RESAMPLING DETECTION OF RECOMPRESSED IMAGES VIA DUAL-STREAM CONVOLUTIONAL NEURAL NETWORK

Antao Zhou\textsuperscript{1,2}, Gang Cao\textsuperscript{1,2,*}, Xianglin Huang\textsuperscript{1,2}, Gege Song\textsuperscript{1,2}, Lifang Yang\textsuperscript{1,2}

\textsuperscript{1}School of Computer Science and Cybersecurity, Communication University of China, Beijing, China. \textsuperscript{2}Key Laboratory of Convergent Media and Intelligent Technology (Communication University of China), Ministry of Education, China.

\textsuperscript{*}Email: gangcao@cuc.edu.cn.

ABSTRACT

Resampling detection plays an important role in identifying image tampering, such as image splicing. Currently, the resampling detection is still difficult in recompressed images, which are yielded by applying resampling and post-JPEG compression to primary JPEG images. Although low quality primary compression benefits the detection, it remains rather challenging due to the widespread use of middle/high quality compression in imaging devices. In this paper, we propose a novel deep learning approach to learn resampling features directly from the recompressed images. To this end, a noise extraction layer based on low-order high pass filters is deployed to yield the image noise residual domain, which is more beneficial to extract manipulation trail features. A dual-stream convolutional neural network (CNN) is presented to capture the resampling trails along different directions, where the horizontal and vertical streams are interleaved and concatenated. Lastly, the learned features are fed into Sigmoid/Softmax layer, which is used as a binary/multiple classifier for achieving the blind detection or parameter estimation of resampling operations, respectively. Extensive experimental results demonstrate that our proposed method could detect resampling effectively in recompressed images and outperform the state-of-the-art detectors.

Index Terms— Image forensics, Resampling detection, Convolutional neural network, Noise extraction, Interleaved stream

1. INTRODUCTION

Digital images play an important part in spreading information, leading public opinion and taking evidence in legal proceedings and criminal investigations. Advances in photo editing and manipulation tools have made it significantly easier to create fake images without leaving any perceptible artifacts than ever before. Malicious image manipulation may lead to serious moral, ethical and legal problems. Therefore, it is significant to detect the image manipulation blindly \cite{1}. Image manipulations typically leave behind unique traces that can be used to detect the type of image editing. Researchers extract features related to these traces and develop associated algorithms to determine the type of processing operation. In the last decade, numerous methods have been developed to detect various types of image manipulations, such as JPEG compression \cite{2,3}, contrast enhancement \cite{4,5,6}, sharpening \cite{7}, median filtering \cite{8,9}, image splicing \cite{10,11,12}, etc.

Forensic detection of image resampling has also drawn wide attention. Existing image resampling detection methods operate by identifying operation traces from spatial \cite{13,14,15,16,17,18} and frequency domains \cite{19,20}. A. C. Popescu et al. propose to detect the statistical correlation of interpolated pixels by expectation maximization (EM) algorithm \cite{13}. Mahdian and Saic propose to detect pixel interpolation traces by capturing the statistical changes on signal covariance structure \cite{17}. In \cite{15}, subspace decomposition and random matrix theory are used to identify resampling traces in upscaled images. Feng et al. exploit normalized energy density to derive a 19-dimensional feature vector, which is fed into a SVM classifier \cite{19}. In addition, Kirchner \cite{20} models the detectable resampling artifacts in spatial and frequency domains by means of the variance of prediction residue.

However, such prior resampling detection methods are rather fragile with post-JPEG compression, even for the quality factors (QF) about 95 \cite{12,25}. The periodic pattern introduced by resampling is disturbed by JPEG blocking artifacts. In order to detect resampling in JPEG images, the periodicity in the second derivative of interpolated images is employed as evidence of resampling detection \cite{21}. Furthermore, Nataraj et al. \cite{22} propose to suppress JPEG periodic patterns by adding Gaussian noise to post-compressed images. However, such prior methods are ineffective in low quality JPEG images. Kirchner and Gloe \cite{23} demonstrated that the previous JPEG compression before resampling can well detect resampling in recompressed images. Although such an approach can detect resampling in the cases of low and moderate quality primary compression, it is invalidated for the high quality case. In \cite{24}, the operation chain including JPEG com-
pression and resampling is detected by the transformed block artifacts extracted from pixel and discrete cosine transforms (DCT) domains.

Recently, convolutional neural network (CNN) is exploited to detect resampling in recompressed images [25, 26]. In order to prevent CNN from tending to learn features representing image content, the constrained convolutional layer is used to suppress image content and learn manipulation features adaptively. Such a method achieves the best performance in resampling detection compared with previous methods.

