ManTra-Net: Manipulation Tracing Network For Detection And Localization of Image Forgeries With Anomalous Features

Yue Wu*, Wael AbdAlmageed*, and Premkumar Natarajan*§
*USC Information Sciences Institute, Marina del Rey, CA, USA
§Amazon, Manhattan Beach, CA, USA
wuayue@amazon.com, {wamageed, pnataraj}@isi.edu

Abstract

To fight against real-life image forgery, which commonly involves different types and combined manipulations, we propose a unified deep neural architecture called ManTra-Net. Unlike many existing solutions, ManTra-Net is an end-to-end network that performs both detection and localization without extra preprocessing and postprocessing. ManTra-Net is a fully convolutional network and handles images of arbitrary sizes and many known forgery types such as splicing, copy-move, removal, enhancement, and even unknown types. This paper has three salient contributions. We design a simple yet effective self-supervised learning task to learn robust image manipulation traces from classifying 385 image manipulation types. Further, we formulate the forgery localization problem as a local anomaly detection problem, design a Z-score feature to capture local anomaly, and propose a novel long short-term memory solution to assess local anomalies. Finally, we carefully conduct ablation experiments to systematically optimize the proposed network design. Our extensive experimental results demonstrate the generalizability, robustness and superiority of ManTra-Net, not only in single types of manipulations/forgeries, but also in their complicated combinations.

1. Introduction

Image forgery has recently become an epidemic, negatively affecting many aspects of our life, e.g., fake news, Internet rumors, insurance fraud, blackmail, and even academic publications [51]. Yet, most cases of image forgeries are not detected. Just in biomedical research publications alone, 3.8% of 20,621 papers (published in 40 scientific journals from 1995 to 2004) contained problematic figures, with at least half exhibiting features suggestive of deliberate manipulation [12]. In 2014, Stern et al. [41] estimate that each of the retracted articles could account for a mean of $392,582 in direct costs, implying much higher indirect costs caused by misled research—and this is only in the biomedical field and these numbers are five years outdated. It is therefore imperative to develop new algorithms to assist in the fight against image manipulation and forgery.

Many image forgery techniques exist. However, splicing [19, 44, 28], copy-move [18, 43, 37, 46, 45], removal [58], and enhancement [9, 10, 17] are the four that have been studied the most. Both splicing and copy-move involve pasting image content to the target (i.e., forged)
image. However, in splicing the added content is obtained from a different image, while in copy-move it is from the target image. Removal, also known as inpainting, removes a selected image region (e.g., hiding an object) and fills the space with new pixel values estimated from background. Finally, image enhancement is a wide collection of local manipulations, such as sharpening, brightness adjustment, etc. Depending on the characteristics of the forgery, different clues can be used as the foundation for detection/localization. These clues include JPEG compression artifacts [25, 30, 5], edge inconsistencies [39, 53], noise pattern [33, 49, 20], color consistency [21], visual similarity [44, 45, 46], EXIF consistency [28], and camera model [14, 13]. However, real-life forgeries are more complex, as illustrated in Fig. 1, and malicious forgers often use a sequence of manipulations to hide the forgery, including up-to-date techniques such as deep neural network-based (DNN) face swapping [36, 57], as shown in Fig. 1-(c). This compels us to develop new unified forgery detection techniques that are not limited to one or several known manipulation types but capable of handling more complicated and/or unknown types.

Another issue that has often been overlooked is forgery region localization. Most of the existing methods [9, 25, 37, 38, 49] only focus on image-level detection—whether or not an image is forged. Furthermore, methods that provide localization capabilities often rely on heavy, time-consuming pre- and/or post-processing, e.g., patch extraction [53], expectation-maximization [19, 20], feature clustering [14, 11, 32, 28], segmentation [32, 28, 15], etc. Finally, the disconnection between feature learning and forgery mask generation suggests an under-optimized forgery detection and localization method.

