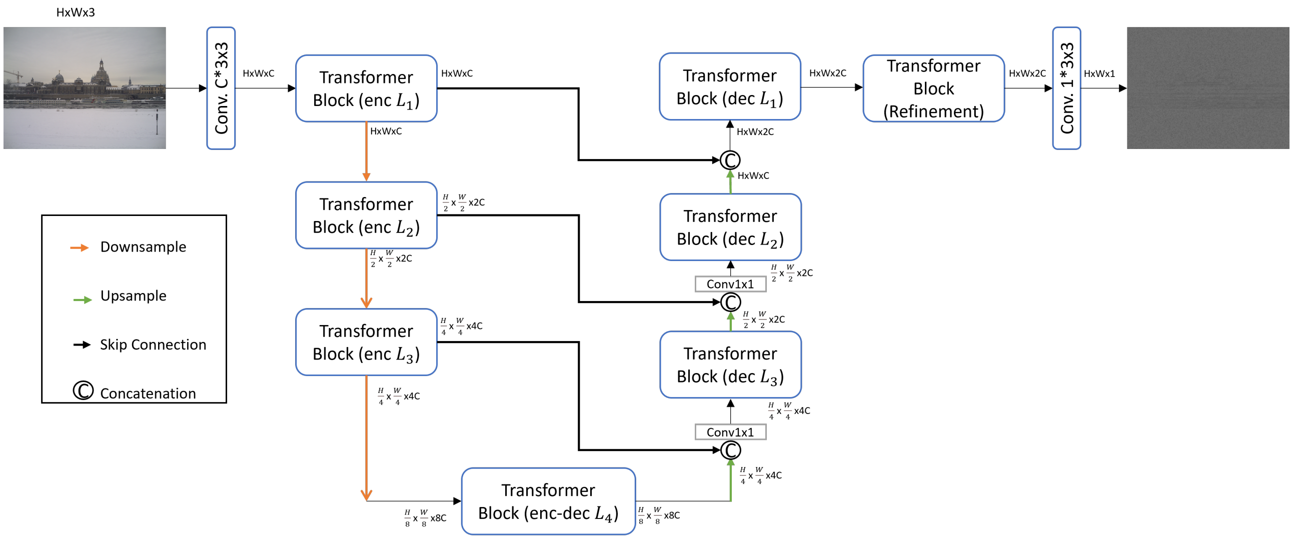
Figure 2 shows a simplified diagram of the Restormer's architecture.

2.2. Noiseprint: a CNN-based camera model fingerprint

One of the main concerns in image forensics are the in-camera and out-camera traces left on an image during its acquisition, also known as camera model related artifacts or fingerprint. These traces are due to several factors; indeed, the digital image acquisition pipeline is not unique, and may differ depending on the vendor, the device model and the available on-board technologies [27].

Initially, the light is captured by the camera's lens, which may contain imperfections that influence the resulting image. Subsequent stages introduce further traces into the image; for example, demosaicing artifacts may emerge during the reconstruction of the image from the color sensor's mosaic. Additionally, diverse digital processing stages like white balancing, color correction and JPEG compression may be incorporated, all of them typically being vendor-specific.

All these traces can be used as fingerprints for accomplishing diverse forensic tasks [2, 5, 8, 10, 18]. For



HxWx3
Conv. C*3x3
HxWxC
Transformer Block (enc L_1)
HxWxC
HxWxC
Transformer Block (enc L_2)
$\frac{H}{2}$x$\frac{W}{2}$x2C
$\frac{H}{2}$x$\frac{W}{2}$x2C
Transformer Block (enc L_3)
$\frac{H}{4}$x$\frac{W}{4}$x4C
$\frac{H}{4}$x$\frac{W}{4}$x4C
$\frac{H}{8}$x$\frac{W}{8}$x8C
Transformer Block (enc-dec L_4)
$\frac{H}{8}$x$\frac{W}{8}$x8C
$\frac{H}{4}$x$\frac{W}{4}$x4C
Conv1x1
$\frac{H}{4}$x$\frac{W}{4}$x4C
Transformer Block (dec L_3)
$\frac{H}{2}$x$\frac{W}{2}$x2C
Conv1x1
$\frac{H}{2}$x$\frac{W}{2}$x2C
Transformer Block (dec L_2)
HxWxC
HxWx2C
Transformer Block (dec L_1)
HxWx2C
Transformer Block (Refinement)
HxWx2C
Conv. 1*3x3
HxWx1
Downsample
Upsample
Skip Connection
© Concatenation

Figure 2: A summarized sketch of the Restormer's architecture. In the original approach, a skip connection is added between the input and the residual map in order to obtain the denoised image, in our case this skip connection is discarded to obtain directly the residual as output.

instance, Cozzolino and Verdoliva [11] proposed a method for extracting the camera model fingerprint from images, called Noiseprint. The purpose of the Noiseprint extractor is to capture this fingerprint from the images and to spot editing-related artifacts due to local image forgery. This is done by suppressing the scene content and keeping only the model-related traces. To do so, the Noiseprint method starts from a pre-trained DnCNN model (the denoising CNN presented in Section 2.1.1) and refines it to acquire the desired output by leveraging a dataset comprising images from various camera models.

The training procedure can be summarized as follows:

- Collect a dataset of pristine images that have been captured using different camera models.
- Extract image patches not only coming from different camera models but also from different pixel positions.
- Feed image patches into the DnCNN, extracting noise patterns.
- Learn the optimal weights such that a distance metric between the noise patterns coming from patches of the same model and the same pixel position is minimized.

By following the above procedure, the DnCNN network does not only learn to discriminate among different camera models, but it can also learn to recognize transformations applied to the images such as translations and rotations, together with different compressions as well. Therefore, the Noiseprint can be used for exposing traces of local manipulation applied to images [11], this thanks to its ability to extract the camera model-related noise. When an image has been manipulated by means of splicing, two different patterns can be clearly seen on the output fingerprint returned by the Noiseprint.

In figure 3, we show an example of a Noiseprint for a manipulated image.

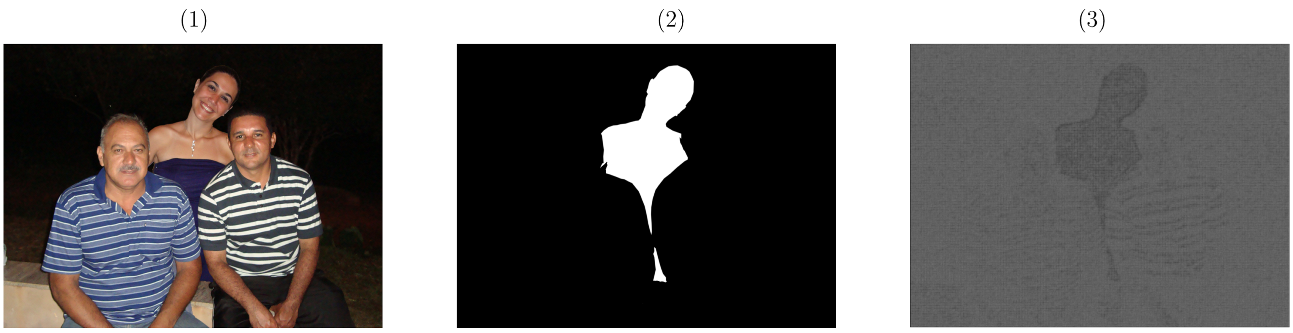


Figure 3: An example of a Noiseprint is showed on the right image. At the center we can see the corresponding ground truth and at the left we observe the manipulated image.

2.3. PaDiM: A Method for Image Local Anomaly Detection

Anomaly Detection (AD) is a technique from data analysis that aims to detect patterns deviating from the expected behavior of the data. In general, AD uses tools from statistical analysis, machine learning and deep learning in order to identify irregularities in the data. Most recent anomaly detection methods are based on deep learning techniques such a CNNs and transformers [28, 33]. Different works have been done in this field and applications go from medical images analysis [33], where AD is used to detect anomalies in tissues or organs, to industrial applications [34], where it is used for instance to automatize manufacturing defects recognition.

In this work, we restrict the field to image local AD, which aims at exposing anomalies occurring on image pixels. The anomaly is identified as a different pixel pattern or different scene content with respect to the standard characteristic of the image. This can be the case of image splicing, where a pixel region of an image is inserted into another image (after potentially applying editing operations) to create a local manipulation.

The purpose of image local AD is to detect whether a pixel in an image should be classified as "normal" or "abnormal". To do so, AD algorithms use a set of images to learn the characteristics of the "normal" class. Then, these characteristics are compared with those of the test image and an anomaly score is assigned to every pixel on it. For instance, in the case of image splicing, it is expected that the region of pixels that was inserted from a second image has different characteristics than those pixels corresponding to the original image.

One of the state-of-the-art algorithms for image local AD is Patch Distribution Modeling (PaDiM) [14]. The idea behind PaDiM is to use a CNN for extracting feature maps from a dataset of "normal" images, constructing a set of reference local embeddings that follow a specific distribution. Then, a generic image can be tested and compared with the trained data distribution to spot local anomalies. The training procedure is composed of the following steps:

- Consider a set of normal images, these images being resized to a common resolution.
- Pass the resized images through a pretrained CNN feature extractor (typically, EfficientNet [23] or ResNet [35] architectures). During this step, a set of activation maps are hooked. In particular, since high-level features are more sophisticated and describe better the specific characteristic of an image, the last three network layers are selected.
- Concatenate the acquired maps from the layers to obtain some corresponding embedding vectors. For this action to be possible, all the maps must have the same spatial dimensions, therefore smaller maps are upsampled to the dimensions of the largest one. After this is done, all the features can be concatenated, forming a set of embedding vectors, one per each pixel position of the largest map. Notice that the largest map itself has smaller spatial size than the input image, meaning that a pixel position on this map represents a patch on the input. For this reason, each embedding vector represents a patch of the original image, therefore we call them patch embedding vectors.
- For each different patch position, associate the patch embeddings related to all training images with a multivariate Gaussian distribution. All the information of the training patch embeddings are contained in these distributions.

  Figure 4 shows a sketch of the procedure.

At deployment stage, the test image is resized to the same size as the training images. Then, the CNN selected as feature extractor is employed to extract the patch embedding vectors. Finally, a distance metric between each patch embedding position and its corresponding training data distribution is computed, resulting into an anomaly score at the patch position, all these scores are combined into a heatmap that quantizes the deviation of every pixel of the test image with respect to the normal distribution.

Figure 5 shows the results of PaDiM on one image of the MVTec-AD dataset [1], a popular dataset for AD on industrial images.

3. Problem Formulation

In this work, we focus on developing a robust method that is able to detect which pixels of an image have been manipulated by means of splicing. Formally, we define a generic pristine image as $\mathbf{I}$, with size $H \times W$, in which a pixel coordinate is defined as (h, w), $h \in [1, H]$ and $w \in [1, W]$. The locally manipulated version of $\mathbf{I}$ can be

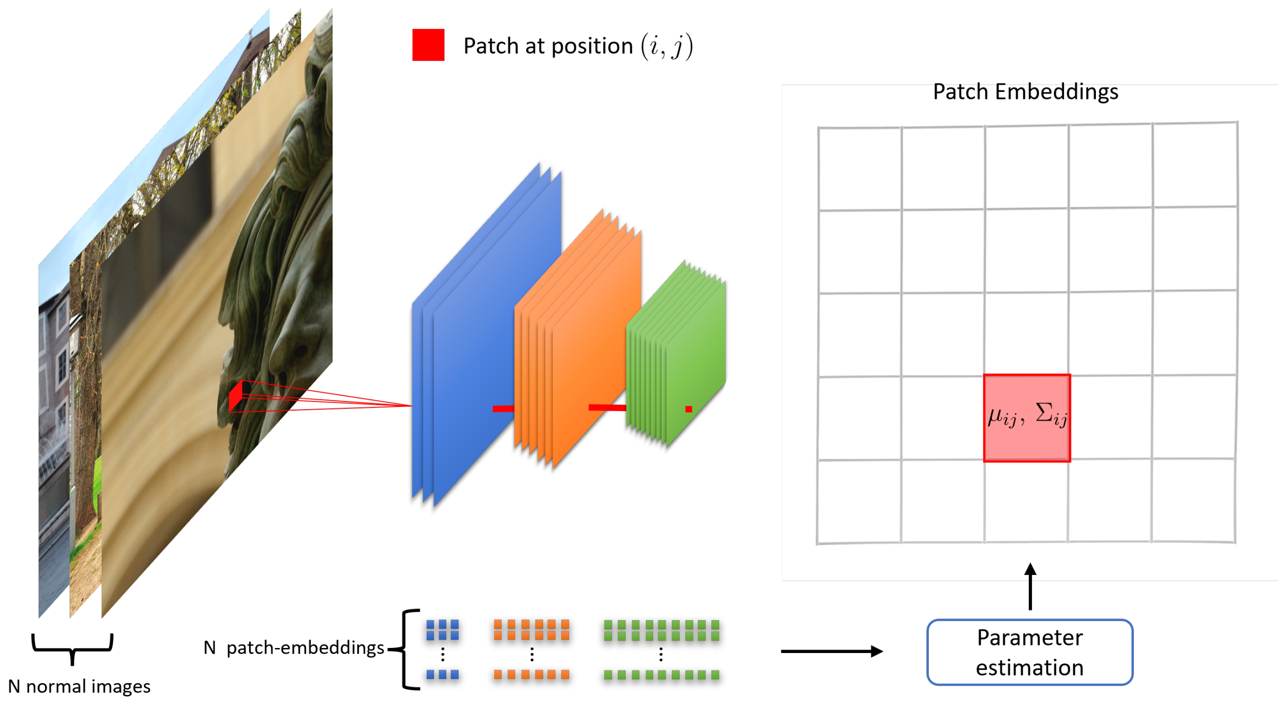


Figure 4: A sketch of the training procedure of PaDiM. From a set of N normal images, N patch-embeddings are extracted at each patch position. Then a multivariate Gaussian is assigned to every position.

