https://doi.org/10.2352/ISSN.2470-1173.2017.7.MWSF-327

A Preliminary Study on Convolutional Neural Networks for Camera Model Identification

Luca Bondi; Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano; Milano, Italy
David Güera; School of Electrical and Computer Engineering, Purdue University; West Lafayette, IN, USA
Luca Baroffio; Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano; Milano, Italy
Paolo Bestagini; Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano; Milano, Italy
Edward J. Delp; School of Electrical and Computer Engineering, Purdue University; West Lafayette, IN, USA
Stefano Tubaro; Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano; Milano, Italy

Abstract

Camera model identification is paramount to verify image origin and authenticity in a blind fashion. State-of-the-art techniques leverage the analysis of features describing characteristic footprints left on images by different camera models from the image acquisition pipeline (e.g., traces left by proprietary demosaicing strategies, etc.). Motivated by the very accurate performance achieved by feature-based methods, as well as by the progress brought by deep architectures in machine learning, we explore in this paper the possibility of taking advantage of convolutional neural networks (CNNs) for camera model identification. More specifically, we investigate: (i) the capability of different network architectures to learn discriminant features directly from the observed images; (ii) the dependency between the amount of training data and the achieved accuracy; (iii) the importance of selecting a correct protocol for training, validation and testing. This study shows that promising results can be obtained on small image patches training a CNN with an affordable setup (i.e., a personal computer with one dedicated GPU) in a reasonable amount of time (i.e., approximately one hour), given that a sufficient amount of training images is available.

Introduction

The rapid proliferation of inexpensive image capturing devices has driven the widespread diffusion of digital pictures on the web. As sharing any type of images through websites and social media is everywhere. Verifying the veracity and authenticity of these widely distributed (and re-distributed) images is far from being an easy task [1, 2]. For this reason, the multimedia forensic community has investigated methodologies to assess the trustworthiness of digital images in the last few years [3, 4].

Among the many investigated forensic issues, great attention has been devoted to camera model identification [5, 6, 7]. Indeed, detecting the camera model used to take a picture can be crucial for criminal investigations and trials. This information can be exploited for solving copyright infringement cases, as well as indicating the authors of illicit material (e.g., child pornography). Even when deeper source identification granularity is needed (i.e., detecting the unique camera instance rather than just the make and model), camera model identification can be considered an important preliminary step to reduce the set of camera instances [7]. Moreover, being able to detect the camera model by analyzing small image patches is a possible way to expose splicing operations [8].

The rationale behind blind state-of-the-art camera model identification detectors is that each camera model performs peculiar operations on each image at acquisition time (e.g., different JPEG compression schemes, proprietary algorithms for CFA demosaicing, etc.). These operations leave on each picture characteristic "footprints" that can be exploited as an asset to reverse-engineer the camera model identity.

Following this idea, some methods focus on capturing characteristic footprints left during one specific step of the image acquisition pipeline. As an example, in [6] noise traces left by sensors on acquired images are exploited. Conversely, in [9] an algorithm tailored to detect lens distortion is proposed. In [10, 11, 12], traces relative to the used demosaicing strategy are investigated. Alternatively, the effect of dust traces on image sensors is studied in [13].

Given the difficulty of properly modeling complex chains of operations typical of the image acquisition pipeline, other camera model identification methods exploit features mainly capturing statistical image properties paired with machine learning classifiers, rather than focusing on specific processing operations. As an example, an technique based on local binary patterns is proposed in [14]. More recently, the authors of [15, 16, 17] exploit pixel co-occurrence statistics computed in different domains fed to a variety of supervised classification techniques. These methods guarantee very accurate results, especially on full resolution images that provide sufficient pixel statistics.

A common aspect of all the aforementioned algorithms is that they rely on manually defined procedures to expose traces characterizing different camera models. This means that they rely on some model assumptions a priori made. However, recent advancements established by deep learning techniques in computer vision [18] have shown that it is possible to improve the accuracy in detection and classification tasks by taking advantage of great amount of data in order to learn characteristic features directly from the data itself. These methods are known as data-driven, as they learn directly from data rather than following an analytic model.

Considering that the feature learning paradigm has recently proved fruitful for forensics applications [19, 20, 21, 22], in this paper we investigate the use of feature learning techniques in the camera model identification context, further investigating our first exploratory solutions [23, 24]. Our objective is to show that it is possible to use convolutional neural networks (CNNs) to learn discriminant features directly from the observed known images,

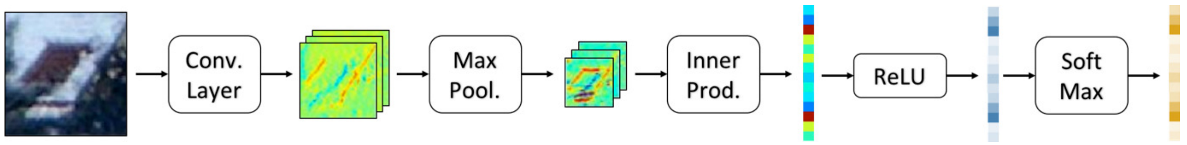

Figure 1. *Simple CNN architecture consisting of commonly used layers. A small color image patch is processed through convolutional, max pooling, inner product and ReLU layers. Finally, SoftMax is used to obtain a probability vector.*

rather than relying on hand-crafted descriptors. In principle, this enables to possibly capture also characteristic traces left by non-linear and difficult to model operations during the acquisition pipeline.

To conduct our study, we investigate the behavior of different CNN architectures to select a proper network for discriminant feature learning on $64 \times 64 \times 3$ (i.e., height $\times$ width $\times$ colors) image patches, while keeping computational complexity at bay. In particular, we compare a series of CNN architectures differing in the number of convolutional, pooling, inner product and rectified linear unit (ReLU) layers. For each type of architecture, several hyper-parameters choices (e.g., kernel size, stride, number of feature maps) are examined.

