BusterNet: Detecting Copy-Move Image Forgery with Source/Target Localization

Yue Wu[1(✉)], Wael Abd-Almageed[1], and Prem Natarajan[1,2]

[1] USC Information Sciences Institute, Marina del Rey, CA 90292, USA
{yue_wu,wamageed,pnataraj}@isi.edu
[2] Amazon Alexa, 101 Main Street, Cambridge, MA 02142, USA

Abstract. We introduce a novel deep neural architecture for image copy-move forgery detection (CMFD), code-named *BusterNet*. Unlike previous efforts, BusterNet is a pure, end-to-end trainable, deep neural network solution. It features a two-branch architecture followed by a fusion module. The two branches localize potential manipulation regions via visual artifacts and copy-move regions via visual similarities, respectively. To the best of our knowledge, this is the first CMFD algorithm with discernibility to localize source/target regions. We also propose simple schemes for synthesizing large-scale CMFD samples using out-of-domain datasets, and stage-wise strategies for effective BusterNet training. Our extensive studies demonstrate that BusterNet outperforms state-of-the-art copy-move detection algorithms by a large margin on the two publicly available datasets, CASIA and CoMoFoD, and that it is robust against various known attacks.

Keywords: Copy-move · Image forgery detection · Deep learning

1 Introduction

Fake news, often utilizing tampered images, has lately become a global epidemic, especially with the massive adoption of social media as a contemporary alternative to classic news outlets. This phenomenon can be largely attributed to the following: (i) the rapidly declining cost of digital cameras and mobile phones, which leads to a proliferation of digital images, and (ii) the availability and ease-of-use of image-editing software (e.g., mobile phone applications and open source tools) which make images editing or manipulating profoundly easy, whether it is for innocent or malicious intent.

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.

V. Ferrari et al. (Eds.): ECCV 2018, LNCS 11210, pp. 170–186, 2018.
https://doi.org/10.1007/978-3-030-01231-1_11

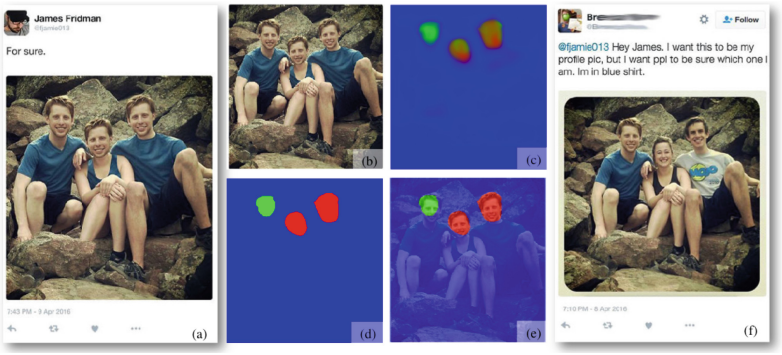

Fig. 1. Whom in photo is *not* manipulated? BusterNet answers this question by not only detecting copy-move regions but also differentiating source (green) and target (red) copies. (a) tweet snapshot of a manipulated photo by James Fridman; (b) input region for analysis; (c) raw BusterNet output; (d) BusterNet output by applying majority rule; (e) overlaid result of (c) on (b); and (f) tweet snapshot of the original photo. (Color figure online)

Copy-move image forgery is one of the most common and easiest-to-perform image tampering schemes (see Fig. 1), in which an image patch, regular or irregular shape, is copied and cloned into the same image. Since the cloned image patch comes from the same image, the photometric characteristic remains largely consistent, which increases the difficulty of detecting this type of image forgery.

The objective of copy-move forgery detection (CMFD) is to determine whether a probe (*i.e.* query) image contains *cloned regions*, as evidence of potential malicious intent. Based on the sophistication of the cloning process, we can generally classify copy-move manipulations as *plain*, *affine* and *complex* forgeries. Let S and T denote the source and target regions, respectively. Plain cloning means that T is simply a translated version of S, i.e., copy S and directly paste it to a new location as T. This is the simplest cloning, which can be done with very basic image editing tools. In contrast, affine cloning means that T is an affine-transformed version of S, implying that additional scaling and rotation changes have been made on S. Similarly, this type of copy-move tampering can be easily performed with image editing tools supporting affine transformations. Finally, complex cloning entails a more complicated relationship between D and T, often with extra diffusion estimation, edge blending, color change or other more sophisticated image processing steps. Complex cloning requires advancing image editing tools, such as *Adobe Photoshop* or *GIMP*.

In this paper we present a novel deep neural architecture for detecting copy-move image forgeries. The proposed architecture addresses two major limitations of state-of-the-art CMFD algorithms—(i) it is an end-to-end DNN solution, and thus can be optimized directly w.r.t. copy-move forgery detection task; and (ii) it not only detects copy-move forgeries, but also produces source and target region masks, as shown in Fig. 1. To the best of our knowledge, our proposed

technology is the first to feature this capability. Discriminating between source and target regions could be significant for forensic investigations. Consider, for example, two people each holding a pistol in a criminal investigation. We not only are interested in knowing that image is manipulated, but also which gun is the original and which is the clone.

The remainder of this paper is organized as follows. Section 2 briefly discusses existing approaches and their limitations. Section 3 introduces the proposed BusterNet. Section 4 discusses BusterNet training details. Section 5 shows extensive experimental results and analysis. Finally, we conclude the paper in Sect. 6.

2 Related Work

Early CMFD work can be traced back to the early 2000s [12,15] when most of the research work focused on plain cloning. As mentioned in Sect. 1, the increasing volume of digital images and availability of editing software meant that "*Doctoring digital photos is easy. Detecting it can be hard*" [11].