In this paper, we propose a new CNN architecture to detect resampling in recompressed images. A noise extraction layer consisting of high pass filters is inserted into the CNN architecture. An input image can be converted into noise residuals by such filters for suppressing the influence of image contents. Meanwhile, the the dual-stream CNN is employed to extract the high level feature of resampling traces. We also note that resampling is often enforced locally to adjust region dimensions for creating realistic spliced images. Additionally, the proposed dual-stream CNN method is applied to the detection of such resampling-involved image splicing. Extensive experimental results verify that the proposed resampling detection method can achieve the state-of-the-art performance.

The rest of the paper is organized as follows. Section 2 describes the proposed resampling detection scheme detailedly. Section 3 shows experimental results and discussions. Finally, the concluding remarks are given in Section 4.

2. PROPOSED METHOD

In this section, the proposed image resampling images forensics scheme is presented detailedly.

2.1. Overview of proposed model

We address resampling detection as a pattern classification problem, which would be resolved by deep learning methodology. Figure 1 depicts the overall design of our proposed dual-stream CNN architecture for resampling detection. It consists of three main components. First, due to resampling features exist in inhorizontal and vertical directions independently, the noise is extracted from adjacent pixel differences in both directions by noise extraction layer. Second, the high level representation of image manipulation features are generated by horizontal and vertical streams. In order to capture the correlation of both directions, horizontal and vertical streams are interleaved and concatenated. Finally, the concatenated features of horizontal and vertical streams are fed into the sigmoid/softmax layer. Note that sigmoid layer is used for binary classification, and softmax layer is used for multiple classifications. Below we present a detailed introduction of such three components and the different layers used in our CNN architecture.

Fig. 1. Proposed dual-stream CNN-based resampling detector. H1-H5, I1-I4 and V1-V5 represent convolution layers of horizontal, interleaved and vertical streams, respectively. GAP means global average pooling.
2.2. Noise extraction

Resampling detection is typically interfered by image content which should be suppressed. It has been found that the resampling trace generally exist in the redundant domain of images and is irrelevant to image content. In our approach, noise is modeled by the residual between a pixel and its estimation yielded by interpolating only neighboring pixels. To accomplish this, a special convolution layer is set as the first layer of the network, namely noise extraction layer. As shown in Figure 2, two low-order high pass filters are chose as convolution kernels of noise extraction layers. Such a layer would not require training and bias. The adjacent pixel differences in horizontal and vertical directions can be captured by the filters shown in Figures 2(a) and (b), respectively. More specifically, a patch sized \(256 \times 256\) pixels from a grayscale input image is first convolved with \(1 \times 3\) and \(3 \times 1\) filters with stride 1 and padding 1. Such filters could learn the prediction error features between the estimated central pixel and its local neighbors. As a result, the noise extraction layers would yield a noise map of prediction residuals with dimension of \(256 \times 256 \times 1\).

2.3. Feature extraction

2.3.1. Horizontal and vertical streams

Noise from horizontal and vertical directions is extracted through the noise extraction layer. Then high level representation of resampling trace would be extracted from such noise. Experimental observations show that, a part of the evidence would be lost and the overall resampling detection performance would degrade if a single, instead of double, direction(s) of noise gradient was used. Therefore, two identical pipelines (horizontal stream and vertical stream) are designed to extract features independently from different directions, and the weights of two pipelines are not shared. As shown in Figure 1, the horizontal and vertical streams are composed of five similar groups. Each group consists of four layers, namely convolution layer, batch normalization layer, activation layer and pooling layer. The fifth group receives additional features from the interleaved stream. Finally, the features generated by the horizontal/vertical and interleaved streams are concatenated.

2.3.2. Interleaved stream

In order to determine the resampling behavior, it requires to fuse the two directional features for making a decision together. So, we propose a feature fusion strategy, namely interleaved stream, to concatenate such two features. The interleaved stream consists of four similar groups. The first group consists of a convolutional layer, a batch normalization layer, and an activation layer. The remaining groups have an additional pooling layer. The features from the first group in horizontal and vertical streams are concatenated, and then enforced by \(1 \times 1\) convolutional kernels with stride 1. Such a type of convolution kernel weights each position of the two feature graphs linearly for fusion. Then the remaining three groups extract advanced representation of the fused features. Finally, the feature map output from interleaved stream is interpolated back to the horizontal and vertical streams. As cooperative learning, the horizontal and vertical streams are aware of each other without affecting the feature extraction in each direction.