In this paper, we address the above issues, and propose a novel solution called ManTra-Net for generalized image forgery localization/detection (IFLD). It detects forged pixels by identifying local anomalous features, and thus is not limited to a specific forgery or manipulation type. It is an end-to-end solution, and thus no need to apply pre- and/or post-processing. It is also composed of all trainable modules, and thus all modules can be jointly optimized towards the IFLD task. The remainder of this paper is organized as follows. Sec. 2 discusses the related works and gives the ManTra-Net overview. Sec. 3 presents our study to obtain robust image manipulation-trace features. Sec. 4 proposes our local anomaly detection network. Sec. 5 shows our experimental results; and we conclude this paper in Sec. 6.

### 2. Manipulation Tracing Network

#### 2.1. Related Works

Table 1 summarizes the most notable image forgery detection and localization work in the last four years. Several trends can be observed — (1) varieties of clue/feature are used, ranging from handcrafted features, such as DCT correlation to completely implicit learned DNN features, (2) even though DNN methods are becoming more popular, no dominant DNN architecture, or more precisely, almost not any two DNN approaches, adopt the same network architecture, and (3) most methods focus on one specific type of forgery. A more comprehensive review can be found in [6].

#### 2.2. Overview

As shown in Fig. 2, the proposed ManTra-Net solution is composed of two sub-networks, i.e., the image manipulation-trace feature extractor that creates a unified feature representation, and the local anomaly detection network (LADN) for directly localizing forgery regions without postprocessing. We make three major contributions to the IFLD community.

First, we reinvent the image manipulation trace feature, which was limited to differentiate a small number of known manipulations [17, 5], but is now capable of distinguishing 385 types of known manipulations, and is robust to encode manipulations of unknown types, even for those DNN-based manipulations (e.g., deep image inpainting) and sequential manipulations (e.g., enhancement, resizing, and compression in a row.) We demonstrate that this feature is suitable to IFLD tasks and that it can be effectively and efficiently learned from the self-supervised learning task – image manipulation classification (IMC).

Second, we abandon the common semantic segmenta-
tion like IFLD formulations [58, 56, 39], but formulate the IFLD task as a local anomaly detection problem to improve the model generalizability. More precisely, we want to learn a decision function mapping from the difference between a local feature and its reference to its forgery label. To fulfill this goal, we invent a simple yet effective LADN architecture that mimics the human decision process by using two novel designs: (1) \( Z \text{Pool2D} \) DNN layer, which standardizes the difference between a local feature and its reference in the \( Z \)-score manner; and (2) the far-to-near analysis, which performs the \( \text{Conv2DLSTM} \) sequential analysis on \( Z \text{Pool2D} \) feature maps pooled from different resolutions.

Finally, we carefully conduct ablation experiments to systematically optimize both the IMC and LADN architectures, and provide theoretical groundings and/or experimental results to support our network designs.

### 2.3. Experimental Setup

To systematically study manipulation trace feature and anomaly detection, we use the following common setup for all ablation experiments, unless otherwise specified.

**For manipulation trace feature**, we use the Dresden Image Database [24] for pristine base images. Training, validation, and testing are divided with respect to image IDs with the ratio of 8:1:1. Each image is further broken into 256×256 patches. After rejecting patches with high homogeneity (i.e., intensity deviation < 32), we have 1.25M patches in total. We synthesize a sample for image manipulation classification by: (1) selecting a random patch \( P \) and a random manipulation \( y(\cdot) \), both in a uniform random manner, (2) applying manipulation \( y \) to \( P \), and 3) cropping a random 128×128 region in \( y(P) \) as \( X \). This \((X, y)\) pair is a sample of input and output for the classification task.

The Kaggle Camera Model Identification (KCMI) dataset [4] is used to check the generalizability and sensitivity of a manipulation classification network. It contains 10 camera models with 2475 samples. To evaluate KCMI performance, we randomly divide the dataset into two halves—one half to fit a \((K=7)\) nearest-neighbor classifier, and the other half to test. The camera model feature is obtained by averaging over all manipulation trace features in the center 512×512 patch of a given image.

**For anomaly detection**, we use four synthetic datasets for training and validation — namely the splicing dataset from [44], the copy-move dataset from [45], the removal dataset synthesized by using built-in OpenCV inpainting function (with Dresden base images), and the enhancement dataset synthesized by using manipulation classification settings discussed previously. More precisely, we synthesize an enhanced sample by (1) introducing a random structured binary mask \( M \) (see \([31]\)), (2) composing a forged image by using \( Z = P \cdot (1-M)+y(P) \cdot M \), where \( P \) and \( y(\cdot) \) are a pristine patch and a random manipulation, respectively. The resulting \((Z, M)\) pair is a sample of input and output for an LADN task. Training patch size is set to 256×256.

