Constrained Convolutional Neural Networks: A New Approach Towards General Purpose Image Manipulation Detection

Belhassen Bayar, *Student Member, IEEE,* and Matthew C. Stamm, *Member, IEEE*

***Abstract*—Identifying the authenticity and processing history of an image is an important task in multimedia forensics. By analyzing traces left by different image manipulations, researchers have been able to develop several algorithms capable of detecting targeted editing operations. While this approach has led to the development of several successful forensic algorithms, an important problem remains: creating forensic detectors for different image manipulations is a difficult and time consuming process. Furthermore, forensic analysts need 'general purpose' forensic algorithms capable of detecting multiple different image manipulations. In this paper, we address both of these problems by proposing a new general purpose forensic approach using convolutional neural networks (CNNs). While CNNs are capable of learning classification features directly from data, in their existing form they tend to learn features representative of an image's content. To overcome this issue, we have developed a new type of CNN layer, called a constrained convolutional layer, that is able to jointly suppress an image's content and adaptively learn manipulation detection features. Through a series of experiments, we show that our proposed constrained CNN is able to learn manipulation detection features directly from data. Our experimental results demonstrate that our CNN can detect multiple different editing operations with up to 99.97% accuracy and outperform the existing state-of-the-art general purpose manipulation detector. Furthermore, our constrained CNN can still accurately detect image manipulations in realistic scenarios where there is a source camera model mismatch between the training and testing data.**

***Index Terms*—Image forensics, deep learning, convolutional neural networks, deep convolutional features.**

I. Introduction

INFORMATION about image authenticity can be used in important settings, such as evidence in legal proceedings and criminal investigations. However, many commonly available tools can allow a user to make visually realistic image forgeries. As a result, image manipulation detection has become a very important task in multimedia forensics. To determine the authenticity and processing history of digital images, researchers have developed numerous forensic approaches during the last decade [1]. Specifically, it has been observed that image manipulations typically leave behind traces unique to the type of editing an image has undergone. Researchers design forensic algorithms that extract features related to these traces and use them to detect targeted image manipulations. This approach has been successful in detecting many types of image tampering such as resizing and resampling [2], [3], [4], [5], [6], median filtering [7], [8], [9], [10], contrast enhancement [11], [12], [13], [14], multiple JPEG compression [15], [16], [17], [18], etc.

Although, research in image forensics has dramatically advanced, these approaches still suffer from important drawbacks. New editing operations are frequently developed and incorporated into editing software such as Adobe Photoshop. As a result, researchers must identify traces left by these new operations and design associated detection algorithms. This is difficult and time consuming since these algorithms are often designed from estimation and detection theory. Furthermore, the forensic algorithms described above are designed to detect a single targeted manipulation. As a result, multiple forensic tests must be run to authenticate an image. This results in several challenges such as fusing the results of multiple forensic tests and controlling the overall false alarm rate among several forensic detectors.

To address these issues, researchers have recently focused on developing general-purpose image forensic techniques to determine if and how an image has undergone processing. Tools from steganalysis have been adapted to perform general-purpose image forensics [19], [20]. Specifically, powerful steganalytic features called the spatial-domain rich model (SRM) [19] have been successfully used to perform universal image manipulation detection [20]. Kirchner *et al.* [7] showed the effectiveness of subtractive pixel adjacent matrix (SPAM) [21] features when performing median filtering detection. Furthermore, Fan *et al.* developed a general-purpose manipulation detector where image manipulation traces are learned from Gaussian mixture model (GMM) parameters of small image patches [22].

While these recent approaches have resulted in noticeable gains in detection accuracy, several important questions remain, including: How should low-level forensic feature extractors be designed? Do they have a common form? Can image tampering traces be learned directly from data? Are there alternative ways of extracting higher-level features for tampering detection from low-level forensic traces?

Deep learning approaches, particularly convolutional neural networks (CNNs), provide a potential answer to these questions. CNNs have attracted a significant amount of attention given their ability to automatically learn classification features

This material is based upon work supported by the National Science Foundation under Grant No. 1553610. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

The authors are with the Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA, 19104 USA (e-mail: bb632@drexel.edu, mstamm@drexel.edu)

directly from data. They have been successfully used on a variety of different types of signals such as images [23], speech [24] and text data [25]. While CNNs provide a promising approach towards automatically learning image manipulation traces, in their existing form they are not well-suited for forensic purposes. This is because existing CNNs tend to learn features representative of an image's content as opposed to manipulation traces, which are content-independent. As a result, forensic researchers may ask: Is it possible to force a CNN to learn manipulation detection features instead of features that represent image's content?

In this paper, we propose a new type of CNN architecture, called a constrained CNN, designed to adaptively learn image manipulation features and accurately identify the type of editing that an image has undergone. We use our constrained CNN to construct a powerful general-purpose manipulation detector called "MISLnet", named after our lab, the Multimedia and Information Security Lab (MISL). To accomplish this, we propose a new type of convolutional layer called a constrained convolutional layer that forces a CNN to learn low-level manipulation features. Many forensic algorithms, such as resampling detectors [2], [3], median filtering detectors [7] and other forensic detectors based on steganalytic features like the SRM [20], operate by first extracting prediction residual features, then by forming higher-level features from these residuals. Inspired by this, our constrained convolutional layer is designed to only learn prediction error filters. This jointly suppresses the image's content and adaptively learns low-level prediction residual features that are optimal for detecting forensic traces. Higher-level forensic features are learned from these residuals by deeper layers of our CNN.

Through a series of experiments, we show that our MISLnet architecture can automatically learn to detect multiple types of image editing directly from data. This removes the need for difficult and time consuming human analysis to design forensic detection features. Our results show that when given comparable amount of training data, our constrained CNN can perform as good or better than state-of-the-art general-purpose detector based on steganalytic features [20]. Furthermore, we show that we can significantly improve the performance of our CNN-based detector by using very large amounts of training data that are computationally prohibitive for forensic detectors based on the SRM and its associated ensemble classifier. These results show that our proposed method can achieve 99.97% accuracy with five different tampering operations using a large scale data.

The major contributions of this paper are as follows: (1) We propose a CNN architecture that is capable of detecting image editing and manipulations. This CNNs architecture is deeper and more sophisticated than the one we initially proposed in [26], and its design choices are systematically investigated through a series of experiments. (2) We introduce our proposed *constrained convolutional layer*, provide a detailed discussion of how it is constructed and trained, as well as provide intuition into why it works. (3) We conduct a large scale experimental evaluation of our MISLnet CNN architecture and show that it can outperform existing image manipulation detection techniques, can differentiate between multiple editing operations even when their parameters vary, can detect sequences of operations, can provide localized manipulation detection results, and can provide extremely accurate manipulation detection results when trained using a large scale training dataset.

The remainder of this paper is organized as follows: In Section II, we present an overview about CNNs in literature. Then, in Section III we describe how CNNs are used in multimedia forensics task using the constrained convolutional layer. Section IV provides details about our proposed CNN architecture. Finally, in Section V we assess our proposed deep learning approach in adaptively extracting image manipulation features through a set of experiments. Lastly, Section VI concludes our work.

II. Convolutional Neural Networks

Deep learning approaches, such as convolutional neural networks [27], are an extended version of neural networks. Their architecture, which is the set of parameters and components that we need to design a network, is based on stacking many hidden layers on top of one another. This has proven to be very effective in extracting hierarchical features. That is, they are capable of learning features from a set of previously learned features. In a CNN architecture, the first layer is a set of convolutional feature extractors applied in parallel to the image using a set of several learnable filters. These filters work as a sliding window that convolves with all regions of the input image with an overlapping distance called the stride and produce outputs known as feature maps. Similarly, the hidden convolutional layers extract features from each lower-level feature maps. Finally, the output of these hierarchical feature extractors is stacked to a fully-connected neural network that performs classification.

The convolutional operation between the input feature maps and a convolutional layer within the CNN architecture is given in Eq. (1):

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

where $*$ denotes a $2d$ convolution, $\boldsymbol{h}_j^{(n)}$ is the j^{th} feature map output in the n^{th} hidden layer, $\boldsymbol{h}_k^{(n-1)}$ is the k^{th} channel in the $(n-1)^{th}$ hidden layer, $\boldsymbol{w}_{kj}^{(n)}$ is the k^{th} channel in the j^{th} filter in the n^{th} layer and $b_j^{(n)}$ is its corresponding bias term.

The filter coefficients in each layer are initially seeded with random values, then learned using the back-propagation algorithm [27], [28]. The convolutional layers are also followed by an activation function to introduce nonlineraity. The set of convolutional layers yields a large volume of feature maps. To reduce the dimensionality of these features, the convolutional layers are followed by another type of layer called pooling. This reduces the training computational cost of the network and decreases the chances of over-fitting. There exist many types of pooling operations such as max, average, and stochastic pooling. In particular, the max-pooling layer works also as a sliding window with a stride distance which retains the maximum value within the dimension of a sliding window.

The training process of a CNN is done through an iterative algorithm that alternates between feedforward and backpropagation passes of the data. The weights of the convolutional filters and fully-connected layers are updated at each iteration of the backpropagation passes. Ultimately, we would like to minimize the average loss E between the true class labels (i.e., unaltered, manipulated, etc.) and the network outputs, i.e.,

$$E = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{c} y_i^{*(k)} \log\left(y_i^{(k)}\right), \tag{2}$$

where $y_i^{*(k)}$ and $y_i^{(k)}$ are respectively the true label and the network output of the i^{th} image at the k^{th} class with m training images and c neurons in the output layer. There have been proposed a variety of solvers [29], [30], [31] to minimize the average loss.

In this paper, we consider the stochastic gradient descent (SGD) to train our model [29]. The iterative update rule for the kernel coefficients $\boldsymbol{w}_{ij}^{(n)}$ in CNN during the backpropagation pass is given below:

$$\begin{aligned} \nabla\boldsymbol{w}_{ij}^{(n+1)} &= \epsilon\cdot\frac{\partial E}{\partial \boldsymbol{w}_{ij}^{(n)}} - m\cdot\nabla\boldsymbol{w}_{ij}^{(n)} + d\cdot\epsilon\cdot\boldsymbol{w}_{ij}^{(n)} \\ \boldsymbol{w}_{ij}^{(n+1)} &= \boldsymbol{w}_{ij}^{(n)} - \nabla\boldsymbol{w}_{ij}^{(n+1)}, \end{aligned} \tag{3}$$

where $\boldsymbol{w}_{ij}^{(n)}$ represents the i^{th} channel from the j^{th} kernel matrix in the n^{th} hidden layer that convolves with the i^{th} channel in the previous feature maps of the $(n-1)^{th}$ layer, $\nabla\boldsymbol{w}_{ij}^{(n)}$ denotes the gradient of $\boldsymbol{w}_{ij}^{(n)}$ and ϵ is the learning rate. The bias term $b_j^{(n)}$ in (1) is updated using the same equations presented in (3). For fast convergence as explained by LeCun *et al.* in [28], we use the decay and momentum strategy which are respectively denoted by d and m in (3).