2.4. Classification

The final features learned from the previous layers are fed into the classifier, which is a fully connected layer using softmax or sigmoid functions. Through such a classifier, the probability that the feature belongs to each category is obtained, and the most probable category is the result of the classifier. The two involved functions are

\[
P(y = 1|x) = \frac{1}{1 + e^{-z}} \quad (1)
\]

\[
P(y = j|x) = \frac{e^{z_j}}{\sum_{k=0}^{K} e^{z_k}} \quad (2)
\]

Eq.(2.1) is a sigmoid function used in binary classification, where \(z\) is the value of neurons in the full connected layer. \(P(y = 1|x)\) is the probability that \(x\) belongs to the positive category. Eq.(2.2) is a softmax function used in multiple classification, where \(z_j\) is the value of the \(j\)th neuron in the full connected layer. \(P(y = j|x)\) belongs to the probability that \(x\) is the \(j\)th category.

2.5. Layer architectures

2.5.1. Convolution layers

The convolution layer in CNN is used to extract features. The convolutional operation between the input feature maps and a convolutional layer within the CNN architecture is defined by

\[
F_j^{(n)} = \sum_{k=0}^{K} F_k^{(n-1)} * \omega_{kj}^{(n)} + b_j^{(n)} \quad (3)
\]

where \(\ast\) indicates a 2d convolution operation. \(F_k^{(n-1)}\) signifies the \(k\)th feature map generated in the \((n-1)\)th layer. \(\omega_{kj}^{(n)}\) indicates the \(k\) channel of the \(j\)th convolution kernel in the \(n\)th layer.
layer. \( b^{(n)} \) is the \( j \)th bias in the \( n \)th layer, \( F^{(n)} \) is the \( j \)th feature map generated in the \( n \)th layer.

Three convolution kernels with different sizes \((1 \times 1, 3 \times 3 \text{ and } 5 \times 5)\) are used in the proposed model. In horizontal/vertical stream, the convolution layers “H/V1~5” are with 64, 64, 64, 128 filters of size \(5 \times 5 \times 1, 3 \times 3 \times 64, 5 \times 5 \times 64, 3 \times 3 \times 64 \) and \(3 \times 3 \times 128\), respectively. The output dimensions of such convolutional layers are \(256 \times 256 \times 64, 127 \times 127 \times 64, 63 \times 63 \times 64, 31 \times 31 \times 64, 15 \times 15 \times 64\), respectively. In interleaved stream, the convolution layers “I1~4” are with 64 filters of size \(1 \times 1 \times 128, 5 \times 5 \times 64, 5 \times 5 \times 64, 3 \times 3 \times 64\), respectively. Their output dimensions are \(127 \times 127 \times 64, 127 \times 127 \times 64, 63 \times 63 \times 64, 31 \times 31 \times 64\), respectively. The stride is set as 1 in all convolution layers.

### 2.5.2. Batch normalization layers

In order to solve internal covariate shift, which is the change of input data distribution in each layer during the training of CNN, we need to normalize the feature maps generated by the convolutional layers. To do this, a batch normalization layer is used between the convolution layer and the activation layer. The batch normalization operation within the CNN architecture is given from Eq.(2.4) to (2.7).

First, the mean and variance of all data in a batch are calculated as shown in Eq. (2.4) and (2.5).

\[
\mu = \frac{1}{m} \sum_{i=0}^{m} x_i
\]

(4)

\[
\sigma^2 = \frac{1}{m} \sum_{i=0}^{m} (x_i - \mu)^2
\]

(5)

where \( m \) denotes the number of data in a batch, \( x_i \) signifies the \( i \)th data in a batch, \( \mu \) and \( \sigma^2 \) indicate the mean and variance, respectively.

Then each data is normalized to generate a new data \( \hat{x}_i \) with a mean of 0 and a variance of 1, as shown in Eq. (2.6).

\[
\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}
\]

(6)

where \( \epsilon \) is a small floating-point number greater than 0 to prevent dividing by zero errors.

Finally, all data is scaled and shifted.

\[
y_i = \gamma \hat{x}_i + \beta
\]

(7)

where \( y_i \) indicates the \( i \)th output of the batch normalization layer. \( \gamma \) and \( \beta \) are parameters which are learned by network.

The newly generated data can make better use of the non-linear function of the activation function. The small changes in front layer parameters from causing huge changes in back layer parameters is prevented by batch normalization layer.

### 2.5.3. Activation layers

In order to compensate for the expressive deficiency of linear models, the convolution layer is generally followed by an activation layer containing a non-linear function called activation function. The features generated by convolution layer are transformed into another space by activation function, and the data can be better classified.

TanH, Sigmoid, and ReLu are the widely applied activation functions. The data is squashed to \([-1,1]\) by TanH function. TanH has a good performance for the features with significant difference, because it can enlarge continuously the effect of features. Besides, TanH is preferred to Sigmoid in practical application due to TanH’s mean value is 0. Although the training speed of ReLu function is faster than that of TanH, it is fragile in training and can not use a larger learning rate. Hence, activation layers are equipped with the TanH function in our network.

