Image Copy-Move Forgery Detection via an End-to-End Deep Neural Network

Yue Wu, Wael Abd-Almageed, and Prem Natarajan
Information Sciences Institute, University of Southern California
4676 Admiralty Way #1001, Marina del Rey, CA 90292, USA
{yue_wu,wamageed,pnataraj}@isi.edu

Abstract

In this paper, for the first time, we introduce a new end-to-end deep neural network predicting forgery masks to the image copy-move forgery detection problem. Specifically, we use a convolutional neural network to extract block-like features from an image, compute self-correlations between different blocks, use a pointwise feature extractor to locate matching points, and reconstruct a forgery mask through a deconvolutional network. Unlike classic solutions requiring multiple stages of training and parameter tuning, ranging from feature extraction to postprocessing, the proposed solution is fully trainable and can be jointly optimized for the forgery mask reconstruction loss. Our experimental results demonstrate that the proposed method achieves better forgery detection performance than classic approaches relying on different features and matching schemes, and it is more robust against various known attacks like affine transformation, JPEG compression, blurring, etc.

1. Introduction

Due to the fast growth of social networks and advances in image editing tools, it is much easier to manipulate an image in a more realistic way—even for a non-professional photo-editing user. Although the purpose of many user manipulations is not malicious, e.g., beautifying one's selfie or removing 'red eye' before uploading the image to facebook, image forgery has become a more serious and pervasive problem in recent years. Indeed, on 22 April 2016, *Nature News* reported *"Problematic images found in 4% of biomedical papers,"* and many papers are retracted because of image forgery. For example, [20] published in *Plant* magazine was retracted, because *"it contains a substantial number of inappropriate image manipulations."* [11] published in the *Cancer Research Journal* was retracted because its "*Fig. 4C are duplicates of the M329-actin bands in Fig. 4A*."

Image copy-move forgery is probably one of the most common image forgeries because it only involves one single image, and it can be easily achieved by using more than a dozen different photo-editing tools, e.g., Adobe Photoshop and GNU Image Manipulation Program (GIMP). Depending on the purpose of image copy-move, we can roughly classify it into two categories: 1) evidence duplication, and 2) evidence removal. Evidence duplication is used to copy and paste a whole evidence object one or more times, such that a resulting image shows more evidence than it actually has. For example, given an image of a birthday party with a single balloon, you may paste the balloon many times. In contrast, evidence removal is used to remove an evidence object by overwriting the evidence object region with a region copied from something else. For example, given an image of a crime scene with a bullet on the floor, you may select some floor region and paste it to the bullet location, such that the original bullet become invisible in the resulting image. Fig. 1 illustrates image copy-move forgery samples using these two methods.

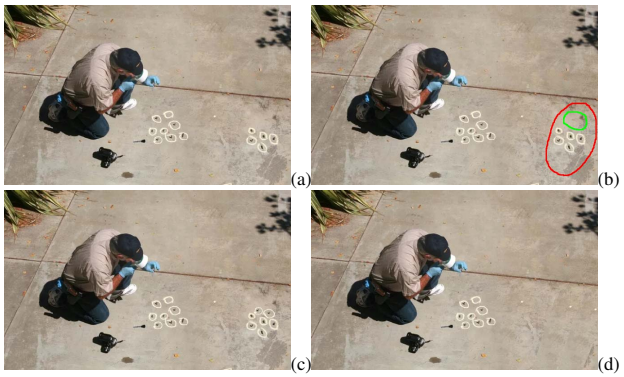

Figure 1. Image copy-move forgery samples: (a) original image without manipulation; (b) original image with highlighted image copy-move forgery regions; (c) *evidence duplication* example, where the duplicated region corresponds to the green circle region in (b); and (d) *evidence removal* example, where the removed region corresponds to the red circle region in (b).

Classic solutions of image copy-move forgery detection [16, 4, 22, 1, 7, 2, 18, 8, 17, 5, 12, 29, 10] often follow three main steps: 1) *unit feature representation*, 2) *unit-level matching*, and 3) *postprocessing* [8, 6]. Unit

978-1-5386-4886-5/18/$31.00 ©2018 IEEE
DOI 10.1109/WACV.2018.00211

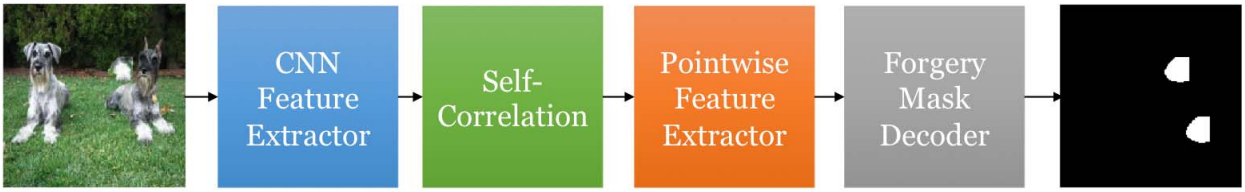

Figure 2. Overview of the proposed DNN-based image copy-move forgery approach.

feature representation expresses an input image in terms of descriptive features. This step contains two essential components—namely region unit (e.g., keypoints [7, 2, 12], image blocks, or other units like super-pixels [18]); and image feature (e.g., SIFT [1, 7, 18], Zernike moments [22, 12], PCA [17, 10] and many others [4, 5, 29]). Unit-level matching performs matching between different units. Although this step used to be done in one shot, recent studies show that iterative matching improves matching performance [8, 28]. Finally, postprocessing reduces false alarms by considering holistic matching between two or more unit groups. More details can be found in comprehensive reviews [25, 3] and reports [6].