In **terms of training settings**, we set batch size to 64 and 1000 batches per epoch, and use the Adam optimizer with the initial learning rate of 1e-4 but without decay. This learning rate will be halved if validation loss fails to improve for 20 epochs. The image manipulation classification and anomaly detection tasks are optimized towards the cross-entropy loss.

### 3. Manipulation-Trace Feature

In this section, we study the image manipulation trace feature extractor (see the yellow shaded block in Fig. 2) via the image manipulation classification problem. Although image manipulation trace feature have been previously used for forgery detection and localization purposes for a long time, the total number of image manipulations was usually below 10 — e.g., [17, 5] use 7 and 9 types, respectively. Such few types of manipulations is clearly inadequate for a unified feature representation. We therefore systematically study manipulations with more types and finer differences, with 385 manipulation types. To the best of our knowledge, we this work is the first to consider this large number of fine-grained manipulation types.

#### 3.1. Study of Backbone Network Architecture

Since no dominant IFLD network architecture (see Table 1) and very few studies on IMC networks, we conduct backbone architecture comparisons among three
networks—VGG [40], ResNet [26], and DnCNN [52], all of which are proposed outside of the IFLD community but used previously for IFLD [20, 28, 44, 45, 55].

For fair comparison, we customize backbone models to have the same receptive field sizes, and similar numbers of filters and hyper-parameters (see Table 2). It is worth noting that all listed manipulation classification models are fully convolutional networks (FCN) (i.e. no down-sampling or Dense layer).

<table>
<thead>
<tr>
<th>FirstBlock</th>
<th>IMC-VGG</th>
<th>IMC-ResNet</th>
<th>IMC-DnCNN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IMC-VGG</td>
<td>IMC-ResNet</td>
<td>IMC-DnCNN</td>
</tr>
<tr>
<td>FirstBlock</td>
<td>16k(3,3)</td>
<td>16k(3,3)</td>
<td>16k(3,3)</td>
</tr>
<tr>
<td>MiddleBlock</td>
<td>16k(3,3)</td>
<td>16k(3,3)</td>
<td>16k(3,3)</td>
</tr>
<tr>
<td>LastBlock</td>
<td>16k(3,3)</td>
<td>16k(3,3)</td>
<td>16k(3,3)</td>
</tr>
</tbody>
</table>

Table 2. IMC-7 network architecture and performance comparisons. Building blocks are shown in brackets. $N@((3,3))$ indicates a Conv2D layer with $N$ filters of kernel size 3-by-3. $k$ in IMC-VGG is the block index, e.g. the number of filters used in block 2 is $32 = 16 \times 2$. $m$ is the unit repetitiveness; e.g. $m = [2, 3, 2]$ indicates three middle blocks repeat the unit 2, 3, and 2 times, respectively. Conv2D of projection shortcut in ResNet is not listed.

To speed up training and offer training to many models, we study the simple IMC-7 problem, i.e., classification on the seven general manipulation families: compression, blurring, morphology, contrast manipulation, additive noise, resampling, and quantization. Specifically, we train each architecture with three models, but only the model with the best validation loss is reported in the lower half of Table 2. It turns out that all three architectures achieve similar IMC-7 performance. However, VGG outperforms the rest with a smaller gap between training and validation, but a much higher accuracy in KCMI testing. We thus use the VGG architecture in the remainder of our studies.

We also study the feature choice of the first layer. We compare the known optimal settings for SRMConv2D from [55] and BayarConv2D from [10] with the classic Conv2D layers, and a combined version of all three, which is simply the feature concatenation as shown in Fig. 2. From Table 3, it is safe to conclude that different feature types make small differences in IMC-7 performance, usually 1% to 2%, while using the combined setting gives the best performance. We therefore use the combined features for the first convolutional layer.