We focus on the importance of a proper training protocol, which is essential to make sure that the CNN learns important characteristics (e.g., properties discriminating camera models) rather than biased information (e.g., the semantic of the captured scenes). To this purpose, strongly inspired by [7], we consider different amounts of training images, depicting either the same or different scenes, and show how different training choices affect classification results.

The rest of the paper is structured as follows. We first report some background on CNNs. Then, we show the algorithmic pipeline devised to perform camera model identification using CNNs. Afterwards, we report all the details about the experimental setup, from the tested CNN architectures to the used dataset splits. Then, we report the numeric results achieved through our study in order to evaluate the different tested setups. Finally, we wrap up our final considerations and conclude the paper.

Background on CNNs

In this section, we provide a brief overview of convolutional neural networks (CNNs) sufficient to understand the rest of the paper. For a more in depth description, please refer to one of the many available tutorial in the literature [18, 25].

Deep learning and in particular CNNs have shown very good performance in several computer vision applications such as image classification, face recognition, pedestrian detection and handwriting recognition [25]. A CNN is a complex computational model partially inspired by the human neural system that consists of a very high number of interconnected nodes, or neurons. Connections between nodes have a numeric weight parameter that can be tuned based on experience, so that the model is able to learn complex functions. The nodes of the network are organized in multiple stacked layers, each performing a simple operation on the input. The set of operations in a CNN typically comprises convolution, intensity normalization, non-linear activation and thresholding and local pooling. By minimizing a cost function at the output of the last layer, the weights of the network (e.g., the values of the filters in the convolutional layers) are tuned so that they are able to capture patterns in the input data and automatically extract distinctive features.

This is different than traditional use of “handcrafted” features, in which the features used are driven by human intuition. In a CNN the features used are driven by data. Such complex models are trained using backpropagation coupled with an optimization method such as gradient descent and the use of large annotated training datasets. The first layers of the networks usually learn low-level visual concepts such as edges, simple shapes and color contrast, whereas deeper layers combine such simple information to identify complex visual patterns. Finally, the last layer consists of a set of data that are combined using a given cost function that needs to be minimized. For example, in the context of image classification, the last layer is composed of N nodes, where N is the number of classes, that define a probability distribution over the N visual category. That is, the value of a given node p_i, $i = 1, \dots, N$ belonging to the last layer represents the probability of the input image to belong to the visual class c_i. Depending on user choices, it is possible to select the class maximizing p_i as classification result, or to use all p_i values as feature vector to train an external classification tool (e.g., a support vector machine).

To train a CNN model for a specific image classification task we need:

1. to define the metaparameters of the CNN, i.e., the sequence of operations to be performed, the number of layers, the number and shape of the filters in convolutional layers, etc;
2. to define a proper cost function to be minimized during the training process;
3. to prepare a (possibly large) dataset of training and test images, annotated with labels according to the specific tasks (i.e., camera models in our work).

Figure 1 shows a minimalistic example of a small CNN architecture depicting some of the commonly used layers. To better understand the role of each layer, we describe the most common building block:

- *Convolution*: each convolution layer is a filterbank, whose filters impulse response h are learned through training. Given an input signal x, the output of each filter is $y = x * h$, i.e., the valid part of the linear convolution. Convolution is typically done on 3D representations consisting of the spatial coordinates (x, y) and the number of feature maps p (e.g., $p = 3$ for an RGB input).
- *Max pooling*: returns the maximum value of the input x evaluated across small windows (typically of 3x3 pixels).
- *ReLU*: Rectified Linear Unit (ReLU) uses the rectification function $y = \max(0, x)$ to the input x, thus clipping negative values to zero [26].
- *Inner Product*: indicates that the input of each neuron of the next layer is a linear combination of all the outputs of the

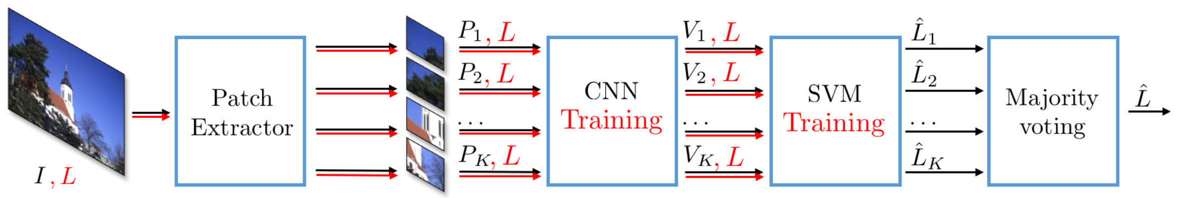

Figure 2. *Proposed pipeline for camera model identification with training steps highlighted in red. During training, patches are extracted from each training image I inheriting the same label L of the image. These are used for CNN and SVM training. During evaluation, for each patch P_i of the image I under analysis, a feature vector V_i is extracted through the CNN. Feature vectors are input into a set of trained linear SVM classifiers in order to associate a candidate label $\hat{L}_i$ to each vector. The predicted label $\hat{L}$ for image I is obtained by majority voting.*

previous layer. Combination weights are estimated during training. The dropout rate indicates the percentage of nodes that are randomly neglected during training in order to avoid data overfitting [27].

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

Camera Model Identification Using CNNs

In this paper, we consider camera model identification as a closed set classification problem. In other words, given an image I under analysis, the goal is to detect which camera model among a set of known N ones has been used to shoot the photograph.

In order to solve this problem, our proposed method follows the pipeline depicted in Figure 2: (i) images are split into patches; (ii) a CNN is first trained, than used to extract meaningful features from each patch; (iii) a set of support vector machines (SVMs) are trained and used to classify each patch; (iv) a final voting procedure is used to take decision at image level aggregating patches scores.

In the following we report details about each step, leaving to the next section the description of the tested CNN architectures, which is object of investigation in this paper.

