A Deep Learning Approach To Universal Image Manipulation Detection Using A New Convolutional Layer

Belhassen Bayar
Drexel University
Dept. of ECE
Philadelphia, PA, USA
bb632@drexel.edu

Matthew C. Stamm
Drexel University
Dept. of ECE
Philadelphia, PA, USA
mstamm@coe.drexel.edu

ABSTRACT

When creating a forgery, a forger can modify an image using many different image editing operations. Since a forensic examiner must test for each of these, significant interest has arisen in the development of universal forensic algorithms capable of detecting many different image editing operations and manipulations. In this paper, we propose a universal forensic approach to performing manipulation detection using deep learning. Specifically, we propose a new convolutional network architecture capable of automatically learning manipulation detection features directly from training data. In their current form, convolutional neural networks will learn features that capture an image's content as opposed to manipulation detection features. To overcome this issue, we develop a new form of convolutional layer that is specifically designed to suppress an image's content and adaptively learn manipulation detection features. Through a series of experiments, we demonstrate that our proposed approach can automatically learn how to detect multiple image manipulations without relying on pre-selected features or any preprocessing. The results of these experiments show that our proposed approach can automatically detect several different manipulations with an average accuracy of 99.10%.

CCS Concepts

•**Computing methodologies** → **Image processing;**

Keywords

Image forensics; Universal forgery detection; Convolutional neural networks

1. INTRODUCTION

Over the past several years, researchers have developed a variety of information forensic techniques to determine the authenticity and processing history of digital images [20]. Much of this research has focused on identifying traces left in an image by specific editing operations, then developing algorithms designed to detect these traces. This approach has been used to develop algorithms targeted at detecting image manipulations such as resizing and resampling [17, 8], median filtering [9, 7], contrast enhancement [19], etc.

While the development of targeted editing detectors has led to many important advances in information forensics, this approach to image authentication suffers an important drawback: Since a forger has many editing operations at their disposal, a forensic investigator must apply a large number of forensic tests to determine if and how an image has been edited. If multiple forensic tests are run on an image, the investigator must address several new problems such as controlling the overall false alarm rate between multiple tests and dealing with conflicting results. Furthermore, as new image editing operations are developed, researchers must identify traces left by these new operations and design associated detection algorithms.

In response to these issues, there has been significant interest in the development of universal forensic algorithms designed to detect many, if not all, editing operations. Recent experimental evidence has shown that tools initially developed to perform steganalysis are capable of detecting a wide variety of image editing operations [18]. These tools from steganalysis operate by building local models of pixel dependencies by analyzing the joint distribution of pixel value prediction errors, then extracting detection features from these joint distributions [16, 4]. Another recent effort towards developing universal forensic detectors operates by building Gaussian mixture models (GMMs) of image patches in unaltered and manipulated images [3]. A series of binary manipulation detectors for several editing operations are used then by comparing the log-likelihood of an image patch under the GMM for different possible manipulations. While these techniques show great promise, they each learn detection features from pre-selected models. As a result, a natural question remains: Can strong universal detection features be discovered without requiring human analysis or imposing a predetermined model on the data?

In this work, we propose a new universal approach for performing image editing detection that is capable of *automatically learning* traces left by editing. To accomplish this, we make use of tools from deep learning known as convolutional neural networks (CNNs). CNNs have recently fueled dramatic advances in image recognition due to their ability to adaptively learn classification features rather than rely on human-selected features [12]. These features are extracted from an image via a set of convolutional filters whose coefficients are learned using a technique known as back-

IH&MMSec 2016, June 20-23, 2016, Vigo, Spain
© 2016 ACM. ISBN 978-1-4503-4290-2/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2909827.2930786

propagation, then aggregated using an operation known as pooling. Though CNNs are able to adaptively learn good features for object recognition, they are not well suited for performing image manipulation detection in their existing form. Instead of learning filters that identify traces left by editing and manipulation, the convolutional layers will extract features that capture an image's content.