Although the general idea of classic solutions is not hard to follow, it is annoying to implement an image copy-move forgery algorithm simply because this requires one to *cleverly* tune many heuristic but data-dependent parameters. For example, one may have to decide how many keypoints to extract for each image, or what the block size should be and the overlapping ratio between image blocks in the *unit feature representation* stage, or what the neighborhood size K is in the approximate K-nearest neighbor search [8] in *unit-level matching*. Moreover, one may also have to tune parameters in postprocessing. For example, in checking holistic matching, one may have to specify the minimum distance between matching candidates, minimum number of correspondences, area threshold, rules to split one unit group into two or to merge two groups into one, and threshold to fill a hole, etc. [6]. All of these parameters could be image-dependent, implying that one set of parameters working well for one image may fail on another. Of course, one could try different parameters and select the one that works well for each testing image, but this approach only works when the number of testing images is small.

Another salient drawback to the classic approaches [16, 4, 22, 1, 7, 2, 18, 8, 17, 5, 12, 29, 10] is that the three steps are often trained disjointedly instead of jointly. For example, none of SIFT, Zernike moments, or PCA features are designed for the image copy-move forgery detection purpose, but it is common to see people use these kinds of general-purpose features as a black-box in the image copy-move forgery detection problem without extra training. Although trainable classifiers are also used to automatically decide things like whether two features match, they are only trained according to their individual tasks instead of the end-to-end copy-move detection task. Deep neural networks (DNN) have been recently introduced to the image forgery detection community, but most works still focus on using them as a replacement for the classic feature extractors, e.g., [14, 19].

In this paper we introduce an end-to-end DNN solution for the image copy-move forgery detection problem. Specifically, we conceptually follow the three main steps widely used in the classic solutions, but implement them within a single end-to-end DNN. In this way, we not only avoid setting various heuristic parameters and thresholds, but also jointly train all modules w.r.t. the forgery mask reconstruction loss. To the best of our knowledge, this is the first end-to-end DNN solution that predicts forgery masks for the image copy-move forgery detection problem. The remainder of this paper is organized as follows: Section 2 describes our DNN solution, module by module, Section 3 discusses training data synthesis with implementation and training details, Section 4 discusses our experiment results, and we conclude our paper in Section 5.

2. Methodology

2.1. Overview

The overall flowchart of the proposed DNN approach is given in Fig. 2. Each of our processing modules is a set of standard or custom DNN layers, and thus when we cascade them together, our model is still end-to-end trainable. We now discuss the details of these modules. To simplify discussion, we ignore the batch dimension because it is a dynamic number that can be changed in training.

2.2. Convolutional Feature Extractor

The goal of the convolutional feature extractor is to extract block-like features for a given image. Compared to classic image-block or keypoint-based features which are originally designed for other image tasks, the resulting CNN feature is clearly better in the sense that it will automatically learn appropriate feature representations for the image copy-move detection problem from training data.

Of course, any CNN architecture can be used here, e.g., ResNet-101 [9] and GoogleNet [24]. For the sake of simplicity, we choose the most common VGG16 CNN feature

extractor [23]. Fig. 3 shows the detailed architecture, where we basically use the `convolution` and `max-pooling` layers in the first four blocks of the VGG16 model. As a result, given an input image X of size $256 \times 256 \times 3$, the VGG16 convolutional feature extractor outputs a feature tensor f_X of size $16 \times 16 \times 512$.

2.3. Self-Correlation

The goal of self-correlation is to compute pairwise similarity scores between any two feature pixels. Specifically, given $f_X(i,j)$ and $f_X(i',j')$, i.e., feature pixels located at (i,j) and (i',j'), we compute the their similarity as the normalized correlation coefficient, as shown in Eq. (1), where $\bar{f_X}$ is the normalized feature tensor w.r.t. its raw mean and standard deviation, as shown in Eq. (2).

$$\text{corr}(f_X(i,j), f_X(i',j')) = \frac{\bar{f_X}(i,j)^T \bar{f_X}(i,j)}{512} \quad (1)$$

$$\bar{f_X}(i,j) = \frac{f_X(i,j) - \mu_X(i,j)}{\sigma_X(i,j)} \quad (2)$$

It is worth noting that this self-correlation module can be efficiently implemented by any deep neural network library in two steps: 1) a custom activation function performing the standard normalization as shown in Eq. (2), and 2) a tensorwise dot product function which is often a low-level but built-in function, e.g., `batched_tensordot` in `Theano` and `matmul` in `TensorFlow`. As a result, the output tensor of self-correlation is four dimensions, where the former two dimensions represent (i,j), and the latter two represent (i',j').