Patch Selection

The first step of the proposed pipeline consists in splitting each image I into a set of K non-overlapping patches P_k, $k \in [1,K]$ of size $64 \times 64 \times 3$ (i.e., height $\times$ width $\times$ color). The rationale behind this choice is twofold: (i) splitting images into patches allows us to obtain a greater amount of data for CNN training; (ii) feeding the CNN with smaller data (i.e., a patches rather than full resolution images) enables working with smaller and lighter CNN architectures.

However, not all patches contain enough statistical information about the used camera model. For instance, it is clear that saturated patches should not be considered during either training or testing. Therefore, we devised a patch selection procedure. Specifically, for each patch P_k within an image, we compute a quality value defined as

$$Q(P_k) = \frac{1}{3} \sum_{c \in [R,G,B]} \left[\alpha \cdot \beta \left(\mu_c - \mu_c{}^2 \right) + (1-\alpha)\left(1 - e^{\gamma \sigma_c}\right) \right], \quad (1)$$

where α, β and γ are empirically set constants (set to 0.7, 4 and $\ln(0.01)$ in our experiments), whereas μ_c and σ_c, $c \in [R,G,B]$ are the average and standard deviation of red, green and blue components (in range $[0,1]$) of patch P_k, respectively. This quality measure tends to be lower for overly saturated or flat patches, whereas it is higher for textured patches showing some statistical variance (as shown in Figure 3). Therefore, we select for each image K patches with the highest Q values.

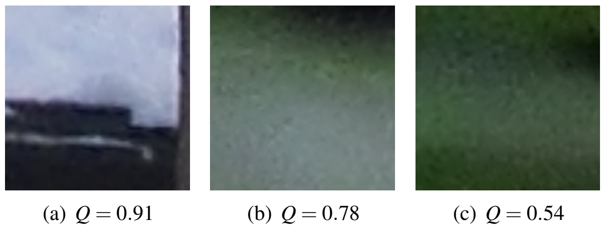

Figure 3. *Patch examples and relative quality measure.*

CNN Training

Given a set of training labeled images coming from N known camera models, we split them into patches and associate to each patch the same label L of the source image to which patches belong. Then, we input all available patch-label pairs for training into the CNN.

The choice of the CNN architecture is a delicate step. As an example, a too deep network may be unnecessary long to train and may contain too many parameters that need a huge training dataset to be properly tuned. However, smaller networks may not achieve accurate enough performance, loosing the ability of well discriminating the used camera models. Moreover, despite the number of used layers, also the choice of filters size and stride plays a crucial role, not to mention the use of inner product layers. For these reasons in this paper we tested different CNN architectures, the detailed description of which is left to the next section.

A common aspect among all tested architectures is that they accept as input patches of size $64 \times 64 \times 3$. The pixel-wise average over the training set is subtracted to each input patch. At the end of the training step, we obtain the CNN model $\mathcal{M}$.

SVM Training

Even though we could perform classification by simply picking the class corresponding to the maximum value from SoftMax layer, we decided to make use of an additional classification tool. Therefore, for each patch, the selected CNN model $\mathcal{M}$ is used to extract a feature vector, stopping the forward propagation at the last-minus-one layer (before the classification inner-product layer). Feature vectors associated to training patches are used to train a battery of $N \cdot (N-1)/2$ linear binary SVM classifiers $\mathcal{S}$ in a One-versus-One fashion.

Majority Voting

When a new image I is under analysis, the camera model used to acquire it is estimated as follows. A set of K patches is obtained from image I according to the quality measures defined in (1). Each patch P_k is processed by the trained CNN model $\mathcal{M}$ in order to extract a feature vector V_k. The set $\mathcal{S}$ of linear SVMs assigns a label $\hat{L}_k$ to each patch by attributing vectors V_k to one of the available N classes (i.e., camera models). The predicted model $\hat{L}$ for image I is obtained through majority voting on $\hat{L}_k, k \in [1,K]$. In case of par, random selection between equally likely models is operated.

Evaluation Setup

Considering the empirical nature of the proposed pipeline, it is essential to investigate through testing the impact of different CNN architectures and training strategies. In this section, we first report a detailed description of all tested CNNs. Then, we introduce the problem of biased training and describe the strategies that we tested to ensure a fair training process.

CNN Architectures

Designing a proper CNN architecture for our camera model identification pipeline is a critical step. The overall accuracy of the system is significantly determined by the extracted feature vectors from each image patch.

There are several key design choices that have to be considered as they determine the final structure of the CNN. The depth of the network, the use of pooling layers and the size of the kernels are examples and are referred to as hyper-parameters. Tuning hyper-parameters is approached in a trial-and-error fashion as there are no hard-quantitative rules that can be followed. This is due to the fact that we approach camera model identification as a data-driven problem and the final architecture of the network depends on the type of data under consideration. In our particular case, we explore networks that accept input patches of size $64 \times 64 \times 3$.

In [23], we presented a CNN used to extract features for camera model identification. A summary of the hyper-parameters used in that CNN is shown in Table 1. The encouraging results demonstrated with that particular architecture motivated us to improve its design in [24]. In particular, we used some of the CNN architecture design guidelines proposed in [28] and proposed a new CNN architecture, hereinafter denoted as $\mathcal{M}_{Conv4}$. In Table 2 we summarize its hyper-parameters. In this paper, we study the behavior of $\mathcal{M}_{Conv4}$ compared to other 3 new networks, which vary in depth as will be explained in this section. In particular, we consider $\mathcal{M}_{Conv4}$ as the base CNN on top of which we develop the 3 remaining deeper architectures.

The changes made in $\mathcal{M}_{Conv4}$ with respect to our first proposed CNN aim to reduce the computational complexity and improve the accuracy. As suggested in [28], in order to keep the computational complexity at bay we use more convolutional layers with smaller kernel sizes instead of using larger kernels and fewer convolutional layers (e.g. 2 stacks of 3×3 convolutional layers vs. a single 7×7 convolutional layer). A similar idea was also proposed by Chatfield et al. in [29]. This change coupled with the reduction in the overall number of convolutional filters aim to be parameter efficient. It also has the added benefit that our convolutional neural network is able to represent more complex functions as we add more layers.