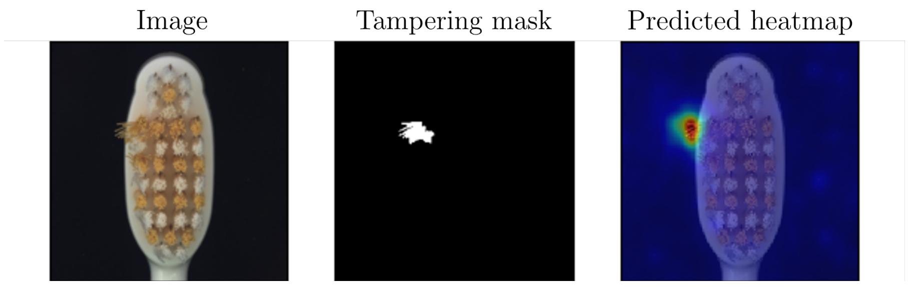


Figure 5: Example of local anomaly detection on one of the classes of the MVTec-AD dataset [1]. At the left we have the test image, in the middle the corresponding mask, and at the right the predicted anomaly heatmap.

written as

$$\bar{\mathbf{I}}(h,w) = \begin{cases} t, & \text{if } (h,w) \in \mathcal{S}, \\ \mathbf{I}(h,w), & \text{if } (h,w) \notin \mathcal{S} \end{cases} \tag{1}$$

where $\mathcal{S}$ is the pixel region under splicing attack and t corresponds to the specific tampered with pixel value.

The local manipulation can be described by a tampering mask, which is a 2D matrix with the same spatial size of the image. We can define the tampering mask as

$$\mathbf{M}(h,w) = \begin{cases} 0, & \text{if } (h,w) \notin \mathcal{S}, \\ 1, & \text{if } (h,w) \in \mathcal{S} \end{cases} \tag{2}$$

Given a generic manipulated image $\bar{\mathbf{I}}$, our goal is to estimate a tampering heatmap capable of distinguishing between non-altered pixels and manipulated ones. A heatmap $\mathbf{H}$ is a single-channel image with the same spatial dimensions as the input. For each pixel location (h, w), the value of the pixel $\mathbf{H}(h, w)$ represents the probability of the image pixel belonging to the manipulated class.

Figure 6 shows a sketch of the tackled problem: given an image $\bar{\mathbf{I}}$, we aim to retrieve a heatmap $\mathbf{H}$ that is as close as possible to the mask $\mathbf{M}$.

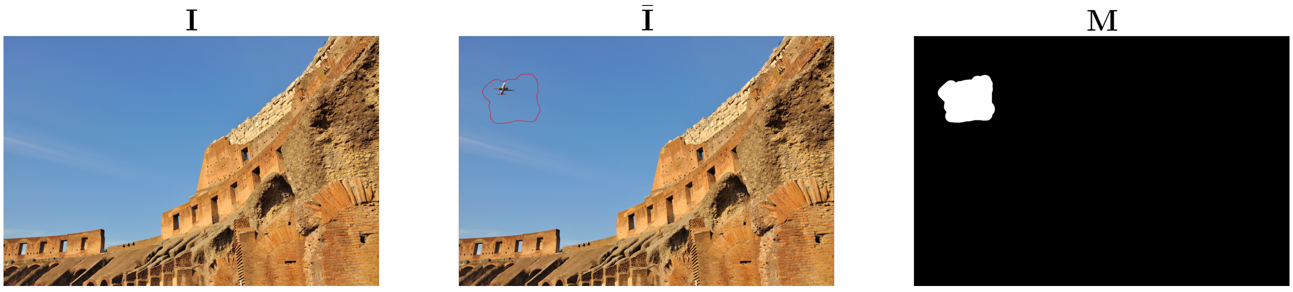


Figure 6: We show the pristine image $\mathbf{I}$, its manipulated version $\bar{\mathbf{I}}$ and the corresponding mask $\mathbf{M}$.

4. Proposed method

The proposed method is designed for detecting forgeries in the case of image splicing. To do so, we take advantage of the unique camera model related artifacts that are present on captured images. We propose a method that can be separated into two main stages:

1. Fingerprint extraction: we use a denoising architecture to retrieve low level information from the image. In particular, the denoiser is capable of extracting camera model-based artifacts. If the image has been manipulated by splicing, the denoising model extracts a noise fingerprint which shows two different patterns, clearly making a distinction between non-manipulated and manipulated pixels.
   This distinction, however, may or may not be perceptible by the human eye, moreover, even if it is perceptible, it might not be easily separable into the two classes, in the sense that a simple thresholding might not be able to separate the two different noisy patterns.
2. Heatmap generation: In this stage, an AD technique is applied in order to generate a tampering heatmap that effectively quantifies the differences between the patterns, resulting in a probability map that can thresholded to obtain a mask that classifies each pixel as belonging to one of the extracted patterns. In particular, we make use of the noise fingerprint and of the network activation maps acquired in the previous stage.

Figure 7 reports a sketch of the proposed methodology.

Every single block needs to be trained separately. In the following, we provide more details on the training phase of each single block. Then, we present details on the complete deployment stage.

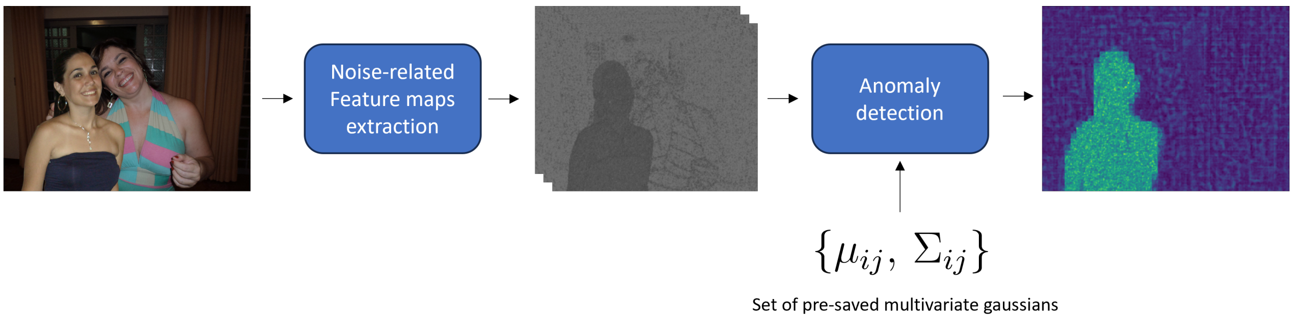

Noise-related Feature maps extraction
Anomaly detection
$\{\mu_{ij},\, \Sigma_{ij}\}$
Set of pre-saved multivariate gaussians

Figure 7: Simplified sketch of the proposed pipeline. We start from a possibly manipulated image. Then we extract noise-related feature maps by using a pretrained fingerprint extractor. Finally, we use the feature maps as inputs of an anomaly detection algorithm.

4.1. Fingerprint extractor training

In this stage, we aim to develop a denoiser that is capable of extracting from images the camera model related artifacts. This should be done by removing all the scene content and other types of noises coming from different sources. To do so, we take inspiration from the original Noiseprint paper [11], proposing a Siamese framework for training a denoising network architecture:

1. We select a number of images from which we know the exact camera model that was used for their acquisition, all these images must be pristine images, this means that they have not been manipulated.
2. We crop images into squared patches $\mathbf{P}^m_{(h,w)}$ of size $P \times P$, where (h, w) represents the patch position, and m the camera model used for capturing the image.
3. Each patch is fed into a denoising network. We extract its related model-related fingerprint, defined as $\mathbf{F}^m_{(h,w)}$.
4. We generate a mini-batch of samples containing the fingerprints of N patches coming from different images and camera models. Then, we pair patches only if they meet the two following conditions: they share the same camera model, and they are extracted from the same pixel positions but from different images. Label 1 is assigned to paired patches, otherwise label 0 is assigned.
5. The weights of the denoising network are iteratively updated by calculating the Distance-Based Logistic (DBL) loss [37] (see Section 4.1.2) between the fingerprints in the mini-batch. The DBL loss minimizes a distance metric between the fingerprints of paired patches, while maximizing the unpaired ones.

Figure 8 shows a sketch of the training process.

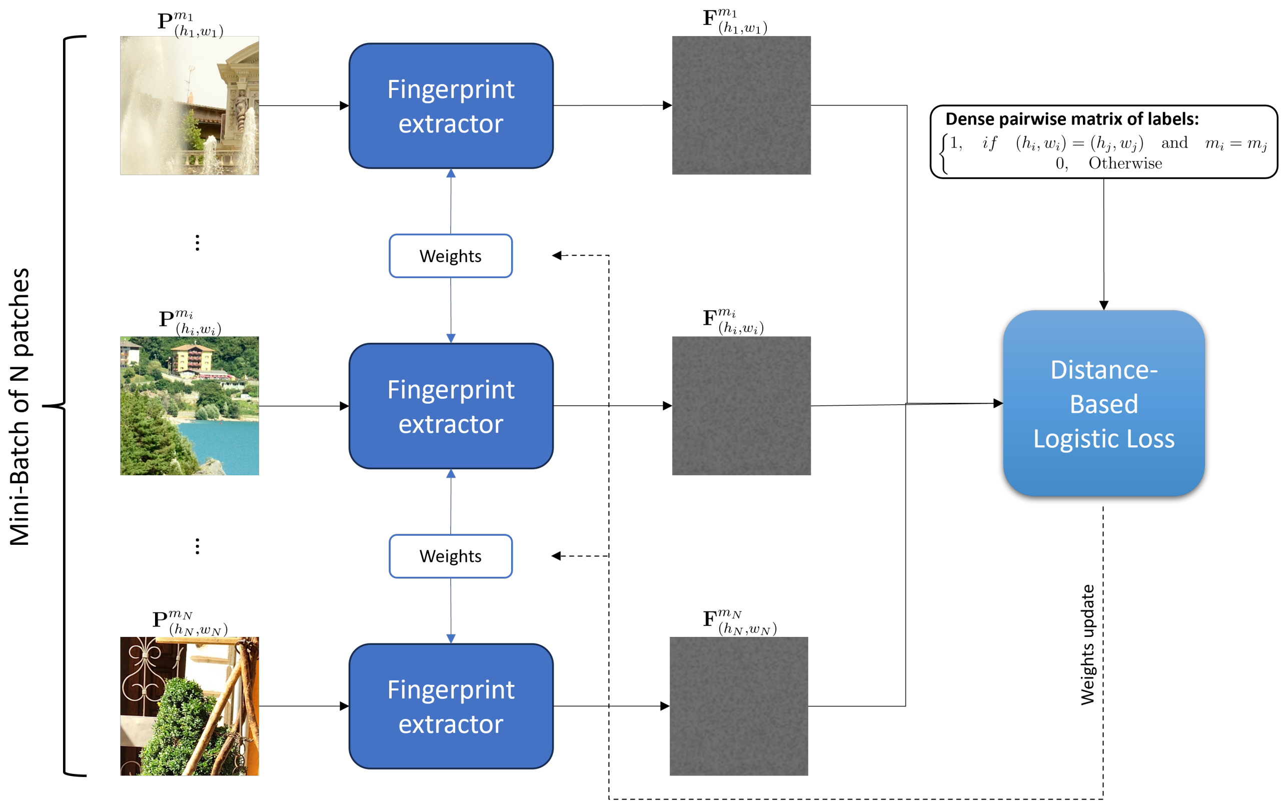
By following the previous procedure, we are training a denoising neural network that is not only able to extract noise fingerprints that are similar for all the images belonging to the same camera model, but also to distinguish between images captured from different models and maximize the dissimilarity between them.

In the next sections, we make a deeper description of how the mini-batch is structured, which are the architectures proposed for the training and which is the specific loss function used.

4.1.1 Mini-batch structure

The network is trained using mini-batches formed by image patches, each mini-batch contains N patches of size 48×48. The patches inside a batch can be gathered together to create groups. These groups are formed in a way such that all the patches inside a group share same camera model and same image position, whereas patches from different groups differ by one or both of the previously mentioned conditions. This structure is needed because there can be non-uniformly distributed noise artifacts caused by image acquisition processes, such as JPEG compression, which imprint a distinctive periodic fingerprint on the image. These artifacts may impact on the noise fingerprint itself, therefore, patches coming from different images positions should not be treated as equals.

A label of 1 or 0 is assigned to each patch pair, 1 if both the patches in the pair come from the same model and same image position, 0 otherwise.