### 2.5.4. Pooling layers

The pooling layer aims to reduce the dimensionality of the feature maps. This reduces the computational cost of training and decreases the chances of over-fitting. In our CNN, two different types of pooling is applied, i.e., max pooling and global average pooling. Max pooling, which retains the maximum value within the local neighborhood of the sliding window, is applied to all pooling layers except for the fifth group in the horizontal and vertical streams. Max pooling layers use kernels of size \(3 \times 3\) and stride of 2, which is the smallest one to capture the notion of left/right, up/down and center. Global average pooling downsamples the feature maps to 1 via the average pooling and it can replace full connection layer to reduce model parameters. Note that global average pooling is only used in the last pooling layer of horizontal and vertical streams.

### 3. EXPERIMENTAL RESULTS

In this section, we first describe the image dataset used in our experiments and the detail setups of the proposed model, and then present extensive experiments to show the effectiveness of the proposed model.

#### 3.1. Performance measure and experimental details

In order to thoroughly measure our model and the baselines, the accuracy is employed as evaluation metrics. For each image, we generate the highest ranked labels and compare the generated labels to the ground truth labels. The accuracy is the number of annotated correctly labels divided by the number of generated labels.

All models in our experiment are trained and tested based on Keras with a single NVIDIA Titan GPU. The network is trained from scratch using the annotated training images. The
weights of all network parameters are initialized by Xavier initialization. We train CNN model using stochastic gradient descent with a batch size of 32 images, momentum [29] of 0.9, and weight decay of 1e-5. The learning rate is initialized at 0.01. Besides, step decay of learning rate was used in our method. We gradually reduced the learning rate during training. We divided the learning rate by 10 when the validation loss stopped improving for three epochs. The number of training epochs is 30 epochs.

3.2. Resampling detection with recompression

In this section, we set up two different sets of experiments on the detection of resampling in the recompressed image. First, we evaluate the performance of our model by specifying the resampling parameter and the quality factor of JPEG compression. In the second part, we challenge a difficult task that the resampling parameters and JPEG quality factors are chosen randomly. The performance of the proposed method on different size images is also evaluated.

3.2.1. On fixed parameters

In our first set of experiments, we use our proposed CNN architecture to detect resampling operation in the recompressed images with different scaling factors and JPEG compression quality factors.

In order to conduct this experiment, we collect 5,000 RAW images of size 3154 × 5286. All operations are performed on RGB color images for a more realistic simulation of the image processing. Each image is divided into 512 × 512 pixels subimages, and seven central subimages are retained. Then JPEG compression is performed using a quality factor in the 95-97 range in order to simulate the built-in compression parameters of cameras. This eventually yields 35000 unaltered image blocks. Next, we created several sets of resampled and recompressed images. We consider the image resampling processing using bilinear interpolation with three different scaling factors, i.e., 50%, 120% and 150%. Each unaltered block is firstly rescaled using each of three different scaling factors, and then the central 256 × 256 block of the scaled block is retained to form the final database. After this, each resampled image is compressed and their corresponding

| Operation parameters | downscale50 | upscale120 | upscale150 |
|----------------------|-------------|------------|------------|
|                      | Q50         | Q60        | Q70        | Q80        | Q90 | Q50         | Q60        | Q70        | Q80        | Q90 | Q50         | Q60        | Q70        | Q80        | Q90 |
| Bayar’s method [25]  | 92.88       | 94.94      | 97.25      | 99.43      | 96.63 | 94.59       | 98.30      | 99.17      | 99.71      | 97.10 | 98.07       | 99.13      | 99.59      | 99.92      |
| Proposed method      | 96.51       | 98.29      | 99.23      | 99.68      | 99.96 | 97.32       | 97.92      | 99.73      | 99.75      | 99.95 | 97.19       | 98.37      | 99.15      | 99.88      | 99.92 |

Finally, we create 16 different databases each of which consists of 70000 images (35000 resampled image and 35000 unresampled images). These images are converted to grayscale by retaining only its green color layer of each image. Each database is divided into three parts: 50000 images are used for training, 10000 images are used for validation, and the rest part is used for testing.

In order to evaluate the effectiveness of the proposed model, such a model is compared with constrained convolutional neural networks [25] using both the same training and testing datasets. The comparison results are illustrated in Table 1. The proposed CNN is superior to constrained convolutional neural networks. Specially, our CNN achieves better identification rates for lower upsampling and lower quality factors. It can be noted from Table 1 that our method is similar to Bayar’s method [25] in performance with high quality post-compression. However, under the post-compression of medium and low quality factors, our method shows a clear advantage. In the case of a QF of 50 and an upsampling of 120, our method improves performance by nearly 4% compared to Bayar’s [25] method; when the downsampling is 50, the performance of our method is also improved by nearly 4%.