We also changed the kernel size of the pooling layers to the most conventional value of 2×2 and changed the stride value to 2. We also added an additional pooling layer. The main idea behind having more pooling layers stems from the fact that the exact location of a feature in the original input patch (i.e. a high activation value occurs) is not as important as its relative location to other features. These layers drastically reduce the spatial dimension of the input volume they receive, which serves two purposes:

- the number of parameters is reduced by 75%, which translates into an efficient use of computer resources.
- controlling overfitting, which happens when a model is excessively fine-tuned to the training examples and it is not able to generalize well for the validation and evaluation sets (e.g. if the number of parameters of the network is high enough, the network could just memorize the training examples).

Finally, because we still want our CNN to be able to model non-linear functions we use a single ReLU layer towards the end of the network. This will make the CNN applicable to a wide range of camera models due to the fact that the non-linearity can be helpful to capture non-trivial classes.

It was shown by the Oxford VGG team in [30] and Szegedy et al. in [31], the representational capacity of a network is largely determined by its depth. This insight was also verified with the recent development of deep residual networks [32], which can have more than 150 layers and were used by the winners of the ILSVRC-2015 competition [33]. Keeping simplicity in mind, and following the strategy that Simonyan et al. devised in [30] to prepare their submission for the ILSVRC-2014, we explored a single family of networks of increasing depth. By doing so, we took advantage of the key design choices made in our base CNN model, $\mathcal{M}_{Conv4}$, such as the stacks of convolutional and pooling layers structure and the kernel sizes of the convolutional layers. A stride value of 1 was chosen for the convolutional layers. This results in no skipping (i.e. our filters are applied to all the values of the input volumes they receive).

The CNN architectures, evaluated in this work, are outlined in Table 3, one per column. $\mathcal{M}_{Conv4}$ was introduced earlier in [24] and we refer to the three remaining networks as $\mathcal{M}_{Conv6}$, $\mathcal{M}_{Conv8}$ and $\mathcal{M}_{Conv10}$. As mentioned earlier, the 4 networks vary only in the depth: from 6 weight layers in the network $\mathcal{M}_{Conv4}$ (4 convolutional and 2 Inner Product layers) to 12 weight layers in the network $\mathcal{M}_{Conv10}$ (10 convolutional and 2 Inner Product layers). The number of filters of each convolutional layer is rather small, starting from 32 in the first layer and then adding 16 more filters after each pooling layer, except for the last one, where we increase the number of filters by a factor of 2 reaching a total of 128 filters.

Training Strategies

As discussed in [7], the training procedure for machine learning-based camera model identification algorithms must be

Table 1: Summary of the hyper-parameters of the first CNN architecture that we presented as a feature extractor for camera model identification in [23]

Layer	Kernel size	Num. filters
Conv-1	7×7	128
ReLU-1	-	-
Pool-1	3×3	-
Conv-2	7×7	512
ReLU-2	-	-
Pool-2	3×3	-
Conv-3	6×6	2048
ReLU-3	-	-
InnerProduct-1	-	2048
ReLU-4	-	-
InnerProduct-2	-	2048
SoftMax	-	-

Table 2: Structure of the reference CNN architecture $\mathcal{M}_{\text{Conv4}}$. *N* is the number of training classes. Feature are extracted after the ReLU-1 layer

Layer	Input size	Kernel size	Stride	Num. filters	Output size
Conv-1	64×64×3	4×4	1	32	61×61×32
Pool-1	61×61×32	2×2	2	-	31×31×32
Conv-2	31×31×32	5×5	1	48	27×27×48
Pool-2	27×27×48	2×2	2	-	14×14×48
Conv-3	14×14×48	5×5	1	64	9×9×64
Pool-3	9×9×64	2×2	2	-	5×5×64
Conv-4	5×5×64	5×5	1	128	1×1×128
InnerProduct-1	1×1×128	-	-	128	128
ReLU-1	128	-	-	-	128
InnerProduct-2	128	-	-	*N*	*N*
SoftMax	*N*	-	-	-	*N*

devised with great attention. While it is important to ensure a sufficient amount of training data, training data cannot be randomly selected. Training data must be carefully chosen in order to avoid over fitting and ensure a wide variety of images covering different scenarios.

In order to further highlight the importance of the training strategy, let us consider the following example. Let us consider camera model identification problem using only two camera models whose labels are L_1 and L_2, respectively. If all images coming from camera L_1 are very dark, and all images from camera L_2 are very bright, the CNN might learn to discriminate luminance levels rather than camera models. It is clear that, in order to avoid such a biased training inevitably leading to incorrect results and conclusions, images from both cameras must depict both dark and bright scenes in this case. Despite the simplicity of this example, the situation becomes less trivial when many different camera models and images with different semantical contents are used.

In order to consider this important issue, in our study we consider the Dresden Image Dataset [34] as reference, as suggested in [7]. This dataset is composed by 73 camera devices from 25 camera models and 14 camera brands. For each device a variable number of shots has been taken in several geographical positions. For each position a set of different motives is shot. Details about the acquisition process are available at [34]. In the following we will refer to *scene* when considering the combination of a geographical position with a specific motive. This results in a total amount of 83 available scenes. We only consider camera models represented by more than one device, in order to ensure that the CNN learns model specific artifacts rather than instance specific ones. This leads to a dataset composed of 18 camera models (as Nikon D70 and D70s basically differ only in their on-device screen), for nearly $15,000$ shots.