In this paper, we propose a new form of convolutional layer designed to suppress an image's content and adaptively learn manipulation detection features. Using this new convolutional layer, we propose a CNN architecture capable of automatically learning how to detect multiple image manipulations without relying on pre-selected features or models. Through a series of experiments, we evaluate our CNN's ability to act as a universal image manipulation detector. The results of these experiments show that our proposed approach can automatically detect several different manipulations with an average accuracy of 99.10%.

2. BACKGROUND

In this section, we give a brief overview of CNNs. A CNN is a special type of multi-layer neural network used in deep learning that has recently gained significant attention in the computer vision and machine learning communities [10, 21]. Convolutional neural networks first appeared in the late 1980's with the handwritten zip code recognition [13] as an extended version of artificial neural networks (ANN). They have been also applied to handwritten digit recognition[11], images, speech and time series data [12]. Instead of relying on hand-designed features, CNNs are able to adaptively learn classification features. A deep learning to constructing CNNs, i.e., stacking many hidden layers on top of one another, has recently proved very effective for problems such as object recognition [10]. These recent advances have been fueled by the use of GPUs to overcome the computational expense of estimating the large number of hyper-parameters that a deep network involves.

While the particular design or "architecture" of CNNs may vary, they are built using a common set of basic elements and share a similar overall structure. The first layer is a convolutional layer, comprising several convolutional filters applied to the image in parallel. These filters act as a set of feature extractors−their outputs are known as feature maps.

In this paper, matrices are denoted by bold letters, e.g., $\boldsymbol{h}$, $\boldsymbol{w}$; and scalars by regular letters. More specifically, $\boldsymbol{w}(i,j)$ denotes the $(i,j)^{th}$ entry in the matrix $\boldsymbol{w}$ and $\boldsymbol{w}_{ij}$ denotes the $(i,j)^{th}$ matrix in a set of matrices. The superscript $\cdot^{(n)}$ denotes the layer order in the network.

More specifically, the analytical expression of the convolution 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 $\boldsymbol{h}_j^{(n)}$ is the j^{th} feature map output in the hidden layer $h^{(n)}$, $\boldsymbol{h}_k^{(n-1)}$ is the k^{th} channel in the hidden layer $h^{(n-1)}$, $\boldsymbol{w}_{kj}^{(n)}$ is the k^{th} channel in the j^{th} filter in $h^{(n)}$ and $b_j^{(n)}$ is its corresponding bias term. The convolutions and associations of these feature maps throughout layers strengthen the learning ability of a CNN model to predict classes.

Though initially seeded with random values, the filter coefficients are learned via a process known as back-propagation algorithm which we explain in details later. A convolutional layer is typically followed by a pooling layer whose purpose is to reduce the dimensionality of the feature maps. This reduces the computational cost associated with training the network and decreases the chances of over-fitting. Pooling layers operate by dividing feature maps into small, possibly overlapping windows, then retaining only single value per window. Two of the most popular forms of pooling are max-pooling and mean-pooling which retain the maximum and mean value of each window respectively.

Most CNN architectures are built by stacking several convolutional layers and pooling layers on top of one another. This enables the CNN to learn a set of low-level features in early layers, then hierarchically group them into high-level features in later layers. After this, the final set of feature maps are passed to a set of fully connected layers that perform the classification. As in a traditional neural network, every neuron in a fully connected layer is connected to each of the outputs of the previous layer. Multiple fully connected layers can be stacked on top of one another to form deep architectures. The final (visible) fully connected layer of neurons is trained to produce class scores for each class. If sigmoids are used as activation functions for each neuron in this layer, the class scores can be interpreted as class probabilities.

During training the coefficients of the convolutional filters $\boldsymbol{w}_{ij}$ are automatically learned using an iterative algorithm which alternates between feedforward and backpropagation passes of the data. The ultimate goal of this algorithm is to minimize the average loss between the actual labels 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.