III. Constrained Convolutional Neural Network

A. Constrained convolutional layer

Instead of relying on hand-designed or predetermined features, we propose a CNN-based approach to image manipulation detection. Our approach is able to use data to directly learn the changes introduced by image tampering operations into local pixel relationships. We note that if CNNs in their standard form (such as AlexNet [23]) are used to perform image manipulation detection, they will learn features that represent an image's content. This will lead to a classifier that identifies scene content associated with the training data as opposed to learning image manipulation fingerprints.

By contrast, our approach is designed to suppress an image's content and adaptively learn image manipulation traces. To accomplish this, we propose a new type of convolutional layer, called a *constrained convolutional layer*, that is designed to be used in forensic tasks. It is inspired by a common process that we have observed in many existing forensic algorithms. Several existing algorithms first use a predetermined predictor to produce a set of pixel value prediction errors. These prediction errors are then used as low-level forensic features from which more sophisticated manipulation detection features are built. To mimic this process, our constrained convolutional layer is designed to only learn prediction error filters. The feature maps it produces correspond to prediction error fields that are used as low-level forensic traces.

The constrained convolutional layer is placed at the beginning of a CNN designed to perform a forensic task. This serves to suppress an image's content (since prediction errors largely do not contain image content) and provide the CNN with low-level forensic features. Deeper layers in the CNN will learn higher-level manipulation detection features from these low-level forensic features.

To describe the constraints we enforce upon the constrained convolutional layer, we adopt the notational conventions that the superscript $^{(\ell)}$ denotes the ℓ^{th} CNN layer, the subscript $_k$ denotes the k^{th} convolutional filter within a layer, and that the central value of a convolutional filter is denoted by spatial index (0,0). We force the CNN to learn prediction error filters by actively enforcing the following constraints

$$\begin{cases} \boldsymbol{w}_k^{(1)}(0,0) = -1, \\ \sum_{m,n\neq 0}\boldsymbol{w}_k^{(1)}(m,n) = 1, \end{cases} \tag{4}$$

on each of the K filters $\boldsymbol{w}_k^{(1)}$ in the constrained convolutional layer during training. Fig. 1 depicts a set of K constrained convolutional filters convolved with an input image.

The prediction error filter constraints in the constrained convolutional layer are enforced through the following training process. Training proceeds by updating the filter weights $\boldsymbol{w}_k^{(1)}$ at each iteration using the stochastic gradient descent (SGD) algorithm during the backpropagation step, then projecting the updated filter weights back into the feasible set of prediction error filters by reinforcing the constraints in (4). The projection is done at each training iteration by first setting the central filter weight to -1. Next, the remaining filter weights are normalized so that their sum is equal to 1. This is done by dividing each of these remaining weights by the sum of all the filter weights excluding the central value. It is worth mentioning that experimentally we have found that using a central value larger than one (i.e. set the central value to -10,000 and ensure the remaining values sum to 10,000) can help improve both the numerical stability of these filters and the convergence of our CNNs loss[1]. Doing this still produces prediction error filters of the same form, but the filter weights are proportionally larger. Pseudocode outlining this process is given in Algorithm 1.

B. Analogy with existing forensic approaches

Many existing state-of-the-art approaches to perform image manipulation detection proceed by extracting hand-designed prediction residual features. Our proposed constrained convolutional layer can be viewed as a generalization of these non-adaptive feature extraction based approaches. Examples of these include resampling detectors that use the variance of prediction residues [3], [32], median filter streaking artifact

[1]The python scripts used to conduct the experiments can be found at misl.ece.drexel.edu/downloads or the project git repository https://gitlab.com/MISLgit/constrained-conv-TIFS2018/.

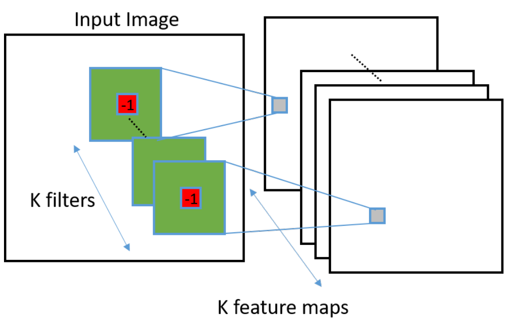

Fig. 1: The constrained convolutional layer. The red coefficient is -1 and the coefficients in the green region sum to 1.

Algorithm 1 Training algorithm for constrained convolutional layer

```
1:  Initilize $\boldsymbol{w}_k$'s using randomly drawn weights
2:  i=1
3:  while $i \leq max\_iter$ do
4:      Do feedforward pass
5:      Update filter weights through stochastic gradient
         descent and backpropagate errors
6:      Set $\boldsymbol{w}_k(0,0)^{(1)} = 0$ for all K filters
7:      Normalize $\boldsymbol{w}_k^{(1)}$'s such that $\sum_{\ell,m\neq 0} \boldsymbol{w}_k^{(1)}(\ell,m) = 1$
8:      Set $\boldsymbol{w}_k(0,0)^{(1)} = -1$ for all K filters
9:      i = i+1
10:     if training accuracy converges then
11:         exit
12: end
```

residuals [7], median filter residual features [8], [33], SPAM features [21], and rich model predictors [19]. These examples of prediction residual features suppress an image's contents but still allow traces in the form of content-independent pixel value relationships to be learned by a classifier.

To provide intuition into this, prediction residual features are formed by using some function $f(\cdot)$ to predict the value of a pixel based on that pixel's neighbors within a local window. The true pixel value is then subtracted from the predicted value to obtain the prediction residual r such that

$$r = f(I) - I, \tag{5}$$

where I is the input image or image patch. Frequently, a diverse set of K different prediction functions are used to obtain many different residual features.

It can easily be shown that the K feature maps produced by a constrained convolutional layer in Eq. (4) are residuals of the form (5). A simple way to see this is to define a new filter $\tilde{\boldsymbol{w}}_k$ as

$$\tilde{\boldsymbol{w}}_k(m,n) = \begin{cases} \boldsymbol{w}_k(m,n) & \text{if } (m,n) \neq (0,0), \\ 0 & \text{if } (m,n) = (0,0). \end{cases} \tag{6}$$

As a result, $\boldsymbol{w}_k$ can be expressed as $\tilde{\boldsymbol{w}} - \delta$ where δ is an impulse filter whose central value is 1 and 0 elsewhere. The feature map produced by convolving an image with the filter $\boldsymbol{w}_k^{(1)}$ in the constrained convolutional layer is

$$h_k^{(1)} = \boldsymbol{w}_k^{(1)} * I = (\tilde{\boldsymbol{w}}_k - \delta) * I = \tilde{\boldsymbol{w}}_k * I - I, \tag{7}$$

where $h_k^{(1)}$ is the k^{th} feature map produced by the k^{th} constrained filter in the first convolutional layer defined in Eq. (1). By defining $f(I) = \tilde{\boldsymbol{w}}_k * I$, we can see these residuals are produced in the same manner that the above mentioned feature extraction based approaches produce prediction residuals r in (5).

The residual predictors used in different multimedia forensic tasks [2], [3], [7], [20] take the form $\tilde{\boldsymbol{w}}_k$ to predict local pixel relationships. Resampling detectors for instance, operate by computing a probability measure called a *p-map* from a prediction residual r of the form shown in Eq. (5). Then higher-level features in the frequency domain of the *p-map* are learned to detect resampling artifacts [2], [3]. To detect median filtering, Kirchner and Fridrich similarly compute low-level residual features [7]. These residual features capture streaking artifacts, then higher-level detection features are learned. Furthermore, the steganalytic rich model method used in forensics [20] operates in the same manner by building several local models of pixel dependencies to compute a diverse set of residual features. Then higher-level features are learned using the co-occurrence of these residuals [19].

As a result, our approach suppresses an image's content in the same manner as prediction residual based methods. In order to capture a large number of different types of dependencies among neighboring pixels, a diverse set of $\tilde{\boldsymbol{w}}_k$'s is typically used. The advantage of modeling the residual instead of the pixel values is that the image content is largely suppressed.

Unlike prior forensic methods which use fixed predictors, however, our approach adaptively learns good predictors for feature extraction through backpropagation. Nonlinearities can be further introduced by the subsequent application of activation functions and pooling layers in higher CNN layers. Thus, the constrained convolutional layer based approach unifies an important amount of research in multimedia forensics.

IV. Network Architecture

In the previous section, we showed how low-level image manipulation features can be adaptively extracted using the constrained convolutional layer approach. We use this approach to design a CNN that is able to identify the type of editing operations in an image. Fig. 2 depicts the overall design of our proposed architecture with details about the size of each layer. Our architecture consists of four different conceptual blocks and has the ability to: (i) jointly suppress an image's content and learn prediction error features while training, (ii) extract higher level representation of the previously learned image manipulation features and (iii) learn new associations between feature maps in deeper layer by using a block that consists of 1×1 convolutional filters. These type of filters learn a linear combination of features located at the same position but in a different feature map across channels. The output of the latter block is fed to the classification block which consists of three fully-connected layers. In this work, the input layer of our CNN is a grayscale image patch sized 256×256 pixels. In what follows, we present a detailed overview of our proposed conceptual blocks as well as the different layers that we have used in our CNN's architecture.