We need to split images into three different datasets: (i) a training set $\mathbb{D}_T$ used for updating CNNs and SVMs parameters; (ii) a validation set $\mathbb{D}_V$ used to decide the stopping point of the training step and avoid over-fitting (i.e., typically the training process is stopped when validation loss, given by SoftMax layer, reaches its minimum); (iii) an evaluation set $\mathbb{D}_E$ used to test the trained architectures. Following the ideas presented by Kirchner et al. [7]

- we selected shots belonging to the evaluation set ($\mathbb{D}_E$) from N_E scenes and a single instance per camera model. The selected images are never used in training or validation.
- we selected shots for training set ($\mathbb{D}_T$) and validation set ($\mathbb{D}_V$) among images from remaining scenes and instances.

Specifically, we define three splitting policies for $\mathbb{D}_T$ and $\mathbb{D}_V$ so

Table 3: The 4 proposed CNN architectures (shown in columns). Added layers are shown in bold and the number of filters for each convolutional layer is shown in parenthesis

CNN Architecture			
$\mathcal{M}_{Conv4}$	$\mathcal{M}_{Conv6}$	$\mathcal{M}_{Conv8}$	$\mathcal{M}_{Conv10}$
6 weight layers	8 weight layers	10 weight layers	12 weight layers
Input (64×64×3 image patch)			
Conv-1 (32)	Conv-1 (32) **Conv-1 (32)**	Conv-1 (32) Conv-1 (32)	Conv-1 (32) Conv-1 (32)
Pool-1			
Conv-2 (48)	Conv-2 (48) **Conv-2 (48)**	Conv-2 (48) Conv-2 (48)	Conv-2 (48) Conv-2 (48)
Pool-2			
Conv-3 (64)	Conv-3 (64)	Conv-3 (64) **Conv-3 (64)**	Conv-3 (64) Conv-3 (64) **Conv-3 (64)**
Pool-3			
Conv-4 (128)	Conv-4 (128)	Conv-4 (128) **Conv-4 (128)**	Conv-4 (128) Conv-4 (128) **Conv-4 (128)**
InnerProduct-1			
ReLU-1			
InnerProduct-2			
SoftMax			

to test for possible over-fitting on scenes content rather than on camera model identification during CNN training. Since the validation set is used to decide when to stop the training process, if its content is too similar to the training set we could easily over-fit. Conversely, if validation set is sufficiently different from training one, we should be able to obtain more generalizable results on the evaluation set. Splitting policies are detailed below:

1. *Fair-N_T*: training and validation shots are split according to the depicted scene. The number of training scenes is set to N_T and shots coming from a specific scene are included only in $\mathbb{D}_T$ or in $\mathbb{D}_V$. In this way, $\mathbb{D}_T$ and $\mathbb{D}_V$ are completely disjoint sets (in terms of scenes), thus should lead to robust training.
2. *Fair-balanced-N_T*: training and validation shots are split according to scenes as for *Fair-N_T*. The number of shots for each device model is the same, leading to a model-balanced training dataset.
3. *Unfair-P_T*: training and validation shots are split regardless of the scene they belong to, fixing the percentage of training shots to P_T. In doing so, the same scene can appear in both training and validation sets, thus possibly leading to over-fitting and less accurate evaluation results.

A small case example for the three splitting strategies is available at Table 4.

Experimental Results

In this section we report the performed tests using different CNNs and training strategies to validate the proposed pipeline in a fair way.

Impact of the CNN Architecture

To evaluate the proposed CNN architectures, we selected a reference splitting policy showing good performance in our initial analysis. We used the *Fair-60* splitting policy and worked with a 10-fold cross validation framework (i.e., the selected splitting policy is tested 10 times on different realizations of scenes). This results in a total number of 40 trained CNN models. For evaluation, we do not to train additional SVMs, but use the CNN output as class prediction. This allows us to study the effect of the different CNN architectures on the accuracy results for a fixed splitting policy.

For each shot in the training, validation and evaluation sets in *Fair-60*, $K = 32$ patches were extracted as described above. Training and validation patches were used to train the proposed CNNs. Specifically, the CNN architectures were trained on $\mathbb{D}_T$ patches until classification loss on $\mathbb{D}_V$ patches was minimized. Once the CNNs were trained, they were used to extract an 18 elements vector V_k for each patch P_k at the end of InnerProduct-2 layer of the CNN. Results aggregation at shot level was performed averaging element by element feature vectors V_k associated to patches P_k belonging to the same shot P, so to obtain an 18 elements score vector V for the shot. Camera model associated to the maximum score was used to predict the shot's class. Shots classification accuracy was computed on $\mathbb{D}_T$, $\mathbb{D}_V$ and $\mathbb{D}_E$ as average over the 10 data realizations.

The results, presented in Table 5, indicate that the classification accuracy increases as we increase the CNN architecture depth: from 6 weight layers in the network $\mathcal{M}_{Conv4}$ to 12 weight layers in the network $\mathcal{M}_{Conv10}$. The camera model classification accuracy of our architecture saturates when the depth reaches 12 layers, but even deeper CNNs might be beneficial for larger datasets with a higher number of classes and training images.

Table 4: A small scale example for three different splitting strategies. Row colors correspond to scenes. First, an instance id and a set of scenes are selected for the evaluation set $\mathbb{D}_E$. Considering the remaining instances and scenes, $\mathbb{D}_T$ and $\mathbb{D}_V$ are built according to what specified in the text. Labels E, V and T denote images associated to $\mathbb{D}_E$, $\mathbb{D}_V$ and $\mathbb{D}_T$ according to each policy.