Without loss of generality, a general copy-move detection framework consists of three major steps [9]: namely (i) *feature extraction* which basically converts an input image X to a set of features of interests $\mho = \{f_1, \cdots, f_k\}$, (ii) *feature matching* which measures the similarity (or distance) between two features f_i and f_j for all $f_i, f_j \in \mho$, and (iii) *post-processing* which usually uses a set of heuristics to further improve CMFD performance, e.g., considering holistic matching between set of features on a higher level of consistency to reduce false alarms and to improve true positive. Copy-move detection frameworks can be broadly classified, based on the underlying feature extraction and subsequent matching schemes, into three main categories: patch/block-based methods such as chroma features [4,9], PCA feature [14], Zernike moments [26], blur moments [20], DCT [21]; keypoint-based methods such as SIFT [1,8,36], ORB [40], triangles [2], SURF [22,27,28], and irregular region-based methods [16,25].

Each category has its own advantages and disadvantages in CMFD. For example, block-based methods are known to be simple but computationally expensive. In contrast, keypoint-based methods are fast and robust against affine transformation. However, keypoint-based method often fail when S and D are homogeneous. This general architecture for CMFD pipelines suffers three inherent limitations: (i) each module is optimized independently, (ii) dependence on hand-crafted features that may not be optimal for the downstream task, and (iii) inclusion of one or more heuristics or manually tuned thresholds in order to reduce false alarm and increase detection rates. For a detailed comparison of existing methods, the reader is referred to [3,5,30,32].

Recently, deep neural network (DNN) has been applied to the image forgery detection research. [18] uses DNN for feature extraction in CMFD. [6] detects manipulated regions via a DNN-based patch classifier. [34] proposes an end-to-end DNN solution for splicing detection and localization. [39] uses DNN to detect tampered faces.

3 Copy-Move Forgery Detection via BusterNet

3.1 BusterNet Overview

To overcome the drawbacks of classic CMFD pipelines, as discussed in Sect. 2, our goal is to design a DNN pipeline that is (i) end-to-end trainable, such that it does not include manually tuned parameters and/or decision rules and (ii) capable of producing distinct source and target manipulation masks (which could be used for forensic analysis).

To achieve the above goals, a valid DNN solution should attain two feature properties simultaneously, (i) source and target features are dissimilar enough to distinguish source from target, and (ii) they are also more similar than those in pristine regions. Of course, one can train a naive DNN, while hoping it could attain these properties magically. However, a better idea is to explicitly consider these properties, and we therefore propose *BusterNet*, a two-branch DNN architecture as shown in Fig. 2.

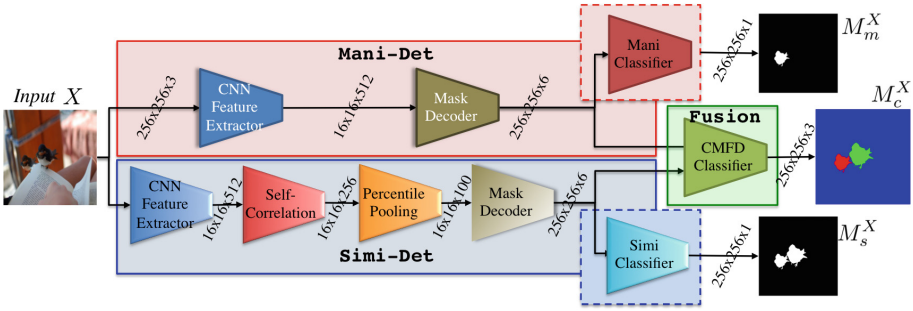

Fig. 2. Overview of the proposed two-branch DNN-based CMFD solution. Dashed blocks are only activated during branch training. Output mask of the main task, *i.e.* M_c^X, is color coded to represent pixel classes, namely *pristine* (blue), *source copy* (green), and *target copy* (red). Output masks of auxiliary tasks, *i.e.* M_m^X and M_s^X, are binary where white pixels indicates manipulated/similar pixels of interests, respectively. (Color figure online)

Specifically, we design `Mani-Det` branch to detect manipulated regions such that its feature is good for property (i), while `Simi-Det` branch to detect cloned regions such that its feature attains property (ii), and finally use both features in `Fusion` to predict pixel-level copy-move masks differentiating *pristine*, *source copy*, and *target copy* classes. To ensure these two branches achieve the desired functionality, we define each branch an auxiliary task, as indicated by the dashed blocks in Fig. 2. More precisely, `Mani-Det`'s and `Simi-Det`'s tasks are to predict a binary manipulation mask M_m^X and a binary copy-move mask M_s^X, respectively, and both binary masks can be derived from the 3-class mask M_c^X.

To simplify discussions, we assume our input image is of size $256 \times 256 \times 3$, but BusterNet is capable of handling images of other sizes.

3.2 Manipulation Detection Branch

The manipulation detection branch (*i.e.* `Mani-Det` as shown by red shaded regions in Fig. 2) can be thought of as a special segmentation network [19] whose aim is to segment manipulated regions. More precisely, it takes input image X, extracts features using `CNN Feature Extractor`, upsamples the feature maps to the original image size using `Mask Decoder`, and applies `Binary Classifier` to fulfill the auxiliary task, *i.e.* producing a manipulation mask M_m^X.

Any convolutional neural network (CNN) can serve as `CNN Feature Extractor`. Here, we use the first four blocks of the VGG16 architecture [29] for its simplicity. The resulting CNN feature f_m^X is of size $16 \times 16 \times 512$, whose resolution is much lower than that is required by the manipulation mask. We therefore need to decode this feature, and apply deconvolution [23] to restore the original resolution via the `Mask Decoder` as shown Fig. 3, which applies `BN-Inception` and `BilinearUpPool2D` [33] in an alternating way and eventually produces a tensor d_m^X of shape $256 \times 256 \times 6$. To be clear, 16 times of the spatial dimension increase is due to the 4 times of `BilinearUpPool2D` (*i.e.* $2^4 = 16$), and the output filter dimension 6 is because of the last `BN-Inception`(2@[5,7,11]), which concatenates 3 `Conv2D` responses, each with 2 output filters but uses kernel sizes at $(5, 5)$, and $(7, 7)$ and $(11, 11)$, respectively (*i.e.* $3 \times 2 = 6$). Finally, we predict pixel-level manipulation mask M_m^X via `Binary Classifier`, which is as simple as a single `Conv2D` layer with 1 filters of kernel size $(3, 3)$ followed by the `sigmoid` activation.