To this aim, a variety of solvers could be used to solve the underlying optimization problem. In this paper we consider the stochastic gradient descent (SGD) to train our model. The iterative update rule for the kernel coefficients $\boldsymbol{w}_{ij}^{(n)}$ in CNN during the backpropagation pass is given below:

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

where $\boldsymbol{w}_{ij}^{(n)}$ represents the i^{th} channel from the j^{th} kernel matrix in the hidden layer $h^{(n)}$that convolves with the i^{th} channel in the previous feature maps denoted by $h_i^{(n-1)}$, $\Delta\boldsymbol{w}_{ij}^{(n)}$ denotes the gradient of $\boldsymbol{w}_{ij}^{(n)}$ and ϵ is the learning rate. The letters m and d are respectively the momentum and the decay. The bias term $b_j^{(n)}$ in (1) is updated using the same equations presented in (3). The use of the *decay* and *momentum* strategy is mainly for fast convergence as explained by LeCun *et al.* in [14].

3. NEW CONVOLUTIONAL LAYER

Though CNNs are able to adaptively learn strong classification features for object recognition, they are ill suited for

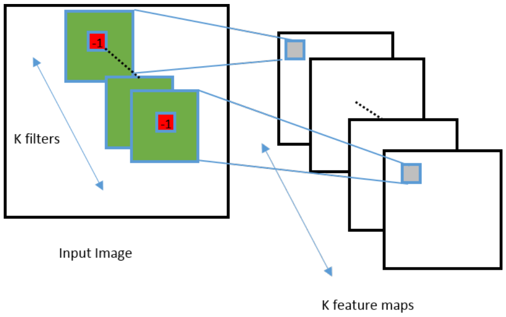

Figure 1: Our proposed convolutional layer. The red coefficient is -1 and the coefficients in the green region sum to 1.

performing manipulation detection in their standard form. This is because in their existing form, CNNs will tend to learn features that represent an image's content rather than manipulation detection features. This effect has recently been observed by Chen et al. during their efforts train a CNN to perform median filtering detection [2]. In their experiments, Chen et al. found that the CNN was not able to learn median filtering detection features if images are directly fed to the input layer. Instead, they first extracted a high dimensional feature set from the image known as the median filter residual, then provided this to the input layer of the CNN.

To overcome this problem, we propose a new form of convolutional layer that will force the CNN to learn manipulation detection features from images without requiring any preliminary feature extraction or pre-processing. The key idea behind developing this layer is that certain local structural relationships exist between pixels independent of an image's content. Manipulations will alter these local relationships in a detectable way. As a result, manipulation detection feature extractors must learn the relationship between a pixel and its local neighborhood while simultaneously suppressing the content of the image so that content dependent features are not learned. For this to occur, the first convolutional layer must not be allowed freely evolve into any set of filters. Instead, it must be constrained to evolve filters with the desired properties described above.

To accomplish this, we propose creating the first layer of our CNN using convolutional filters that are constrained to learn only a set of prediction error filters. Prediction error filters are filters that predict the pixel value at the center of the filter window, then subtract this central value to produce the prediction error. More explicitly, each of the K filters $\boldsymbol{w}_k^{(1)}$ in the first layer of the CNN have the following constraints placed on them:

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

where $\boldsymbol{w}_k^{(1)}(\ell,m)$ is the filter weight at the (ℓ,m) position and $\boldsymbol{w}_k^{(1)}(0,0)$ is the filter weight at the center of the filter window. Each filter in this layer is initialized by randomly choosing each filter weight, then enforcing the constraints in (4). During training, the constraints in (4) are again enforced during each iteration after the filter weights have undergone their stochastic gradient descent updates. This allows the CNN to adaptively learn a strong set of manipulation detection feature extractors, rather than having the chosen a priori.