3.2. Study of Fine-Grained Manipulation Types

To make the manipulation trace feature more sensitive and robust, we study the IMC problem for more and finer manipulation types. Specifically, we gradually break down the seven manipulation families (hierarchy level 0) until they are individual algorithms (hierarchy level 5). For example, the blurring family is broken down to Gaussian blurring, box blurring, wavelet denoising, and median filtering for hierarchy level 1. Then we proceed to an even finer level by specifying algorithm parameters, e.g., Gaussian blurring w.r.t. small kernel sizes like 3, 5, and 7 for hierarchy level 2. This continues on until reaching the individual kernel size for hierarchy level 5. The complete hierarchy map is included in our code repository, as there are different hierarchy levels for 7, 25, 49, 96, 185, and 385 classes for manipulation classification.

All IMC models in this study share the same VGG network architecture discussed earlier, except for the number of output classes in the decision block (see Table 2). Their scores are listed in Table 4. Because of the predefined hierarchy map, an IMC trained on hierarchy $i$ can be used to also predict labels of hierarchy $j$ for $i > j$. All underlined scores in Table 4 are obtained in this way. It is clear that fine-grained manipulation classes help improve, not only validation accuracy for lower hierarchies, but also the KCMI accuracy from 57.2% to 82.6%.

<table>
<thead>
<tr>
<th>Hierarchy Level</th>
<th>HL0</th>
<th>HL1</th>
<th>HL2</th>
<th>HL3</th>
<th>HL4</th>
<th>HL5</th>
</tr>
</thead>
<tbody>
<tr>
<td># IMC Classes</td>
<td>7</td>
<td>25</td>
<td>49</td>
<td>96</td>
<td>185</td>
<td>385</td>
</tr>
<tr>
<td>IMC-7 Valid Acc.</td>
<td>93.4%</td>
<td>95.1%</td>
<td>96.1%</td>
<td>96.3%</td>
<td>96.3%</td>
<td>96.2%</td>
</tr>
<tr>
<td>IMC-25 Valid Acc.</td>
<td>85.1%</td>
<td>85.7%</td>
<td>85.4%</td>
<td>85.5%</td>
<td>85.7%</td>
<td>85.7%</td>
</tr>
<tr>
<td>IMC-49 Valid Acc.</td>
<td>77.3%</td>
<td>79.6%</td>
<td>78.9%</td>
<td>79.9%</td>
<td>79.2%</td>
<td>79.2%</td>
</tr>
<tr>
<td>IMC-96 Valid Acc.</td>
<td>72.4%</td>
<td>73.7%</td>
<td>73.2%</td>
<td>73.2%</td>
<td>73.2%</td>
<td>73.2%</td>
</tr>
<tr>
<td>IMC-185 Valid Acc.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IMC-385 Valid Acc.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4. IMC performance analysis w.r.t. manipulation types.

IMC-385 validation accuracy (47.3%) is relatively low. We therefore adjust the baseline IMC-VGG architecture in two orthogonal directions—(1) make it wider [50], i.e., using more filters in each convolutional layer, and (2) make it deeper, i.e., using more convolutional blocks. Both attempts improve the baseline performance, and the combination of wider and deeper (W&D) improves even more. Table 5 shows these results. We therefore use the IMC-VGG W&D architecture excluding the decision block for the manipulation trace feature extractor (see Fig. 2).

3.3. Discussions

The IMC performance can be further improved if a larger receptive field size is used. We, however, stop exploration...
and stick to the IMC-VGG-W&D architecture to ensure the feature sensitivity to small manipulated regions.

Regarding the IMC-385 performance, Fig. 3-(a) illustrates the IMC-VGG-W&D confusion matrix at the hierarchy level 1 (with 25 classes). It is quite close to the identity matrix, and thus the most IMC-385 errors happen within the same type of manipulations, but with different parameters. Indeed, the only salient error in the confusion matrix is to misclassify JPEGCompression to JPEGDoubleCompression, possibly because most pristine images in the Dresden dataset are of the JPEG format, indicating that they are already compressed.