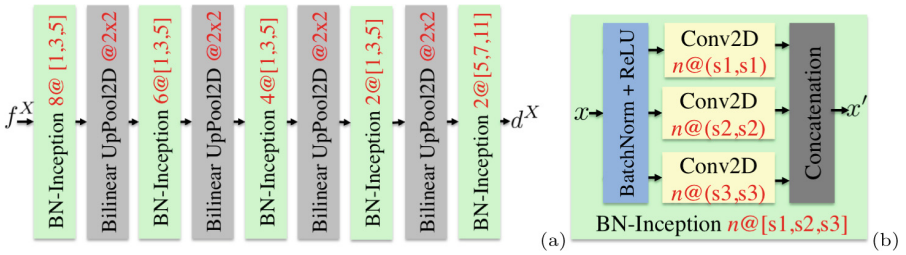

Fig. 3. Inception-based mask deconvolution module. (a) Mask deconvolution network and (b) parametric BN-inception module, where $s1, s2$ and $s3$ indicates the kernel sizes used in three `Conv2D` layers, respectively, and n stands for the number of filters.

3.3 Similarity Detection Branch

The similarity detection branch (*i.e.* `Simi-Det` as shown by blue shaded regions in Fig. 2) takes an input image X, extracts features using `CNN Feature Extractor`, computes feature similarity via `Self-Correlation` module, collects useful statistics via `Percentile Pooling`, upsamples feature maps to the original image size using `Mask Decoder`, and applies `Binary Classifier` to fulfill the auxiliary task, *i.e.* producing a copy-move mask M_m^X at the same resolution of X. It is worthy to stress that modules shared in both branches, *e.g.* `CNN Feature Extractor`, only share the network architecture, but not weights.

Like `Mani-Det` branch, `Simi-Det` branch starts with feature representation via `CNN Feature Extractor`. It again produces a feature tensor f_s^X of size $16 \times 16 \times 512$, which can be also viewed as 16×16 patch-like features, *i.e.* $f_s^X = \{f_s^X[i_r, i_c]\}_{i_r, i_c \in [0,\cdots,15]}$, and each with 512 dimensions. Since our goal is to recover the potential copy-move regions, we have to mine useful information to decide what are matched patch-like features. To do so, we first compute all-to-all feature similarity score using `Self-Correlation`, and collect meaningful statistics to identify matched patches via `Percentile Pooling`.

Specifically, given two patch-like feature $f_m^X[i]$ and $f_m^X[j]$ where $i = (i_r, i_c)$ and $j = (j_r, j_c)$, we use the Pearson correlation coefficient ρ to quantify the feature similarity as shown in Eq. (1)

$$\rho(i, j) = (\tilde{f}_m^X[i])^T \tilde{f}_m^X[j]/512 \tag{1}$$

where $(\cdot)^T$ is the transpose operator, and $\tilde{f}_m^X[i]$ is the normalized version of $f_m^X[i]$ by subtracting its mean $\mu_m^X[i]$ and dividing by its standard deviation $\sigma_m^X[i]$ as shown in Eq. (2).

$$\tilde{f}_m^X[i] = (f_m^X[i] - \mu_m^X[i])/\sigma_m^X[i] \tag{2}$$

For a given $f_m^X[i]$, we repeat the process over all possible $f_m^X[j]$, and form a score vector $S^X[i]$, namely

$$S^X[i] = [\rho(i, 0), \cdots, \rho(i, j), \cdots, \rho(i, 255)] \tag{3}$$

As a result, `Self-Correlation` outputs a tensor S^X of shape $16 \times 16 \times 256$.

When f_m^X and Pearson correlation coefficient are both meaningful, it is clear that if $f_m^X[i]$ is matched, some score $S^X[i][j]$ with $j \neq i$ should be significantly greater than the rest of the scores $S^X[i][k]$ with $k \notin \{i, j\}$. Since we do not know the corresponding $f_m^X[j]$ in advance, it is difficult to check this pattern in the context of DNN. Alternatively, it is easier to check this pattern in a sorted score vector. Specifically, `Percentile Pooling` first sorts a score vector $S^X[i]$ to $S'^X[i]$ in the descending order as shown Eq. (4).

$$S'^X[i] = \text{sort}(S^X[i]) \tag{4}$$

Imagining plotting a curve about $(k, S'^X[i][k])$ for $k \in [0, \cdots, 255]$, we are supposed to see a monotonic decreasing curve with an abrupt drop at some point if $f_m^X[i]$ is matched. This indicates that this sorted version of score vector contains sufficient information to decide what feature is matched in future stages.

One can directly feed S'^X to future modules to decide matched features. However one drawback of doing so is that the resulting network loses the capability of accepting an input of an arbitrary size, because the length of score vector is dependent on the input size. To remove this dependency, `Percentile Pooling` also standardize the sorted score vector by only picking those scores at percentile ranks of interests. In other words, regardless the length L of raw sorted score vector, we always pick K scores to form a pooled percentile score vector $P^X[i]$ as shown in Eq. (5)

$$P^X[i][k] = S'^X[i][k'] \tag{5}$$

where $k \in [0, \cdots, K-1]$, and k' is the index of raw sorted score vector after mapping a predefined percentile $p_k \in [0, 1]$ according to L as shown in Eq. (6)

$$k' = \text{round}(p_k \cdot (L-1)) \tag{6}$$

Another advantage of the above standardization is dimension reduction, because only a small portion of all scores is kept. Once `Percentile Pooling` is done, we use `Mask Decoder` to gradually upsample feature P^X to the original image size as d_s^X, and `Binary Classifier` to produce a copy-move mask M_s^X to fulfill the auxiliary task. Again, both `Mask Decoder` and `Binary Classifier` only have the same architecture as those in `Mani-Det`, but with distinctive weights.