Brand	Model	Instance	Scene	*Fair*	*Fair-balanced*	*Unfair*
Canon	Ixus 70	0	Kohlenstrasse Back view ZINT	E	E	E
Canon	Ixus 70	0	Home III Trees in a garden II			
Canon	Ixus 70	0	Kaethe-Kollwitz-Ufer Blue Wonder			
Canon	Ixus 70	1	Kohlenstrasse Back view ZINT			
Canon	Ixus 70	1	Home III Trees in a garden II	V	V	T
Canon	Ixus 70	1	Kaethe-Kollwitz-Ufer Blue Wonder	T	T	T
Canon	Ixus 70	1	Kaethe-Kollwitz-Ufer Blue Wonder	T		V
Kodak	M1063	0	Kohlenstrasse Back view ZINT	E	E	E
Kodak	M1063	0	Home III Trees in a garden II			
Kodak	M1063	0	Kaethe-Kollwitz-Ufer Blue Wonder			
Kodak	M1063	1	Kohlenstrasse Back view ZINT			
Kodak	M1063	1	Home III Trees in a garden II	V	V	V
Kodak	M1063	1	Kaethe-Kollwitz-Ufer Blue Wonder	T	T	T

Table 5: CNN classification accuracy for the different sets of *Fair-60*

CNN Architectures	$\mathbb{D}_T$ accuracy (%)	$\mathbb{D}_V$ accuracy (%)	$\mathbb{D}_E$ accuracy (%)
$\mathcal{M}_{Conv4}$	97.69	97.74	94.51
$\mathcal{M}_{Conv6}$	97.82	97.53	94.67
$\mathcal{M}_{Conv8}$	97.95	97.62	94.79
$\mathcal{M}_{Conv10}$	98.01	97.81	94.93

Impact of Training Strategy

After testing different architectures, we focused on a reference CNN showing good performances and performed an extensive set of experiments over the three splitting policies described in the previous section. In particular, we selected the $\mathcal{M}_{Conv4}$ CNN detailed in Table 2. For evaluation, we decided not to train additional SVMs, but to use the CNN output as class prediction. This allows us to study the effect of different training and validation split on the CNN only.

For this analysis, we fixed the number of evaluation scenes N_E to 10. For splitting policies *Fair* and *Fair-balanced* the number of training scenes N_T was varied in $\{10,15,20,30,40,50,60\}$ over the 73 available scenes. The remaining $73-N_T$ scenes were used for validation. For splitting policy *Unfair* the percentage of training shots P_T was varied in $\{10,20,30,40,50,60,70,80,90\}$. The remaining shots were assigned to the validation set. This resulted in 23 splitting policies.

Figure 4 shows results using the *Fair* splitting policy to select train and validation datasets. Shots classification accuracy is reported as function of number of training scenes. Train and validation curves, in blue and green respectively, are almost always aligned. The red curve refers to the performance on the evaluation set. When using a small number of scenes for training (i.e., $N_t = 10$) , the small amount of data limits the CNN capabilities of learning from data. Once the number of training scenes is sufficiently large (i.e., $N_t > 15$) the results increase reaching an evaluation accuracy up to 94.5%.

Figure 5 shows results using the *Fair-balanced* splitting policy to select train and validation datasets. In this case the small amount of data severely limits the CNN capabilities of learning from data. In fact, in the Dresden Dataset, some camera models are represented by only a few number of shots. In the best situation (*Fair-balanced-60*), the evaluation accuracy reaches 92.6%.

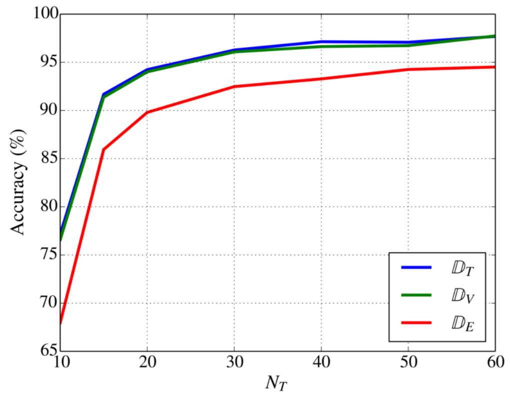

***Figure 4.** Fair splitting policy results. Training (blue), validation (green) and evaluation (red) set.*

Figure 6 shows results using the *Unfair* splitting policy to select train and validation datasets. Also in this case the small amount of training data impairs CNN learning capabilities when $P_T = 0.1$. However, as soon as the percentage of training data is increased, the evaluation accuracy reaches 94.4%.

Both the *Fair* and the *Unfair* splitting policies show a gap around 3.3% between validation and evaluation accuracies. This kind of behavior might be an indicator of some instance specific features learned during the training process.

A comparison between the *Fair* (Figure 4) and the *Unfair* (Figure 6) shows that there is not much gain in carefully splitting

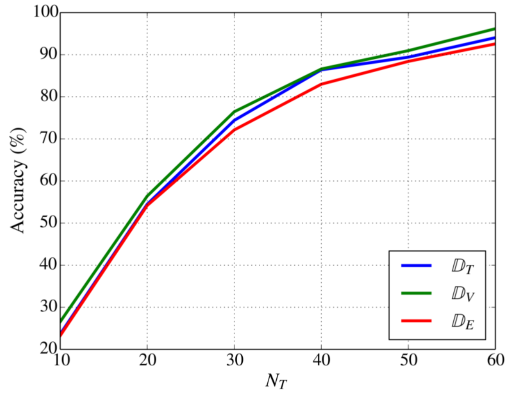

Figure 5. *Fair balanced splitting policy results. Training (blue), validation (green) and evaluation (red) set.*

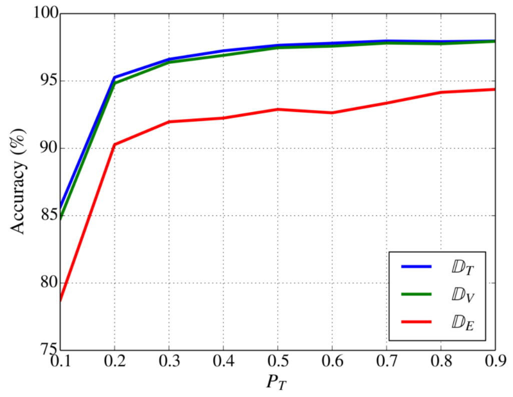

Figure 6. *Unfair splitting policy results. Training (blue), validation (green) and evaluation (red) set.*

training and validation scenes. A possible motivation for this results stands in the small size of the patches used in this context. In fact, a $64 \times 64 \times 3$ patch extracted from a full resolution picture (as the ones in the Dresden Image Dataset) contains only a few details from the image, and rarely some scene specific content that might be found only in larger patches. This motivates even further the use of small patches for this learning task.