Pseudocode summarizing the training algorithm for our new layer is shown below:

Algorithm 1 Training algorithm for our new convolutional layer

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

We note that our proposed constrained convolutional layer takes inspiration from a wide array of previous information forensic and steganographic research. Many forensic and steganalysis algorithms can be viewed as specific forms of the following detection approach: Predict each pixel value on the basis of its neighbors according to a fixed rule, calculate the prediction error, create a lower dimensional feature vector or test statistic from these prediction errors, then make a decision on the basis of this feature vector or test statistic. This approach has also been recently applied to camera model identification [1]. It is quite easy to see that steganalysis algorithms such as rich models [4] and SPAM [16] are very successful instances of this approach. Furthermore, forensic algorithms for detecting several manipulations such as resizing (using linear predictor residues [8]) and median filtering (using streaking artifacts [9] or median filter residuals [7]) can also be viewed as specific forms of this approach.

While each of these algorithms discussed above rely on a fixed predictor or set of predictors chosen a priori, our proposed constrained convolutional layer enables a set of predictors to be *learned directly from the training data.* Furthermore, the higher layers of our CNN (described in Section 4) are able to *learn* the appropriate method for extracting low dimensional detection features from the high dimensional prediction errors. As a result, our proposed universal forensic approach does not require analysis by a human expert to create a detector for a new manipulation. This is particularly important because the design of detection features and an appropriate detection rule by a human expert is both time consuming and difficult.

4. NETWORK ARCHITECTURE

In this section, we present our proposed CNN architecture for performing manipulation detection. Fig. 2 shows our proposed CNN architecture as well as detailed information about the size of each layer. As depicted in Fig. 2, our network contains 8 layers, namely our proposed new convolutional layer, 2 convolutional layers, 2 max-pooling layers and 3 fully-connected layers. Images are fed into the CNN

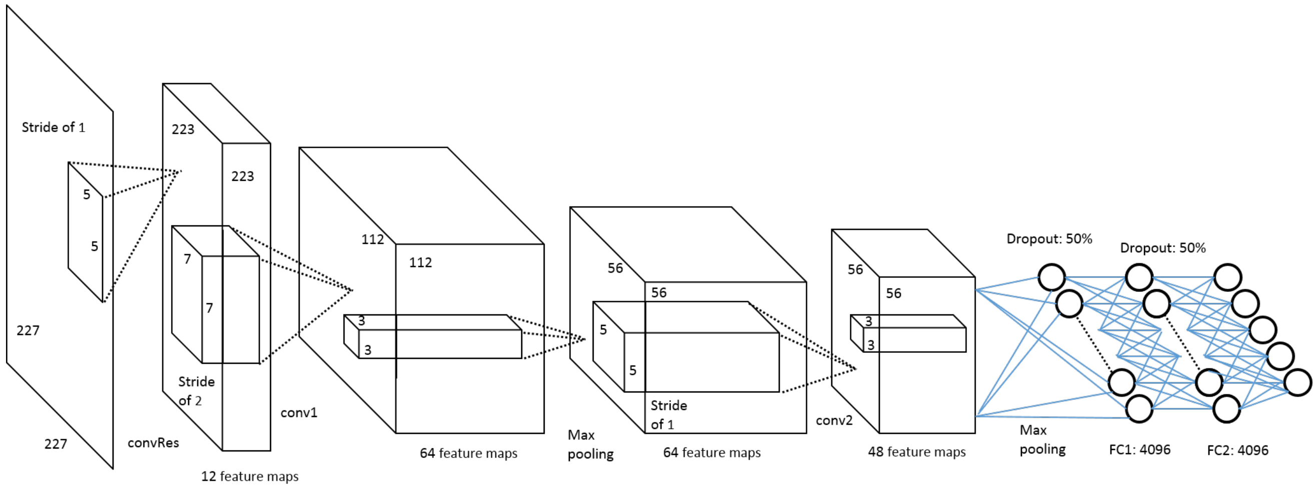

Figure 2: An illustration of the proposed CNN Architecture. The network's input dimension is 51529 neurons and the remaining 8 layers have 596748, 802816, 200704, 150528, 37632, 4096, 4096 and 5 neurons respectively

through an input layer, also known as the data layer. The first layer of our network is 227 × 227 grayscale image.