2.4. Pointwise Feature Extractor

Traditionally, people look for matching pairs after computing pairwise feature similarity scores. Regardless of how to determine what feature pairs match each other, these traditional approaches often have two common drawbacks: 1) relying on handcrafted parameters to determine the neighbor size in retrieving candidate pairs, and 2) relying on heuristic thresholds/rules to determine matching or non-matching pairs [6].

Although a natural solution to avoiding these two drawbacks is to design a DNN module that mimics the process of determining matched pairs, we find that the proposed pointwise feature extractor shown in Fig. 4 is a more reasonable and better alternative. Deciding what are matched pairs requires analyzing a total of $256 \times 255/2 = 32,640$ unique pairs, where $256 = 16 \times 16$ is the number of feature pixels. However, determining what features are matched only requires 256 assessments. We therefore do not attempt to make hard decisions regarding which features match, but make soft decisions via pointwise features at different holistic levels, namely 1×1, 3×3 and 5×5.

The `Reshape` layer suppresses the latter two axes representing feature location at (i',j') into a single dimension, and treats it as a new feature dimension. The `Argsort` layer is introduced to standardize features such that the resulting feature does not depend on (i',j'), but only on the statistics of matching scores. The `BatchNormalization` layer helps capture the local statistics better, and the three `Convolutional` layers extract pointwise features at different holistic levels. Finally, the `Concatenate` layer forces later layers to consider holistic pointwise features as a whole instead of disjointly.

Fig. 5 shows the sample results of pointwise feature extraction. Each mega-row visualizes the eight filters of kernel sizes 1×1, 3×3 and 5×5, respectively. As one can see, the color of forgery regions is either very red or blue, indicating that forgery-related information is captured. Finally, all features are considered jointly to predict the forgery mask (see result in the lower-left corner of Fig. 5).

2.5. Forgery Mask Decoder

The goal of the forgery mask decoder is to use all pointwise matching information for the copy-move forgery mask prediction. As one may notice, the resolution of the pointwise feature (16×16) in the previous module is much smaller than that of the required forgery mask (256×256). We therefore upsample the pointwise feature and then reconstruct the mask. This process is known as image deconvolution [15]. Like the CNN feature extractor, this module can also be implemented by using existing network architectures, e.g., [15] and [27]. We choose to use the following architecture shown in Fig. 6. Specifically, we use the `BilinearUpsampling` layer to gradually increase resolution and the Google `Inception` module [24] to decode the forgery mask.

2.6. Discussion

It is worth pointing out the differences between the proposed method and the recent end-to-end DNN solution for image splicing detection [27]. First, our method is designed for the image copy-move forgery detection problem with a single image as input, while [27] takes a pair of images as input. Second, we use normalized Pearson correlation coefficients in computing feature-wise similarity scores, while [27] uses cross-correlation. Third, [27] tries to recover translation parameters between matching groups, while we are only interested in exploring pointwise features.

More importantly, [27] simply does not work for the image copy-move problem because it is common to see cases with more than one source region and/or multiple destination regions in the image copy-move forgery detection problem. This implies that the number of correct matching pairs can be large; e.g. there are $\binom{N+1}{2}$ matching pairs for the case of one source region copied and pasted N times. As a

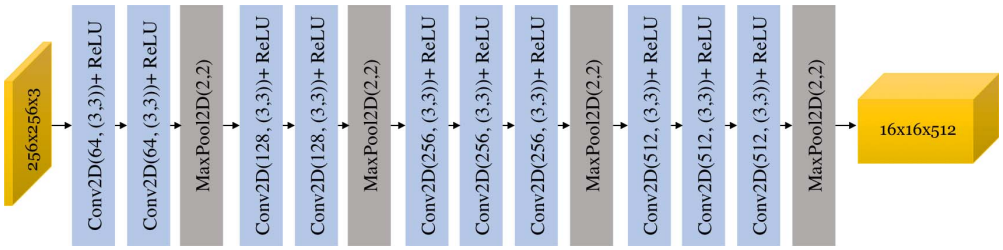

Figure 3. The network architecture of convolutional feature extractor.

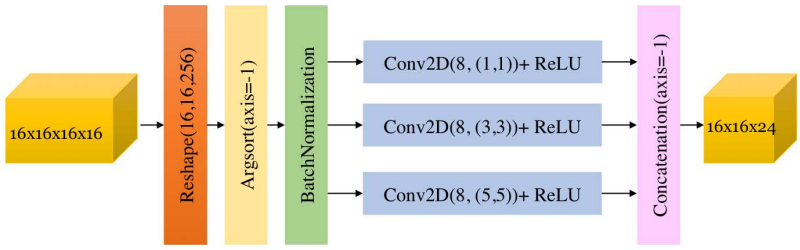

Figure 4. Network architecture of pointwise feature extractor.