3.4 BusterNet Fusion

As illustrated in Fig. 2, `Fusion` module takes inputs of the `Mask Decoder` features from both branches, namely d_m^X and d_s^X, and jointly considers these two branches and make the final CMFD prediction. More precisely, we (i) concatenate feature d_m^X and d_s^X, (ii) fuse feature using the `BN-Inception` with parameter set $3@[1, 3, 5]$ (see Fig. 3(b)), and (iii) predict the three-class CMFD mask using a `Conv2D` with one filter of kernel size 3×3 followed by the `softmax` activation.

4 BusterNet Implementation and Training

4.1 Custom Layer Implementation

It is clear that except `Self-Correlation` and `Percentile Pooling` modules, all other modules are either standard or can be built from standard layers. `Self-Correlation` requires implementing Eqs. (1) and (2). We compute (i) tensor $\tilde{f}_m^X$ using Eq. (2), and (ii) correlation scores for all i, j pairs in one-shot via tensor dot product operator[1], instead of computing Eq. (1) one by one. Both operations are differentiable.

`Percentile Pooling` is essentially a pooling layer, which has no trainable parameters but a deterministic pooling function. As one may notice, neither Eq. (4) nor (5) is differentiable. However, all we need to do is to perform back-propagation similar to that is performed in standard `MaxPooling` (*i.e.* only the neuron corresponding to the max receives the gradient).

4.2 Training Details

As shown in Table 1 of [38], and Table 12 of [32], publicly available dataset are very small (typically around a few thousands). More importantly, none of existing CMFD dataset provides ground truth masks differentiating source and target copies. We therefore create a synthetic dataset for training BusterNet.

Inspired by [13] and also [10], we start with an original image X with an associated object mask M_s, randomly generate an affine matrix m to transform both the source mask and the source object image. We then use the transformed mask as the target mask M_t, paste the transformed object back to image

[1] This operator is known as `batched_tensordot` in Theano, and `matmul` in TensorFlow.

X and obtain a copy-move forgery sample X'. In particular, we use the MIT SUN2012 dataset [35], and the Microsoft COCO dataset [17] as image sources. All generated manipulation are either affine [34] or complex[2]. Complex clones use *Poisson image editing* [24], and perform complicated blending far beyond naive region pasting. Parameters used in affine matrix are: rotation $\in [-30°, 30°]$, scale $\in [0.5, 2]$, and translation is also uniformly picked within regions.

To further encourage more real-like training samples, we train a binary classifier to predict whether a sample is real unmanipulated or synthetic copy-move forged. Synthetic samples that fail to fool this classifier are not used for training BusterNet. Figure 4 shows some synthetic samples of our dataset, and they look quite natural in general. This dataset can be provided upon request.

Fig. 4. Synthetic CMFD samples with ground truth masks. Pixel colors blue, green and red in masks denotes *pristine*, *source* and *target copy* classes, respectively. (Color figure online)

In total, we collected 100,000 quality synthetic samples for copy-move detection, each with one three-class mask distinguishing source and destination copies and two binary masks auxiliary task training. The synthetic training data is split into training, validation and testing splits with 8:1:1 ratio. This synthetic dataset is used to train both auxiliary tasks and the main task of BusterNet.

It is worth noting that external image manipulation dataset from IEEE IFS-TC First Image Forensics Challenge[3] and the Wild Web tampered image dataset [37] are also used for training `Mani-Det` branch, because we want `Mani-Det` learns to identify more manipulated regions beyond the fixed set of manipulations in our synthetic dataset.

To train BusterNet, we initialize all parameters from random weights except for using a pretrained VGG16 on ImageNet for `CNN Feature Extractor` in `Simi-Det`. Instead of training BusterNet all modules together, we adopt a three-stage training strategy—(i) train each branch with its auxiliary task independently, (ii) freeze both branches and train `fusion` module, and (iii) unfreeze entire network and fine tune BusterNet end-to-end. Specifically, for auxiliary tasks, we use `Adam` optimizer with initial learning rate of 1e−2 and `binary_crossentropy` loss. Whenever validation loss reaches plateaus after 10

[2] https://github.com/fbessho/PyPoi.git.
[3] http://ifc.recod.ic.unicamp.br/.

epochs, we reduce learning rate by half until improvement stops for 20 epochs. For main task, we also `Adam` optimizer with `categorical_crossentropy` loss, but use initial learning rate of 1e−2 for `fusion` training while 1e−5 for finetuning. Pretrained models can be also found in our open repository https://github.com/isi-vista/BusterNet.git.

Table 1. Comparing different training strategies on Synthetic 10K testing set

Metrics	Recall			Accuracy
Class	Pristine	Source	Target	3-Class
Simi-Det Only	92.57%	32.28%	38.97%	92.57%
Direct BusterNet	93.70%	34.12%	47.37%	92.74%
Stage-wise BusterNet	93.83%	41.64%	53.61%	93.02%

This stage-wise training strategy is important to ensure the functionality of each branch, and further to achieve BusterNet's goals. Table 1 compares BusterNet performance on our synthetic testing dataset. Specifically, *Simi-Det Only* denotes to perform the main task but only use the `Simi-Det` branch (in other words, a naive solution as we discussed in Sect. 3.1), *Direct BusterNet* means to train BusterNet without auxiliary tasks, and *Stage-wise BusterNet* is our proposed training strategy. As one can see, it is easier to predict target copies than source copies in general, possibly because target copies may contain visual artifacts to help classification; and differences of prediction accuracy between systems are small, due to the dominant *pristine* class. However, the impact of two-branch design and stage-wise training should not be under-looked. As shown in Table 1, it turns out that two-branch model(*Direct BusterNet*) outperforms one-branch model (*Simi-Det Only*), and stage-wise training further improves recall on all classes by a large margin, especially on source and target classes.