4.1 Convolutional Layer

In our architecture, we use two types of convolutional layers, i.e., regular and our proposed type of convolution. Throughout the regular convolutional layers, inputs are called images. Specifically, an input image of dimension $w \times l \times c$ from a hidden layer $h^{(n-1)}$ is convolved with k different kernels of dimension $s \times s \times c$ where w and l are respectively, the width and the height of the input image, c is the number of feature maps in the hidden layer $h^{(n-1)}$ and s is the filter size. The number of feature maps in the input image of the hidden layer $h^{(n)}$ is k. The convolution is applied to all the $s \times s$ local regions of the image, also called receptive fields, with an overlapping distance called stride.

In our model, we have 2 non-constrained convolutional layers respectively called conv1 and conv2. As can be seen in Fig. 2, conv1 has 64 kernels that we depict only one of size $7 \times 7 \times 12$ with stride of 2 which yields $112 \times 112 \times 64$ feature maps. conv2 has 48 kernels of size $5 \times 5 \times 64$ with stride of 1 which yields $56 \times 56 \times 48$ feature maps

A convolutional layer is commonly followed by a nonlinear mapping applied in an activation layer. An activation layer is simply a nonlinear function applied to each pixel value. In our work we use the Rectified Linear Unit (ReLU), i.e., $f(x) = \max(0, x)$ [15]. Krizhevsky *et al.* showed that in practice, CNNs with ReLU neural activations train several times faster than other activation functions [10].

The first layer in our CNN is our proposed constrained layer discussed in Section 3. We refer to this layer as convRes (see Figs. 1 and 2). Specifically, we have a set of 5×5 constrained prediction error filters with stride of 1. From Fig. 2, we use 12 kernels whose outputs are 12 223×223 feature maps. These latter convolutional outputs are considered as a new image with 12 channels. Therefore, the next convolutional kernel in conv1 has 12 channels. This type of convolution is not followed by a ReLU mapping mainly because the output feature maps carry the fingerprints left by a tampering operation which can be destroyed by the nonlinear operator.

4.2 Max-Pooling

We use the overlapping max-pooling layer similarly to [10] which is a subsampling approach. The goal of pooling layer is to reduce the resolution of the feature map and make it robust to variations for previous learned features. Explicitly, this method consists of computing the maximum value in each neighborhood at different positions. We use a kernel size of 3 and a stride of 2. We can see from Fig. 2 that the size of the feature map in conv1 is reduced from $112 \times 112 \times 64$ to $56 \times 56 \times 64$ after the first max-pooling layer. The output of the second max-pooling layer is a set of $37,632$ neurons, i.e. feature maps of size $28 \times 28 \times 48$, reduced from the previous feature maps of size $56 \times 56 \times 48$.

Furthermore, the max-pooling layers are followed by a Local Response Normalization (LRN) similarly applied in [10] where the central value in each neighborhood is normalized by the surrounding pixel values. This type of operation is also called "brightness normalization".

4.3 Dropout & Fully-Connected Layers

The dropout technique [5] applied in the fully-connected layers fc1 and fc2 consists of setting to zero the neurons of the hidden layer with probability 0.5. It reduces the complex co-adaptations of neurons and forces them to learn more robust features. From Fig. 2 we have 4096 neurons in fc1 and fc2. Therefore, only 2048 neurons contribute in the forward pass and the backpropagation.

Finally, the output layer has one neuron corresponding to each possible class, i.e. one neuron for unaltered images as well as a neuron for each possible manipulation. In the experiments conducted in this paper, we considered 4 different manipulations, therefore out output layer has 5 neurons. A softmax operator is applied in fc3 to scale the output values as the probabilities of an example to belong to each class.