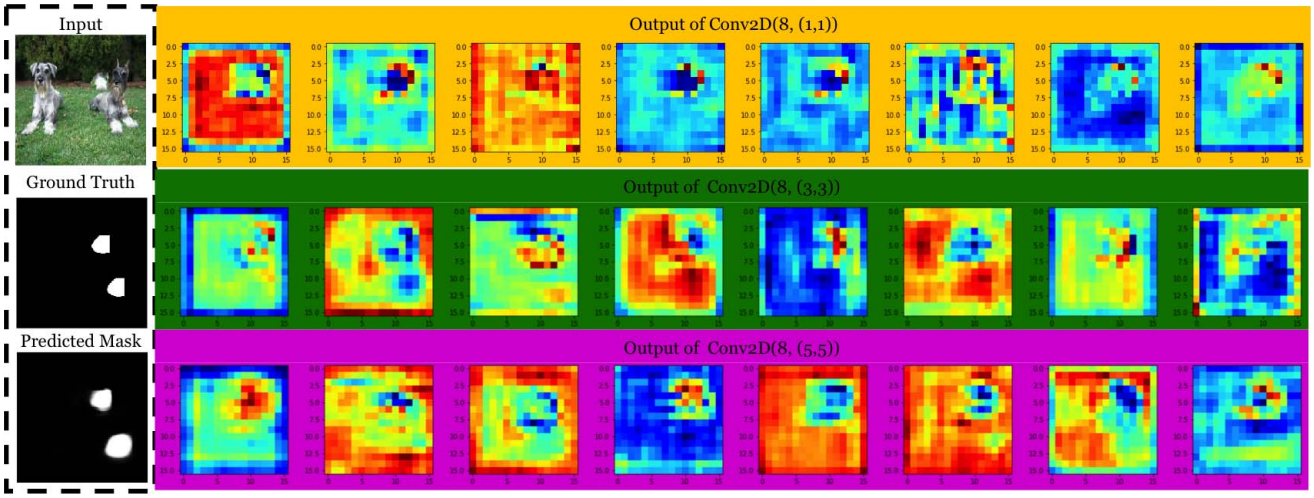

Figure 5. Sample output of pointwise feature extraction results. Each filter output is normalized to (0,1) before visualization using the `jet` color map (0:blue, 1:red).

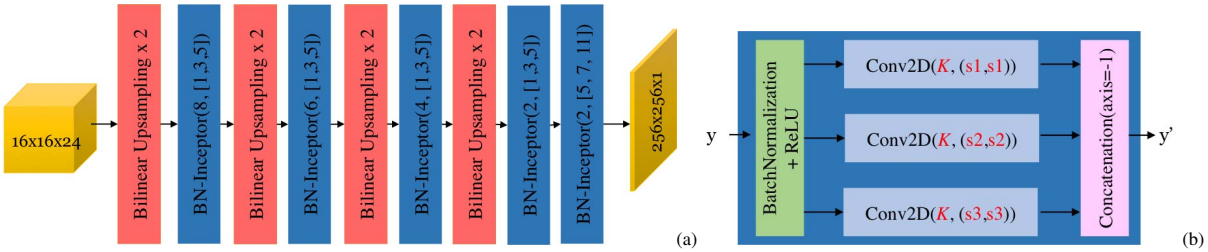

Figure 6. Network architecture of the forgery mask decoder (a) and batch-normalization-based inception module (b).

result, one may fail to recover all possible matching groups by pooling out a small number (which is 6 in [27]) of matching candidates. What makes the situation even worse is the existence of very high self-correlation scores in adjacent image blocks. [27] may fail to differentiate truly matched source and destination regions from self-similar patterns or homogeneous blocks in the image copy-move problem, because its pooling criteria is based on the overall matching responses. In contrast, we focus on pointwise features, and let the later forgery mask decoder module consider holistic matching between candidates.