Mini-Batch of N patches
$\mathbf{P}^{m_1}_{(h_1,w_1)}$
$\mathbf{P}^{m_i}_{(h_i,w_i)}$
$\mathbf{P}^{m_N}_{(h_N,w_N)}$
Fingerprint extractor
Weights
Fingerprint extractor
Weights
Fingerprint extractor
$\mathbf{F}^{m_1}_{(h_1,w_1)}$
$\mathbf{F}^{m_i}_{(h_i,w_i)}$
$\mathbf{F}^{m_N}_{(h_N,w_N)}$
Dense pairwise matrix of labels:
$\begin{cases} 1, & if \quad (h_i,w_i)=(h_j,w_j) \quad \text{and} \quad m_i=m_j \\ 0, & \text{Otherwise} \end{cases}$
Distance-Based Logistic Loss
Weights update

Figure 8: Siamese architecture for training the fingerprint extractors, each possible pair of patches is compared according to their labels by means of the distance-based logistic loss.

4.1.2 Distance-based logistic loss

A common choice when using Siamese network training is the DBL loss function [37]. This loss is used in order to maximize the dissimilarity between non-related pairs of samples while maximizing the similarity between related pairs. To do so, first a distance metrics should be chosen, in our case we are using the simple Euclidean distance

$$d_{ij} = \|\mathbf{F}_i - \mathbf{F}_j\|^2 \tag{3}$$

where $\mathbf{F}_1, ...\mathbf{F}_n$ are the fingerprints of the N input patches used at each mini-batch iteration and d_{ij} is the Euclidean distance between the residuals $\mathbf{F}_i$ and $\mathbf{F}_j$. Then we can define the probability distribution $p_i(j)$, which represents how the distance from the fingerprint of the i-th patch to all the other patches in the mini-batch is distributed

$$p_i(j) = \frac{e^{-d_{ij}}}{\sum_{j \neq i} e^{-d_{ij}}} \tag{4}$$

In specific, this represents the probability that the fingerprint of the patch j presents a specific distance d_{ij} from the fingerprint of the i-th patch. Recalling that our goal is to minimize the fingerprint distances when both patches come from the same group, we can observe that this requirement is translated into maximizing $p_i(j)$ when the pair (i, j) belongs to the same group, while minimizing it when the pair belongs to different groups. From this we can construct our loss function by saying that maximizing $p_i(j)$ is equivalent to minimize

$$L_i = -log \sum_{j:l_{ij}=1} p_i(j) \tag{5}$$

Where the sum goes over the pairs for which the label l_{ij} is equal to 1, which means that the pair belong to the same group. Finally, we obtain the batch loss by summing over all mini-batch patches

$$L = \sum_{i=1}^{N} -log \sum_{j:l_{ij}=+1} p_i(j) \tag{6}$$

4.1.3 Backbones of the denoising network

We employ the algorithm described at the beginning of this section to train two distinct models.

The first model is the DnCNN architecture presented in Section 2.1.1. Since the input for the DnCNN is a grayscale image, the preprocessing that we apply to images is composed by two steps, first we convert the RGB image to grayscale, and then we normalize the pixels by dividing the grayscale image over 255.

It is important to notice that, even if the DnCNN is trained using patches of size 48×48, being a fully convolutional network it allows us to consider images of any sizes. DnCNN is built as a residual mapper, which means that its output is directly the noise fingerprint, so no changes have to be done to the original architecture.

The second network model used is the transformer-based denoiser called Restormer presented in Section 2.1.2. In the original approach, the output of the network is the restored image $\mathbf{x}$. We know that it is obtained by subtracting the residual to the corrupted image: $\mathbf{x} = \mathbf{y} - \mathbf{v}$, since for our purposes only the residual is needed, we make a modification on the network, so its output is now $\mathbf{v} = \mathbf{y} - \mathbf{x}$. Restormer was also trained using patches of the same size as DnCNN, however, the architecture does not allow us to use it on any image. Specifically, it is only possible to use it on images in which their dimensions are multiples of the patch size used during training, so at inference time, it is necessary to either process them by patches, or pad them until the dimensions fulfill the requisite.

4.2. Heatmap generator training

For the heatmap generation stage, we take inspiration from the PaDiM algorithm proposed in [14] (see Section 5.2.2). In particular, we propose an Anomaly Detection (AD) algorithm to infer distributions of the embedding features from a set of "normal" training data, which in our case are non-manipulated pristine images. To do so, we exploit denoising network architectures pretrained as shown in Section 4.1.

In the following lines, we report more details on the training steps required and on the feature selection phase.

4.2.1 Training steps

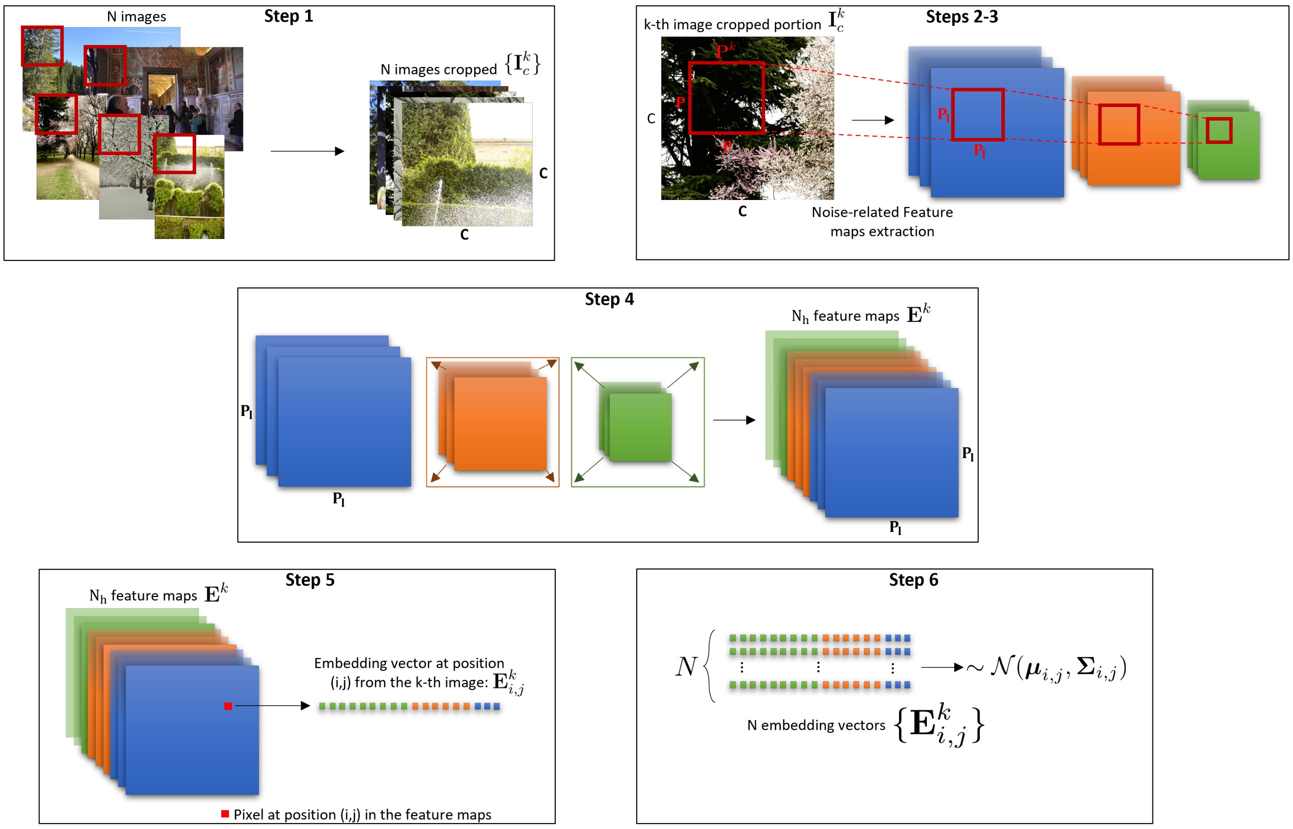
To have a more clear idea of how our proposed AD method works, we explain the training steps as follows:

1. We select a dataset containing N pristine images coming from M different camera models. Since all these images come from different models and have different sizes, we crop them to a common size of $C \times C \times 3$ pixels, starting from the top-left corner of the image. Let us call the set $\{\mathbf{I}_c^k\}$, where $k = 1, ..., N$ enumerates the N training images, and c is used just to clarify that it is a cropped image.
2. The cropped images $\{\mathbf{I}_c^k\}$ are fed into one of the proposed fingerprint extractors in Section 4.1.3. Then, N_h activation maps are selected and hooked from some of the layers of the extractor, these maps might or might not include the last layer corresponding to the actual fingerprint. The selection of these activation maps is discussed in the next section.
3. From the previously acquired maps, only those pixels corresponding to a patch $\mathbf{P}^k$ of size $P \times P \times 3$ inside every original cropped image $\mathbf{I}_c^k$ are kept. The position from which the patch is going to be extracted is selected by making two restrictions; first, the patch should not be taken from the borders of the original image in order to avoid border artifacts; second, the patch must be aligned with the 8×8 JPEG grid. The motivation behind this last decision is discussed in Section 6.5. Notice that, in the most general case, feature maps coming from different network layers may have different spatial size, in particular, their size is reduced over the depth of the network. Therefore, every original input patch $\mathbf{P}^k$ has a corresponding activated region in the selected maps which is typically smaller than $P \times P$ pixels, its size changing according to the activation map depth.
4. We upscale all the selected activation regions to have equal spatial dimensions. We end up with all the regions having the same spatial size of $P_l \times P_l$ pixels, were P_l and P_l represents the size of the largest selected activation region.
5. We build the corresponding embedding vectors by concatenating the resulting feature maps extracted from all the considered layers. Each $\mathbf{P}^k$ has its own embeddings $\mathbf{E}^k$ with size $N_h \times P_l \times P_l$. Notice that, by following similar considerations to those done in step 3, every $\mathbf{E}_{i,j}^k$, which has N_h total elements and $(i, j) \in [1, ...P_l] \times [1, ...P_l]$, corresponds to a small pixel area in the input patch $\mathbf{P}^k$. Therefore, every

$\mathbf{E}^k_{i,j} \in \mathbb{R}^{N_h}$ is defined as the embedding vector of a specific pixel region of the input $\mathbf{P}^k$.

6. Considering the contributions of all training images, we end up having a set of N embeddings $\{\mathbf{E}^k\}, k = 1, ..., N$. For each pixel position (i, j) of the embeddings, we estimate one multivariate Gaussian distribution, by computing the corresponding mean $\boldsymbol{\mu}_{ij}$ and covariance matrix $\boldsymbol{\Sigma}_{ij}$ over the set of N samples. In particular, $\boldsymbol{\mu}_{ij} \in \mathbb{R}^{N_h}$ and $\boldsymbol{\Sigma}_{ij} \in \mathbb{R}^{N_h \times N_h}$. We end up with a set of P_l^2 reference multivariate Gaussian distributions $\{\mathcal{N}(\boldsymbol{\mu}_{ij}, \boldsymbol{\Sigma}_{ij})\}, (i, j) \in [1, ...P_l] \times [1, ...P_l]$. Each distribution describes the statistics of the "normal" images in a specific pixel region.

Figure 9 shows a scheme of the previous procedure.



Step 1
N images
N images cropped $\{\mathbf{I}^k_c\}$
C
C
Steps 2-3
k-th image cropped portion $\mathbf{I}^k_c$
P^k
P
P
P
C
C
P_l
P_l
Noise-related Feature maps extraction
Step 4
P_l
P_l
N_h feature maps $\mathbf{E}^k$
P_l
P_l
Step 5
N_h feature maps $\mathbf{E}^k$
Embedding vector at position (i,j) from the k-th image: $\mathbf{E}^k_{i,j}$
Pixel at position (i,j) in the feature maps
Step 6
N
$\sim \mathcal{N}(\boldsymbol{\mu}_{i,j}, \boldsymbol{\Sigma}_{i,j})$
N embedding vectors $\{\mathbf{E}^k_{i,j}\}$

Figure 9: Scheme of the procedure for training the Heatmap generator.

4.2.2 Feature selection

An important aspect is the fact that in the step 2 of the training phase, we have to hook a certain number of network layers. In principle, we can expect that using the activations from all the layers would lead to better results. However, even if this were true, we have to take into account that the memory consumption of the algorithm grows not only with the resolution of the used images, but also with the length of the embedding vectors and the number of images used at the training step. For this memory limitation issues, a selection of layers must be done.

Formally, we have D layers from which we can extract the activation maps $\mathbf{A}_i, i \in [1, D]$. To do so, we can select any number $N_l \in [1, D]$ of layers. In particular, the authors of PaDiM did different experiments to investigate on the trade-off between memory consumption and number of layers used [14]. They found that selecting only the last three activation maps is an optimal choice. So we use this as a starting point to chose our final configuration.

For DnCNN, we discovered that the best results are obtained by using the layers (L_{D-2}, L_{D-1}, L_D), where D is the total depth of the network. The fingerprint itself would be the output of the last layer L_D. Contrary, for Restormer, we obtained the best results by selecting only the output layer, i.e, the fingerprint itself.

4.2.3 Differences with respect to PaDiM

With respect to the PaDiM original training pipeline described in Section 2.3, we introduce several differences in order to take into account our specific tackled goal.

First, notice that in the original version of PaDiM the neural networks used for extracting the feature maps are standard CNNs (i.e., ResNet [35] or EfficientNet [23]) pretrained on ImageNet dataset [15]. Indeed, these original extractors are specialized in extracting object-specific features from natural images, being able to associate different semantic categories with different features (e.g., cats and dogs are reasonably associated with diverse embeddings). However, in our case, these kinds of features are not meaningful for the task at end. In fact, we aim at inferring low level camera-model noise-related details for exposing potential tampering traces. Therefore, we propose to exploit the denoising network architectures defined in Section 2.1, which are trained to extract noise-related low-level features from the input images (see Section 4.1). By doing so, the embedding vectors that we can extract from the intermediate layers contain the information that the fingerprint extractor uses for constructing the residual image, which are in fact meaningful features for our task.