Table 5. IMC-385 performance comparisons using different architectures. Building blocks are shown in brackets. \( N@(3,3) \) indicates a Conv2D layer with \( N \) filters of kernel size 3-by-3.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Baseline</th>
<th>Wider(W)</th>
<th>Deeper(D)</th>
<th>W&amp;D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FirstBlock</strong></td>
<td>16×16/2/ReLU x2</td>
<td>32×32/3/ReLU x2</td>
<td>64×64/3/ReLU x2</td>
<td>128×128/3/ReLU x2</td>
</tr>
<tr>
<td><strong>MiddleBlock</strong></td>
<td>32×32/3/ReLU x2</td>
<td>64×64/3/ReLU x2</td>
<td>128×128/3/ReLU x2</td>
<td>256×256/3/ReLU x2</td>
</tr>
</tbody>
</table>
| **LastBlock** | \( [32×32/3, 
L2Norm] \times 2 \) | \( [256×256/3, 
L2Norm] \times 2 \) | \( [512×512/3, 
L2Norm] \times 2 \) | \( [1024×1024/3, 
L2Norm] \times 2 \) |
| **DecisionBlock** | \( [385×385/3, + \text{Softmax}] \) | \( [385×385/3, + \text{Softmax}] \) | \( [385×385/3, + \text{Softmax}] \) | \( [385×385/3, + \text{Softmax}] \) |
| **#Conv2D** | 11 | 11 | 14 | 14 |
| **#Receptive Field** | \( 23 \times 23 \) | \( 23 \times 23 \) | \( 29 \times 29 \) | \( 29 \times 29 \) |
| **IMC-385 Top-1** | 47.3% | 48.6% | 45.7% | 51.8% |
| **IMC-385 Top-5** | 75.3% | 79.5% | 79.4% | 81.1% |
| **IMC-385 Top-10** | 85.8% | 90.7% | 92.5% | 93.1% |
| **KCMI Test Acc** | 82.6% | 83.1% | 83.0% | 83.6% |

Table 5. IMC-385 performance comparisons using different architectures. Building blocks are shown in brackets. \( N@(3,3) \) indicates a Conv2D layer with \( N \) filters of kernel size 3-by-3.

4. Local Anomaly Detection Network

In this section, we propose a novel deep anomaly detection network architecture. As shown in Fig. 2, it is composed of three stages: (1) adaptation, which adapts the manipulation trace feature for the anomaly detection task; (2) anomalous feature extraction, which is inspired by human thinking and extracts anomalous features; and (3) decision, which holistically consider anomalous features and classify whether a pixel is forged or not. Since both adaptation and decision stages are straightforward, we focus on the discussion of anomalous feature extraction.

4.1. Anomalous Feature Extraction

Given a feature map (e.g. the bottom row in Fig. 3-(b)), how a human identifies potential forged regions. Though this question can be answered differently, one can first identify the dominant feature of an image, and any feature sufficiently different from this dominant feature is thus anomalous. In the rest of section, we follow this intuition and discuss the solutions to the two key tasks (1) what is a dominant feature, and how to compute it, and (2) how to quantify the difference between a local feature and a reference dominant feature, and what is the best way in practice.

Let us start with simple solutions. One choice for the dominant feature is the average feature defined in Eq. (1)

\[
\mu_F = \sum_{i=1}^{H} \sum_{j=1}^{W} F[i, j] / (HW)
\]

where \( F \) is a raw feature tensor of size \( H \times W \times L \). Similarly, one may use the raw difference in Eq. (2) to quantify the difference between a local feature and its reference.

\[
D_F[i, j] = F[i, j] - \mu_F
\]

Considering the generalizability, the normalized \( Z \)-score defined in Eq. (3) works better, (see Table 6)

\[
Z_F[i, j] = D_F[i, j] / \sigma_F
\]

where \( \sigma_F \) is the standard deviation of \( F \) as shown in Eq (4).

\[
\sigma_F^2 = \sum_{i=1}^{H} \sum_{j=1}^{W} F[i, j]^2 / (HW) - \mu_F^2
\]

In practice, we replace \( \sigma_F \) with \( \sigma_F^* \) as shown in Eq. (5)

\[
\sigma_F^* = \text{maximum}(\sigma_F, \epsilon + w_\epsilon),
\]

where \( \epsilon = 1e-5 \) and \( w_\epsilon \) is a learnable non-negative weight vector of the same length as \( \sigma_F \).