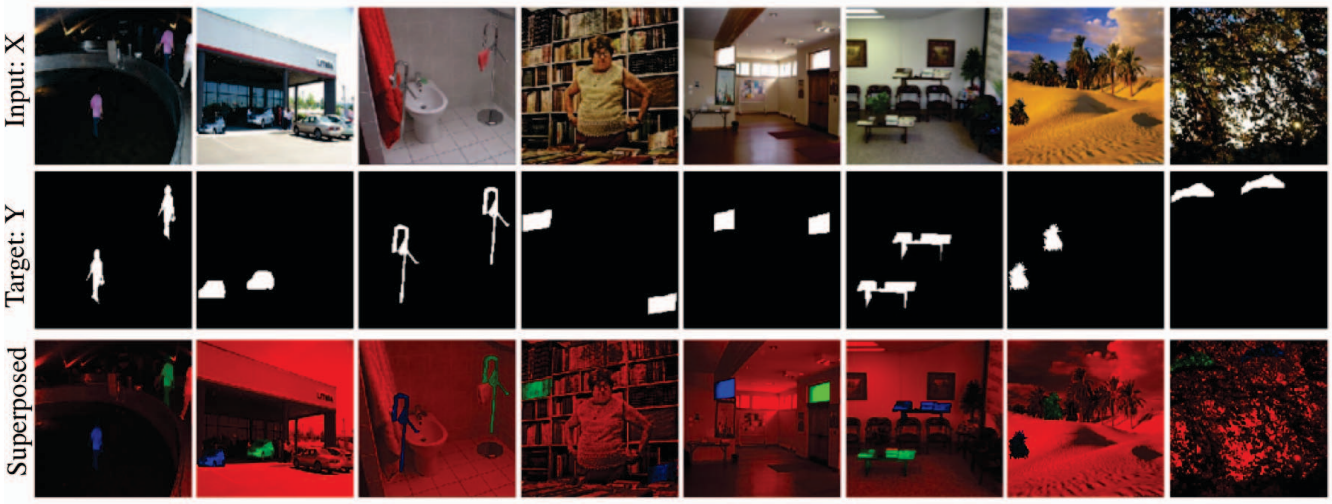

Figure 7. Synthesized samples used in training. Top row: network input (synthesized images); middle row: network target (synthesized masks); and bottom row: superposed images (green and blue colors highlight the source and destination copy-move regions).

3. Training Data and Strategy

To the best of our knowledge, no existing public dataset in the image forgery detection domain is large enough ($>$ 1 million) to train the proposed image copy-move forgery DNN model. However, it is not difficult to synthesize training data automatically, because all we need is to randomly select a source region in an image and paste its copy somewhere else in the image.

The SUN2012 object detection dataset [26] and the MS COCO dataset [13] provide instance-level polygon annotations, and thus we can easily synthesize a sample by selecting a source region via the given polygon annotation and randomly paste it back to the image. To further increase the number of training samples, we also use the ImageNet dataset [21]. Because the ImageNet dataset does not contain instance-level annotation, we synthesize samples by randomly selecting a region of irregular shape and paste it back the image. It is worth noting that we apply random affine transforms during data synthesis. Specifically, rotation angle and scale change are uniformly distributed in $\mathbb{U}(-10^o, 10^o)$ and $\mathbb{U}(0.75, 1.25)$. Translation change is also uniformly distributed over the entire image. A random affine transform is then tested by a *trial and error* algorithm to ensure that the source region and the destination region do not overlap. We also uniformly perturb one color channel's intensity by a random value in $\mathbb{U}(-8, 8)$. In this way, we prepare about 1.5 million training and 0.3 million validation samples offline. Fig. 7 shows eight randomly pulled samples. Note, we only train our model with synthetic data, but no external data.

With regard to model training, we implement the proposed image copy-move forgery DNN solution in the *Keras* deep learning library with the *TensorFlow* backend. We use the pretrained VGG16 weights from the official Keras release to initialize our CNN feature extractor to speed up training, while using random weights for the rest of the model. The model is trained with the *Adadelta* optimizer w.r.t. the *binary crossentropy* loss.

To improve the model robustness under known attacks, we apply online data augmentation for JPEG compression, blurring, and additive white Gaussian noise during training. Specifically, given a raw sample, we recompress it for a JPEG quality factor in $\mathbb{U}(20, 100)$, blur it for a window size in $\mathbb{U}(0, 7)$, and perturbe it for a noise image with the standard deviation in $\mathbb{U}(0, 25)$. Common data augmentation techniques like mirroring and channel shift are also used. Finally, we subtract the intensity means learned from ImageNet for each input image, and feed pairwise (image, target) to the network for training.

Training speed is about 40 minutes per epoch on a Nividia `Titan-X` 4-GPU cluster, and each epoch scans 32,768 training samples and 32,768 validation samples. The model converges after about 80 epochs, and reaches accuracy of 98.2% and 97.1% on training and validation data at epoch 132, respectively.

4. Experimental Results

4.1. Baselines and Settings

To compare the performance of the proposed method, we select three representative algorithms as baselines—the classic Zernike moments-based block matching [22] with nearest-neighbor search, the SURF feature-based keypoint matching [6] and the dense field matching [8]. All baselines used are implemented by either a third-party or by the

authors of the corresponding papers.[1]

With regard to preprocessing and postprocessing, we always resize an image to 256×256 to fit the input size of our network, and we apply zero postprocessing to our network output. For all baseline algorithms, we simply keep their default pre- and postprocessing parameters and settings unchanged. To fairly compare the speed of algorithms, we run our trained model and all baseline algorithms on a Linux server with `Intel Xeon` CPU E5-2695@2.40GHz.