5 Experimental Evaluation

5.1 Metrics and Baseline Settings

We use precision, recall and F_1 scores to report CMFD performance [7,9,34]. For a testing image, we compute the true positive (TP), false positive (FP) and false negative (FN) at pixel level. Of course, we have to treat pixels classified to source and target both as *forged*, so that the proposed BusterNet could be fairly compared with all classic CMFD methods which only predict binary masks.

Based on how F_1 is calculated, two protocols are used for **pixel-level evaluation**: (A) aggregate all TP, FP, and FN numbers over the whole dataset, and report precision, recall and F_1 scores [7,34]; and (B) compute precision, recall, F_1 scores for each image, and report the averaged scores [25]. Protocol A better captures overall performance including non-forged images, while protocol B

only works for a subset of forged images (F_1 score is ill-defined when TP is zero), but better quantifies the localization performance. We use both protocols in our evaluations. If any pixels in a testing image are detected as forged, the testing image is labeled as forged. We compare a predicted image label with its ground truth to compute image-level TP, FP, and FN, and report precision, recall and F_1 scores over an entire dataset as **image-level evaluation protocol**.

Furthermore, we use *area under the receiver operating characteristic (ROC) curve (AUC)* to evaluate overall performance, where the ROC curve is a function of *true positive rate (TPR)* in terms of *false positive rate (FPR)*. AUC quantifies the overall ability of the network to discriminate between two classes.

We use four methods as baselines for comparison—a block-based CMFD with the Zernike moment feature [26], a keypoint-based CMFD with the SIFT feature [7], a dense field-based CMFD [9], and a deep matching and validation network (DMVN) [34]. All method implementations are either provided by paper authors or from reliable third-party implementation in [7]. Since DMVN is originally designed for image splicing detection where the input is a pair of images, we recursively split the input image into two halves X_1 and X_2 along X's longer axis, and feed DMVN this pair of (X_1, X_2). If DMVN finds anything spliced, this means X contains copy-move regions (one in X_1 and the other in X_2). If not, we then split X_1 and X_2 into halves again, and apply DMVN to detect whether splicing has happened within X_1 and X_2. This recursion continues until it reaches the smallest patch size (we use 16×16) for analysis or successfully finding splicing regions. Default parameters are used for all baseline methods without any preprocessing. Speed measurement is based on an Intel Xeon CPU E52695@2.4GHz with a single Nivdia Titan-X GPU.

5.2 Evaluation Data

We use two standard datasets for evaluation. The first dataset is the CASIA TIDEv2.0 dataset,[4] which is the largest public accessible image forgery detection benchmark, in which all manipulations are created manually. It contains 7491 authentic and 5123 tampered color images. However, it does not specify which images are manipulated in a copy-move manner and does not provide ground truth manipulation masks. We therefore manually verify that 1313 out of 5123 tampered samples are of copy-move forgery. These 1313 CMFD samples and their authentic counterparts together form the testing dataset (total 2626 samples) we used later. We refer to it as the CASIA CMFD dataset.

The second dataset is the CoMoFoD dataset [31], which contains 200 base forged images and 25 categories (total 5000 images). Each category is made by applying postprocessing/attacks to the **base** category images to hide forgery clues (e.g., JPEG compression, etc.). Detailed attack descriptions and settings can be found in [31].

Finally, to evaluate BusterNet discernibility, we need testing data with ground truth masks distinguishing source and target. However, neither the

[4] http://forensics.idealtest.org/casiav2.

CASIA CMFD nor the CoMoFoD datasets provide such masks. We therefore synthesize them by comparing a forged image with its authentic counterpart for both datasets. All synthesized masks are manually verified, and can be found in our code repository.

5.3 Overall CMFD Performance Analysis

Table 2 shows the overall performance on the CASIA CMFD dataset. BusterNet's F_1 score outperforms all others by a large margin on all three evaluation protocols; it is also is the fastest solution. By comparing the performance of `Simi-Det` branch and the full BusterNet, one can see that end-to-end fine-tuning improves AUC by 3–4% at both pixel- and image-level as shown in Fig. 5.

Table 2. Performance analysis on CASIA CMFD dataset.

	Methods				Ours	
	[26]	[7]	[9]	[34]	Simi. Det.	BusterNet
	Image level evaluation protocol					
Precision	97.01%	68.49%	**99.51%**	66.37%	71.53%	78.22%
Recall	24.47%	67.82%	30.61%	73.59%	**80.73%**	73.89%
F-score	39.08%	68.15%	46.82%	69.80%	75.85%	**75.98%**
	Pixel level evaluation protocol - A					
Precision	**94.46%**	64.84%	83.12%	17.06%	56.52%	77.38%
Recall	25.05%	0.17%	51.28%	10.60%	**62.06%**	59.15%
F-score	39.59%	0.34%	63.43%	13.08%	59.16%	**67.05%**
	Pixel level evaluation protocol - B					
Precision	22.71%	37.09%	24.92%	23.97%	47.23%	**55.71%**
Recall	13.36%	0.14%	26.81%	13.79%	**48.44%**	43.83%
F-score	16.40%	0.23%	25.43%	14.64%	43.72%	**45.56%**
	Processing speed					
Sec./Img.	5.11	0.97	1.78	0.95	**0.44**	0.62

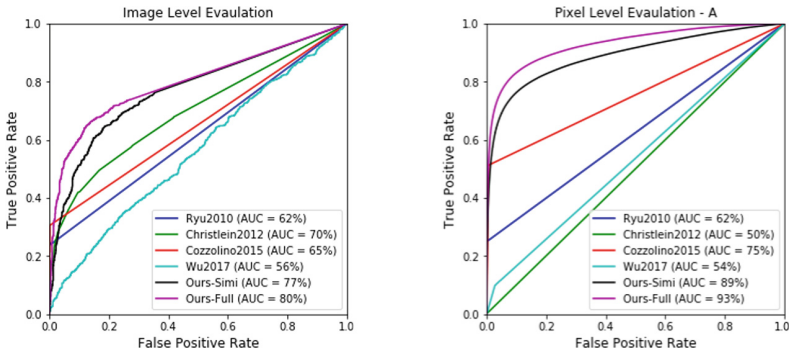

Fig. 5. AUC performance comparison on CASIA CMFD dataset.