To this end, feature \( Z_F \) encodes how different each local feature is from a reference feature, but \( Z_F \) suffers one major drawback when two more regions are manipulated differently. Say an image contains two disjoint forged regions.
$R_1$ and $R_2$, while the rest is pristine background region $B$. Depending on the relative relationship among $\mu_{R_1}$, $\mu_{R_2}$ and $\mu_B$, feature $\mu_F$ may fail to represent the dominant $\mu_B$. To simplify discussion, let $F$’s feature dimension be 1. When $\mu_{R_1} \gg \mu_{R_2} > \mu_B$, $\mu_F$ can be some value much closer to $\mu_{R_2}$ than $\mu_B$, implying $Z_F$ is incapable of capture the anomalous region $R_2$.

One quick remedy is to compute the reference feature from a local but big enough window, which mitigates if not excludes the influence of features from other forged regions. Specifically, we compute the window-wise deviation feature,

$$D_F^{n \times n}[i, j] = F[i, j] - \mu_F^{n \times n}[i, j] \quad (6)$$

where $\mu_F^{n \times n}[i, j]$ is the average feature computed within the $n \times n$ window centered at $(i, j)$ location through the standard AveragePool2D layer. However, we have no idea what $n$ should be for a testing sample. We therefore follow the common multi-resolution analysis (e.g. [47]), and collect a series of Z-score features w.r.t. different window sizes $n_1$ through $n_k$ as shown in Eq. (7).

$$Z_F^k = [Z_F^{n_1 \times n_1}, \ldots , Z_F^{n_k \times n_k}, Z_F] \quad (7)$$

The process of converting from input feature $F$ to a Z-score feature is referred as to $Z$Pool2D in Fig. 2.

Although one can concatenate $Z_F^k$ along the feature dimension and produce a 3D feature (size of $H \times W \times (k+1)L$) to represent the difference feature, this fails to capture the essence of human decision progress – the far-to-near analysis, i.e. one will move closer if he can’t see something clearly. We, therefore, concatenate $Z_F^k$ along the new artificial time dimension and produce a 4D feature of size $(k+1) \times H \times W \times L$. By using the ConvLSTM2D layer [48], the proposed anomaly detection network analyzes the Z-score deviation belonging to different window sizes in a sequential order. In other words, we look into a fine-grained Z-score map if we are uncertain, and thus conceptually follows the far-to-near analysis.

### 4.2. Anomaly Detection Ablation Experiment

We conduct a set of ablation experiments to study the performance of previously mentioned anomalous features using the ManTra-Net solution shown in Fig. 2. To ensure fair comparisons, all experiments (1) differ from each other only in the used anomaly detection feature; (2) share the same pretrained manipulation feature extractor; and (3) the manipulation trace feature extractor is set to non-trainable. One may refer to Sec. 2.3 for other settings.

Table 6 compares all features in terms of validation $F_1$ scores. It is clear that the Z-score difference is better, and that the more window sizes we consider, the better the overall performance is. For the sake of efficiency, we stop analyzing more windows. Compared to the feature-axis-concatenation (FAC), using the time-axis-concatenate (TAC) for $Z_F^k$ feature further boosts performance by roughly 7% in absolute and 15% in relative.

### 5. Experimental Evaluation

We have previously demonstrated the effectiveness of the used image manipulation-trace feature and the local anomaly detection network. In this section, we focus on evaluating the performance of the end-to-end ManTra-Net w.r.t. generalizability, sensitivity, robustness to postprocessing, and standard benchmarks.

Regarding evaluation metrics, we use the pixel-level area under the receiver operating characteristic curve (AUC) unless otherwise specified. It is important to note that due to the nature of local anomaly detection, ManTra-Net will label pristine pixels as forged if they are minorities. However, this behavior should not be penalized. We thus negate a ManTra-Net predicted mask when more than 50% of pixels are forged in ground truth, as suggested in [28].

#### 5.1. Pre-trained Models and Generalizability Test

We train ManTra-Net models in the end-to-end manner using the four synthetic datasets mentioned in Sec. 2.3. The pretrained ManTra-Net models are available at 1.