4.2. Dataset and Metrics

To evaluate image copy-move forgery performance, we use two datasets: 1) synthesized 10K dataset, and 2) CASIA TIDE v2.0 dataset.[2] The former one is synthesized by using the same strategies discussed in Section 3, but using different base images which we ensure are not seen in training and validation. It contains 5,000 positive and 5,000 negative samples (non-manipulated). The CAISA TIDE v2.0 contains 7491 authentic and 5132 tampered color images coming from nine categories: scene, animal, architecture, character, article, plant, nature, indoor, and texture. All tampered images are obtained through realistic manual manipulations (e.g., resize, deform, rotation, and blurring) and postprocessing (e.g., blurring boundary area of a tampered region) in *Adobe Photoshop CS3*. Among all tampered images, 3,274 images belong to the image copy-move forgery problem (the rest belong to the image splicing problem), and thus we use them as positive samples We regard the remaining 7,491 as negative samples. However, this dataset does not provide pixel-level ground truth, and thus we only use it to evaluate image-level performance.

We follow the widely used precision, recall and f-score metrics [6] to evaluate both pixel-level and image-level image copy-move forgery detection performance. Specifically, let true positive rate T_p be the ratio of images that have been correctly detected as forged, false positive rate F_p be the ratio of images that have been erroneously detected as forged, and false negative rate F_n be the ratio of images that have been erroneously missed. We compute the image-level precision, recall and f-score as follows.

$$\text{precision} = \frac{T_p}{T_p + F_p} \tag{3}$$

$$\text{recall} = \frac{T_p}{T_p + F_n} \tag{4}$$

$$\text{fscore} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{5}$$

[1]Available at `https://www5.cs.fau.de/research/software/copy-move-forgery-detection/`, `http://www.grip.unina.it/research/83-image-forensics/90-copy-move-forgery.html` as of August 30, 2017.

[2]Available at `http://forensics.idealtest.org/casiav2/` as of August 30, 2017.

In the same manner, we also compute pixel-level precision, recall and f-score.

4.3. Results and Discussions

Table 1 compares baseline approaches with the proposed solution on both image-level and pixel-level detection performance using the synthesized 10K dataset. As one can see, the proposed DNN solution outperforms the baseline methods by a noticeable margin. Method [22] achieves a high precision score but a low recall on image-level detection. Method [6] achieves a relatively high recall score on image-level detection but a much lower recall score on pixel-level detection. Considering the synthesized data used, this phenomenon is explainable. Both irregular copy-move region and additional affine transformation could confuse a block-matching-based algorithm like [22]. In contrast, [6] depends on key-points to reconstruct a forgery mask. When the number of matched keypoints is small, its forgery mask recovery performance is also low. Fig. 8 compares detected forgery masks from different methods.

Table 1. Performance Comparison on Syn10K Dataset

	Image-Level			Pixel-Level		
Method	**Precision**	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
[22]	93.16%	16.06%	27.39%	76.69%	15.64%	25.98%
[6]	58.19%	53.42%	55.70%	31.55%	10.22%	12.69%
[8]	91.24%	64.47%	75.53%	-	-	-
Ours	80.35%	87.11%	83.52%	80.12%	69.49%	74.43%

Table 2 compares baseline approaches with the proposed solution on image-level detection performance using the CASIA TIDE v2.0 dataset. Again, the proposed DNN solution outperforms the baseline algorithms by a large margin in both quality and speed. Fig. 9 shows the qualitative detection results of using the proposed DNN method. Here we emphasize that time complexity is computed based on decoding with CPUs, but the proposed DNN solution could be further boosted by 10+ times if we switch to decoding with GPUs.

Table 2. Performance Comparison on CASIA TIDE v2.0 Dataset

Method	**Precision**	**Recall**	**F-score**	**Time (sec/img)**
[22]	88.49%	16.53%	27.86%	5.11
[6]	39.02%	56.43%	46.14%	0.97
[8]	81.87%	61.34%	70.13%	1.78
Ours	67.83%	85.69%	75.72%	0.42

Fig. 9 shows sample detection results on the CASIA TIDE v2.0 dataset using the proposed DNN solution. As one can see, the proposed solution gracefully handles several known challenges, including but not limited to: 1) copy-move under affine transform with rotation, scale, and perspective changes; and 2) multiple source and/or multiple destination regions. Unlike classic approaches relying on additional postprocessing to handle different cases, we handle all challenges implicitly in our end-to-end DNN solution—that is to say, we do not need to specify parameters such as how many source regions to analyze, what is the

Figure 8. Sample image copy-move forgery detection results on synthesized 10K dataset. Rows from top to bottom are: input image X, ground truth target Y, the superposed image of X and Y, block-based algorithm [22], keypoint-based algorithm [6], and our results.

largest distance to reject a matching, or any heuristic parameters, but simply let the network do the prediction. Furthermore, our detected forgery masks are very meaningful in the sense of capturing "object" concept, which is extremely helpful in computer-aided image forensics analysis.

With regard to drawbacks, we noticed that the proposed DNN method 1) tends to predict "blob"-like regions; 2) makes mistakes in pure texture images; 3) sometimes erroneously predicts the genuine but visually similar regions as forged regions.