Furthermore, we never resize the input images, contrarily to what is proposed in the original PaDiM approach, which applies a resizing to the input image instead of patching it. We avoid resizing the images as it inevitably results in a loss of the low level model-related artifacts.

4.3. Deployment stage

When a query image has to be analyzed, we pass it through the trained fingerprint extractors to extract features by selecting specific network layers. Then, we exploit our proposed AD algorithm to find local anomalies in the analyzed image.

Notice that, to correctly apply the algorithm, we have to make a one-to-one comparison of patch embeddings from test and train images. This means that, the spatial size of the input images at test step must be the same as that of the patches used at train step, i.e, $P \times P$. To avoid any resizing operation, we operate in a patch-wise framework.

In particular, the procedure done for extracting the heatmap from an image of size $H \times W$ is the following:

1. First, the image has to be correctly pre-processed as required by the fingerprint extractors (DnCNN or Restormer).
2. We pad the images to avoid unwanted border artifacts. In particular, we use reflection padding.
3. We proceed by feeding the fingerprint extractor with the padded images. Here we have to make a distinction between the pipelines of DnCNN and Restormer. As mentioned before, DnCNN can process the whole image, while Restormer needs to process the image by patches. Specifically, we use square patches of size $P \times P$, and instead of using overlapping, we pad the patches by using their surrounded pixels, this way we do not add border artifacts on this step. As previously mentioned, for DnCNN we hook the last three layers, while for Restormer we only use the reconstructed fingerprint.
4. Then we create the patch embedding vectors in the same way they were created at training step (See 4.2.1). In the case of Restormer, we already have the patch embeddings for an input of size $P \times P$, while for DnCNN, we have to properly divide the complete activation maps into patches of activation maps corresponding to inputs of size $P \times P$.
5. Then, the Mahalanobis distance [26] between each embedding vector and the corresponding pre-computed multivariate Gaussian distribution is calculated, resulting in an anomaly score at every position (i, j). This is done for all the patches into which the original image was decomposed, resulting into score maps of size $P \times P$. Finally, the heatmap of the whole input image is reconstructed by properly joining all the obtained score maps.

It is important to point out that as in the original approach, we must have the same number of embeddings at both training and testing time, this means that during the deployment stage we have to use inputs with the same size as the patches $\mathbf{P}_k$, i.e, $P \times P$.

5. Experimental Setup

5.1. Datasets

As previously mentioned, the pipeline can be divided into two distinct tasks. First, the training of the fingerprint extractor was conducted, followed by the use of an anomaly detection method to derive a heatmap from the fingerprints. Consequently, two different types of datasets were employed. To train the fingerprint extractor, datasets comprising images from various camera models were necessary. Conversely, for testing the results and assessing performance, datasets containing manipulated images along with their respective masks were used. These manipulated images and masks were essential for measuring different metrics and determining the method's performance. For clarity's sake, Table 1 show a resume of the used datasets. In the following lines, we provide more details for each of them.

Table 1: Resume of the used datasets, including their format, number of images and purpose.

Set	Format	# of images	Type	Training	Testing
Dresden	.jpeg	16682	Pristine	✓	
Vision	.jpeg	7566	Pristine	✓	
IEEE SPC2018	.jpeg	2750	Pristine	✓	
DSO-1	.png	100	Manipulated		✓
JPEG-based GBMD	.jpeg	200	Manipulated		✓
PNG-based GBMD	.png	200	Manipulated		✓

5.1.1 Pristine image datasets

Dresden: The Dresden dataset is a public dataset produced by Gloe and Böhme [20] with the purpose of developing and evaluating performance of camera-based image forensic techniques. The dataset consists of 16692 images taken from 25 different camera models, and the images were taken in controlled indoor and outdoor scenarios in such a way that all the acquisitions share the same conditions, due to the fact that most of the models compress by default the images, they used the highest quality compression available by each device, apart from the natural JPEG images, Dresden provides dark and flat field images for facilitating the estimation of model-specific noises. For our purposes, only the natural JPEG images were used.
Vision: Vision is another public dataset published by Shullani et al [32], this dataset includes images and videos taken from 35 different portable devices corresponding to 29 different models, the complete dataset is composed by 34427 images and 1914 videos, all of them acquired in different four formats, the native camera, and three through social media platforms (WhatsApp, YouTube and Facebook), however, for our purposes we only used 7566 images corresponding to the natural JPEG images captured with the native camera. Vision also provides flat field images for the estimation of the model noise, again, for the training of the noiseprint extractor we only used the default natural JPEG compressed images, as for Dresden, Vision used the highest JPEG compression quality available by each device.
IEEE SPC2018: This dataset consists of 2750 images from 10 different portable camera models, however, some of them are shared with the Vision dataset.

5.1.2 Manipulated image datasets

In order to validate our results, we need some datasets containing manipulated images and their respective masks. The following datasets are used for measuring the performance of our method.

Spliced-Based Manipulated Dataset (SBMD): First we constructed a spliced-based manipulated dataset, to do so, we make use of the pristine datasets introduced in 5.1.1. We start by selecting a random background image from a specific camera model, then, we randomly sample another image from a different model, we will call this the splice image, we proceed by extracting a patch from a random position of the splice image and pasting it into another random position of the background image.

For testing purposes, we created three different configurations of this dataset. As we have mentioned before, PaDiM requires a set of training images for extracting the characteristics of the normal class, so we want to investigate how the performance of PaDiM is affected in the cases where it is used on images either captured by camera models seen or not seen during training.

To do so, we divide the 64 camera models into two groups, one that we call known classes, and the other called unknown classes. Then the three sub-datasets are build as follows

1. The first sub-dataset is done by randomly sampling both the background image and the splice image from the group of known classes
2. The next one is built by randomly sampling the background image from the known classes and randomly sampling the splice image from the unknown classes.
3. Finally, the last sub-dataset is done by sampling both the background image and the splice image from the unknown classes.

In all the cases, the corresponding mask is also computed. This way we are able to evaluate the performance on the different configurations.

DSO-1: The DSO-1 dataset was introduced by Carvalho et al. [12]. This dataset comprises 200 images, including indoor and outdoor scenes, with 100 being original and 100 manipulated images. The manipulations involve the addition of one or more persons into images that originally contained individuals. These alterations were executed through splicing techniques and, in some cases, further enhanced using post-processing methods to refine the results. Alongside the manipulated images, the authors provide the corresponding ground truth masks for validation and performance measuring.

Generative-Based Manipulated Dataset (GBMD): Apart from a spliced-based manipulated dataset, we decided to also include forgeries created by a different technique. In particular, we used DALL·E [30] as a generative AI tampering generator. DALL·E is a generative diffusion-based AI technology that allows us to introduce realistic forgeries inside images.

Let us first illustrate better the process of generating one tampered image with DALL·E.

1. We begin by uploading a color image of size $H \times W$.
2. DALL·E works on a 1024×1024 generation frame, this means that if any of the image dimension is lower than 1024, DALL·E will pad the image and complete the scene in the border. In the case that $H > 1024$ or $W > 1024$, DALL·E will either ask to crop the image to 1024×1024 or select the generation frame inside the image. In our particular case, all the images fulfil this last condition. We proceed by selecting the 1024×1024 generation frame inside our image instead of cropping it.
3. Inside the generation frame, we erase a portion of the image, inside this portion DALL·E will recreate the image.
4. After both the generation frame and the erased section have been selected, DALL·E allows inputting a prompt specifying the instructions for reconstructing the erased area.
5. Finally, DALL·E offers four possible generations from which we can choose only one to remain on the complete $H \times W$ image. However, all the four generations will be saved but cropped to the generation frame.
6. We end up with four 1024×1024 cropped images and one complete image.

To create the manipulations, we start from images taken from the Raise dataset [34]. This dataset is composed of 8156 RAW images from which we do not know the camera models used. To simulate realistic scenarios, we consider input images to be either uncompressed PNG or JPEG-compressed. To do so, we randomly select 50 images from the Raise dataset and convert them to both types. By following this procedure, we create two datasets, a JPEG-based GBMD and a PNG-based GBMD, both of them containing 200 images, corresponding to four cropped frames per each of the 50 selected images.

Notice that ther two datasets may be created by using the complete versions of the manipulated images. However, for this work, we decided not to use them, as they require important memory consumption and long computation time. Moreover, the smaller image size of the cropped image datasets enables us to evaluate the detector performance over a bigger amount of samples, thus providing more general results.

Figure 10 shows an example of the five generated images starting from one pristine image.

5.2. Training

In this section, we review the training process, including the setup and the hyperparameters used.

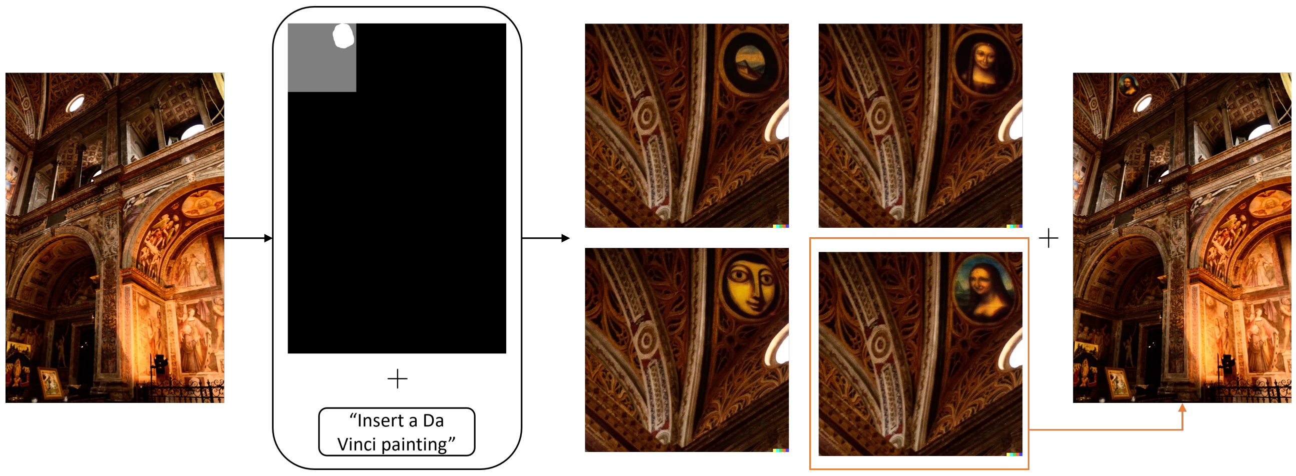


Figure 10: Processing of creating manipulated images using DALL·E. We start from an input pristine image, then we provide a 1024×1024 generation frame (Gray square). Inside this generation frame, we erase a portion of the image (white area). Finally, we provide a text prompt with the instructions for the generation. Four 1024×1024 output images are provided, from which, we select one to complete the entire input image.

5.2.1 Fingerprint Extractor

When combining all the datasets, we have in total a bit more than 27000 images captured from 64 different devices.

For training the DnCNN extractor, we used mini-batches of 200 patches coming from 100 different images at each training iteration, i.e, for each image we extracted two patches. These 100 images correspond to 25 different camera models, so 4 images per model are selected. It is important to notice that the two patches are taken at random positions, but these two positions are maintained over all the images in the batch. To deal with the fact that images may have different sizes, we sampled the patch positions, taking into account the dimensions of the lowest resolution image of the batch. After the 200 patches have been extracted, we compute the 200×200 pairwise dense matrix with all the possible combinations of patches, assigning the corresponding label to each pair, which is 1 when both patches come from the same camera model and have been extracted at the same image position, and 0 otherwise.

On the other hand, when training the Restormer version of our model, instead of sampling 25 camera models for each batch, we only sample 4 models, so instead of having 100 images and 200 patches, we have 16 images and 32 patches for each batch, this because of memory limitations due to the high size of Restormer.

In both cases (DnCNN and Restormer), we used pretrained weights shared by the original authors of the architectures in order to initialize our weights. Both authors proposed different configurations for their denoisers, one for real imaging denoising, three for Gaussian gray scale image denoising with values of $\sigma = 15, 25, 50$, where σ is the noise level, and three for Gaussian color images with the same values of σ. We tried to train our denoisers starting for all these possible configurations. We discovered that in both cases starting from the weights of the gray scale Gaussian image denoisers not only helped us with the convergence of the network, but also gave betters results in terms of loss and accuracy.

As far as the optimizer is concern, we used Adam with a starting learning rate of 0.001, we also added a learning rate scheduler in order to reduce the learning rate value by 10 when a certain patience is reached, and finally we used early stopping in order to avoid overfitting.

5.2.2 Anomaly Detector

The images used for training our AD come from the three pristine datasets introduced in 5.1.1. Again we have more than 27000 images belonging to 64 classes. One drawback of the proposed method is that, in order to compute the multivariate Gaussian, we must have the embedding vectors stored in memory. This means that the number of images used at the training step is limited by the RAM capacity. However, we selected the images in such a way that all the 64 camera models are seen during training. We discovered that using 50 images per model is enough to reach optimal results, in the sense that increasing the number of images per model over this number will give no benefit.