5.4 BusterNet Robustness Analysis

To evaluate BusterNet robustness against various attacks/postprocessing, we test it and all baseline methods on the CoMoFoD dataset. Table 3 shows the number of *correctly detected* images under each attack (containing 200 samples). An image is to referred as *correctly detected* if its pixel-level F_1 score is higher than 0.5 [31]. BusterNet outperforms baseline methods on all but one attack.

To better understand the BusterNet performance against the state-of-the-art, we conduct performance analysis over the entire dataset. Figure 6 shows the detailed pixel-level F_1 scores under protocol B for all attacks. As one can see, except for attacks of severe JPEG compression (*e.g.* JC1 compression quality 10), BusterNet is quite robust against various attacks. The performance analysis on the base category (i.e., no attack) is shown in Table 3. On the left half of the table, we follow the modified protocol used by [18,31] to report average scores only on the correctly detected subset, while on the right half of table we continuously report our performance using pixel-level protocol B. It is worthy to mention that although [9] leads the proposed BusterNet by 7% in F_1 score when only considering correctly detected samples, BusterNet correctly detected 24 more

Table 3. Number of correctly detected images on CoMoFoD dataset under attacks.

Attack	Methods							
	[31]	[26]	[16]	[28]	[18]	[9]	[34]	Ours
Base	53	90	102	88	97	93	53	**117**
BC1	-	91	-	-	-	94	50	**116**
BC2	-	89	-	-	-	94	53	**115**
BC3	42	89	99	90	94	88	48	**109**
CA1	-	93	-	-	-	98	50	**117**
CA2	-	93	-	-	-	96	50	**116**
CA3	45	92	99	90	94	96	48	**116**
CR1	44	92	90	82	72	97	51	**117**
CR2	-	91	-	-	-	95	50	**116**
CR3	-	90	-	-	-	92	54	**116**
IB1	-	90	91	94	104	91	53	**113**
IB2	47	87	-	-	-	88	32	**98**
IB3	-	84	-	-	-	84	26	**93**
JC1	-	43	-	-	-	**69**	18	60
JC2	-	63	-	-	-	73	21	**77**
JC3	-	72	-	-	-	75	26	**86**
JC4	5	73	-	-	-	77	29	**103**
JC5	-	76	-	-	-	81	38	**99**
JC6	-	80	-	-	-	83	33	**101**
JC7	-	86	-	-	-	87	42	**107**
JC8	-	88	-	-	-	92	42	**109**
JC9	-	81	89	31	78	87	36	**106**
NA1	-	24	-	-	-	41	38	**100**
NA2	3	42	-	-	-	66	39	**102**

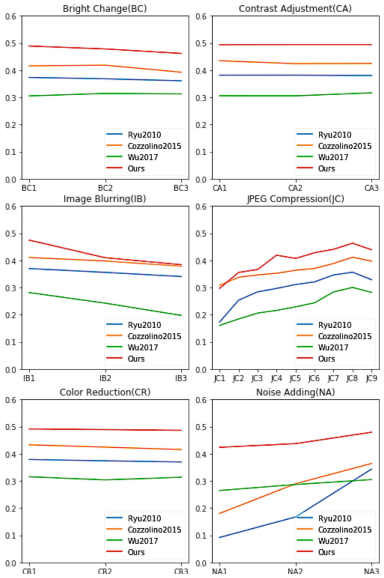

Fig. 6. Pixel-level F1 scores (y-axis) on CoMoFoD under attacks (x-axis).

Table 4. CMFD performance comparisons on CoMoFoD dataset with no attack.

Methods	Correctly detected average				Protocol B (overall average)		
	#Passed	Precision	Recall	F1	Precision	Recall	F1
[31]	50	**96.77**%	83.91%	**88.72**%	-	-	-
[26]	90	96.27%	69.84%	79.93%	45.78%	34.35%	37.37%
[16]	102	54.46%	85.04%	59.54%	-	-	-
[28]	100	54.37%	74.19%	54.60%	-	-	-
[18]	97	59.27%	82.20%	63.18%	-	-	-
[9]	93	84.22%	**93.58**%	87.82%	39.92%	47.61%	41.83%
[34]	53	61.11%	71.48%	63.13%	36.29%	40.41%	31.13%
Ours	**117**	83.52%	78.75%	80.09%	**57.34**%	**49.39**%	**49.26**%

samples than [9], indicting BusterNet is better in general. Indeed,BusterNet outperforms [9] by 7% on the performance over the entire dataset (Table 4).

5.5 Source/Target Discernibility Analysis

To the best of our knowledge, none of CMFD state-of-the-art methods is capable of localizing, and differentiating between, the source and target of the manipulation. One of the prominent features of BusterNet, however, is the ability to localize source and target regions. In order to evaluate the accuracy of localization, we compare the predicted forgery region labels with those from ground truth. For each predicted mask, we merge the source and destination channels to find all forged regions using the connected component (CC) analysis, and use the dominant class of all its pixels as the label of a CC. If no CC is found, this is a miss. If all CCs in a sample have the same label, we opt-out this sample. Otherwise, this is an opt-in sample for analysis, and we label it "correct" only when both source and target forgery regions are correctly classified.

Visual examples are shown in Fig. 7. Table 5 summarizes the discernibility performance of BusterNet on both the CASIA CMFD (manipulated only) and CoMoFoD datasets, where *miss* indicates those missed samples, *overall accuracy* is the ratio of corrected samples to total samples, and opt-in accuracy is the ratio of corrected samples to opt-in samples. The overall ∼12% accuracy does not seem not high. However, one should consider the fact that BusterNet is only trained with synthetic data with a limited number of real manipulation samples, and the used simple CC-based label assignment scheme is also simple for complicated real cases. As one can see in Fig. 7(b), BusterNet sometimes correctly captures target manipulation at least partially (*e.g.*, the left-most bird sample and the right-most spider sample), but the simple CC-based label scheme fails to assign correct labels. Indeed, if we consider the accuracy only for opt-in samples, the accuracy of the proposed BusterNet jumps to 78% as shown in Table 5.