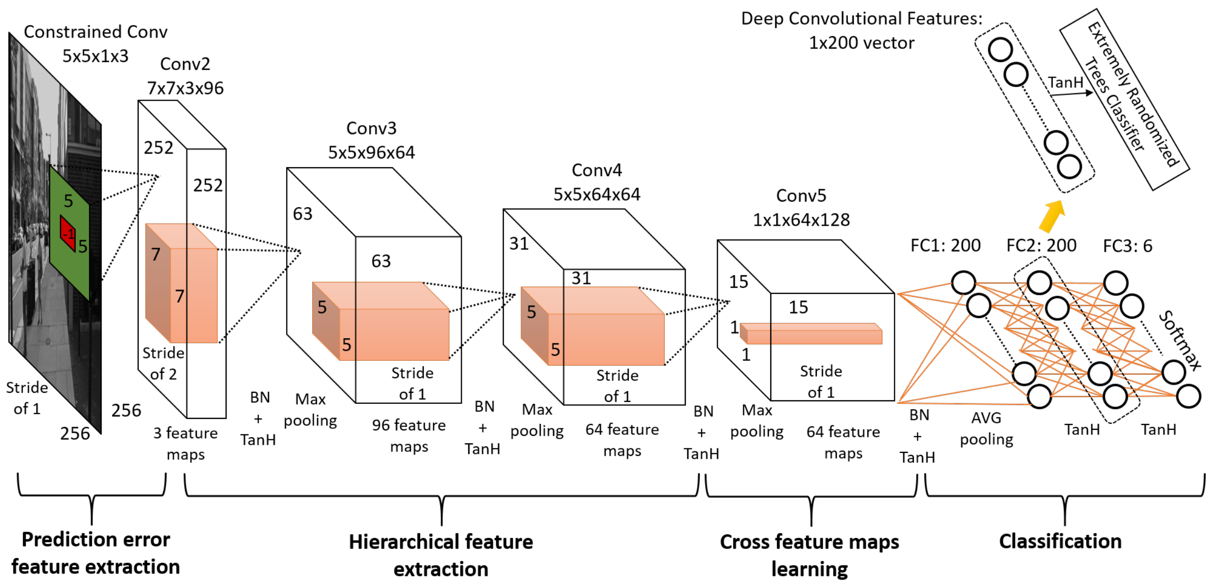

Fig. 2: MISLnet: our proposed constrained CNN architecture; BN:Batch-Normalization Layer; TanH: Hyperbolic Tangent Layer.

A. Conceptual Blocks

a) Prediction error feature extraction: CNNs in their existing form tend to learn content-dependent features. Therefore, in our proposed architecture the first block consists of a constrained convolutional layer [26], [34]. This suppresses the content and constrains CNN to learn prediction error features in the first layer [35], [36]. As a result, the first conceptual block learns low-level pixel value dependency features. These features are fragile and vulnerable to be destroyed by many types of nonlinear operations [37] such as pooling and activation layers which are explained later. Therefore, the output of this block is directly passed to a regular convolutional layer.

b) Hierarchical feature extraction: In order to learn higher-level prediction error features, we use a conceptual block that consists of a set of three consecutive convolutional layers each followed by a batch normalization, activation function and pooling layers. Each convolutional layer will learn a new representation of feature maps learned by the preceding convolutional layer (lower-level features).

c) Cross feature maps learning: The previously learned hierarchical features are produced by learning local spatial association within a receptive field (local region/patch convolved with a filter) in the same feature map. Next, a new association is learned between these feature maps. In order to constrain CNN to learn only association accross feature maps, we use $1{\times}1$ convolutional layer after the hierarchical feature extraction conceptual block. This has been demonstrated to improve the learning ability of CNN in steganalysis [38]. In our architecture, this layer also followed by a a batch normalization, activation function and pooling layers.

d) Classification: The deepest convolutional features learned by the previous conceptual block are directly passed to a classification block. This block consists of a regular neural network that will learn to classify the input data from the previously learned features throughout CNN. To improve the performance of our CNN we use the deep features approach [39] that we explain in Section IV-G. In what follows we give more details about each type of layer we used in our CNN.

e) Differences from original architecture: Differences from original architecture: Compared to our original CNN architecture proposed [26], our new CNN architecture has gone through substantial design refinement. Specifically, this new architecture contains fewer filters in the constrained convolutional layer, uses a different filter size in the Conv3 (referred to as Conv2 in [26]) convolutional layer, uses a different number of filters in the Conv3 and Conv4 layers, includes one more ‘traditional’ convoltional layer than our architecture in [26], adds an additional $1{\times}1$ layer after the ‘traditional’ convolutional layers, uses a different type of pooling before the fully connected layers, uses different activation functions, uses batch normalization instead of local response normalization, contains a different number of neurons in each fully connected layer (this network uses noticeably fewer neurons in these layers), and uses the activations of the last fully connected layer as “deep features” that are provided to an extremely randomized tree classifier. These design choices have been motivated by an extensive series of experiments, the most important of which are discussed in detail in Section V. Additionally, while training this network we use a different initial learning rate as well as a learning rate that decreases during training [26] uses a fixed learning rate) and train using a different batch size. Training information for this network is discussed in detail in Section V.

B. Convolutional Layers

From Fig. 2, one can notice that we use three different types of convolutional layers, namely one “Constrained Conv” layer which is the constrained convolutional layer presented in Section III, three regular convolutional layers and the $1{\times}1$ convolutional filters in “Conv5”. More specifically, a patch sized 256×256 from a grayscale input image is first convolved with three different 5×5 constrained convolutional filters with a stride equal to 1. These filters learn the prediction error features between the estimated center pixel and it's local neighbors. The constrained convolutional layer yields feature maps of prediction residuals of dimension $252 \times 252 \times 3$.

To learn higher-level representative features and new associations between the prediction residual feature maps, we use three regular convolutional layers, namely "Conv2" with 96 filters of size $7 \times 7 \times 3$ and stride of 2, "Conv3" with 64 filters of size $5 \times 5 \times 96$ and stride of 1 and "Conv4" with 64 filters of size $5 \times 5 \times 64$ and stride of 1. The output dimensions of these convolutional layers are respectively, $126 \times 126 \times 96$, $63 \times 63 \times 64$ and $31 \times 31 \times 64$. Finally, we use 128 different 1×1 convolutional filters with stride of 1 in "Conv5". This type of layer learns the association across feature maps, i.e., linear combination of features across channels located at the same spacial location. The output dimension of this convolutional layer is $15 \times 15 \times 128$. Finally, from our architecture one can notice also that we use a batch normalization layer after every regular convolutional layer. A brief overview about the batch normalization layer is given in Section IV-E.

C. Fully-connected Layers

To identify the type of the processing operation that an input image has undergone, the output of all these convolutional layers is fed to a *classification* block which consists of a fully-connected neural network defined by three layers. More specifically, the first two fully-connected layers contain 200 neurons. These layers learn new association between the deepest convolutional features in CNN. The output layer, also called classification layer, contains one neuron for each possible tampering operation and another neuron that corresponds to the unaltered image class.

D. Activation Function

A convolutional layer is typically followed by a nonlinear mapping called an activation function. This type of function is applied to each value in the feature maps of every convolutional layer. There exist many types of activation functions. In computer vision applications, the ReLU activation function has been used successfully [23], [40]. He *et al.* [41] proposed another type of activation function called PReLU that leads to surpass human-level performance on visual recognition challenge [42]. Additionally, Clever *et al.* [43], proposed the exponential linear units (ELU) activation function, which considerably speeds up learning and obtains less than 10% classification error compared to a ReLU network with the same architecture.

In our proposed CNN, as depicted in Fig. 2 we propose to constrain the range of data values with the saturation regions of hyperbolic tangent (TanH) activation function at every stage of the network. Introducing nonlinearity throughout the network layers strengthens CNN capability to separate the feature space. However, one can notice that feature maps learned by the constrained convolutional layer are not followed by a TanH layer. This is mainly because the learned prediction error features can easily be destroyed by many types of nonlinear operations including activation functions. In our experiments (see Section V-G), we compare the performance of our proposed TanH based CNN architecture to CNN models with different choices of activation functions that we mentioned above.

Finally, the output layer is followed by a softmax activation function. This type of activation function maps features learned by the last fully-connected layer to a set of probability values where the output of all neurons in this layer sum up to 1. The identification of the image manipulation types in subject images can be performed by choosing the editing operation associated with the neuron in the softmax layer with the highest activation level.

E. Batch Normalization

Researchers in computer vision have developed several techniques to normalize the data throughout the CNN architecture. Early deep learning architectures use the local response normalization (LRN) layer which normalizes the central coefficient within a sliding window in a feature map with respect to its neighbors. Recently Ioffe *et al.*, proposed in [44] the batch normalization layer which dramatically accelerates the training of deep networks. This type of mechanism minimizes the internal covariate shift which is the change in the input distribution to a learning system.

This is done by a zero-mean and unit-variance transformation of the data while training the CNN model. The input to each layer gets affected by the parameters of all previous layers and even small changes get amplified. Thus, this type of layer addresses an important problem and increases the final accuracy of a CNN model. Therefore, in our proposed architecture we use a batch normalization layer after each regular convolutional layer. However, the prediction error convolutional filters outputs are directly convolved with the next convolutional layer without using the batch normalization layer.

F. Pooling

In our CNN, we use two different types of pooling, i.e., three max-pooling and one average-pooling. We experimentally demonstrate our choice of pooling layers in Section V. Similarly to [23], we use an overlapping kernel with size 3×3 and stride of 2. Explicitly, the max-pooling layer retains the maximum value within the local neighborhood of the sliding window, whereas, the average-pooling layer retains the average in a local neighborhood. The purpose of this type of layer is to reduce the dimensionality of the feature maps. This reduces the computational cost of training and decreases the chances of over-fitting. More specifically, the set of parallel convolutional operations yields a high dimensional feature maps volume. Therefore, pooling layers are useful for subsampling as well as improving the accuracy by retaining the most representative features.

In our architecture, the four used pooling layers have respectively reduced the feature maps dimensions from $126 \times 126 \times 96$ to $63 \times 63 \times 96$, from $63 \times 63 \times 64$ to $31 \times 31 \times 64$, from $31 \times 31 \times 64$ to $15 \times 15 \times 64$ and finally from $15 \times 15 \times 128$ to $7 \times 7 \times 128$.

G. Deep Convolutional Features

As depicted in Fig. 2, to classify the output features learned by the set of convolutional layers we use a neural network classifier with a softmax activation function in the output layer. However, other approaches to producing a final classification decision may work better than a fully-connected and softmax layer. Therefore, we propose to extract the output of the activation function [39] from the second fully-connected layer "FC2" by doing a feedforward pass of the training and testing data after completing the training of our CNN. Subsequently, we train an extremely randomized trees classifier using the new collected data. Each 256×256 patch in the training and testing data has its corresponding 200 features vector.

V. Experiments

To assess the performance of our proposed constrained CNN for performing image manipulation detection, we conducted a set of experiments and analysis. In these experiments, we first evaluated our proposed CNN's ability to detect a single manipulation. This was done for each of the editing operations and parameters listed in Table I. Next, we evaluated our CNN's ability to be used as a multiclass classifier to perform general image manipulation detection with five different editing operations listed in Table III. We also evaluated our approach in two more challenging scenarios, i.e., when the editing parameters are chosen to be arbitrary as listed in Table V, and when the editing operation can be followed by another editing operation. We compared our CNN-based approach to an existing general-purpose manipulation detection approach that uses steganalysis features [19] to perform manipulation detection [20]. Next, we compared the performance of our proposed architecture with different structural design choices, e.g., the choice of pooling and activation function layers. Finally, we tested the performance of our proposed CNN trained with a large scale dataset, then we evaluated it in a real world scenario. The results of these experiments demonstrate that our CNN can accurately learn detection features directly from data and achieve state-of-the-art performance.