5. EXPERIMENTAL RESULTS

5.1 Experimental Setup

To evaluate the performance of our proposed CNN model for image editing detection, we first built an experimental

Table 1: CNN detection accuracy rate for binary detectors

	Median Filtering	Gaussian Blurring	AWGN, $\sigma = 2$	Re-sampling
Accuracy	99.31%	99.32%	99.68%	99.40%

Table 2: Confusion matrix showing the detection accuracy of our multiclass CNN

	Original	Median Filtering	Gaussian Blurring	AWGN, $\sigma = 2$	Re-sampling
Original	**98.40%**	0.52%	0.29%	0.34%	0.44%
Median Filtering	0.23%	**98.27%**	1.24%	0.12%	0.12%
Gaussian Blurring	0.00%	0.18%	**99.75%**	0.00%	0.06%
AWGN, $\sigma = 2$	0.03%	0.04%	0.14%	**99.77%**	0.00%
Resampling	0.27%	0.20%	0.15%	0.00%	**99.35%**

database of unaltered and edited images. Our experimental image datasets have been collected from 12 different camera models and devices with no previous tampering or preprocessing. We created a set of grayscale images by retaining only the green color layer from each image. We cropped each original image in the center, then subdivided it into 256×256 blocks. More explicitly, every block corresponds to a new image that has its corresponding different tampered images. In total we created a set of $261,800$ unaltered blocks. Next, we generated a set of altered images. We did this by applying the following operations to a set of unaltered image:

- Median Filtering with a 5×5 kernel.
- Gaussian Blurring with a 5×5 kernel and a standard deviation $\sigma = 1.1$.
- Additive White Gaussian Noise (AWGN) with standard deviation 2.
- Resampling (resizing) using bilinear interpolation and a scaling factor 1.5.

We then cropped these images into 256 by 256 blocks to create a total of $333,200$ manipulated blocks. During training and testing all the blocks are further cropped to 227 by 227 blocks.

We implemented all of our CNNs using the Caffe deep learning framework [6]. We ran our experiments using one Nvidia GeForce GTX 980 GPU with 4GB RAM. To facilitate this, we converted our datasets to the lmdb format. We set the training parameters of the stochastic gradient descent as follows: $momentum = 0.9$, $decay = 0.0005$, and a fixed learning rate $\epsilon = 10^{-6}$ over all iterations. The choice of the learning rate ϵ is very crucial for both accuracy and the stability of the weights gradient. A larger learning rate would yield numerically unstable filters weights. We set the batch size for training and testing to 32 images.

5.2 Results

In what follows, we use our suggested CNN model as a binary and multi-class classifier and we present our simulation results.

5.2.1 *Binary Classification Approach*

In our first set of experiments, we trained different CNNs to detect each of the four manipulations discussed in Section 5.1. Each CNN corresponds to a binary classifier that detects one type of possible image operation with the same architecture outlined in Section 4. The output layer corresponds to two neurons, i.e., original v.s. tampered image. Decision made by picking the class corresponding to the neuron with the highest activation. We chose $43,500$ unaltered blocks and their corresponding tampered blocks to build our training data for each type of forgery. Similarly we picked $16,000$ unaltered blocks and their corresponding tampered blocks to build our testing data for each type of forgery. That is, for every binary classifier we have a total number of $87,000$ blocks for training and $32,000$ blocks for testing.

Table 1 summarizes the performance of our proposed model for binary classification to detect the underlying image operations. We can see from this table that our CNNs are able to distinguish between unaltered and manipulated images with at least 99.31% accuracy. We also note that we chose to stop the training process after achieving an accuracy higher than 99% since it increases slowly above that rate. Therefore, these results represent a lower-bound accuracy of what our model can achieve.

Our model converges to a very high accuracy after few thousands of iterations. Furthermore, we note that on our machines this typically takes less than one hour.

5.2.2 *Multi-class Classification Approach*

In our second experiment we trained a multiclass CNN to detect multiple types of image forgery, i.e., median filtering, guassian blurring, additive white gaussian noise and re-sampling v.s. authentic image. Following the first set of experiments, a decision is made by picking the class corresponding to the neuron with the highest activation. Given the memory constraints of our machines, we picked $17,400$ unaltered blocks and their four corresponding tampered blocks to build our training data. Similarly we picked $6,400$ unaltered blocks and their four corresponding tampered blocks to build our testing data. That is, we have a total number of $87,000$ blocks for training and $32,000$ blocks for testing.