Comparison with the State-of-the-Art

After validating the performance of CNNs standalone (i.e., using the InnerProduct-2 output as score for each class), we focused on the evaluation of the entire pipeline (i.e., with SVMs and majority voting) in comparison with the recently proposed state-of-the-art method by Chen et al. [15]. In particular we stopped the forward step of $\mathcal{M}_{Conv4}$ at the end of the ReLU-1 layer in order to extract from each patch P_k a feature vector V_k. As dataset, we considered the *Fair-balanced-60* splitting policy.

Figure 7 shows how the average classification accuracy on shots from $\mathbb{D}_E$ varies while increasing the number of voting patches for each image. The proposed CNN-based approach is depicted by the green line. Benchmark result using the approach proposed by Chen et al. [15] on 64×64 color patches followed by majority voting is shown with the red line. As the method proposed by Chen et al. is not specifically tailored to small patches, we also tested it on full resolution images without voting procedures (i.e., blue line of Figure 7). It is worth noting that, despite the high accuracy obtained by Chen et al., our method approaches within 1% their result by using considerably less input data (i.e., just a few patches and not the full image).

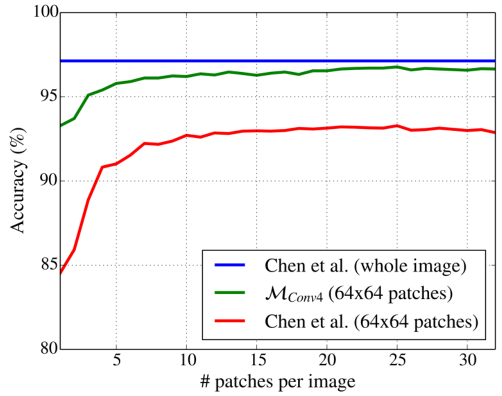

Figure 7. *Comparison between the overall pipeline considering the CNN $\mathcal{M}_{Conv4}$ trained with Fair-balanced-60 splitting policy and the state-of-the-art algorithm by Chen et al. [15].*

As a final remark, notice that the number of features generated at the output by the CNN for each patch is only 128, less than one tenth with respect to the $1,372$ generated by Chen et al. This confirms that we are able to characterize camera models in a space with reduced dimensionality. In principle, this enables the use of simple classifiers, which can be trained more efficiently.

Conclusions

In this paper we showed the possibility of using CNNs for camera model identification. Specifically, we focused on a processing pipeline making use of a CNN for feature extraction and a set of SVMs for classification. We first tested different CNN architectures, in order to select a valid structure candidate balancing accuracy and computational complexity. We then investigated the effect of training a CNN on different data splits in order to highlight the dependency between accuracy, training set size, and training-testing splitting policy.

This study shows that it is possible to achieve high camera model attribution accuracy (i.e., around 96%) even with fairly small network architectures (i.e., four convolutional layers), provided that a minimum amount of training images are available. Indeed, the use of larger configurations determines a negligible accuracy increment, at least on the selected dataset of 18 camera models.

Future work will be devoted to studying the impact of large networks on bigger datasets, in which the increased number of layers may be crucial. We will also explore the possibility of devising ad-hoc data augmentation routines to enlarge the used training set.

Acknowledgments

This material is based on research sponsored by DARPA and Air Force Research Laboratory (AFRL) under agreement number

FA8750-16-2-0173. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA and Air Force Research Laboratory (AFRL) or the U.S. Government.

References

[1] H. Farid, "Seeing is not believing," *IEEE Spectrum*, vol. 46, no. 8, pp. 44–51, August 2009.

[2] A. Rocha, W. Scheirer, T. Boult, and S. Goldenstein, "Vision of the unseen: Current trends and challenges in digital image and video forensics," *ACM Computing Surveys*, vol. 43, no. 4, pp. 26:1–26:42, October 2011.

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

[4] M. C. Stamm, Min Wu, and K. J. R. Liu, "Information Forensics: An Overview of the First Decade," *IEEE Access*, vol. 1, pp. 167–200, 2013.

[5] M. Kharrazi, H. T. Sencar, and N. Memon, "Blind source camera identification," *Proceedings of the IEEE International Conference on Image Processing*, pp. 709–712, October 2004, Singapore.

[6] T. Filler, J. Fridrich, and M. Goljan, "Using sensor pattern noise for camera model identification," *Proccedings of the IEEE International Conference on Image Processing*, pp. 1296–1299, October 2008, San Diego, CA.

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

[8] A. Swaminathan, M. Wu, and K. J. R. Liu, "Digital image forensics via intrinsic fingerprints," *IEEE Transactions on Information Forensics and Security*, vol. 3, no. 1, pp. 101–117, March 2008.

[9] K.S. Choi, E.Y. Lam, and K.K.Y. Wong, "Automatic source camera identification using the intrinsic lens radial distortion," *Optics Express*, vol. 14, no. 24, pp. 11551–11565, November 2006.

[10] S. Bayram, H. Sencar, N. Memon, and I. Avcibas, "Source camera identification based on CFA interpolation," *Proceedings of the IEEE International Conference on Image Processing*, pp. III–69–72, September 2005, Genova, Italy.

[11] H. Cao and A. C. Kot, "Accurate detection of demosaicing regularity for digital image forensics," *IEEE Transactions on Information Forensics and Security*, vol. 4, no. 4, pp. 899–910, December 2009.

[12] S. Milani, P. Bestagini, M. Tagliasacchi, and S. Tubaro, "Demosaicing strategy identification via eigenalgorithms," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2659–2663, May 2014.