In every experiment, we trained each CNN for 60 epochs, where an epoch is the total number of iterations needed to pass through all the data samples in the training set. Additionally, while training our CNNs, their testing accuracies on a separate testing dataset were recorded every $1,000$ iterations to produce tables and figures in this section. Note that training and testing processes were disjoint. We implemented all of our CNNs using the Caffe deep learning framework [45]. We ran our experiments using an Nvidia GeForce GTX 1080 GPU with 8GB RAM. The datasets used in this work were all converted to the lmdb format.

A. Single manipulation detection

In our first set of experiments, we used our proposed CNN architecture in Fig. 2 to distinguish between images edited with one particular type of manipulation and unaltered images. The output layer of our CNN consisted of two neurons, i.e., original (OR) versus manipulated image. As described in Section IV-G, we also used the deep convolutional features extracted from our CNNs to train an extremely randomized trees (ET) classifier in order to identify manipulated images in testing data.

TABLE I: Editing parameters used to create our 10 experimental databases for CNN-based single manipulation detection.

Editing operation	**Parameters**
Median Filtering (MF)	$K_{size} = 3, 5$
Gaussian Blurring (GB) with $\sigma = 1.1$	$K_{size} = 3, 5$
Additive White Gaussian Noise (AWGN)	$\sigma = 0.5, 2$
Resampling (RS) using bilinear interpolation	Scaling $= 1.2, 1.5$
JPEG compression	$QF = 70, 80$

To conduct these experiments, we created 10 different databases, where each database corresponded to one type of manipulation with different editing parameters. Each database consisted of $60,000$ grayscale images of size 256×256 pixels. These were created using images from the first IEEE IFS-TC image forensics challenge[2]. We used $3,334$ images of size 1024×768 for the training and testing data. Each image was converted to grayscale by retaining only its green color layer. Next, each grayscale image was divided into 256×256 pixel subimages, then nine central subimages were retained.

To train our CNNs, we used $50,000$ grayscale patches of size 256×256 for each type of manipulation. This consisted of $25,000$ unaltered patches and $25,000$ manipulated patches. These grayscale patches were created in the same manner described above by randomly selecting $2,778$ images. Each block corresponds to a new image that has its corresponding tampered images created by the 10 different editing operations listed in Table I, i.e., five types of image manipulations with two different editing parameters for each.

When training our CNN, we set the batch size equal to 64 and the parameters of the stochastic gradient descent as follows: $momentum = 0.95$, $decay = 0.0005$, and a learning rate $\epsilon = 10^{-3}$ that decreases every six epochs (approximately $4,700$ iterations) by a factor $\gamma = 0.5$. We trained the CNN in each experiment for 60 epochs (approximately $47,000$ iterations).

To evaluate the performance of our proposed approach, we used $10,000$ grayscale patches of size 256×256 for each type of manipulation listed in Table I. Each testing dataset consisted of $5,000$ unaltered patches and $5,000$ manipulated patches. These grayscale patches were made from 556 images not used for the training in the same manner described above. Thus, each training dataset has its corresponding testing dataset for the 10 types of manipulations. Table II depicts the detection rate for every type of manipulation using the softmax classification layer and the ET classifier associated with the deep convolutional features extracted from the second fully-connected layer as explained in Fig. 2. We trained the ET classifiers by varying the number of trees from 100 to 600 with a step of 100 then we reported the best detection rates that our ET-based CNN achieved.

From Table II, one can notice that our proposed CNN can achieve at least 99.36% (JPEG with $QF = 70$) detection rate with all types of manipulations. Noticeably, it can achieve 99.95% detection rate with 5×5 Gaussian bluring database.

[2]Dataset website: http://ifc.recod.ic.unicamp.br/fc.website/index.py?sec=5

TABLE II: CNN identification rate for single manipulation detection.

Classifiers	MF		GB		AWGN		RS		JPEG	
	3×3	**5×5**	**3×3**	**5×5**	$\sigma = 0.5$	$\sigma = 2$	**Scaling= 1.2**	**Scaling= 1.5**	**QF = 70**	**QF = 80**
Sotmax	99.58%	99.57%	99.74%	99.95%	99.82%	99.93%	99.40%	99.53%	99.36%	99.66%
ET	99.82%	99.71%	99.85%	99.87%	99.85%	99.96%	99.62%	99.55%	99.97%	99.89%

TABLE III: Editing parameters used to create our experimental database for CNN-based general purpose manipulation detection.

Editing operation	**Parameter**
Median Filtering (MF)	$K_{size} = 5$
Gaussian Blurring (GB) with $\sigma = 1.1$	$K_{size} = 5$
Additive White Gaussian Noise (AWGN)	$\sigma = 2$
Resampling (RS) using bilinear interpolation	Scaling $= 1.5$
JPEG compression	$QF = 70$

We also can notice that the ET classifier has improved the detection rate of each corresponding manipulation except with the 5×5 Gaussian bluring database. Our ET-based CNN approach can achieve at least 99.55% (re-sampling with scaling 1.2) detection rate with all types of manipulations. Our ET-based CNN can noticeably achieve 99.97% in identifying JPEG compression with $QF = 70$. These results show the ability of our constrained CNN in extracting good image manipulation features directly from data with different editing parameters for binary detection. Additionally, these results are very promising since our proposed deep learning approach was able to accurately detect several types of single manipulations using the same network architecture.

B. Multiple manipulation detection

In the previous experiments, our proposed constrained CNN was effective at extracting image manipulation features with different types of tampering for single manipulation detection. In this part, we evaluate our proposed approach in performing multiple image manipulation detection. Similarly to the previous set of experiments, we used the images from the 1st IEEE IFS-TC image forensics challenge website to create a database that consisted of $132,000$ grayscale patches of size 256×256. We used $2,445$ images of size 1024×768 for the training and testing data.

To train our CNNs, we used $100,000$ grayscale patches of size 256×256 in which $16,667$ patches were unaltered. These grayscale patches were made from randomly selected $1,852$ images in the same manner described in the previous experiment. The altered patches were created using the five tampering operations listed in Table III. When training our CNN, we used the same training parameters of the stochastic gradient descent that we used for the binary classifier with a learning rate $\epsilon = 10^{-3}$ that decreased every six epochs ($9,375$ iterations) by a factor $\gamma = 0.5$. We trained the CNN in each experiment for 60 epochs ($93,750$ iterations). We subsequently created a testing dataset that consisted of $32,000$ grayscale patches of size 256×256 where $5,337$ patches were not edited. These grayscale patches were made from the remaining 593 images not used for the training in the same manner described above. To create the altered patches we used the same editing operations listed in Table III.

We used our trained CNN to classify each of the images in the testing dataset. The overall manipulation identification rate of our CNN was 99.66%. Table IV shows the confusion matrices of our two proposed methods. From this table, we can see that each type of manipulation was identified with an accuracy typically greater than 99% except for the original and re-sampled images which were detected with an accuracy of 98.70% and 98.87% respectively. These results demonstrate that our proposed CNN was able to both accurately detect tampered images and identify the type of tampering.

Similarly to our single manipulation detection approach, we compared the performance of using a softmax layer versus the "deep features" approach with an extremely randomized trees (ET) classifier. More specifically, we used the activated deep convolutional features [39] that we extracted from the second fully-connected layer of our network to train an ET classifier with 700 trees. In the rest of our experiments, all the ET classifiers were trained with the same number of trees, i.e., 700 trees. Our experimental results show that the ET classifier increased the overall classification rate from 99.26% to 99.66%. We compare the results of ET-based CNN classifier to our proposed CNN with a softmax classification layer in Table IV. We can notice that the ET-based CNN method increased the identification rate of each tampering operation. The lowest detection accuracy was 99.46% for Gaussian blurring, which is still very high.

As we noted previously, the constrained convolutional layer was designed to suppress the scene and learn prediction error features. Fig. 3 depicts the output of the three filters learned by the constrained convolutional layer for three different grayscale images. One can notice that our proposed constrained convolutional layer was able to successfully suppress each image's content.

C. Multiple manipulation detection with arbitrary parameters

We assessed the performance of our approach at performing image manipulation detection in a more general scenario where editing parameters are chosen to be arbitrary. To accomplish this, we created a training and testing datasets using the same unaltered 256×256 grayscale patches that we collected in the previous experiment in Section V-B. We created modified patches using each of the manipulations and associated parameter values listed in Table V. When modifying a patch with using a specific manipulation, the associated parameter was chosen uniformly at random from the set of possible values. Additionally, Gaussian blurring was implemented using OpenCV which determines the blur variance as a function of the filter size according to the equation $\sigma^2 = \big(0.3 \times ((K_{size} - 1) \times 0.5 - 1) + 0.8\big)^2$.

In total, our database consisted of $100,000$ training patches and $32,000$ testing patches. Table VI shows the confusion

TABLE IV: Confusion matrix for identifying the manipulations listed in Table III using MISLnet.