5. Conclusion

In this paper, we introduce an end-to-end DNN solution to the image copy-move forgery detection problem. Compared to classic approaches composed of multiple stages with parameter tuning and training, this new approach is fully trainable. Due to this nature, all modules are optimized jointly for the forgery mask reconstruction loss. We also showed that this model can be trained with purely synthesized training data, while achieving much better performance than the classic methods. This initial solution clearly reveals the bright future of using DNNs in the image copy-move forgery detection problem and also other image forgery detection tasks like image splicing detection.

Acknowledgement

This work is based on research sponsored by the Defense Advanced Research Projects Agency under agreement number FA8750-16-2-0204. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Project Agency or the U.S. Government.

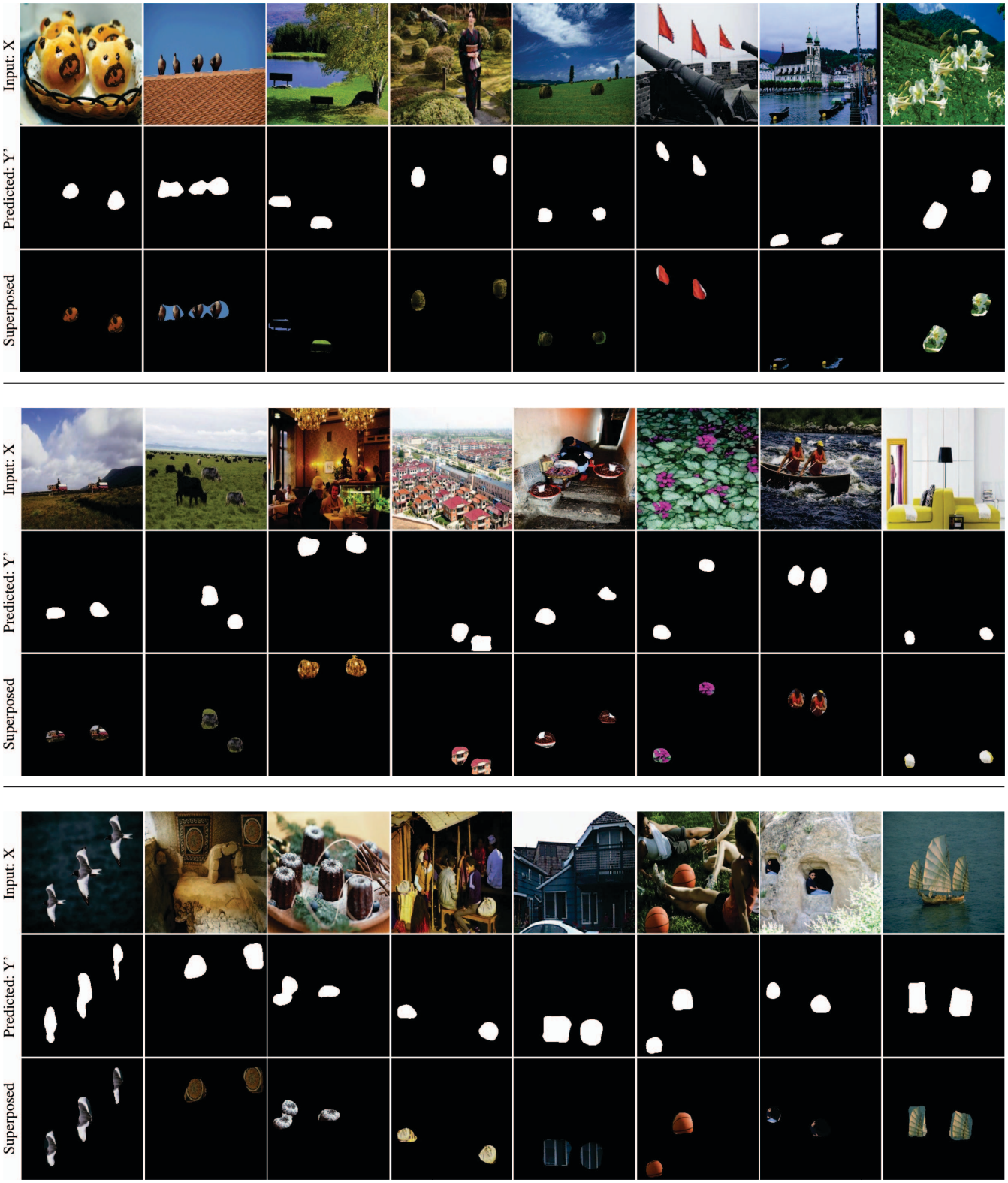

Figure 9. Sample image copy-move forgery detection results using the proposed DNN solution on CASIA TIDE v2.0 dataset. In total, three mega rows of results are shown. In each mega row, an upper row shows the original CASIA TIDE image samples, a middle row shows our detection results, and a lower row shows the superposed results by blacking out all non-detected regions in original images.