As mentioned before, we have to select a patch size for training our AD. This because of two reasons, first, all the images must have the same size since we need to have the same number of patch embeddings for all the images, and second, we cannot use high-resolution images because of memory limitations. We have then to select a portion of the training images, since, as mentioned before, resizing is not feasible.

The selected patch size is 48×48, and the cropping size mentioned in 4.2.1 is 192×192, this decision was done by means of cross validations and is valid for both DnCNN and Restormer cases.

5.2.3 Performance metrics

To evaluate the performance of the proposed method, we use the ROC Area Under the Curve (ROCAUC) and the Matthews Correlation Coefficient (MCC) metrics. We propose these two functions because they are known to work well on unbalanced data. Indeed, we expect to have a high percentage of non-manipulated pixels in comparison with the manipulated ones.

ROCAUC and MCC are defined from the four most basic metrics of binary classification,

1. TP (true positive): number of positive pixels declared positive.
2. TN (true negative): number of negative pixels declared negative.
3. FP (false positive): number of negative pixels declared positive.
4. FN (false negative): number of positive pixels declared negative.

The MCC is a very meaningful metric for our case, not only because it is suitable for unbalanced datasets, but also because it takes into account all possible scenarios by using all the four metrics mentioned before. MCC can be calculated as

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}} \tag{7}$$

where the MCC takes values from -1 to 1, 1 meaning all the points were correctly classified, 0 meaning that the performance is the same as a random classifier and -1 meaning total disagreement between predictions and ground truth.

To define the ROCAUC value, first we have to define the Receiver Operating Characteristic (ROC) curve. The ROC curve is a graphical representation of a classifier's performance that illustrates the trade-off between true positive rate (TPR) and false positive rate (FPR) at different classification thresholds. TPR and FPR are defined as the ratio between the TP over all positive observations and the ratio between the FP and all negative observations, respectively. The ROC curve is then created by plotting TPR against FPR for different values of threshold, after having the ROC curve, the ROCAUC is simply the area under its curve. ROCAUC can take values from 0 to 1, the closer the ROCAUC value is to 1, the better the model's ability to distinguish between the two classes, a ROCAUC value of 0.5 means that the model's performance is the same as that of a random classifier.

One of the advantage of the ROCAUC metric is that it is also suitable for unbalanced data. On top of that, contrary to the MCC, the ROCAUC is threshold independent, this means that it provides an overview of model performance across all possible threshold settings without being sensitive to a specific threshold value.

An important aspect to take into account is that after the image is completely processed and the heatmap is generated, we cannot control whether the final configuration is such that higher values inside the map correspond to manipulated pixels and lower to non-manipulated or if it is the other way around. Moreover, the configuration of the heatmap may vary from image to image, for this reason we have to take into account both cases. This is done either by inverting the heatmap values or directly the mask values. In particular, we consider the two possible mask configurations; first, we assign to the tampered pixels a value of 1, and 0 to the non-manipulated ones, just as stated in equation 2. Then, we invert the mask by computing $\mathbf{M}'(h, w) = 1 - \mathbf{M}(h, w)$. We proceed by calculating the performance metrics for both cases and keep only the better results, this is done for each image.

6. Results

In this section, we present the results and performance of our method on the proposed datasets. First, we investigate the behavior of our method in a controlled scenario by making use of our automatic generated dataset. Then we report the outcomes in different scenarios such as cropped or resized images. Finally, we make a comparison with the state-of-the-art techniques.

6.1. Know vs unknown camera models

We start by investigating the behavior of our detector in different training-testing scenarios. In particular, we want to inspect how the knowledge of a certain camera model during training affects the performance. For achieving this, we propose three different experiments making use of the SBMD dataset introduced in Section 5.1.2. These are described in the following lines:

1. **Known-known camera models.** We use random images from the group of known classes to train our AD detector. Then, we run the detector on a test set composed of tampered with images created only by using images from the group of known classes.
2. **Known-unknown camera models.** As before, we employ random images from the group of known classes to train our AD detector. Then, we run the detector on a test set composed of tampered with images in which the background comes from the group of known classes, while the splicing area belongs to the group of unknown classes.
3. **Unknown-unknown camera models.** We train our AD detector by using images from the group of unknown classes. Then, we run the detector on a test set composed of tampered with images created by considering only images from the group of unknown classes.

To understand the purpose of these experiments, let us first recall the general reasoning behind AD algorithms. These algorithms use a set of images from the "normal class" to encode their characteristics, which are compared with those of the testing images to quantize how deviated they are from the training distribution. In our case, the "normal" class corresponds to pristine images. However, if both the camera model of the background image and that of the spliced region were seen at the training step, our AD detector might recognize both patterns as "normal". If this were the case, even if we were able to see two distinct patterns in the noise fingerprint, the heatmap would risk to identify all pixels in the image as pristine pixels. By running these experiments, we investigate if we are still able to discriminate between the two patterns, even if these are identified as "normal" during the AD training phase.

The obtained values for both MCC and ROCAUC for all the experiments are reported in the tables 2, 3 and 4.

Table 2: Experiment 1 (Known-knwon).

	DnCNN	**Restormer**
ROCAUC	0.958	0.899
MCC	0.677	0.578

Table 3: Experiment 2 (Known-unknown).

	DnCNN	**Restormer**
ROCAUC	0.966	0.918
MCC	0.735	0.606

From these results, we can observe that for both models, the worst case scenario is when the camera models of the test images have already been seen during training. These results confirm what anticipated; however, we still achieve very good results even in this case. When we compare the worst case (known-known) with the best case (unknown-unknown), the difference in the ROCAUC values is about 3% in the case of Restormer, while only 1.5% for DnCNN. When it comes to the MCC, both model's performance is affected by around a 3.5%.

Table 4: Experiment 3 (Unknown-unknown).

	DnCNN	**Restormer**
ROCAUC	0.974	0.932
MCC	0.749	0.649

Another result that we can see from these experiments is that, for this specific dataset, DnCNN outperforms Restormer by approximately 5% accuracy in all the cases.

6.2. Test on the uncontrolled Datasets

Now we proceed to test our method on the two dataset over which we have no control, DSO-I and GBMD. By no control we mean that, in the case of DSO-I, we have no information on the devices and camera models used for capturing the images, while on GBMD we have no information on how the image is exactly processed by DALL·E.

Tables 5 reports the results of each of our models on DSO, PNG-based GBMD and JPEG-based GBMD datasets.

Table 5: Results of testing our method on the three proposed datasets. In bold, the best results per dataset.

	DSO		JPEG-based GBMD		PNG-based GBMD	
	DnCNN	Restormer	DnCNN	Restormer	DnCNN	Restormer
AUC	0.951	**0.968**	**0.975**	0.952	**0.834**	0.750
MCC	0.731	**0.843**	0.838	**0.859**	**0.545**	0.520

Looking at the general results on each dataset, we can observe that both models obtain remarkable results on both DSO and JPEG-based GBMD datasets, while on the PNG-based GBMD dataset even if the metrics are not as high as in the other two cases, the performance is still promising. Recalling that MCC can take values from -1 to 1, while ROCAUC takes values from 0 to 1, we can see that in general, the results of both metrics are highly in agreement with each other.

Looking deeply at the results of Table 5, we observe better performance coming from the Restormer model, outperforming in a moderate quantity the DnCNN model. The highest contrast can be seen on the MCC metric, where Restormer passes DnCNN by about 0.11. The contrary comes out when testing on the PNG-based GBMD dataset, where DnCNN performs slightly better than Restormer. In particular, the highest difference is on the ROCAUC, where DnCNN is above Restormer by roughly 0.08. Finally, on the JPEG-based GBMD dataset, table 5 shows that each denoising model stands out on one of the metrics, but the overall performances are valid for both of them.

Moreover, from these results, we can note how the model's performance is highly affected in the case we use PNG images for generating the GBMD dataset compared with the case in which we use JPEG images. In the case of the DSO dataset, which is shared as PNG images, the results lead us to think that, in some step of the dataset production, JPEG compression was applied.

Figures 11, 12 and 13 show three examples of Fingerprints and heatmaps using both our models for the DSO, JPEG-based GBMD and PNG-based GBMD datasets respectively.

6.3. Test on JPEG compressed images

We want now to explore how JPEG compression applied to the manipulated images can affect the performance. For this task, we resort to the DSO-1 dataset, which is conformed by PNG images. As we have seen in Section 6.2, our method has a great performance on this dataset. We want now to investigate if JPEG compressing on the tampered images erases the camera model traces and, thus, influences in the method's performance. To do so, we add a new step on the image preprocessing, applying JPEG compression before feeding the image

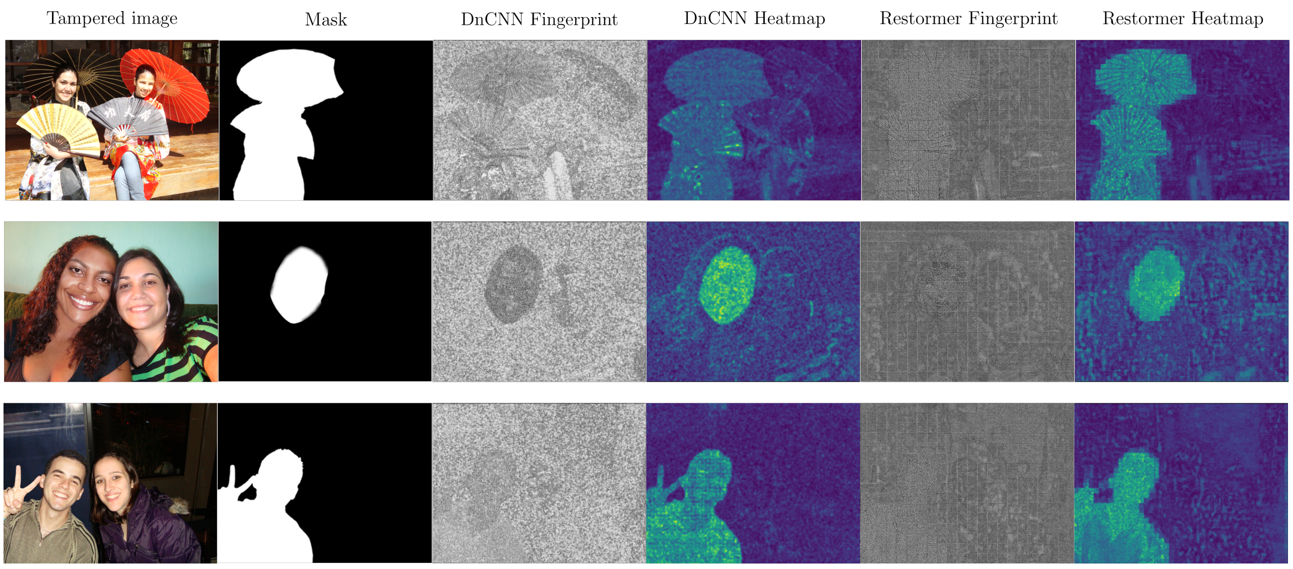


Figure 11: Results on the DSO dataset. From left to right we observe the tampered image, the corresponding mask, the fingerprint extracted using DnCNN, the heatmap generated using DnCNN, the fingerprint extracted using Restormer and the heatmap generated using Restormer.

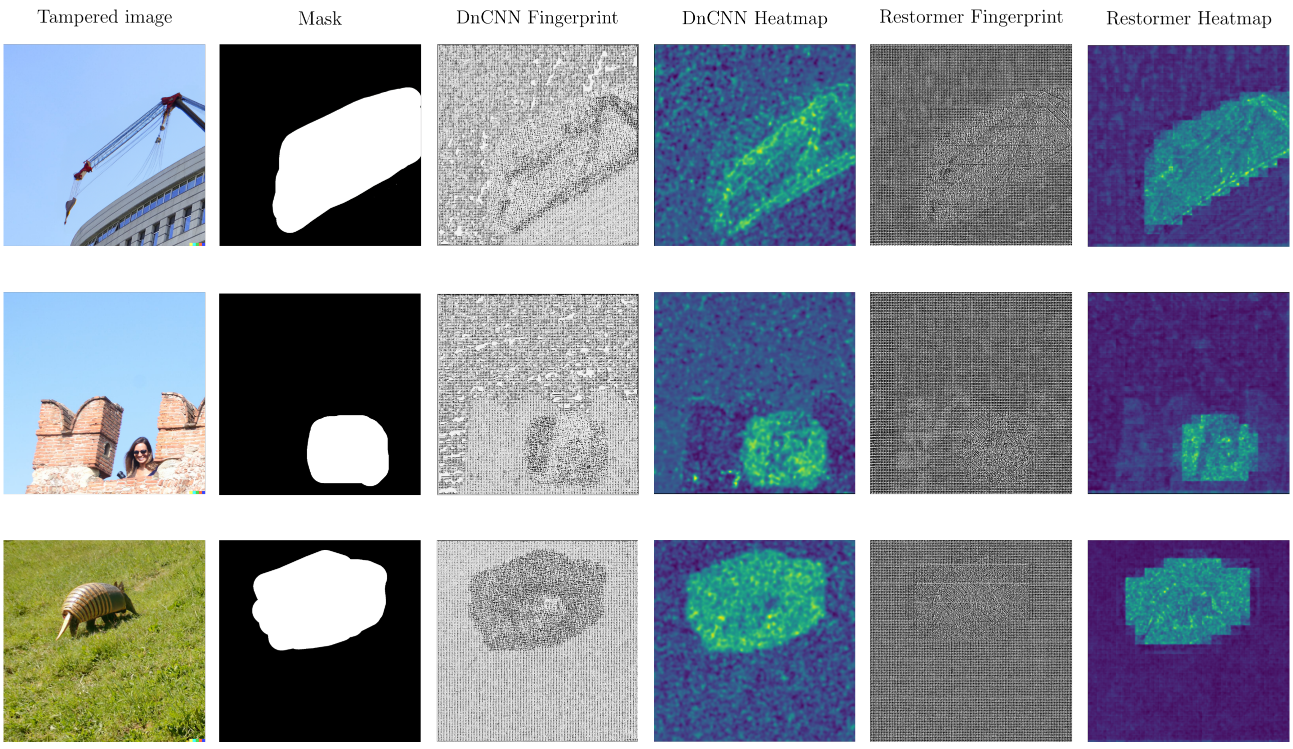


Figure 12: Results on the JPEG-based GBMD dataset. From left to right we observe the tampered image, the corresponding mask, the fingerprint extracted using DnCNN, the heatmap generated using DnCNN, the fingerprint extracted using Restormer and the heatmap generated using Restormer.

into the fingerprint extractor. We do this for two different compression quality factors to investigate if this parameter affects the results.