		Predicted Class						Predicted Class					
		Softmax						**Extremely Randomized Trees**					
		OR	MF	GB	AWGN	RS	JPEG	OR	MF	GB	AWGN	RS	JPEG
True Class	OR	**98.70%**	0.67%	0.01%	0.01%	0.13%	0.45%	**99.49%**	0.15%	0.09%	0.02%	0.13%	0.11%
	MF	0.01%	**99.08%**	0.07%	0.00%	0.00%	0.82%	0.07%	**99.77%**	0.11%	0.00%	0.04%	0.00%
	GB	0.00%	0.05%	**99.15%**	0.00%	0.00%	0.78%	0.02%	0.41%	**99.46%**	0.00%	0.11%	0.00%
	AWGN	0.03%	0.00%	0.00%	**99.96%**	0.00%	0.00%	0.02%	0.00%	0.00%	**99.98%**	0.00%	0.00%
	RS	0.05%	0.00%	0.01%	0.00%	**98.87%**	1.05%	0.07%	0.36%	0.00%	0.00%	**99.51%**	0.06%
	JPEG	0.07%	0.00%	0.00%	0.00%	0.13%	**99.79%**	0.06%	0.00%	0.00%	0.00%	0.15%	**99.79%**

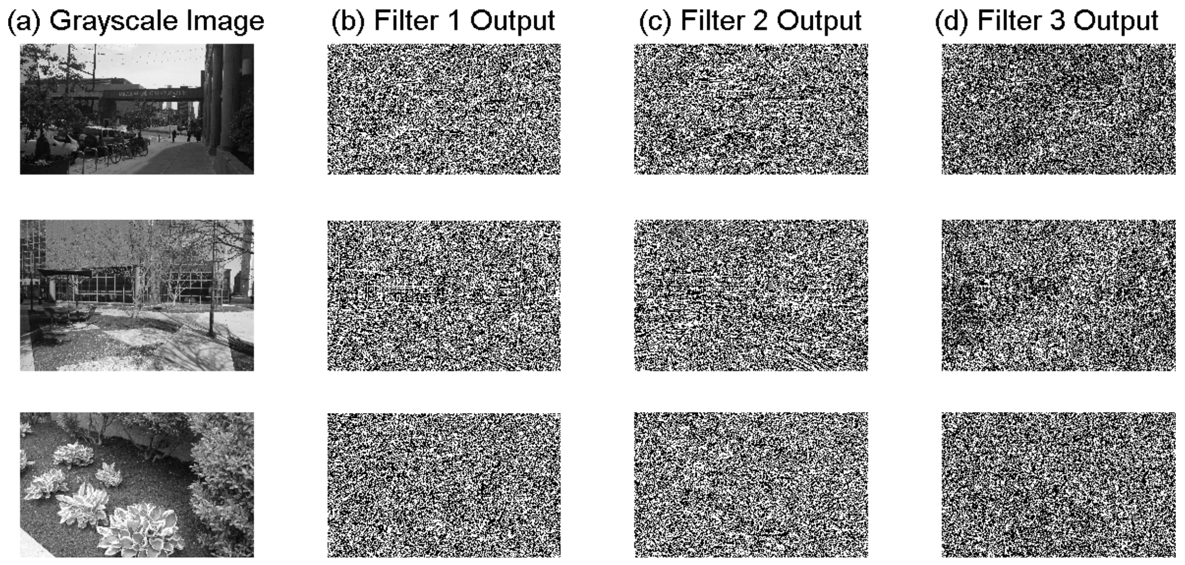

Fig. 3: Output of the three learned filters in "Constrained Conv" layer using three different grayscale images.

TABLE V: Editing parameters used to create our experimental database for CNN-based general purpose manipulation detection with arbitrary parameters; Gaussian blur with adaptive $\sigma = 0.3 \times ((K_{size} - 1) \times 0.5 - 1) + 0.8$.

Editing operation	**Parameter**
Median Filtering (MF)	$K_{size} = 3, 5, 7, 9$
Gaussian Blurring (GB) with adaptive σ	$K_{size} = 3, 5, 7, 9$
Additive White Gaussian Noise (AWGN)	$\sigma = 1.4, 1.6, \cdots, 2$
Resampling (RS) using bilinear interpolation	Scaling $= 1.2, 1.4, \cdots, 2$
JPEG compression	$QF = 60, 61, \cdots, 89, 90$

matrices of our two proposed approaches, namely softmax-based CNN and ET-based CNN. Experiments showed that our approach can achieve 98.82% and 98.99% identification rates respectively with the softmax and the ET classifiers. Noticeably, MISLnet can achieve 99.89% identification rate for JPEG compression manipulation with continuous quality factor parameter between 60 and 90, which is particularly high.

These results demonstrate that even in this more challenging scenario, our CNN is still able to accurately identify image manipulations. Additionally, in order to have a better representation for each type of manipulation this task requires to train a CNN with a larger training dataset compared to the task where images have been manipulated with fixed parameters. In Section V-H, we demonstrate that one can significantly improve CNN's performance using a larger size of the training dataset.

D. Manipulation chain detection with re-compression

To evaluate the performance of MISLnet in another challenging scenario, we used our CNN to identify an image's manipulation history when more than one manipulation could be applied, followed by JPEG re-compression. To demonstrate this, we conducted another experiment where each image patch was edited by a sequence of up to two different manipulations, then JPEG compressed using a quality factor of 90. By re-compressing each image after manipulation, we can mimic conditions similar to those in social networking applications which typically re-compress each image before distributing it.

We created data for this experiment by using 256×256 grayscale image patches collected from the Dresden Image Database [46]. This was done by retaining the 16 central blocks of the green color layer of each image. These image patches were then manipulated using a sequence of the following manipulations: median filtering (MF) using a $5{\times}5$ kernel, Gaussian blurring (GB) with $\sigma = 1.1$ using a $5{\times}5$ kernel, resizing (RS) by a factor of 1.5 using bilinear interpolation. Each manipulation sequence consisted of up to two of these operations. We adopt the notation X-Y to

TABLE VI: Confusion matrix for identifying the manipulations listed in Table V with arbitrary editing parameters using MISLnet.

		Predicted Class						Predicted Class					
		Softmax						**Extremely Randomized Trees**					
		OR	MF	GB	AWGN	RS	JPEG	OR	MF	GB	AWGN	RS	JPEG
True Class	OR	**97.21%**	0.47%	0.02%	1.78%	0.00%	0.53%	**97.73%**	0.11%	0.04%	1.95%	0.06%	0.11%
	MF	0.04%	**99.01%**	0.24%	0.02%	0.02%	0.68%	0.07%	**99.59%**	0.26%	0.02%	0.06%	0.00%
	GB	0.00%	0.71%	**98.73%**	0.00%	0.00%	0.56%	0.04%	1.03%	**98.93%**	0.00%	0.00%	0.00%
	AWGN	1.56%	0.02%	0.00%	**98.39%**	0.00%	0.04%	1.31%	0.02%	0.00%	**98.61%**	0.00%	0.06%
	RS	0.11%	0.36%	0.02%	0.00%	**99.16%**	0.36%	0.13%	0.19%	0.02%	0.00%	**99.64%**	0.02%
	JPEG	0.09%	0.00%	0.00%	0.02%	0.00%	**99.89%**	0.17%	0.26%	0.02%	0.04%	0.04%	**99.47%**

denote a sequence where the patch was first edited using manipulation X, then subsequently edited using manipulation Y (i.e. MF-RS corresponds to first applying median filtering, then applying resizing). We divided these into sets of training and testing patches created from two separate sets of images of total size $2,175$, resulting in a set of $296,000$ training patches and $52,000$ testing patches for both. In total, $29,600$ training patches and $5,200$ testing patches were unaltered. We then trained MISLnet to distinguish between each sequence of editing operations.

Tables VII shows the confusion matrix obtained from this experiment when a softmax is used to perform manipulation chain classification, while Table VIII shows the confusion matrix obtained using an extremely randomized trees (ET) classifier. Results of these experiments demonstrated that we can achieve an accuracy of 92.90% with the softmax-based CNN and 94.19% using the ET-based CNN. From Table VIII, one can observe that we can identity the type of processing operation with an accuracy typically higher than 91%. Noticeably, we can achieve 99.17% identification rate at detecting median filtering followed by resampling and 96.69% at detection Gaussian blur followed by resampling, which is particularly high. Additionally, one can also observe that the ET classifier significantly improved the detection rate of the processing operations followed by median filtering (i.e., GB-MF and RS-MF). These results demonstrate the robustness of our proposed CNN at performing image manipulation detection in a challenging and realistic scenario.

E. Comparison with SRM-based approach

We compared our trained MISLnet CNN for multiple manipulations to the rich model approach [19], [20] using both the same training and testing datasets described in Section V-B. Table IX displays a confusion matrix containing the manipulation detection results obtained using the rich model based approach. The results of these experiments showed that the rich model approach was able to achieve an average manipulation identification accuracy of 99.63%. By contrast, our constrained CNN was able to achieve an accuracy of 99.66% on the same dataset. From Tables IV and IX one can notice that our CNN achieved better identification rates for median filtering, Gaussian bluring and re-sampling.

These results demonstrate that our CNN based detector can perform as well as, or slightly better than, the rich model based detector. In Section V-H, we show that we can achieve an even higher detection accuracy with a larger amount of training data. These results are very important because our approach can learn salient image manipulation detection features directly from data. This may allow us to learn better feature extractors than the human designed incorporated into the rich model feature extractors.

Training time is an important factor when devising a data-driven manipulation detection approach. Our CNN based approach took approximately six hours to train on this database. By contrast, a multi-threaded implementation (using eight threads) of the rich model took over 58 hours to perform only feature extraction on this database using the same computer. Training the classifier for the SRM took several additional hours. As a result, it becomes extremely challenging, if not infeasible, to train the SRM on a very large database.

F. Prediction error feature extractor design choices

The overall performance of MISLnet depends on several design choices. An important one of these is the design of first CNN layer, which extracts prediction error features in our network. To determine the optimal design of this layer, we conducted several experiments to examine the influence of other filter type choices for the first CNN layer including the use of a fixed high-pass filter [37], [38], as well as not using a prediction error feature extraction block (i.e. beginning our CNN with a standard convolutional layer). Additionally, we conducted experiments to determine the optimal number and size of the filters in the constrained convolutional layer of MISLnet.

Choice of prediction error feature extractor: We evaluated the advantage of using the constrained convolutional layer as MISLnet's first layer through two sets of experiments. In each experiment, we trained and evaluated our proposed CNN architecture using three different choices for the first layer: (1) using a constrained convolutional layer, (2) without using a constrained convolutional layer, and (3) replacing the constrained convolutional layer in MISLnet with a generic fixed high-pass filter. In these experiments, we used the same high-pass commonly employed in forensics and steganalysis [37], [38].

To assess the performance gains achieved by the constrained convolutional layer, we report the classification accuracy MISLnet achieved using each choice of filter for the beginning of our CNN. Additionally, we report the *relative error reduction* achieved by using the constrained convolutional layer instead of each alternative. Relative error reduction measures the reduction in error achieved by a classifier normalized by total

TABLE VII: Confusion matrix for identifying manipulation chains followed by re-compression using MISLnet with a softmax.

		Predicted Class									
		OR	MF	GB	RS	MF-GB	GB-MF	MF-RS	RS-MF	GB-RS	RS-GB
True Class	OR	**99.27%**	0.06%	0.12%	0.52%	0.02%	0.00%	0.02%	0.00%	0.00%	0.00%
	MF	0.13%	**90.54%**	0.02%	0.02%	1.08%	3.02%	0.10%	5.10%	0.00%	0.00%
	GB	0.00%	0.23%	**93.56%**	0.25%	0.44%	0.04%	0.04%	0.04%	0.02%	5.38%
	RS	0.19%	0.06%	2.12%	**97.15%**	0.04%	0.00%	0.12%	0.08%	0.00%	0.25%
	MF-GB	0.00%	0.10%	0.23%	0.00%	**98.08%**	0.77%	0.02%	0.04%	0.23%	0.54%
	GB-MF	0.00%	1.40%	0.12%	0.00%	8.65%	**80.13%**	0.08%	9.48%	0.04%	0.10%
	MF-RS	0.02%	0.23%	0.00%	0.19%	0.06%	0.50%	**97.69%**	1.04%	0.27%	0.00%
	RS-MF	0.00%	2.90%	0.02%	0.00%	1.65%	10.75%	0.21%	**84.21%**	0.06%	0.19%
	GB-RS	0.00%	0.08%	0.00%	0.02%	0.87%	0.02%	0.94%	0.12%	**93.94%**	4.02%
	RS-GB	0.00%	0.25%	1.06%	0.02%	1.23%	0.08%	0.06%	0.10%	2.71%	**94.50%**

TABLE VIII: Confusion matrix for identifying manipulation chains followed by re-compression using MISLnet with an ET classifier.