References

[1] I. Amerini, L. Ballan, R. Caldelli, A. Del Bimbo, and G. Serra. A sift-based forensic method for copy–move attack detection and transformation recovery. *IEEE Transactions on Information Forensics and Security*, 6(3):1099–1110, 2011.

[2] E. Ardizzone, A. Bruno, and G. Mazzola. Copy–move forgery detection by matching triangles of keypoints. *IEEE Transactions on Information Forensics and Security*, 10(10):2084–2094, 2015.

[3] K. Asghar, Z. Habib, and M. Hussain. Copy-move and splicing image forgery detection and localization techniques: a review. *Australian Journal of Forensic Sciences*, 49(3):281–307, 2017.

[4] S. Bayram, H. T. Sencar, and N. Memon. An efficient and robust method for detecting copy-move forgery. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 1053–1056. IEEE, 2009.

[5] X. Bi, C.-M. Pun, and X.-C. Yuan. Multi-level dense descriptor and hierarchical feature matching for copy–move forgery detection. *Information Sciences*, 345:226–242, 2016.

[6] V. Christlein, C. Riess, J. Jordan, C. Riess, and E. Angelopoulou. An evaluation of popular copy-move forgery detection approaches. *IEEE Transactions on information forensics and security*, 7(6):1841–1854, 2012.

[7] A. Costanzo, I. Amerini, R. Caldelli, and M. Barni. Forensic analysis of sift keypoint removal and injection. *IEEE Transactions on Information Forensics and Security*, 9(9):1450–1464, 2014.

[8] D. Cozzolino, G. Poggi, and L. Verdoliva. Efficient dense-field copy–move forgery detection. *IEEE Transactions on Information Forensics and Security*, 10(11):2284–2297, 2015.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] D.-Y. Huang, C.-N. Huang, W.-C. Hu, and C.-H. Chou. Robustness of copy-move forgery detection under high jpeg compression artifacts. *Multimedia Tools and Applications*, 76(1):1509–1530, 2017.

[11] A. R. Jazirehi, S. Baritaki, R. C. Koya, B. Bonavida, and J. S. Economou. Molecular mechanism of mart-1+/a* 0201+ human melanoma resistance to specific ctl-killing despite functional tumor–ctl interaction. *Cancer research*, 71(4):1406–1417, 2011.

[12] J. Li, F. Yang, W. Lu, and W. Sun. Keypoint-based copy-move detection scheme by adopting mscrs and improved feature matching. *Multimedia Tools and Applications*, pages 1–15, 2016.

[13] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[14] Y. Liu, Q. Guan, and X. Zhao. Copy-move forgery detection based on convolutional kernel network. *arXiv preprint arXiv:1707.01221*, 2017.

[15] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[16] B. Mahdian and S. Saic. Detection of copy–move forgery using a method based on blur moment invariants. *Forensic science international*, 171(2):180–189, 2007.

[17] T. Mahmood, T. Nawaz, A. Irtaza, R. Ashraf, M. Shah, and M. T. Mahmood. Copy-move forgery detection technique for forensic analysis in digital images. *Mathematical Problems in Engineering*, 2016, 2016.

[18] C.-M. Pun, X.-C. Yuan, and X.-L. Bi. Image forgery detection using adaptive oversegmentation and feature point matching. *IEEE Transactions on Information Forensics and Security*, 10(8):1705–1716, 2015.

[19] Y. Rao and J. Ni. A deep learning approach to detection of splicing and copy-move forgeries in images. In *Information Forensics and Security (WIFS), 2016 IEEE International Workshop on*, pages 1–6. IEEE, 2016.

[20] S. Rivas, T. Mucyn, H. A. Van Den Burg, J. Vervoort, and J. D. Jones. An 400 kda membrane-associated complex that contains one molecule of the resistance protein cf-4. *The Plant Journal*, 29(6):783–796, 2002.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[22] S.-J. Ryu, M.-J. Lee, and H.-K. Lee. Detection of copy-rotate-move forgery using zernike moments. In *Information hiding*, volume 6387, pages 51–65. Springer, 2010.

[23] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[25] N. B. A. Warif, A. W. A. Wahab, M. Y. I. Idris, R. Ramli, R. Salleh, S. Shamshirband, and K.-K. R. Choo. Copy-move forgery detection: Survey, challenges and future directions. *Journal of Network and Computer Applications*, 75:259–278, 2016.

[26] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 3485–3492. IEEE, 2010.

[27] W. A.-A. Yue Wu and P. Natarajan. Deep matching and validation network: An end-to-end solution to constrained image splicing localization and detection. In *MM'17*.

[28] M. Zandi, A. Mahmoudi-Aznaveh, and A. Talebpour. Iterative copy-move forgery detection based on a new interest point detector. *IEEE Transactions on Information Forensics and Security*, 11(11):2499–2512, 2016.

[29] Y. Zhu, X. Shen, and H. Chen. Copy-move forgery detection based on scaled orb. *Multimedia Tools and Applications*, 75(6):3221–3233, 2016.