3.2.2. On random parameters and different image sizes

In the previous experiments, one certain scaling factor and quality factor are only included in each of databases. Next, we assess the performance of our approach at performing image resampling detection in a more complex scenario where the quality factors and the resampling parameters are chosen arbitrarily. Additionally, the effectiveness of our method is also evaluated under different sizes of images. To accomplish this, we create a training and testing datasets using the same unaltered 512 × 512 image blocks that we collected in the previous experiment. The images are firstly sampled using bilinear interpolation with the upsampling and downsampling scaling factor which is chosen uniformly at random from the range of possible values. The range of upsampling and downsampling factors is respectively from 110% to 200% and from 50% to 90%, and the interval is 10. Then we retain the central blocks of the resampled images and their corresponding
unaltered version with different dimension, i.e., 256 × 256, 128 × 128, 128 × 128. After this, each of the image blocks is compressed using selected quality factor arbitrarily in the range of 50-100 with the interval of 10.

Finally, three databases with different dimension are created, where each database corresponds to three sub databases, i.e., upsampling-jpeg, downsampling-jpeg and unaltered-jpeg. Each sub database consists of 35,000 images in which 25,000 images are used for training, 5,000 images are used for validation, and the rest part is used for testing. We compare the results of our proposed method and Bayar’s method [25] under images with different dimension, as shown in Table 2. The experimental results demonstrate that even in this more challenging scenario, our CNN is still able to accurately identify image resampling. Our approach increases the overall classification rate from 88.46% to 99.07%. Although the lowest detection accuracy was 88.46% for our approach, it is still very high compared with Bayar’s method. We can notice that the Bayar’s method performance degrades severely when reducing the dimension of image.

### 3.3. Resampling detection with post-processing

In order to evaluate the performance of our method in another challenging scenario, we used our CNN to detect image resampling when one manipulation is applied for resampled images, followed by JPEG recompression. To demonstrate this, we conducted another experiment where each image patch is first resampled, then it is edited by one manipulations, and finally JPEG compressed using different quality factors. Five editing operations and parameters are listed in Table 4, and they are Gamma correction (GC), Mean filtering (MeanF), Gaussian filtering (GF), Median filtering (MedF) and Wiener filtering (WF). We create data for this experiment by using 35,000 unaltered 512 × 512 image blocks collected from the first experiment (Section 3.2.1). The image blocks are firstly sampled using bilinear interpolation with the upsampling and downsampling scaling factor which is randomly chosen. The range of upsampling and downsampling factors are respectively from 110% to 200% and from 50% to 90%, and the interval is 10. Then resampled image blocks and their corresponding unaltered version are modified with using a specific manipulation, the associated parameter was chosen uniformly at random from the set of possible values. After this, each manipulated image block is compressed using selected quality factor arbitrarily in the range of 50-100 with the interval of 10 and then the central 256 × 256 block is retained. Finally, five databases are built which are GC, MeanF, GF, MedF and WF. Table 4 depicts the detection rate for our proposed method and Bayar’s method [25] when one manipulation is applied for resampled images. From Table 3, the performances of our method and Bayar’s method have different degree of reduce for different post-processing operations. Noticeably, our proposed method is superior to Bayar’s method. For upsampled images, our proposed CNN can achieve at least 87.90% (MedF) detection rate with all types of manipulations, and it can achieve 97.85% (GC) detection rate. Both values have significantly higher than 81.89% and 96.50% of Bayar’s method. Although the influence of post-processing operation on downsampling is greater than that on upsampling, our proposed method results are much bet-

### Table 2. Resampling detection accuracy of different methods in different image sizes.

| Image size    | 256 × 256 | 128 × 128 | 64 × 64 |
|---------------|-----------|-----------|---------|
| Resampling type | Upsampling | Downsampling | Upsampling | Downsampling | Upsampling | Downsampling |
| Bayar’s method [25] | 97.29 | 97.24 | 95.19 | 91.93 | 88.19 | 80.03 |
| Proposed method | 99.07 | 98.45 | 97.06 | 95.47 | 92.49 | 88.46 |

### Table 3. Resampling detection accuracy of different methods against different post-processing.

| Post-processing type | GC | MeanF | GF | MedF | WF |
|----------------------|----|-------|----|------|----|
| Resampling type | Up | Down | Up | Down | Up | Down | Up | Down | Up | Down |
| Bayar’s method [25] | 96.50 | 95.46 | 81.89 | 67.03 | 85.32 | 73.41 | 82.39 | 67.91 | 84.81 | 71.14 |
| Proposed method | 97.85 | 97.37 | 86.69 | 73.88 | 89.14 | 84.26 | 87.90 | 77.12 | 90.45 | 84.46 |