		Predicted Class									
		OR	MF	GB	RS	MF-GB	GB-MF	MF-RS	RS-MF	GB-RS	RS-GB
True Class	OR	**99.33%**	0.06%	0.1%	0.05%	0.00%	0.00%	0.02%	0.00%	0.00%	0.00%
	MF	0.15%	**91.77%**	0.02%	0.02%	0.52%	2.12%	0.29%	5.10%	0.00%	0.02%
	GB	0.00%	0.21%	**95.00%**	0.87%	0.42%	0.04%	0.04%	0.00%	0.06%	3.37%
	RS	0.17%	0.00%	0.65%	**98.94%**	0.02%	0.00%	0.08%	0.08%	0.02%	0.04%
	MF-GB	0.00%	0.31%	0.19%	0.00%	**95.87%**	2.48%	0.02%	0.29%	0.42%	0.42%
	GB-MF	0.00%	1.44%	0.00%	0.02%	3.38%	**86.02%**	0.13%	8.87%	0.06%	0.08%
	MF-RS	0.02%	0.04%	0.00%	0.01%	0.00%	0.08%	**99.17%**	0.38%	0.21%	0.00%
	RS-MF	0.00%	3.54%	0.02%	0.00%	0.17%	9.38%	0.65%	**86.00%**	0.17%	0.06%
	GB-RS	0.00%	0.02%	0.00%	0.00%	0.40%	0.02%	0.96%	0.06%	**96.69%**	1.85%
	RS-GB	0.00%	0.10%	2.08%	0.06%	0.63%	0.13%	0.04%	0.23%	3.56%	**93.17%**

TABLE IX: Confusion matrix for identifying the manipulations listed in Table III using the rich model.

		Predicted Class					
		OR	MF	GB	AWGN	RS	JPEG
True Class	OR	**99.83%**	0.07%	0.00%	0.00%	0.02%	0.07%
	MF	0.02%	**99.23%**	0.06%	0.00%	0.13%	0.56%
	GB	0.08%	0.09%	**99.42%**	0.00%	0.04%	0.38%
	AWGN	0.00%	0.00%	0.00%	**100%**	0.00%	0.00%
	RS	0.17%	0.04%	0.00%	0.00%	**99.47%**	0.32%
	JPEG	0.02%	0.04%	0.00%	0.02%	0.04%	**99.89%**

error reduction that is possible. For reference, the relative error reduction (RER) is calculated according to the formula $RER = (e_1 - e_2)/e_1$ where e_1 corresponds to the error achieved by the lower performing method and where e_2 corresponds to the error achieved by the higher performing method.

In our first experiment, we examined the effect of each filter choice when performing manipulation detection in the same manner as in Section V-B, i.e. images altered using the manipulations listed in Table III. This was done by using the same training and testing datasets described in Section V-B, as well as the same training procedures (i.e. batch size, learning rate, etc.). Each trained CNN was then used to classify each image patch in the test set.

The results of this first experiment are shown in Table X. From this table, we can see that when MISLnet was trained without the constrained convolutinal layer, it's performance decreased by 0.90%. This corresponds to a relative error reduction of $RER = 54.87\%$ achieved by using the constrained convolutional layer. Additionally, we can see that using the constrained convolutional layer achieved an accuracy 0.31% higher detection than when a fixed high-pass filter was used. This corresponds to a relative error reduction of $RER = 29.54\%$ over a fixed high-pass filter. These results demonstrate the advantage of using the constrained convolutional layer.

The full benefit of using a constrained convolutional layer can be seen when considering more challenging forensic scenarios. To demonstrate this, we conducted another set of experiments where each image patch was edited by a sequence of up to two different manipulations, then JPEG compressed using a quality factor of 90. By re-compressing each image after manipulation, we can mimic conditions similar to those in social networking applications which typically re-compress each image before distributing them.

We built our experimental database by using the same training and testing datasets in Section V-D. We then trained MISLnet to distinguish between each sequence of editing operations using both a constrained convolutional layer, a fixed

TABLE X: MISLnet accuracy with different settings in the prediction error feature extraction block for multiple manipulation detection.

Prediction error conceptual block	Accuracy	RER (w.r.t. ours)
Constrained conv layer	**99.26%**	–
W/out constrained conv layer	98.36%	54.87%
High-pass filter	98.95%	29.54%

TABLE XI: MISLnet performance with different settings in the prediction error feature extraction block for sequence of manipulations detection; Image patches were JPEG post-compressed with QF=90.

Prediction error conceptual block	Accuracy	RER (w.r.t. ours)
Constrained Conv layer	**92.90%**	–
W/out constrained Conv layer	84.10%	55.34%
High-pass filter	80.63%	63.35%

high-pass filter, and without using a prediction error block (i.e. a normal convolutional layer). Next, we used each version of our CNN to determine the manipulation sequence of each patch in the testing set.

The classification accuracies obtained by our CNN using each choice for the first layer is shown in Table XI. From this table, we can see that when the constrained convolutional layer was used, MISLnet can identify the sequence of manipulations used to modify each patch with 92.90% accuracy. By contrast, using a fixed high-pass filter yields an accuracy of 80.63%, while using a normal convolutional layer yields an accuracy of 84.10%. In these experiments, using a constrained convolutional layer produces a relative error reduction of 63.35% over a fixed high-pass filter and a relative error reduction of 55.34% over a normal convolutional layer. These results clearly demonstrate the advantage of using the constrained convolutional layer in challenging manipulation detection scenarios.

TABLE XII: MISLnet detection accuracy with different number of filters in the "Constrained Conv" layer.

#. Filters	Testing Accuracy
1	99.04%
2	98.02%
3	**99.26%**
4	99.11%
5	99.05%
6	98.97%

"Constrained Conv" layer parameters: We conducted two sets of experiments to investigate the impact of the number of filters and their dimension in the "Constrained Conv" layer on CNN's performance. To accomplish this, we used the same training and testing datasets that we described in Section V-B. In our first experiment, we identified the optimal number of filters to use in the constrained convolutional layer by letting the number of filters vary from 1 to 6 and evaluating the manipulation detection accuracy achieved by our CNN under each scenario. Table XII shows the results of our experiments. We can notice that our proposed MISLnet architecture with the choice of three constrained filters maximizes CNN's performance and outperforms the other choices of filter numbers by at least 0.15%.

In our second experiment, we examined the effect of the size of the filters in the constrained convolutional layer. This was done by evaluating the detection accuracy of our CNN using filters of size 3×3, 5×5 and 7×7 in the constrained convolutional layer. We kept the total number of filters in the "Constrained Conv" layer fixed (3 filters). Experiments showed that our choice of 5×5 "Constrained Conv" layer with 99.26% identification rate outperforms the other dimension choices. More specifically, with 3×3 constrained filters MISLnet can achieve 98.91% identification rate and 99.07% identification rate when using 7×7 constrained filters in the "Constrained Conv" layer. Taken together, the results of these two experiments show that using three 5×5 filters in the constrained convolutional layer maximizes the performance of our CNN.

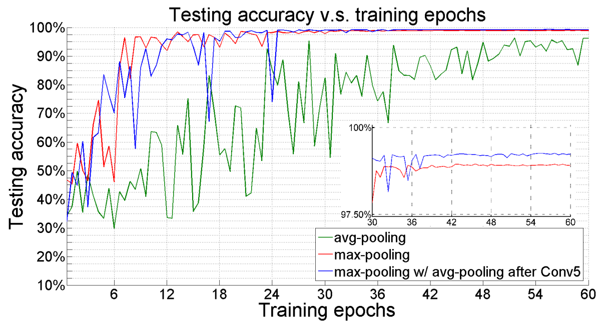

Fig. 4: CNN testing accuracy v.s. training epochs, blue: max-pooling with avg-pooling after Conv5, red: max-pooling, green: avg-pooling.

TABLE XIII: CNN testing accuracy with different pooling operations.

Pooling operations	Accuracy
Avg-pooling	96.35%
Max-pooling	98.95%
Max-pooling w/ avg-pooling after Conv5	**99.26%**

G. Architecture design choices

The structural design of a CNN's architecture has a large impact on its final accuracy. We ran several sets of additional experiments related to the structural design of our CNN's architecture. In this paper we present three of these experiments, namely (1) the choice of the pooling layer, (2) the choice of the activation function, and (3) the choice of the stride size in the "Conv2" layer with different input patch sizes (i.e., 256×256, 128×128 and 64×64). To accomplish this, we started with the fixed architecture defined in Fig. 2. Then we changed one architectural design choice in each experiment such as the choice of pooling layer or activation function. For smaller patch sizes, each image patch in the training and testing datasets were cropped in the center.

Pooling layer: We first evaluated the impact on our CNN's performance using different types of pooling layers. We trained three CNN models using the architecture described in Fig. 2 with different choices of pooling layers, i.e., max-pooling, average-pooling and max-pooling with average pooling after the "Conv5" layer. In our CNN we used 1×1 convolutional filters in the "Conv5" layer to learn new association between feature maps. Because of this, the choice of pooling

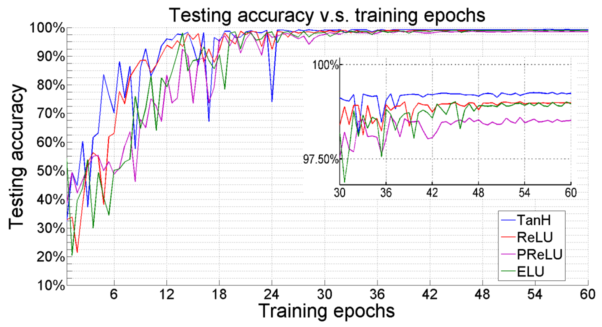

Fig. 5: CNN testing accuracy v.s. training epochs with different activation functions, blue: TanH, red: ReLU, purple: PReLU, green: ELU.

TABLE XIV: CNN testing accuracy with different activation functions.