To evaluate the generalizability of these models, the latest partial convolution-based CNN inpainting method [31] is selected as one typical out-of-domain DNN-based manipulation. In addition, the PhotoShop-battle dataset [27] is also used, because it is large (total 102,028 samples) and diverse (contributed from 31,272 online artists), and it reflects the level of real-life image manipulation. Since it only provides the image-level annotation (i.e., pristine or forged) instead of the pixel-level, we evaluate model performance on this dataset by computing the image-level AUC, where the likelihood that an image is manipulated is simply computed as the average likelihood of all pixels.

As one can see in Table 7, the fully random model trained with full random weights does not generalize well because it overfits to the synthesized data, while the forgery clues presented in the used synthesized dataset are very different from those in the real world. The half freeze model trained by freezing the image manipulation-trace features (IMTF) and with random LADN weights does prevents

---

1 https://github.com/ISICV/ManTraNet.git

9548
overfitting, but eliminates the hope of finding better features for other forgery types, because the manipulation-trace feature is known to be optimized to the enhancement dataset (see the Enhance column in Table 6), but not to splicing, copy-move or removal. In contrast, the half random model that allows these weights to be updated at a lower learning rate of 5e-5 prevents overfitting and converges to a better feature representation for all forgery types. We thus use the ManTra-Net half random model in later experiments.

<table>
<thead>
<tr>
<th>Name</th>
<th>IMTF Setting</th>
<th>Testing F1</th>
<th>[31] F1</th>
<th>[27] AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Random (FR)</td>
<td>Random initialization</td>
<td>71.21%</td>
<td>68.37%</td>
<td>61.85%</td>
</tr>
<tr>
<td>Half Freeze (HF)</td>
<td>Freezed IMC-385</td>
<td>48.39%</td>
<td>72.54%</td>
<td>70.33%</td>
</tr>
<tr>
<td>Half Random (HR)</td>
<td>IMC-385 initialization</td>
<td>68.61%</td>
<td>78.32%</td>
<td>75.88%</td>
</tr>
</tbody>
</table>

Table 7. ManTra-Net performance under different settings.

5.2. Sensitivity and Robustness Evaluation

To evaluate how accurate ManTra-Net is to manipulations of different distortions, we conduct the following sensitivity study: (1) we synthesize manipulated samples using a manipulation function \( f \) and a method parameter \( p \) for 5,000 patches in the Dresden testing split; (2) we evaluate ManTra-Net on this synthesized dataset; and (3) we report its performance as one data point in Fig. 4. As shown in Fig. 4-(a), ManTra-Net is very accurate to additive noise and blurring methods, even for subtle manipulations like \( 3 \times 3 \) GaussianBlur, while less accurate to compression methods, especially when the quality factor is above 95.

In real life, one may disguise a forged image \( X \) with additional post-processing. Here we considered the three common postprocessing methods: (1) resizing \( X \) to a smaller size, (2) compressing \( X \) with a lower quality factor, and (3) smoothing \( X \) around the edges of forgeries. Instead of a raw testing sample from the four synthesized datasets, we feed in the pretrained ManTra-Net with the post-processed version, and compute the testing performance decay. These results are shown in Fig. 4-(b). ManTra-Net’s overall performance almost drops linearly for LinearResize and JPEGCompress, which is much slower than the quadratic pixel reducing rate in resizing. Finally, though local blurring is known to be very effective in fooling edge-based forgery detection methods, ManTra-Net is quite immune to this type of attack.

5.3. Comparison Against SOTA Methods

Following [55], we compare ManTra-Net’s performance against numbers reported in [55], which provides scores of the classic unsupervised methods: ELA [29], NOII [34], CFAI [22], and the latest DNN-based solutions, MFCN [39] and J-LSTM [7] on the four benchmark datasets, namely NIST 2016 [3], CASIA [2], COVERAGE [43], and Columbia dataset [1]. These four datasets contain 564, 6044, 100, and 180 samples, respectively. It worth noting that we (1) use a pretrained model instead of a finetuned one, and (2) evaluate performance on the full dataset instead of a small testing split.