[13] A.E. Dirik, H.T. Sencar, and N. Memon, "Digital single lens reflex camera identification from traces of sensor dust," *IEEE Transactions on Information Forensics and Security*, vol. 3, no. 3, pp. 539–552, September 2008.

[14] G. Xu and Y. Q. Shi, "Camera model identification using local binary patterns," *Proceedings of the IEEE International Conference on Multimedia and Expo*, pp. 392–397, July 2012, Melbourne, Australia.

[15] C. Chen and M. C. Stamm, "Camera model identification framework using an ensemble of demosaicing features," *Proceedings of the IEEE International Workshop on Information Forensics and Security*, pp. 1–6, November 2015, Rome, Italy.

[16] F. Marra, G. Poggi, C. Sansone, and L. Verdoliva, "Evaluation of residual-based local features for camera model identification," in *New Trends in Image Analysis and Processing – ICIAP 2015 Workshops*, Vittorio Murino, Enrico Puppo, Diego Sona, Marco Cristani, and Carlo Sansone, Eds., pp. 11–18. Springer International Publishing, Cham, Switzerland, 2015.

[17] A. Tuama, F. Comby, and M. Chaumont, "Camera model identification based on machine learning approach with high order statistics features," *Proceedings of the IEEE European Signal Processing Conference*, pp. 1183–1187, August 2016, Budapest, Hungary.

[18] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, January 2009.

[19] M. Buccoli, P. Bestagini, M. Zanoni, A. Sarti, and S. Tubaro, "Unsupervised feature learning for bootleg detection using deep learning architectures," *Proceedings of the IEEE International Workshop on Information Forensics and Security*, pp. 131–136, December 2014, Atlanta, GA.

[20] C. Jiansheng, K. Xiangui, L. Ye, and Z. J. Wang, "Median filtering forensics based on convolutional neural networks," *IEEE Signal Processing Letters*, vol. 22, no. 11, pp. 1849–1853, November 2015.

[21] B. Bayar and M. C. Stamm, "A deep learning approach to universal image manipulation detection using a new convolutional layer," *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*, pp. 5–10, June 2016, Vigo, Spain.

[22] G. Xu, H. Z. Wu, and Y. Q. Shi, "Structural design of convolutional neural networks for steganalysis," *IEEE Signal Processing Letters*, vol. 23, no. 5, pp. 708–712, May 2016.

[23] L. Baroffio, L. Bondi, P. Bestagini, and S. Tubaro, "Camera identification with deep convolutional networks," *arXiv:1603.01068*, March 2016.

[24] L. Bondi, L. Baroffio, D. Guera, P. Bestagini, E. J. Delp, and S. Tubaro, "First steps towards camera model identification with convolutional neural networks," *IEEE Signal Processing Letters (SPL)*, vol. PP, no. 99, pp. 1–1, 2016.

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

[26] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines," *Proceedings of the International Conference on Machine Learning*, pp. 807–814, June 2010, Haifa, Israel.

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, January 2014.

[28] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *arXiv:1512.00567v3*, December 2015.

[29] K. Chatfield, K. Simonyan, Andrea Vedaldi, and Andrew Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," *Proceedings of the British Machine Vision Conference*, September 2014, Nottingham, UK.

[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Proceedings of the International Conference on Learning Representations*, May 2015, San Diego, CA.

[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *Proceedings of the IEEE Conference on Computer*

Vision and Pattern Recognition, pp. 1–9, June 2015, Boston, MA.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv:1512.03385*, December 2015.

[33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, December 2015.

[34] T. Gloe and R. Böhme, "The Dresden image database for benchmarking digital image forensics," *Journal of Digital Forensic Practice*, vol. 3, no. 2-4, pp. 150–159, December 2010.

Author Biography

Luca Bondi received the M.Sc. degree in Computer Engineering in December 2014 from Politecnico di Milano. He is currently pursuing a Ph.D. degree at the Department of Electronics, Informatics and Bioengineering, Politecnico di Milano. His research activities are focused on deep neural networks for efficient visual features extraction and compression.

David Güera received his B.E. (Hons.) in Telecommunications Systems Engineering from Polytechnic University of Catalonia - BarcelonaTech, Spain in July 2016. He is currently pursuing a Ph.D. at the Video and Image Processing Laboratory (VIPER), School of Electrical and Computer Engineering, Purdue University, USA. His research interests are video/image processing and deep learning.

Luca Baroffio received the M.Sc. degree (2012, cum laude) in Computer Engineering and the Ph.D. (2016, cum laude) in Information Technology from Politecnico di Milano, Milan, Italy. In 2013 he was visiting scholar at "Instituto de Telecomunicações", Lisbon, Portugal. His research interests are in the areas of multimedia signal processing and visual sensor networks.

Paolo Bestagini received the M.Sc. in Telecommunications Engineering and the Ph.D. in Information Technology at the Politecnico di Milano, Italy, in 2010 and 2014, respectively. Since 2016 he is Assistant Professor at the Department of Electronics, Informatics and Bioengineering, Politecnico di Milano. His research activity is focused on multimedia forensics and acoustic signal processing for microphone arrays.

Edward J. Delp was born in Cincinnati, Ohio. He is currently The Charles William Harrison Distinguished Professor of Electrical and Computer Engineering and Professor of Biomedical Engineering at Purdue University. His research interests include image and video processing, image analysis, computer vision, image and video compression, multimedia security, medical imaging, multimedia systems, communication and information theory. Dr. Delp is a Life Fellow of the IEEE, a Fellow of the SPIE, and a Fellow of IS&T.

Stefano Tubaro completed his studies in Electronic Engineering at the Politecnico di Milano, Italy, in 1982. He joined the Politecnico di Milano, first as a researcher and in 1991 as an Associate Professor. Since 2004 he has been appointed as Full Professor of Telecommunications. His current research interests are on advanced algorithms for video and sound processing and for image and video tampering detection. He authored over 180 scientific publications and more than 15 patents.