Activations functions	Accuracy
PReLU	98.63%
ReLU	99.02%
ELU	99.02%
TanH	**99.26%**

before the fully-connected layers that perform classification is important. Table XIII summarizes the best identification rates achieved by the different pooling choices that we have considered in this experiment. We can observe that the choice of max-pooling with average pooling after the "Conv5" layer outperforms the other pooling choices and can achieve an average accuracy of 99.26%. From Fig. 4, one can also notice that the average-pooling layer based CNN converges considerably slower to a lower overall accuracy than the other two alternatives.

Activation function: In our second experiment, we evaluated the choices of activation functions. We compared our TanH-based network in Fig. 2 to ELU, ReLU and PReLU based networks. We report the best achieved identification rates of these networks in Table XIV. We can notice that the TanH network performance is 0.63% better than a PReLU network and 0.24% better than ReLU and ELU networks. Fig. 5 depicts the testing accuracy versus the training epochs curves for the four choices of activation function. One can observe from this that TanH and ReLU networks converges slightly quicker to a higher accuracy.

Convolutional Stride Size: The choice of the convolutional stride size is important since it will determine the dimension of features throughout CNN. The bigger the convolutional stride, the smaller the dimension of the feature maps produced by the CNN. This is critical when using relatively small input patch sizes. We evaluated the impact of the stride size in the "Conv2" layer on the CNN's performance. When performing image manipulation detection using different patch sizes, the CNN architecture with 256×256 input layer doesn't necessarily achieve the best identification rate. Therefore, we compared the identification rate of our CNN using a stride of 1 versus using a stride of 2 in "Conv2" layer (see Fig. 2) with different input patch size.

The results of our experiments are shown in Table XV. We

TABLE XV: Identification rate of CNN when trained with different image patch sizes; stride of 1 versus stride of 2 in "Conv2" layer.

Patch size	Stride of 1	Stride of 2
256×256	98.93%	99.26%
128×128	98.48%	98.25%
64×64	97.80%	97.33%

can notice that with 128×128 and 64×64 patches, a CNN that uses a stride of 1 in "Conv2" layer outperforms one that uses a stride of 2 in the "Conv2" layer. Thus, the "Conv2" layer with a smaller stride extracts higher dimensional features which may lead deeper CNN layers to extract better high-level image manipulation features. However, with 256×256 patches, a CNN with a stride of 2 in the "Conv2" layer can achieve a higher identification rate. These experiments show that using different choices in the structural design of CNN we can still improve the identification rate with smaller input patches. In the rest of our experiments we use a stride of 1 in the "Conv2" layer only with patches smaller than 256×256.

H. *Effect of training set size*

We have shown through our experiments that our proposed constrained CNN is very effective at learning manipulation detection features and accurately detecting multiple manipulations. A CNN's performance, however, is dependent on the size and quality of the training set [47], [48]. We conducted an experiment to show the effect of the training set size on our CNN's accuracy. This experiment shows that we can improve our results even further using a large database of training data. Given the limited number of images in the 1st IEEE IFS-TC image forensics challenge database, we used the Dresden Image Database [46] to build our training and testing data. In total, our experimental database consisted of $1,250,000$ grayscale patches of size 256×256.

We built our training dataset in the same manner that we described in the previous experiments. Our training dataset consisted of 1.2 million patches where $200,024$ patches were unaltered and $999,976$ patches were altered. To accomplish this, we randomly selected $16,483$ images that we divided into 256×256 blocks and we retained all the nine central patches from the green layer of the first $14,816$ images and the 40 central patches from the green layer of the remaining images. We then created their corresponding edited patches using the five tampering operations listed in Table III.

We trained our CNN with different input patch sizes (i.e., 256, 128 and 64) and different numbers of training patches (i.e., $100K$, $200K$, $400K$, $800K$ and $1.2M$). For training, we used the same parameters that we used in the previous set of experiments, where the learning rate was decreased every six epochs and every CNN was trained for 60 epochs.

To evaluate our trained CNNs, we built a testing dataset that consisted of $50,000$ grayscale patches of size 256×256 where $8,350$ patches were not tampered. These grayscale patches were made from 334 images not used for the training. Similarly to the training dataset, all the images were divided into 256×256 blocks and we retained all the 25 central patches

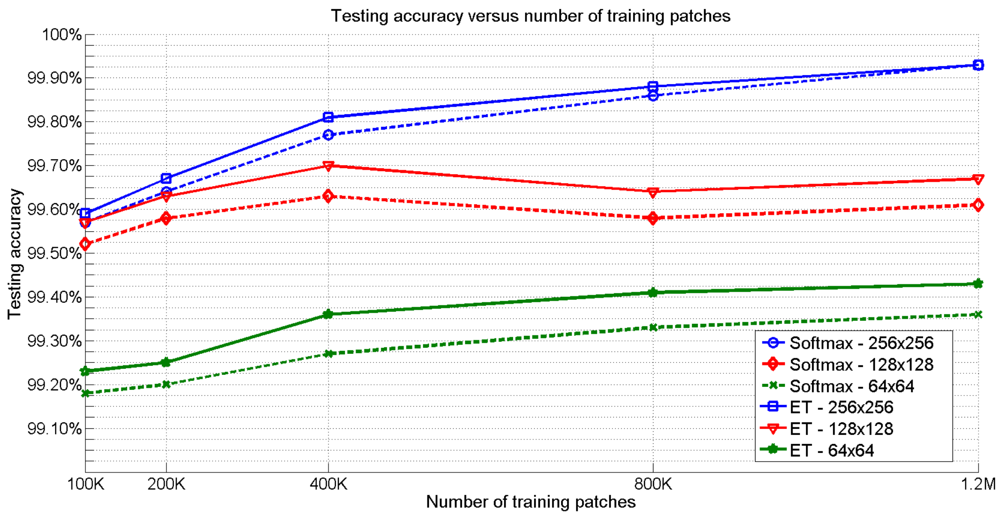

Fig. 6: CNN testing accuracy v.s. number of training patches from Dresden database [46] using Softmax (dashed line) and Extremely Randomized Trees (ET) (solid line) with different patch sizes; blue: 256×256, red:128×128, green: 64×64.

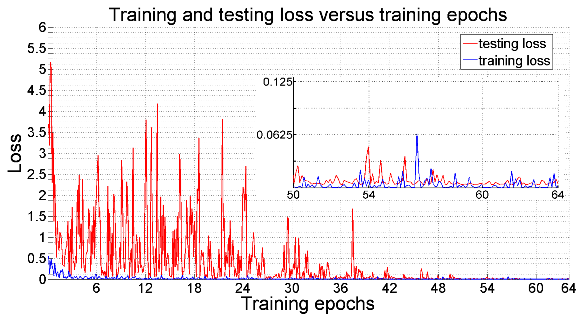

Fig. 7: Training and testing loss versus training epochs using 1.2 million training patches and $50K$ testing patches from Dresden Database of size 256×256. Losses were recorded every $1K$ iterations

TABLE XVI: MISLnet performance when trained using 1.2 million patches from the Dresden Image Database [46] and tested on patches from the both the Dresden Image Database and from a dataset of images taken by 34 different camera models (External Testing Data).

	Dresden Testing Data		External Testing Data	
Patch size	**Softmax**	**ET**	**Softmax**	**ET**
256×256	99.93%	99.93%	99.27%	99.33%
128×128	99.61%	99.67%	99.60%	99.69%
64×64	99.36%	99.43%	99.38%	99.51%

from the green layer. The edited testing patches were created using the five tampering operations listed in Table III.

Fig. 6 depicts the testing accuracy of our constrained CNN versus the number of training patches. Results are shown for different input patch sizes using both softmax and ET classifiers. We can notice that the performance of our CNN is almost always improved when we used a larger number of training patches. One can observe that when the number of training patches is sufficiently large, the softmax and ET classifiers achieve almost the same testing accuracy. Our CNN was able to achieve at least 99.18% with $100,000$ training patches of size 64×64. Additionally, we can observe that when CNN trained with 1.2 million training patches of size 256×256 we can achieve 99.93% using softmax and ET classifiers. This is noticeably better than the results achieved by the SRM.

To differentiate overfitting from poor optimization, machine learning researchers monitor the loss rate on the training set and the testing set [48]. If the training and testing losses converge to approximately the same rate, and the testing loss rate does not increase over training cycles (i.e., epochs), it is an evidence that the CNN is not overfitting [48].

Fig. 7 depicts the training and testing loss versus the training epochs of our CNN trained with 1.2 million patches and tested with 50 thousand patches of size 256×256. Both training and testing datasets were collected from the Dresden database and used in the previous experiment in Section V-H where CNN was trained with different size of large scale data. Recall that training and testing are disjoint in all our experiments.

One can observe that the testing loss decreases throughout the training epochs. We can notice that after 48 epochs as we add more training cycles (i.e., epochs) the testing loss does not increase and both training and testing converge to approximately the same loss rate (see Fig. 7). Thus, we experimentally demonstrated that the proposed CNN is not overfitting [48]. This demonstrates also that we can avoid overfitting when training a reasonably small CNN with a large scale training data [48]. It is worth mentioning that in all our experiments CNN achieves its best identification rate when the testing loss decreases and converges to the same training loss rate during the last several epochs of training. In what follows we evaluate our proposed approach in real world scenario.

I. External experimental database

In a real world scenario, a forensic investigator must examine images captured by several different and possibly unknown cameras. These may be different than the cameras used to train their CNN. It is important for their results to hold consistent when using two different sets of data captured by different devices for training and testing. To mimic this scenario, we used our CNN trained with 1.2 million patches

TABLE XVII: Confusion matrix for identifying the operation types using MISLnet (w/ softmax) trained with 2.5 million patches.

		Predicted Class					
		OR	MF	GB	AWGN	RS	JPEG
True Class	OR	**99.95%**	0.01%	0.00%	0.00%	0.00%	0.04%
	MF	0.02%	**99.89%**	0.02%	0.00%	0.02%	0.04%
	GB	0.00%	0.00%	**100%**	0.00%	0.00%	0.00%
	AWGN	0.00%	0.00%	0.00%	**100%**	0.00%	0.00%
	RS	0.00%	0.00%	0.00%	0.00%	**99.99%**	0.01%
	JPEG	0.01%	0.00%	0.00%	0.00%	0.00%	**99.99%**

from the Dresden database to perform general-purpose image manipulation detection on images taken using 34 different camera models that we have manually collected.

To evaluate the performance of our trained CNN on a different dataset, we built a "wild" testing dataset of images taken by 34 new camera models. This dataset consisted of $50,000$ grayscale patches of size 256×256, where $8,350$ patches were not edited. This was accomplished by retaining all the 25 central 256×256 blocks from the green layer of 334 randomly selected images. Next, similarly to our previous set of experiments, we created their corresponding edited patches using the five tampering operations listed in Table III.