### Table 4. Parameters of image post-processing operations used in the experiments.

| Operation type | Parameters |
|----------------|------------|
| Gamma correction(GC) | γ: 0.5, 0.6, 0.7, 0.8, 0.9, 1.2, 1.4, 1.6, 1.8, 2.0 |
| Mean filtering(MeanF) | window size: 3 × 3, 5 × 5, 7 × 7 |
| Gaussian filtering(GF) | window size: 3 × 3, 5 × 5, 7 × 7, σ: 0.8-1.6 |
| Median filtering(MedF) | window size: 3 × 3, 5 × 5, 7 × 7 |
| Wiener filtering(WF) | window size: 3 × 3, 5 × 5, 7 × 7 |
Table 5. Confusion matrix for identifying resampling parameters using the proposed method. The relative number of correct and incorrect classifications are shown and the number are written in percent. Tru and Det represent the true label and detection label, respectively.

| Tru\Det | 50   | 60   | 70   | 80   | 90   | 110  | 120  | 130  | 140  | 150  | 160  | 170  | 180  | 190  | 200  |
|---------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 50      | 91.56| 4.32 | 2.54 | 0.76 | 0.4  | 0.14 | 0.02 | 0.1  | 0.02 | 0    | 0.04 | 0.04 | 0.06 | 0    |      |
| 60      | 4.04 | 93.2 | 0.96 | 0.26 | 0.18 | 0.08 | 0.04 | 0.04 | 0.12 | 0.02 | 0.04 | 0.04 | 0.08 | 0    |      |
| 70      | 2.3  | 1.5  | 93.72| 1.28 | 0.7  | 0.1  | 0.1  | 0.06 | 0.08 | 0.02 | 0.02 | 0.06 | 0.02 | 0.02 |      |
| 80      | 0.62 | 0.86 | 1.5  | 95.46| 0.32 | 0.02 | 0.02 | 0.1  | 0.04 | 0.02 | 0    | 0.04 | 0.04 | 0.08 | 0    |
| 90      | 0.36 | 0.3  | 0.84 | 0.88 | 96.9 | 0.26 | 0.12 | 0.08 | 0.02 | 0.02 | 0.02 | 0.06 | 0.02 | 0.06 | 0    |
| 110     | 0.08 | 0.3  | 0.16 | 0.46 | 0.46 | 97.12| 0.56 | 0.26 | 0.14 | 0.14 | 0.02 | 0.06 | 0.14 | 0.04 |      |
| 120     | 0    | 0    | 0.1  | 0.14 | 0.06 | 0.58 | 98.38| 0.38 | 0.06 | 0.1  | 0.04 | 0.06 | 0.04 | 0.04 | 0.02 |
| 130     | 0.08 | 0.02 | 0.02 | 0    | 0.06 | 0.14 | 0.24 | 98.86| 0.1  | 0.16 | 0.08 | 0.08 | 0.1  | 0.04 | 0.02 |
| 140     | 0.06 | 0.02 | 0.08 | 0.18 | 0.08 | 0.18 | 0.12 | 0.42 | 97.58| 0.46 | 0.24 | 0.34 | 0.1  | 0.06 | 0.08 |
| 150     | 0.04 | 0.08 | 0.02 | 0    | 0.06 | 0.08 | 0.08 | 0.1  | 0.24 | 98.64| 0.08 | 0.32 | 0.06 | 0.14 | 0.06 |
| 160     | 0    | 0    | 0    | 0    | 0.02 | 0.02 | 0.06 | 0.14 | 0.14 | 0.14 | 98.92| 0.06 | 0.36 | 0.12 | 0.1  |
| 170     | 0.04 | 0.02 | 0.02 | 0    | 0.06 | 0.02 | 0.02 | 0.02 | 0.06 | 0.18 | 0.12 | 98.96| 0.12 | 0.24 | 0.1  |
| 180     | 0.02 | 0.02 | 0    | 0.04 | 0.04 | 0    | 0.06 | 0.12 | 0.14 | 0.14 | 98.98| 0.16 | 0.32 | 0.16 | 0.1  |
| 190     | 0.02 | 0    | 0    | 0.02 | 0.02 | 0.02 | 0.02 | 0.1  | 0.08 | 0.1  | 0.12 | 98.98| 0.16 | 0.16 | 0.1  |
| 200     | 0.02 | 0    | 0    | 0    | 0    | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.08 | 0.1  | 0.16 | 0.18 | 99.58|

Table 6. Average detection accuracy for upsampling and downsampling on image blocks with size of 64×64 pixels.

|          | Upsampling | Downsampling |
|----------|------------|--------------|
| Accuracy | 89%        | 77%          |

3.4. Estimation of resampling parameters

In the previous experiments, the proposed CNN is effective at detecting image resampled features in recompressed images with different scaling factors and JPEG compression quality factors. In this part, in order to evaluate the performance of the proposed approach in resampling parameters evaluation, two multiclass classifiers are respectively performed for parameter estimation of upsampling and downsampling.