Table 6 shows the results of our model on the DSO dataset after JPEG compressing the images. In particular, we show the results for quality factors of 90, 95 and 100.

From table 6 we can notice that JPEG compressing the image with the highest possible quality factor does not have a big impact on the model's performance. In fact, if we compare these results with those of table 5, in which the configuration of the experiment was the same except for the fact that the images were not JPEG compressed, we note by looking at both metrics, that the impact is less than 2% for both models.

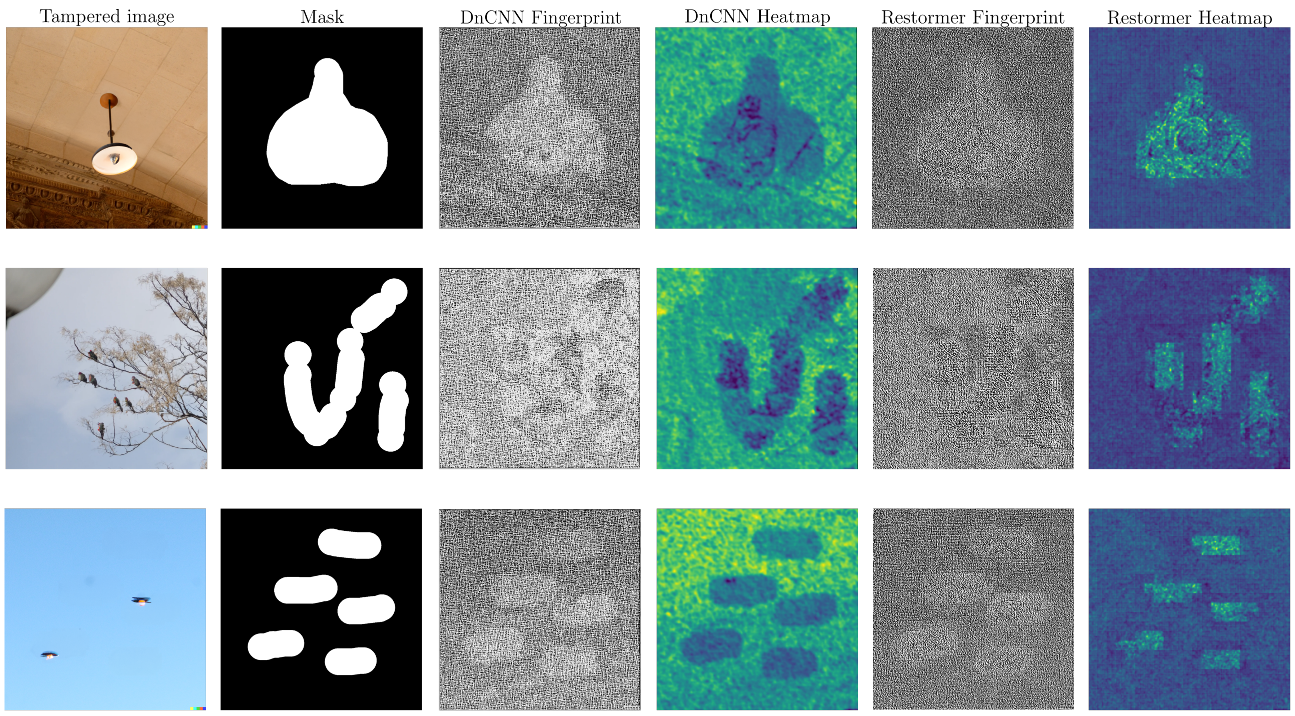


Figure 13: Results on the PNG-based GBMD dataset. From left to right we observe the tampered image, the corresponding mask, the fingerprint extracted using DnCNN, the heatmap generated using DnCNN, the fingerprint extracted using Restormer and the heatmap generated using Restormer.

Table 6: Results on JPEG compressed images with quality factors 90, 95 and 100.

	DnCNN			**Restormer**		
QF	100	95	90	100	95	90
ROCAUC	0.943	0.682	0.643	0.959	0.644	0.598
MCC	0.722	0.229	0.178	0.832	0.187	0.135

On the other hand, we observe that reducing the quality factor has an unwanted effect on the results which are very poor, with this effect being more critical on the Restormer model, even for a relatively high value of 95. We can conclude that JPEG compressing images without using the highest quality factor has a great impact on the camera-model artifacts present on them, almost completely erasing them.

6.4. Test on resized images

In this experiment, we test the performance of our method when applying different resizing factors to the input images. To do so, we rely only on the DSO dataset. We report the results of applying 0.8×, 0.9×, 1.1× and 1.2× scaling in Table 7.

We can notice a huge impact in the performance for both models, even at 0.9× and 1.1× scaling, being DnCNN the most affected one. Restormer is the most robust model, obtaining a ROCAUC higher than 0.7 for three of the four cases analyzed. An interesting behavior can be observed when looking at the difference between upsampling and downsampling factors. In fact, table 7 shows that Restormer is noticeably more successful than DnCNN in the cases of 0.8× and 0.9×, while DnCNN outperforms Restormer by a moderate difference in the cases of 1.1× and 1.2×. Surprisingly, both models obtain better results for an upscaling factor of 1.2× than for 1.1×. With these results, we confirm that the camera-model traces that we want to isolate are highly affected when applying re-scaling to the images.

Figure 14 shows two downsampled examples, while figure 15 shows two upsampled examples from the DSO dataset. We notice that, even when having questionable average results on the MCC and ROCAUC, for some of the images we can still recognize the affected area on the heatmaps, specially for the case of the model based

Table 7: Results of testing our method on resized images for scaling factors of 0.8×, 0.9×, 1.1× and 1.2×.

	DnCNN				**Restormer**			
Resizing factor	0.8	0.9	1.1	1.2	0.8	0.9	1.1	1.2
ROCAUC	0.618	0.657	0.696	0.706	0.706	0.714	0.686	0.703
MCC	0.155	0.170	0.221	0.227	0.274	0.279	0.204	0.223

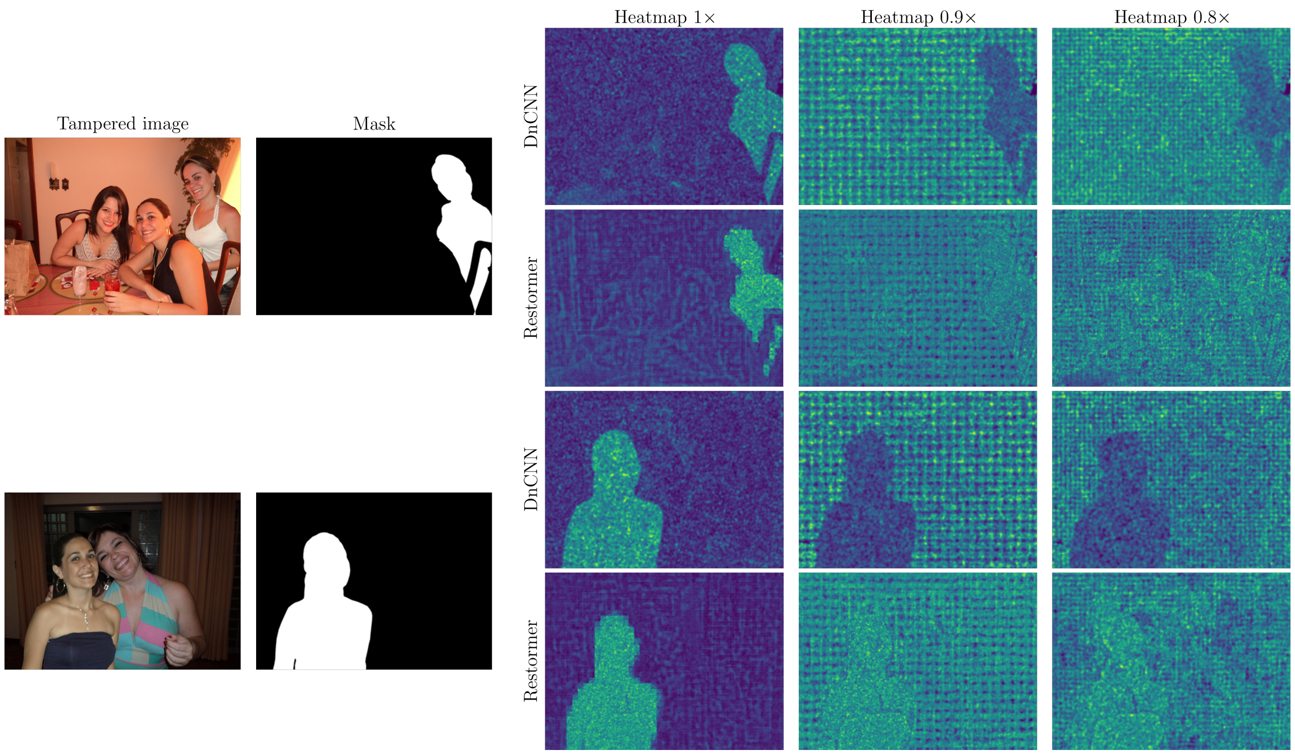


Figure 14: Results on resized images of the DSO dataset. We show two different examples, for each of them we report the 1×, 0.9× and 0.8× heatmaps obtained with both of our models.

on the DnCNN denoiser. This, however, does not generalize through the whole dataset.

6.5. Test on differently cropped images

We have mentioned before (see Section 4.1.1, that the fingerprint traces that we are extracting usually follow a periodic pattern, moreover, this pattern change for different noise sources as well as its periods. This means that, cropping or misaligning the images may affect drastically the results. For this reason, we aim to investigate how different cropping sizes impacts the forgery localization performance.

So far, during the training stage of our anomaly detector, we have been selecting patches aligned with the JPEG 8×8 grid. However, in the case where the test images are unaligned to the JPEG grid, also their feature maps (extracted by our AD algorithm) will be in a way unaligned. As a consequence of this, we would be comparing JPEG-aligned maps from training images with JPEG-unaligned maps from testing images. This could insert some errors on the forgery localization, thus affecting the performance of the method.

For evaluating the effects of the alignment on the results, we propose the following experiments:

1. We train our anomaly detector on patches aligned with the 8×8 JPEG grid, then we test the performance on randomly cropped images. With this experiment, we want to evaluate how robust our proposed detector is when facing random image cropping scenarios. We know that camera-model artifacts stamp different periodic patterns on images, meaning that extracting these patterns from cropped images is a more difficult task, specifically for neural network-based algorithms that may learn and overfit the original configuration and periods of these traces.

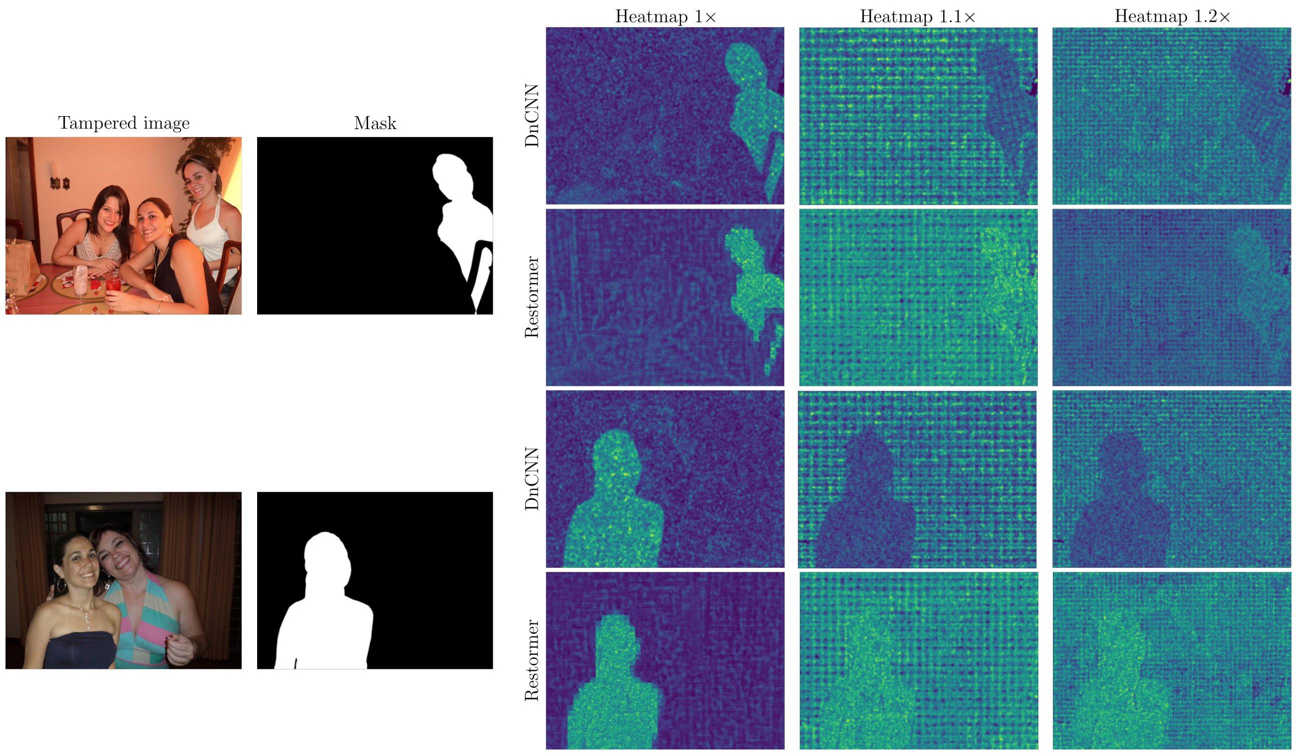


Figure 15: Results on resized images of the DSO dataset. We show two different examples, for each of them we report the 1×, 1.1× and 1.2× heatmaps obtained with both of our models.