The CNN was trained over $56,000$ iterations and then fine-tuned for $9,000$ iterations with fixed filters in all the convolutional layers. Since CNNs have many hyper-parameters that must be learned throughout all layers, this constraint during fine-tuning helps to direct the neurons in the fully connected layers toward more optimal weights. That is, we need to direct the gradient direction to a better minima by fine-tuning only the fully-connected layers. This procedure has increased the accuracy of our model by $\approx 1\%$. The entire training typically converges in less than 6 hours.

Our simulation results are summarized in Table 2. Our proposed model achieves an accuracy of 99.10% of detecting the different four types of forgery. From this confusion matrix, we can see that our CNN can detect each manipulation with very high accuracy.

These results are significant for several reasons. First, they show that our CNN represents a universal manipulations detection approach since it can be trained to detect multiple manipulations without altering its architecture. Second, and perhaps most surprisingly, our CNN can be trained to automatically learn detection features for each manipulation without requiring human intervention. This suggests that as new manipulations are considered or developed, our CNN can potentially learn to detect them without needing a human expert to identify detection features.

6. CONCLUSION

In this paper, we proposed a novel CNN-based universal forgery detection technique that can automatically learn how to detect different image manipulations. To prevent the CNN from learning features that represent an image's content, we proposed a new form of convolutional specifically designed to suppress an image's content and learn manipulation detection features. We accomplished this by specifically constraining this new convolutional layer to learn prediction error filters. Through a series of experiments, we demonstrated that our CNN-based universal forensic approach can automatically learn how to detect multiple image manipulations without relying on pre-selected features or any preprocessing. The results of these experiments demonstrated that our proposed approach can automatically detect several different manipulations with an average accuracy of 99.10%.

7. REFERENCES

[1] C. Chen and M. C. Stamm. Camera model identification framework using an ensemble of demosaicing features. In *Information Forensics and Security (WIFS), 2015 IEEE International Workshop on*, pages 1–6. IEEE, 2015.

[2] J. Chen, X. Kang, Y. Liu, and Z. J. Wang. Median filtering forensics based on convolutional neural networks. *IEEE Signal Processing Letters*, 22(11):1849–1853, Nov. 2015.

[3] 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 (WIFS)*, pages 1–6, Nov. 2015.

[4] J. Fridrich and J. Kodovskỳ. Rich models for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security*, 7(3):868–882, 2012.

[5] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[6] 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.

[7] X. Kang, M. C. Stamm, A. Peng, and K. J. R. Liu. Robust median filtering forensics using an autoregressive model. *IEEE Transactions on Information Forensics and Security,*, 8(9):1456–1468, Sept. 2013.

[8] 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*, MM&Sec '08, pages 11–20, New York, NY, USA, 2008. ACM.

[9] M. Kirchner and J. Fridrich. On detection of median filtering in digital images. In *IS&T/SPIE Electronic Imaging*, pages 754110–754110. International Society for Optics and Photonics, 2010.

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

[11] B. B. Le Cun, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. Citeseer, 1990.

[12] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[13] 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*, 1(4):541–551, 1989.

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

[15] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, pages 807–814, 2010.

[16] T. Pevny, P. Bas, and J. Fridrich. Steganalysis by subtractive pixel adjacency matrix. *IEEE Transactions on Information Forensics and Security*, 5(2):215–224, June 2010.

[17] A. C. Popescu and H. Farid. Exposing digital forgeries by detecting traces of resampling. *IEEE Transactions on Signal Processing*, 53(2):758–767, Feb. 2005.

[18] X. Qiu, H. Li, W. Luo, and J. Huang. A universal image forensic strategy based on steganalytic model. In *Proceedings of the 2nd ACM workshop on Information hiding and multimedia security*, pages 165–170. ACM, 2014.

[19] M. C. Stamm and K. J. R. Liu. Forensic detection of image manipulation using statistical intrinsic fingerprints. *IEEE Trans. on Information Forensics and Security*, 5(3):492 –506, 2010.

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

[21] 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*, pages 1–9, 2015.