Similarly to the previous set of experiments, 35000 unaltered image blocks of size 512×512 from the first experiment are used to build the database. The images are respectively sampled using bilinear interpolation with fifteen different scaling factors, i.e., 50%, 60%, 70%, 80%, 110%, 120%, 130%, 140%, 150%, 160%, 170%, 180%, 190% and 200%. And then the central 256×256 blocks of the sampled image is retained to form the final database. Next the generated blocks of size 256×256 pixels are compressed using a quality factor which is randomly selected in the range of 50-100 at the interval of 10. Eventually, 15 different databases are created. In the each database, we convert these images to grayscale by retaining only its green color layer of each image, and we used 25,000 images for the training, 5,000 images for the validation and 5,000 images for testing.

Table 5 shows the confusion matrix achieved by the proposed CNN. Actuals belong on the side of the confusion matrix and predictions are across the top. The accuracies for the proposed CNN with different scaling factors are greater than 91% and the scaling factor of 200 performs the best results of 99.58%. For the classifier some images with different scaling factors, such as the scaling factor of 60 and 70, seem to be difficult to distinguish from the scaling factor of 50.

3.5. Image splicing detection with resampling

We assessed the performance of our approach at performing image splicing detection. In this experiment, 64×64 is chosen as a small size to detect resampling. In order to train the variant CNN, we created a dataset using the same unaltered 512×512 pixels image blocks that we collected in the first experiment. These image blocks are first cut to 64×64 pixels subblocks, and then 300,000 subblocks are randomly selected, followed by resampling and JPEG recompression. Scaling factors and quantity factors are randomly chosen in the range of 0.5 to 2 at the interval of 0.01 and the range of 50 to 100 at the interval of 1, respectively. Finally, we create a database which consists of 900,000 images of size 64×64 pix-
The Wild forensics dataset \cite{30, 10} is applied to verify the effectiveness of the proposed approach in image splicing detection. The tampered areas of the test images are resampled by unknown parameters and compressed by jpeg. As illustrated in Figure 3, our method successfully detects and locates the resampling region in some images, which indicates that the resampling feature is helpful for image splicing detection. However, such a method can only detect the resampled part of the resampled image. If the mosaic part has not been resampled or the image has been processed by a variety of post-processing such as sharpening and filtering, our method will not work.

3.6. Effectiveness validation of model architecture

The structural design of a CNN’s architecture has a large impact on its final accuracy. We conduct several sets of additional experiments related to the structural design of our CNN’s architecture. In this subsection, we try to present some experimental results to further validate the rationality of the proposed model. Three parts of the proposed model have been considered, including the noise extraction layers, horizontal stream or vertical stream and the activation functions. The sub database (Gamma correction, down-sampling) that we generated in the previous experiment in Section 3.4 is utilized to accomplish this experiment.

3.6.1. Effectiveness of noise extraction layers

As described in Section 2.2, two low-order high pass filters are used to extract noise maps in our proposed method. A set of experiments are conducted to prove the effectiveness of noise extraction layers in this section. We consider the three
different cases about the noise extraction layer, including using two high-order high pass filters (denoted as “high-order”) from Figure 4, using two low-order high pass filters that used in our method (“low-order”) and removing the noise extraction layer (“no noise layer”).

The experimental results are shown in Figure 5. We can observe that low-order high pass filter achieves the best performance, and its accuracy is distinctively higher than other methods. The accuracy of high-order high pass filter is slightly higher than removing noise extraction layer.

3.6.2. Performance of horizontal/vertical and interleaved streams

In order to assess the performance of horizontal stream and vertical stream, four cases are studied, which are only using horizontal stream, only using vertical stream, using horizontal and vertical stream without interleaved stream, and using horizontal and vertical stream with interleaved stream. Table 7 depicts that the performance of our model is superior to other model, and the accuracy is 97.37%. The accuracies of horizontal and vertical stream models are 95.52% and 96.25%, respectively. Both methods are significantly lower than that in the other two methods. The primary reason is that only one directional feature is considered. The accuracy rate typically greater than 97% when the horizontal and vertical features are considered simultaneously. Noticeably, the performance of model with interleaved stream is better than the model without interleaved stream. This is because that the correlation of directions is further strengthened. Thus, the proposed dual-stream method and interleaved stream can improve the performance of the model.

3.6.3. Selection of activation function

Selection of activation function is an important part in the design of convolutional neural networks and different activation functions have great influence on network performance. In order to evaluate the effectiveness of the proposed model with different activation function, two commonly used activation functions, TanH and ReLu, are chosen. From Figure 6, we can notice that TanH is more suitable for our model. The performance of a network with TanH is better than that of network with ReLu. Furthermore, TanH has better stability and faster convergence than ReLu.