2. We train our anomaly detector on randomly positioned image-patches (misaligned with the JPEG grid), then we test the performance on JPEG-unaligned images. With this second experiment, we aim to explore if training with patches extracted at random positions can provide better results than those of the previous experiment when applied to randomly cropped images.
3. We train our anomaly detector on randomly positioned image-patches, then we test the performance on images aligned with the 8×8 JPEG grid. This experiment is proposed to check if, by training our anomaly detector on patches misaligned with the JPEG grid, we obtain better results than training in an aligned fashion, in case of testing on JPEG-aligned images.
4. We train our anomaly detector on patches not aligned with the 8×8 JPEG grid but extracted from a known pixel position, then we test the performance on images cropped in such a way that their pixel grid alignment is the same as in the training patches.

Table 8: Experiment 1.

	DnCNN	**Restormer**
ROCAUC	0.881	0.727
MCC	0.596	0.427

Table 9: Experiment 2.

	DnCNN	**Restormer**
ROCAUC	0.874	0.746
MCC	0.592	0.448

All the experiments are done using both our denoising models (DnCNN and Restormer). The results are shown in Tables 8, 9, 10 and 11. We comment the results for each experiment as follows:

1. We can see from Table 8 that both our denoising models are affected by the modification, specially in the case of Restormer, where we can observe a notable drop in the performance. In particular, comparing

Table 10: Experiment 3.

	DnCNN	Restormer
ROCAUC	0.860	0.735
MCC	0.554	0.431

Table 11: Experiment 4.

	DnCNN	Restormer
ROCAUC	0.947	0.961
MCC	0.704	0.825

with the JPEG-aligned results in Table 5, we report a reduction of around 20% on both metrics. On the other hand, this decrement is more moderate for the DnCNN model, in fact, we observe a reduction of 0.135 in the MCC and only 0.07 in the ROCAUC, which represent less than 7% in both cases. As contrary to the previous experiments, in this special case we found that DnCNN is remarkably more robust than Restormer.

2. From Table 9, we observe that the Restormer model achieves a small improvement of around 2% is obtained with respect to the previous scenario, shown in Table 9. However, for DnCNN we obtain a performance reduction.
3. As we can see from Table 10, both models perform worse in this case than when trained on aligned patches (See Table 5).
4. If we observe Table 11, we notice that the results are very close to those of Table 5, meaning that tampering localization on cropped images can still be done with high performance. This, however, is not an efficient solution since it would require testing the images on 63 different models (trained by considering all the possible pixel grid misalignments inside the 8×8 grid).

From the previous experiments, we note a strong dependence of our model on the alignment used. In fact, optimal performance is only obtained when both train and test inputs are aligned together.

As mentioned before, a solution for this drawback is simply training our anomaly detector on all the possible unaligned positions. By doing so, we could feed the input images to the different models and when the misalignment of the images matches that of the model, we will reasonably obtain the best possible results. Nonetheless, this approach is extremely ineffective in terms of deployment time, so we should go to the root of the problem to solve it. As we have seen, the fingerprint extractors were trained with datasets solely composed by JPEG images. Furthermore, cropping augmentation was not used during training, leading to possible overfitting in the JPEG compression artifacts. Given these considerations, we expect that retraining the fingerprint extractors by possibly expanding the training datasets to different format images and applying cropping augmentation will result into more robust models capable of overcome the mentioned drawbacks.

6.6. Comparison with state-of-the-art

We proceed to compare our results with state-of-the-art methods, in particular, with the method from which we take inspiration, the Noiseprint [11].

Table 12: Comparison on DSO dataset. In bold the best results.

	DnCNN	Restormer	Noiseprint
ROCAUC	0.951	**0.968**	0.926
MCC	0.731	**0.843**	0.722

We make the comparison by taking as a reference our best models, namely the Restormer-based detector in the cases of the DSO and JPEG-based GBMD, and the DnCNN-based one for PNG-based GBMD. By looking at the results on the DSO dataset in Table 12, we observe a considerable advantage for our method. In the

Table 13: Comparison on JPEG-based GBMD dataset.

	DnCNN	**Restormer**	**Noiseprint**
ROCAUC	**0.975**	0.952	0.961
MCC	0.838	**0.859**	0.841

Table 14: Comparison on PNG-based GBMD dataset.

	DnCNN	**Restormer**	**Noiseprint**
ROCAUC	0.834	0.750	**0.938**
MCC	0.545	0.520	0.**769**

case of the JPEG-based GBMD dataset (see Table 13), both models have almost the same performance. The Noiseprint method surpasses our method by 0.009 when looking at the ROCAUC, while we have an advantage of 0.018 on the MCC. Finally, taking a look at Table 14, we observe a noticeable out-performance by the Noiseprint. However, this result was expected from the previously analyzed experiments. Indeed, we already noticed that our method has disadvantages when used on images that were not JPEG compressed before the forgery was added.

6.7. Deeper look at JPEG compression

As we have seen in Section 6.3, JPEG compressing by quality factors different from 100 significantly affects the performance of our method. However, we want to take a deeper look at the results. On figure 16, we observe one example of a JPEG compressed image by quality factors of 90 and 95, fingerprints and heatmaps are shown for both of our models. In particular, we can observe how specially in the case of DnCNN, the fingerprints clearly show the two different patterns, suggesting a tampering on the image. Nonetheless, the heatmaps do not correctly capture these patterns, resulting on the unsatisfactory results reported before.

This behavior expose a possible drawback on the Anomaly Detection (AD) method. Our proposed AD proved to be very efficient on most of the tested cases, however in this particular case we observe that it may be failure prone. Recall that our AD method has some hyperparameters that have to be tuned, for instance, the selected layers for hooking the activation maps. In the previous sections, we presented what we found to be the most robust configuration overall, nevertheless, there are particular cases in which specific configurations may result in better performance. Figure 17 shows three more examples of the presented case only for the DnCNN model. In the last column, we add the heatmap generated by hooking the activation map only from the last layer. We observe how, by changing the configuration of the AD method, the resulting heatmap retrieve better the two patterns even if using the same fingerprint extractor.

7. Conclusions

In this work, we faced the problem of image forgery localization, in particular, the cases of image splicing and image manipulation with generative AI technologies. These two types of forgeries can be made with tools that are available to almost any person in possession of a smartphone, which makes the dissemination of manipulated images a problem of great concern.

Inspired by two well known state-of-the-art algorithms, we proposed a method capable of exposing tampering regions inside images. To do so, we trained a denoiser capable of extracting a camera-model fingerprint from images. Then, we use activation maps coming from different layers of the denoiser to apply an AD procedure to these fingerprints, resulting into a heatmap that can be interpreted as a probability map of tampered pixels.

Our technique showed promising results on both types of dataset used: one containing splicing manipulations, and two containing generative AI manipulations. In most of the considered experiments, our method outperformed one of the top state-of-the-art techniques. However, when we applied it to images that have never been JPEG compressed or resized, the performance is affected considerably. Future works will be dedicated to

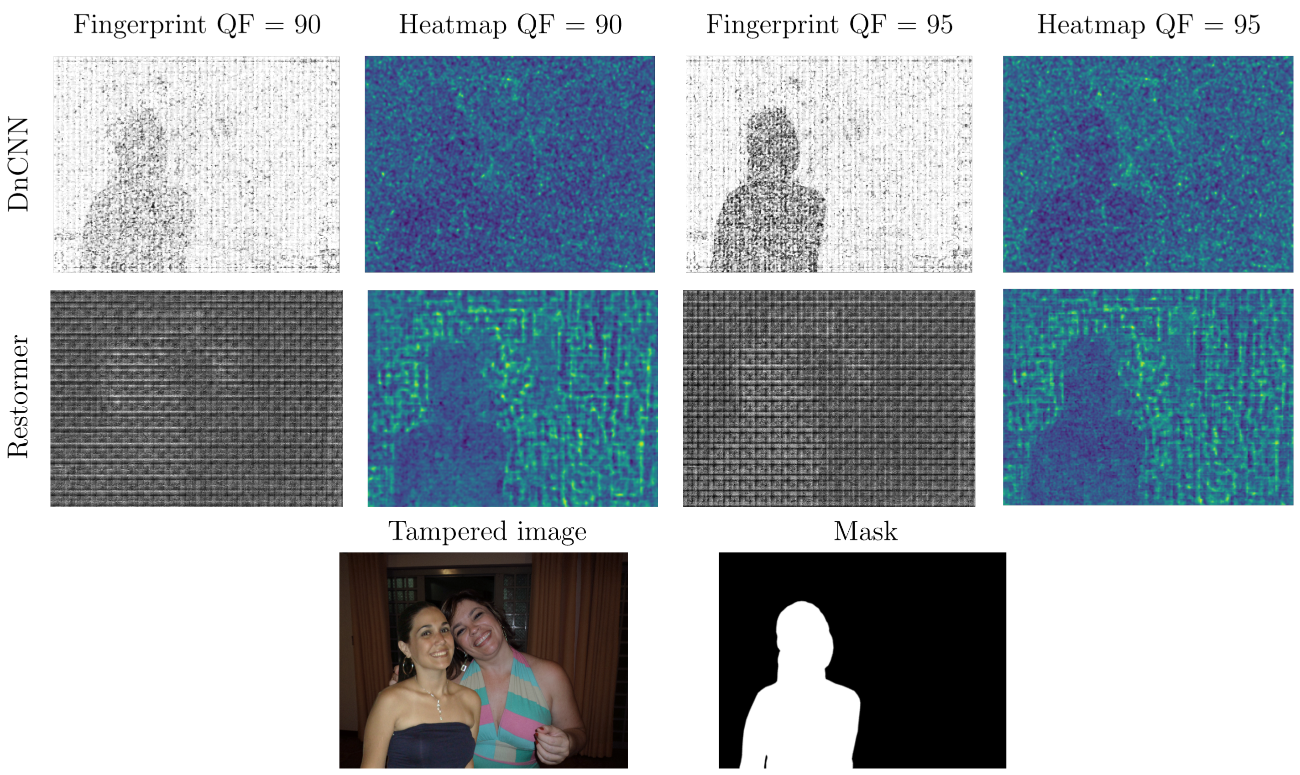


Figure 16: Results on JPEG compressed images of the DSO dataset. We show the fingerprints and heatmaps for JPEG compressions with quality factors of 95 and 90.

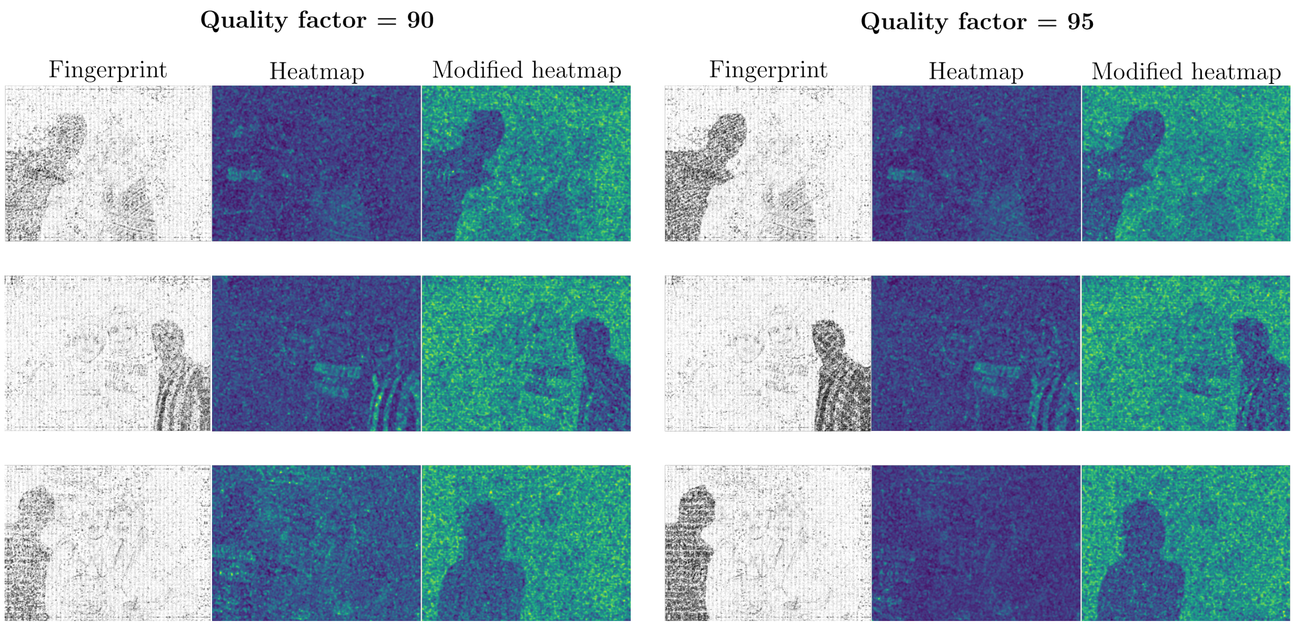


Figure 17: Results on JPEG compressed images of the DSO dataset by modifying the AD method's configuration. We show the fingerprints, heatmaps and modified heatmaps for JPEG compressions with quality factors of 95 and 90.

further investigations for enhancing the robustness on post-processed images.