Table 5. Source/target discernibility performance of BusterNet.

Dataset	Number of images					Accuracy	
	Total	Miss	Opt-out	Opt-in	Corr.	Overall	Opt-in
CASIA CMFD	1313	542	581	190	146	11.11%	76.84%
CoMoFoD	200	83	76	41	33	16.50%	80.49%
Overall	1513	625	657	231	179	11.83%	77.49%

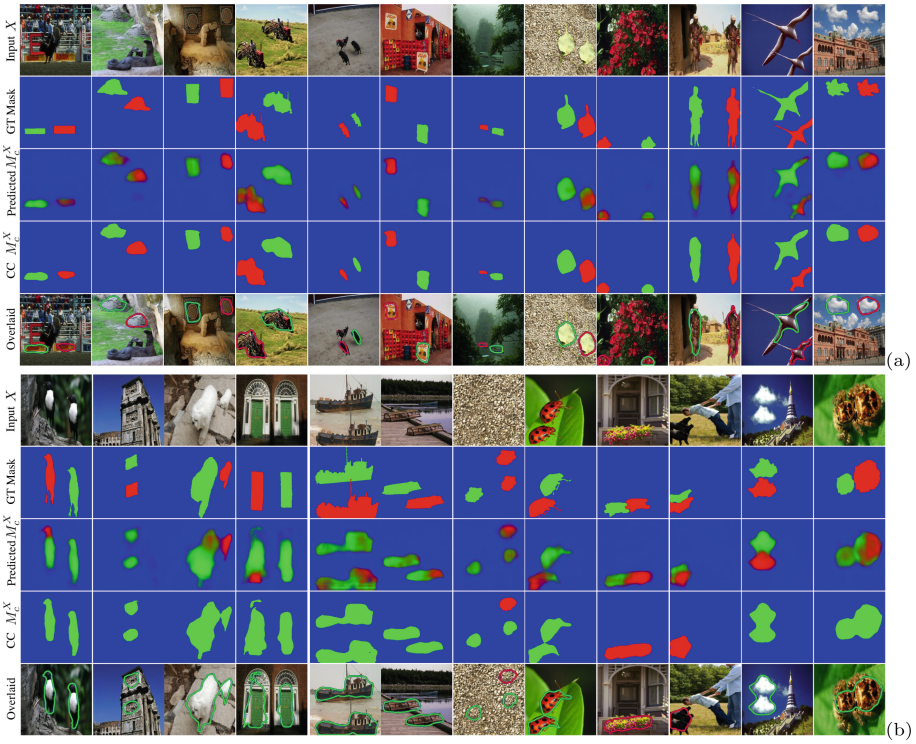

Fig. 7. BusterNet detection results on testing dataset. (a) samples that BusterNet correctly distinguishes source/target copies; and (b) samples that BusterNet fails to distinguish source/target copies. blue: *pristine*, green: *source copy* and red: *target copy*. Note many object classes, *e.g. flower*, *sand*, and *ladybug*, are not included in SUN or COCO dataset, indicating the generalizability of BusterNet to unseen classes. (Color figure online)

6 Conclusion

We introduce BusterNet, an end-to-end DNN solution to detecting copy-move forged images with source/target localization with two branches as shown in Fig. 2. We show how to design auxiliary tasks for each branch to ensure its functionality and feature properties. We also demonstrate how to overcome the

training data shortage by synthesizing a large scale of realistic and quality CMFD samples from out-of-domain datasets. Our evaluation results demonstrate that BusterNet outperforms state-of-the arts methods by a large margin, and is also robust against various known CMFD attacks. More importantly, BusterNet has the prominent advantage, over any existing CMFD solutions, of distinguishing source/target copies. This is a desired capability for forensic experts.

References

1. Amerini, I., Ballan, L., Caldelli, R., Del Bimbo, A., Serra, G.: A SIFT-based forensic method for copy-move attack detection and transformation recovery. IEEE Trans. Inf. Forensics Secur. **6**(3), 1099–1110 (2011)
2. Ardizzone, E., Bruno, A., Mazzola, G.: Copy-move forgery detection by matching triangles of keypoints. IEEE Trans. Inf. Forensics Secur. **10**(10), 2084–2094 (2015)
3. Asghar, K., Habib, Z., Hussain, M.: Copy-move and splicing image forgery detection and localization techniques: a review. Aust. J. Forensic Sci. **49**(3), 281–307 (2017)
4. Bayram, S., Sencar, H.T., Memon, N.: An efficient and robust method for detecting copy-move forgery. In: IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2009, pp. 1053–1056. IEEE (2009)
5. Birajdar, G.K., Mankar, V.H.: Digital image forgery detection using passive techniques: a survey. Digit. Investig. **10**(3), 226–245 (2013)
6. Bunk, J., et al.: Detection and localization of image forgeries using resampling features and deep learning. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 1881–1889. IEEE (2017)
7. Christlein, V., Riess, C., Jordan, J., Riess, C., Angelopoulou, E.: An evaluation of popular copy-move forgery detection approaches. IEEE Trans. Inf. Forensics Secur. **7**(6), 1841–1854 (2012)
8. Costanzo, A., Amerini, I., Caldelli, R., Barni, M.: Forensic analysis of SIFT keypoint removal and injection. IEEE Trans. Inf. Forensics Secur. **9**(9), 1450–1464 (2014)
9. Cozzolino, D., Poggi, G., Verdoliva, L.: Efficient dense-field copy-move forgery detection. IEEE Trans. Inf. Forensics Secur. **10**(11), 2284–2297 (2015)
10. Dwibedi, D., Misra, I., Hebert, M.: Cut, paste and learn: surprisingly easy synthesis for instance detection. In: The IEEE International Conference on Computer Vision (ICCV), October 2017
11. Farid, H.: Seeing is not believing. IEEE Spectr. **46**(8) (2009)
12. Fridrich, A.J., Soukal, B.D., Lukáš, A.J.: Detection of copy-move forgery in digital images. In: Proceedings of Digital Forensic Research Workshop. Citeseer (2003)
13. Gupta, A., Vedaldi, A., Zisserman, A.: Synthetic data for text localisation in natural images. In: IEEE Conference on Computer Vision and Pattern Recognition (2016)
14. Huang, D.Y., Huang, C.N., Hu, W.C., Chou, C.H.: Robustness of copy-move forgery detection under high JPEG compression artifacts. Multimed. Tools Appl. **76**(1), 1509–1530 (2017)
15. Ke, Y., Sukthankar, R., Huston, L.: An efficient parts-based near-duplicate and sub-image retrieval system. In: Proceedings of the 12th Annual ACM International Conference on Multimedia, pp. 869–876. ACM (2004)