Table XVI shows the performance of our CNN in a "real world" scenario with different input patch sizes. One can notice that we can still identify the type of image editing with at least 99.33% accuracy when we use 256×256 testing input patches. We also can notice that with smaller patch sizes our deep learning method is able to detect the type of image editing with an accuracy higher than when tested on patches from the Dresden testing dataset. More specifically, in the real world scenario, our proposed ET-based constrained CNN can identify the different types of image manipulations in 64×64 and 128×128 input image patches with accuracies of 99.51% and 99.69% respectively, whereas with our Dresden experimental testing dataset it can achieve accuracies of 99.43% and 99.67% respectively. These results demonstrate the robustness of our proposed approach when used in real world scenarios.

Finally, to further improve the learning ability of CNN we built a new training dataset that contains 2.5 million patches from our collected '34-camera-model' database to classify each patch in the testing dataset. To accomplish this, we similarly retained all the 25 central patches of size 256×256 from the green layer of the remaining $16,667$ images not used for testing. Then we created their corresponding edited patches using the five tampering operations. In total, the unaltered training patches consisted of $416,675$. We finally trained our CNN using the same parameters that we have used in the previous experiments.

We used our re-trained CNN to identify the type of manipulation the testing dataset described in the previous experiment. Our proposed constrained CNN was able to classify the testing dataset with 99.97% accuracy. Table XVII shows the confusion matrix of the trained CNN. Our method can achieve at least 99.89% accuracy for all manipulations considered. Noticeably, our approach can identify Gaussian blur and AWGN operations with 100% accuracy. Additionally, compared to the performance using $100K$ training patches (see Table IV), one can notice that our CNN trained with significantly more data can accurately distinguish between JPEG compressed images and both resampled and unaltered images. Similarly to the experiments in Section V-H, these results demonstrate again that with a larger scale of training data we can improve our CNN's performance.

VI. Conclusion

In this paper, we proposed a novel deep learning based approach to perform general-purpose image manipulation detection. Unlike existing approaches that rely on hand-designed features, our proposed CNN is able to jointly suppress an image's content and adaptively learn image manipulation detection features directly from data. To accomplish this, we developed a new type of layer, called a constrained convolutional layer, that forces our CNN to learn prediction error filters that produce low-level forensic features. Using this layer, we designed a new CNN architecture that is able to accurately detect multiple types of image manipulations.

Through a series of experiments, we assessed the ability of our proposed constrained CNN to perform image manipulation detection. The results of these experiments showed that our CNN can be trained to accurately detect individual targeted manipulations as well as multiple types of manipulations. To further assess the performance of our constrained CNN, we compared it to the SRM-based general purpose image manipulation detection approach (i.e. the current state-of-the-art detector) using five different image manipulations. This experimental comparison showed that our proposed CNN architecture can outperform the SRM, particularly when using large scale training data. Additionally, we conducted a set of experiments to mimic a realistic scenario where a forensic investigator will use our CNN to analyze images from different, possibly unknown source devices than we used to train our CNN. These experimental results show that our CNN can still accurately detect image manipulations even when there is a source mismatch between the data used to train our CNN and an image under investigation.

References

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

[2] A. C. Popescu and H. Farid, "Exposing digital forgeries by detecting traces of resampling," *IEEE Transactions on Signal Processing*, vol. 53, no. 2, pp. 758–767, Feb. 2005.

[3] M. Kirchner, "Fast and reliable resampling detection by spectral analysis of fixed linear predictor residue," in *Proceedings of the 10th ACM Workshop on Multimedia and Security*, ser. MM&Sec '08. New York, NY, USA: ACM, 2008, pp. 11–20.

[4] N. Dalgaard, C. Mosquera, and F. Pérez-González, "On the role of differentiation for resampling detection," in *IEEE International Conference on Image Processing (ICIP)*. IEEE, 2010, pp. 1753–1756.

[5] X. Feng, I. J. Cox, and G. Doerr, "Normalized energy density-based forensic detection of resampled images," *IEEE Transactions on Multimedia*, vol. 14, no. 3, pp. 536–545, 2012.

[6] B. Mahdian and S. Saic, "Blind authentication using periodic properties of interpolation," *IEEE Transactions on Information Forensics and Security*, vol. 3, no. 3, pp. 529–538, 2008.

[7] M. Kirchner and J. Fridrich, "On detection of median filtering in digital images," in *IS&T/SPIE Electronic Imaging*. International Society for Optics and Photonics, 2010, pp. 754 110–754 110.

[8] X. Kang, M. C. Stamm, A. Peng, and K. J. R. Liu, "Robust median filtering forensics using an autoregressive model," *IEEE Trans. Information Forensics and Security,*, vol. 8, no. 9, pp. 1456–1468, Sep. 2013.

[9] G. Cao, Y. Zhao, R. Ni, L. Yu, and H. Tian, "Forensic detection of median filtering in digital images," in *IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2010, pp. 89–94.

[10] C. Chen and J. Ni, "Median filtering detection using edge based prediction matrix," *Digital Forensics and Watermarking*, pp. 361–375, 2012.

[11] M. C. Stamm and K. J. R. Liu, "Forensic detection of image manipulation using statistical intrinsic fingerprints," *IEEE Trans. on Information Forensics and Security*, vol. 5, no. 3, pp. 492 –506, 2010.

[12] H. Yao, S. Wang, and X. Zhang, "Detect piecewise linear contrast enhancement and estimate parameters using spectral analysis of image histogram," in *International Communication Conference on Wireless Mobile and Computing*. IET, 2009.

[13] M. Stamm and K. R. Liu, "Blind forensics of contrast enhancement in digital images," in *IEEE International Conference on Image Processing*. IEEE, 2008, pp. 3112–3115.

[14] M. C. Stamm and K. R. Liu, "Forensic estimation and reconstruction of a contrast enhancement mapping," in *IEEE International Conference on Acoustics Speech and Signal Processing*. IEEE, 2010, pp. 1698–1701.

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

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

[17] R. Neelamani, R. De Queiroz, Z. Fan, S. Dash, and R. G. Baraniuk, "Jpeg compression history estimation for color images," *IEEE Transactions on Image Processing*, vol. 15, no. 6, pp. 1365–1378, 2006.

[18] Z. Qu, W. Luo, and J. Huang, "A convolutive mixing model for shifted double jpeg compression with application to passive image authentication," in *IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2008, pp. 1661–1664.

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

[20] X. Qiu, H. Li, W. Luo, and J. Huang, "A universal image forensic strategy based on steganalytic model," in *Workshop on Information hiding and multimedia security*. ACM, 2014, pp. 165–170.

[21] T. Pevny, P. Bas, and J. Fridrich, "Steganalysis by subtractive pixel adjacency matrix," *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 2, pp. 215–224, Jun. 2010.

[22] W. Fan, K. Wang, and F. Cayre, "General-purpose image forensics using patch likelihood under image statistical models," in *IEEE International Workshop on Information Forensics and Security*, Nov. 2015, pp. 1–6.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[24] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on audio, speech, and language processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

[25] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

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

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

[28] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[29] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[30] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[31] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[32] M. Kirchner and T. Gloe, "On resampling detection in re-compressed images," in *2009 First IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, 2009, pp. 21–25.

[33] J. Chen, X. Kang, Y. Liu, and Z. J. Wang, "Median filtering forensics based on convolutional neural networks," *IEEE Signal Processing Letters*, vol. 22, no. 11, pp. 1849–1853, Nov. 2015.

[34] B. Bayar and M. C. Stamm, "On the robustness of constrained convolutional neural networks to jpeg post-compression for image resampling detection," in *The 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 2152–2156.

[35] B. Bayar and M. C. Stamm, "Towards order of processing operations detection in jpeg-compressed images with convolutional neural networks," in *International Symposium on Electronic Imaging: Media Watermarking, Security, and Forensics*. IS&T, 2018.

[36] B. Bayar and M. C. Stamm, "A generic approach towards image manipulation parameter estimation using convolutional neural networks," in *Proceedings of the 5th ACM Workshop on Information Hiding and Multimedia Security*. ACM, 2017.

[37] B. Bayar and M. C. Stamm, "Design principles of convolutional neural networks for multimedia forensics," in *International Symposium on Electronic Imaging*. IS&T, 2017.

[38] 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, 2016.

[39] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition." in *ICML*, 2014, pp. 647–655.

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

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[42] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[43] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[44] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

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

[46] 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, 2010.

[47] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis." in *ICDAR*, vol. 3, 2003, pp. 958–962.

[48] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 437–478.

Belhassen Bayar (S'11) received the B.S. degree in Electrical Engineering from the Ecole Nationale d'Ingénieurs de Tunis (ENIT), Tunisia, in 2011, and the M.S. degree in Electrical and Computer Engineering from Rowan University, New Jersey, in 2014. After graduating from ENIT, he worked as a Research Assistant at the University of Arkansas at Little Rock (UALR). In Fall 2014, he joined Drexel University, Pennsylvania, where he is currently a Ph.D. candidate with the Department of Electrical and Computer Engineering. Bayar won the Best Paper Award at the IEEE International Workshop on Genomic Signal Processing and Statistics in 2013. In summer 2015 he interned at Samsung Reasearch America in Mountain View, California. His main research interests are in image forensics, machine learning and signal processing.

Matthew C. Stamm (S'08–M'12) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of Maryland at College Park, College Park, MD, USA, in 2004, 2011, and 2012, respectively.

Since 2013 he has been an Assistant Professor with the Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA, USA. He leads the Multimedia and Information Security Lab (MISL) where and his team conduct research on signal processing, machine learning, and information security with a focus on multimedia forensics and anti-forensics.

Dr. Stamm is the recipient of a 2016 NSF CAREER Award and the 2017 Drexel University College of Engineering's Outstanding Early-Career Research Achievement Award. He was the General Chair of the 2017 ACM Workshop on Information Hiding and Multimedia Security and is the lead organizer of the 2018 IEEE Signal Processing Cup competition. He currently serves as a member of the IEEE SPS Technical Committee on Information Forensics and Security and as a member of the editorial board of IEEE SigPort. For his doctoral dissertation research, Dr. Stamm was named the winner of the Dean's Doctoral Research Award from the A. James Clark School of Engineering. While at the University of Maryland, he was also the recipient of the Ann G. Wylie Dissertation Fellowship and a Future Faculty Fellowship. Prior to beginning his graduate studies, he worked as an engineer at the Johns Hopkins University Applied Physics Lab.