<table>
<thead>
<tr>
<th>Methods</th>
<th>NIST</th>
<th>Columbia</th>
<th>COVERAGE</th>
<th>CASIA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forgery Types</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELA [29]</td>
<td>0%</td>
<td>42.9%</td>
<td>0%</td>
<td>58.3%</td>
</tr>
<tr>
<td>EOH [33]</td>
<td>0%</td>
<td>48.7%</td>
<td>0%</td>
<td>58.7%</td>
</tr>
<tr>
<td>CFAI [22]</td>
<td>0%</td>
<td>50.1%</td>
<td>0%</td>
<td>48.5%</td>
</tr>
<tr>
<td>J-LSTM [7]</td>
<td>72%</td>
<td>76.4%</td>
<td>N/A</td>
<td>61.4%</td>
</tr>
<tr>
<td>RGB-N [55]</td>
<td>72%</td>
<td>93.7%</td>
<td>85.8%</td>
<td>75%</td>
</tr>
<tr>
<td>ManTra-Net</td>
<td>0%</td>
<td>79.3%</td>
<td>0%</td>
<td>81.9%</td>
</tr>
</tbody>
</table>

Table 8. (training%, AUC) performance comparisons. Highest scores and training% are highlighted in blue and red, respectively. Forgery types are color coded as: ■ splicing, ✿ copy-move, ✤ removal, and ◆ enhancement.

These results are listed in Table 8. We rank second-place in the NIST and Columbia databases. A big performance gap between ManTra-Net and that of the RGB-N method is found in the NIST dataset, possibly because this dataset contains many samples forged from the exact same or very similar base images—where finetuning could definitely help. On the Columbia dataset, we fall just slightly behind the best method RGB-N by 3%, since we do not rely on any specific clue. The RGB-N method explicitly analyzes the noise pattern, which is known to be super effective to the Columbia dataset [55].

On the COVERAGE and CASIA datasets, however, we achieve even better performance on a larger evaluation portion than the J-LSTM and RGB-N methods, which both apply dataset finetuning. One possible explanation why we performed better is that images in these dataset are much smaller than those in NIST and Columbia (e.g., a typical CASIA image is of size 256×384, while it is common to see images larger than 1000×1000 in NIST), and are closer to the image size we used in training, which is 256×256.

It is safe to conclude that ManTra-Net: (1) clearly outperforms those classic unsupervised methods by a large margin, and (2) is comparable to those state-of-the-art DNN methods, even though we do not apply any model finetuning or post-processing. Importantly, one especially note-
worry feature is that the proposed ManTra-Net achieves very consistent performance across all testing datasets, indicating that it does generalize well on different datasets. Qualitative results can be found in Fig. 5. In terms of processing speed, ManTra-Net takes roughly 0.8 seconds per image (1024×768) on a single NVIDIA 1080Ti GPU.

5.4. Limitations

Detection of real-life image forgery is a difficult problem. We observed that ManTra-Net may fail when: (1) a forged image is fully regenerated (e.g., using the style transfer [23]), see Fig. 6-(a); (2) a forged image is intentionally contaminated with highly correlated noise, see Fig. 6-(b); and (3) multiple regions are manipulated differently, see Fig. 6-(c). As shown in Fig. 6-(c), both the text region and the wombat region are manipulated. ManTra-Net finds the text region but not the wombat region. We find that a quick remedy is to ask a user to select a region of interest before applying ManTra-Net, and this time we successfully catch the wombat. This indicates that ManTra-Net can be a computer-aid IFLD tool for humans.

6. Conclusion

In this paper, we introduce a novel end-to-end DNN solution to image forgery localization called ManTra-Net. It first extracts image manipulation trace features for a testing image, and identifies anomalous regions by assessing how different a local feature is from its reference features. Our extensive experimental results using only pretrained models demonstrate that the proposed ManTra-Net is sensitive to subtle manipulations, and robust to postprocessing disguising manipulations, and that it attains good generalizability to unseen data and unknown manipulation types, even for those latest DNN-based manipulations like face swapping [36] and deep image inpainting [32]. One may further improve the ManTra-Net performance or adapt it to new forgery types by simply introducing more manipulation types to the IMC task and/or adding more training samples to the end-to-end IFLD task.

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 Projects Agency or the U.S. Government.
References