4. CONCLUSION

In this paper, we presented a new deep learning-based method to detect resampling in recompressed images. A noise extraction layer is used to extract noise residual and a dual-stream CNN is proposed to extract resampling features from noise maps in different directions. The experimental results show that the proposed method could not only detect the resampling of recompressed images, but also achieve excellent robustness against some additional post-processing operations. Moreover, we apply the global resampling detection method to resampling parameter estimation and image splicing detection. It should be noted that the proposed detect resampling traces if the image has undergone complex post-processing or anti-forensic manipulations. In the future, we would extend our work to identify more types of resampling in the existence of complex operation chain and anti-forensics attacks.

5. REFERENCES

[1] Matthew C. Stamm, Min Wu, and K. J. Ray Liu, “Information forensics: An overview of the first decade,” IEEE Access, vol. 1, pp. 167–200, 2013.
[2] Hany Farid, “Exposing digital forgeries from jpeg ghosts,” *IEEE Transactions on Information Forensics and Security*, vol. 4, no. 1, pp. 154–160, 2009.

[3] Weiqi Luo, Jiwu Huang, and Guoping Qiu, “Jpeg error analysis and its applications to digital image forensics,” *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 3, pp. 480–491, 2010.

[4] Gang Cao, Yao Zhao, Rongrong Ni, and Xuelong Li, “Contrast enhancement-based forensics in digital images,” *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 3, pp. 515–525, 2014.

[5] Longyin Wen, Honggang Qi, and Siwei Lyu, “Contrast enhancement estimation for digital image forensics,” 2017.

[6] Ping Wang, Fenlin Liu, Chunjun Yang, and Xiangyang Luo, “Blind forensics of image gamma transformation and its application in splicing detection.” *Journal of Visual Communication and Image Representation*, 2018.

[7] Gang Cao, Yao Zhao, and Rongrong Ni, “Detection of image sharpening based on histogram aberration and ringing artifacts,” in *IEEE International Conference on Multimedia and Expo*, 2009, pp. 1026–1029.

[8] Xiangui Kang, Matthew C. Stamm, Anjie Peng, and K. J. Ray Liu, “Robust median filtering forensics using an autoregressive model,” *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 9, pp. 1456–1468, 2013.

[9] Gang Cao, Yao Zhao, Rongrong Ni, Lifang Yu, and Huawei Tian, “Forensic detection of median filtering in digital images,” in *IEEE International Conference on Multimedia and Expo*, 2010, pp. 89–94.

[10] Minyoung Huh, Andrew Liu, Andrew Owens, and Alexei A Efros, “Fighting fake news: Image splice detection via learned self-consistency,” 2018.

[11] Peng Zhou, Xintong Han, Vlad I Morariu, and Larry S Davis, “Learning rich features for image manipulation detection,” 2018.

[12] Jason Bunk, Jawadul H. Bappy, Tajuddin Manhar Mohammed, Lakshmanan Nataraj, Arjuna Flenner, B. S. Manjunath, Shivkumar Chandrasekaran, Amit K. Roy-chowdhury, and Lawrence Peterson, “Detection and localization of image forgeries using resampling features and deep learning,” pp. 1881–1889, 2017.

[13] A. C. Popescu and H. Farid, “Exposing digital forgeries by detecting traces of resampling,” *IEEE Transactions on Signal Processing*, vol. 53, no. 2, pp. 758–767, 2005.
to jpeg post-compression for image resampling detection,” in IEEE International Conference on Acoustics, Speech and Signal Processing, 2017, pp. 2152–2156.

[26] Belhassen Bayar and Matthew C. Stamm, “Constrained convolutional neural networks: A new approach towards general purpose image manipulation detection,” IEEE Transactions on Information Forensics and Security, vol. 13, no. 11, pp. 2691-2706, 2018.

[27] Jessica Fridrich and Jan Kodovsky, “Rich models for steganalysis of digital images,” IEEE Transactions on Information Forensics and Security, vol. 7, no. 3, pp. 868–882, 2012.

[28] Bolin Chen, Haodong Li, and Weiqi Luo, “Image processing operations identification via convolutional neural network,” 2017.

[29] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton, “On the importance of initialization and momentum in deep learning,” in International Conference on International Conference on Machine Learning, 2013, pp. III–1139.

[30] Daniel Moreira, Aparna Bharati, Joel Brogan, Allan Pinto, Michael Parowski, Kevin W. Bowyer, Patrick J. Flynn, Anderson Rocha, and Walter J. Scheirer, “Image provenance analysis at scale,” 2018.