16. Li, J., Li, X., Yang, B., Sun, X.: Segmentation-based image copy-move forgery detection scheme. IEEE Trans. Inf. Forensics Secur. **10**(3), 507–518 (2015)
17. Lin, T.-Y., et al.: Microsoft COCO: common objects in context. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8693, pp. 740–755. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10602-1_48
18. Liu, Y., Guan, Q., Zhao, X.: Copy-move forgery detection based on convolutional kernel network. Multimed. Tools Appl. 1–25 (2017)
19. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3431–3440 (2015)
20. Mahdian, B., Saic, S.: Detection of copy-move forgery using a method based on blur moment invariants. Forensic Sci. Int. **171**(2), 180–189 (2007)
21. Mahmood, T., Nawaz, T., Irtaza, A., Ashraf, R., Shah, M., Mahmood, M.T.: Copy-move forgery detection technique for forensic analysis in digital images. Math. Probl. Eng. **2016**, 1–13 (2016)
22. Manu, V.T., Mehtre, B.M.: Detection of copy-move forgery in images using segmentation and SURF. In: Thampi, S., Bandyopadhyay, S., Krishnan, S., Li, K.C., Mosin, S., Ma, M. (eds.) Advances in Signal Processing and Intelligent Recognition Systems. AISC, vol. 425, pp. 645–654. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-28658-7_55
23. Noh, H., Hong, S., Han, B.: Learning deconvolution network for semantic segmentation. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1520–1528 (2015)
24. Pérez, P., Gangnet, M., Blake, A.: Poisson image editing. ACM Trans. Graph. (TOG) **22**, 313–318 (2003)
25. Pun, C.M., Yuan, X.C., Bi, X.L.: Image forgery detection using adaptive oversegmentation and feature point matching. IEEE Trans. Inf. Forensics Secur. **10**(8), 1705–1716 (2015)
26. Ryu, S.-J., Lee, M.-J., Lee, H.-K.: Detection of copy-rotate-move forgery using Zernike moments. In: Böhme, R., Fong, P.W.L., Safavi-Naini, R. (eds.) IH 2010. LNCS, vol. 6387, pp. 51–65. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16435-4_5
27. Shivakumar, B., Baboo, S.: Detection of region duplication forgery in digital images using SURF. Int. J. Comput. Sci. Issues **8**(4), 199–205 (2011)
28. Silva, E., Carvalho, T., Ferreira, A., Rocha, A.: Going deeper into copy-move forgery detection: exploring image telltales via multi-scale analysis and voting processes. J. Vis. Commun. Image Represent. **29**, 16–32 (2015)
29. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR abs/1409.1556 (2014)
30. Soni, B., Das, P., Thounaojam, D.: CMFD: a detailed review of block based and key feature based techniques in image copy-move forgery detection. IET Image Process. **12**, 167–178 (2017)
31. Tralic, D., Zupancic, I., Grgic, S., Grgic, M.: CoMoFod—new database for copy-move forgery detection. In: 2013 55th International Symposium on ELMAR, pp. 49–54. IEEE (2013)
32. Warif, N.B.A., et al.: Copy-move forgery detection: survey, challenges and future directions. J. Netw. Comput. Appl. **75**, 259–278 (2016)
33. Wojna, Z., et al.: The devil is in the decoder (2017)

34. Wu, Y., Abd-Almageed, W., Natarajan, P.: Deep matching and validation network: an end-to-end solution to constrained image splicing localization and detection. In: Proceedings of the 2017 ACM on Multimedia Conference, MM 2017, pp. 1480–1502 (2017)
35. Xiao, J., Hays, J., Ehinger, K.A., Oliva, A., Torralba, A.: Sun database: large-scale scene recognition from abbey to zoo. In: 2010 IEEE conference on Computer Vision and Pattern Recognition (CVPR), pp. 3485–3492. IEEE (2010)
36. Yang, B., Sun, X., Guo, H., Xia, Z., Chen, X.: A copy-move forgery detection method based on CMFD-SIFT. Multimed. Tools Appl. **77**, 1–19 (2017)
37. Zampoglou, M., Papadopoulos, S., Kompatsiaris, Y.: Detecting image splicing in the wild (web). In: 2015 IEEE International Conference on Multimedia & Expo Workshops (ICMEW), pp. 1–6. IEEE (2015)
38. Zampoglou, M., Papadopoulos, S., Kompatsiaris, Y.: Large-scale evaluation of splicing localization algorithms for web images. Multimed. Tools Appl. **76**(4), 4801–4834 (2017)
39. Zhou, P., Han, X., Morariu, V.I., Davis, L.S.: Two-stream neural networks for tampered face detection. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 1831–1839. IEEE (2017)
40. Zhu, Y., Shen, X., Chen, H.: Copy-move forgery detection based on scaled ORB. Multimed. Tools Appl. **75**(6), 3221–3233 (2016)