By running different experiments, we got to the conclusion that our denoisers overfitted the JPEG compression noise pattern. This due to the fact that we only trained them on JPEG images. Knowing this, we proposed a solution for cases in which the possibly tampered input images may be cropped, this solution consists on training our anomaly detector also on cropped images. By doing so, we are still able to detect manipulations with a high efficacy, however, even doing this, we still cannot obtain good results on images that were no JPEG compressed before the manipulation.

Being aware of our drawback, we propose for future works, retraining the denoisers using a more diverse set of data, which includes different format and cropped images. This way, we expect to overcome the encountered problem and generalize our method to a wide range of real case-scenarios, making it more robust to several configurations of the input images.

References

[1] Paul Bergmann, Kilian Batzner, Michael Fauser, David Sattlegger, and Carsten Steger. The mvtec anomaly detection dataset: a comprehensive real-world dataset for unsupervised anomaly detection. *International Journal of Computer Vision*, 129(4):1038–1059, 2021.

[2] Tiziano Bianchi and Alessandro Piva. Image forgery localization via block-grained analysis of jpeg artifacts. *IEEE Transactions on Information Forensics and Security*, 7(3):1003–1017, 2012.

[3] Nicolo Bonettini, Luca Bondi, David Güera, Sara Mandelli, Paolo Bestagini, Stefano Tubaro, and Edward J Delp. Fooling prnu-based detectors through convolutional neural networks. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 957–961. IEEE, 2018.

[4] Edoardo Daniele Cannas, Nicolò Bonettini, Sara Mandelli, Paolo Bestagini, and Stefano Tubaro. Amplitude sar imagery splicing localization. *IEEE Access*, 10:33882–33899, 2022.

[5] Can Chen, Scott McCloskey, and Jingyi Yu. Image splicing detection via camera response function analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5087–5096, 2017.

[6] Jiaxin Chen, Xin Liao, Wei Wang, Zhenxing Qian, Zheng Qin, and Yaonan Wang. Snis: A signal noise separation-based network for post-processed image forgery detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 33(2):935–951, 2023.

[7] Shengda Chen, Shunquan Tan, Bin Li, and Jiwu Huang. Automatic detection of object-based forgery in advanced video. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(11):2138–2151, 2015.

[8] Giovanni Chierchia, Giovanni Poggi, Carlo Sansone, and Luisa Verdoliva. A bayesian-mrf approach for prnu-based image forgery detection. *IEEE Transactions on Information Forensics and Security*, 9(4):554–567, 2014.

[9] Davide Cozzolino, Giovanni Poggi, and Luisa Verdoliva. Splicebuster: A new blind image splicing detector. In *2015 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, 2015.

[10] Davide Cozzolino and Luisa Verdoliva. Camera-based image forgery localization using convolutional neural networks. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 1372–1376, 2018.

[11] Davide Cozzolino and Luisa Verdoliva. Noiseprint: A cnn-based camera model fingerprint. *IEEE Transactions on Information Forensics and Security*, 15:144–159, 2020.

[12] Tiago José de Carvalho, Christian Riess, Elli Angelopoulou, Hélio Pedrini, and Anderson de Rezende Rocha. Exposing digital image forgeries by illumination color classification. *IEEE Transactions on Information Forensics and Security*, 8(7):1182–1194, 2013.

[13] DeepfakesWeb.com. Deepfakes web, 2023.

[14] Thomas Defard, Aleksandr Setkov, Angelique Loesch, and Romaric Audigier. Padim: A patch distribution modeling framework for anomaly detection and localization. In Alberto Del Bimbo, Rita Cucchiara, Stan Sclaroff, Giovanni Maria Farinella, Tao Mei, Marco Bertini, Hugo Jair Escalante, and Roberto Vezzani, editors, *Pattern Recognition. ICPR International Workshops and Challenges*, pages 475–489, Cham, 2021. Springer International Publishing.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[16] Shilpa Dua, Jyotsna Singh, and Harish Parthasarathy. Image forgery detection based on statistical features of block dct coefficients. *Procedia Computer Science*, 171:369–378, 2020.

[17] Clare Duffy. Puffer coat pope. musk on a date with gm ceo. fake ai 'news' images are fooling social media users. *CNN*, April 2023.

[18] Pasquale Ferrara, Tiziano Bianchi, Alessia De Rosa, and Alessandro Piva. Image forgery localization via fine-grained analysis of cfa artifacts. *IEEE Transactions on Information Forensics and Security*, 7(5):1566–1577, 2012.

[19] Shreyasi Ghose, Nishi Singh, and Prabhishek Singh. Image denoising using deep learning: Convolutional neural network. In *2020 10th International Conference on Cloud Computing, Data Science and Engineering (Confluence)*, pages 511–517, 2020.

[20] Thomas Gloe and Rainer Böhme. The 'dresden image database' for benchmarking digital image forensics. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, page 1584–1590, New York, NY, USA, 2010. Association for Computing Machinery.

[21] Shital Jadhav, Rashmika Patole, and Priti Rege. Audio splicing detection using convolutional neural network. In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–5. IEEE, 2019.

[22] Alexander Kolesnikov, Alexey Dosovitskiy, Dirk Weissenborn, Georg Heigold, Jakob Uszkoreit, Lucas Beyer, Matthias Minderer, Mostafa Dehghani, Neil Houlsby, Sylvain Gelly, Thomas Unterthiner, and Xiaohua Zhai. An image is worth 16x16 words: Transformers for image recognition at scale. 2021.

[23] Brett Koonce and Brett Koonce. Efficientnet. *Convolutional Neural Networks with Swift for Tensorflow: Image Recognition and Dataset Categorization*, pages 109–123, 2021.

[24] Alexandra Levine. In a new era of deepfakes, ai makes real news anchors report fake stories. *Forbes*, October 2023.

[25] Jan Lukáš, Jessica Fridrich, and Miroslav Goljan. Detecting digital image forgeries using sensor pattern noise. In *Security, steganography, and watermarking of multimedia contents VIII*, volume 6072, pages 362–372. SPIE, 2006.

[26] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Sankhyā: The Indian Journal of Statistics, Series A (2008-)*, 80:S1–S7, 2018.

[27] Sara Mandelli, Nicolò Bonettini, and Paolo Bestagini. Source camera model identification. In *Multimedia Forensics*, pages 133–173. Springer Singapore Singapore, 2022.

[28] Pankaj Mishra, Riccardo Verk, Daniele Fornasier, Claudio Piciarelli, and Gian Luca Foresti. Vt-adl: A vision transformer network for image anomaly detection and localization. In *2021 IEEE 30th International Symposium on Industrial Electronics (ISIE)*, pages 01–06. IEEE, 2021.

[29] Niki J. Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning (ICML)*, 2018.

[30] Mr D Murahari Reddy, Mr Sk Masthan Basha, Mr M Chinnaiahgari Hari, and Mr N Penchalaiah. Dall-e: Creating images from text. *UGC Care Group I Journal*, 8(14):71–75, 2021.

[31] Mubbashar Saddique, Khurshid Asghar, Usama Ijaz Bajwa, Muhammad Hussain, and Zulfiqar Habib. Spatial video forgery detection and localization using texture analysis of consecutive frames. *Advances in Electrical & Computer Engineering*, 19(3), 2019.

[32] Dasara Shullani, Marco Fontani, Massimo Iuliani, Omar Al Shaya, and Alessandro Piva. Vision: a video and image dataset for source identification. *EURASIP Journal on Information Security*, 2017(1):1–16, 10 2017.

[33] Nina Shvetsova, Bart Bakker, Irina Fedulova, Heinrich Schulz, and Dmitry V. Dylov. Anomaly detection in medical imaging with deep perceptual autoencoders. *IEEE Access*, 9:118571–118583, 2021.

[34] Benjamin Staar, Michael Lütjen, and Michael Freitag. Anomaly detection with convolutional neural networks for industrial surface inspection. *Procedia CIRP*, 79:484–489, 2019. 12th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 18-20 July 2018, Gulf of Naples, Italy.

[35] Sasha Targ, Diogo Almeida, and Kevin Lyman. Resnet in resnet: Generalizing residual architectures. *arXiv preprint arXiv:1603.08029*, 2016.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[37] Nam N Vo and James Hays. Localizing and orienting street views using overhead imagery. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 494–509. Springer, 2016.

[38] Zhifeng Wang, Yao Yang, Chunyan Zeng, Shuai Kong, Shixiong Feng, and Nan Zhao. Shallow and deep feature fusion for digital audio tampering detection. *EURASIP Journal on Advances in Signal Processing*, 2022(1):69, 2022.

[39] S. Zamir, A. Arora, S. Khan, M. Hayat, F. Khan, and M. Yang. Restormer: Efficient transformer for high-resolution image restoration. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5718–5729, Los Alamitos, CA, USA, jun 2022. IEEE Computer Society.

[40] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.

[41] Kai Zhang, Wangmeng Zuo, and Lei Zhang. Ffdnet: Toward a fast and flexible solution for cnn-based image denoising. *IEEE Transactions on Image Processing*, 27(9):4608–4622, 2018.

[42] Ying Zhang, Jonathan Goh, Lei Lei Win, and Vrizlynn LL Thing. Image region forgery detection: A deep learning approach. *SG-CRC*, 2016:1–11, 2016.

[43] Peiyu Zhuang, Haodong Li, Shunquan Tan, Bin Li, and Jiwu Huang. Image tampering localization using a dense fully convolutional network. *IEEE Transactions on Information Forensics and Security*, 16:2986–2999, 2021.

Abstract in lingua italiana

Al giorno d'oggi, catturare, modificare e condividere immagini è diventata una pratica comune nelle nostre routine quotidiane. Con l'avvento di nuove e avanzate tecnologie, siamo ora in grado di manipolare le immagini semplicemente utilizzando i nostri smartphone. Queste tecnologie evolvono così rapidamente che i tradizionali metodi di rilevamento e localizzazione delle frodi sono diventati obsoleti. Molte volte, questi strumenti vengono utilizzati per scopi innocenti, ma sfortunatamente, vengono spesso impiegati anche per scopi maligni come manipolazioni politiche, false pubblicità, estorsioni o addirittura furto di identità, diventando un potenziale rischio non solo per gli individui, ma anche per la società. Per questo motivo, c'è un interesse costante nella comunità scientifica nel creare nuovi e potenti metodi di localizzazione delle frodi capaci di rilevare le manipolazioni effettuate sulle immagini. Questi metodi di solito si basano sull'esplorazione di specifiche caratteristiche delle fotocamere, come il CFA o il CRF. Una delle caratteristiche più sfruttate è probabilmente il PRNU, un residuo di rumore dipendente dal dispositivo. Alcuni dei metodi più efficaci per la localizzazione delle frodi si basano sull'idea di estrarre il PRNU dalle immagini e utilizzarlo a fini forensi. La stessa idea può essere estesa a livello di modello di fotocamera, affermando che ogni modello di fotocamera lascia la propria impronta sulle immagini catturate. Il nostro scopo è quello di costruire un metodo in grado di estrarre questa impronta dalle immagini per poi utilizzarla come strumento di localizzazione delle frodi. Per raggiungere questo obiettivo, ci affidiamo a due tipologie differenti di denoiser, uno basato sull'architettura CNN e uno basato sull'architettura Transformer. Successivamente, utilizziamo un insieme di mappe di attivazione estratte da questi reti per eseguire un compito di rilevamento delle anomalie, al fine di costruire una mappa di calore che mostri le potenziali tracce di manipolazione sull'immagine in ingresso. I risultati mostrano che il metodo da noi proposto supera le tecniche all'avanguardia nella maggior parte dei casi.

Parole chiave: Localizzazione per falsificazione di immagini, deep learning, rilevamento delle anomalie, forense multimediale.

Acknowledgements

First, I would like to thank my family, the most important thing in my life, my mother Celina, my aunt Fanny, and my uncle Lalo, you've guided and supported me during all my academic career, without you nothing would be possible.

I am also very grateful with the person who guided me through all this work, answering my questions, discussing every problem, and proposing solutions, thank you Sara. A special thank also to prof. Bestagini, you inspired me at the beginning of this journey with your lessons.

Thanks also to my most special friends, Juan Diego, and Catalina, you supported me from the distance and were always there when I needed you. Finally, I would like to thank my colleagues, who became my friends and an essential part of this journey, thanks for all the moments we shared together, María and Gabry.