Citation for published version:

Published In:
Multi-task Learning for Detecting and Segmenting Manipulated Facial Images and Videos

Huy H. Nguyen*, Fuming Fang†, Junichi Yamagishi*†¶, and Isao Echizen*†‡

*SOKENDAI (The Graduate University for Advanced Studies), Kanagawa, Japan
†National Institute of Informatics, Tokyo, Japan; ‡The University of Tokyo, Tokyo, Japan
¶The University of Edinburgh, Edinburgh, UK
Email: {nhhuy, fang, jyamagis, iechizen}@nii.ac.jp

Abstract

Detecting manipulated images and videos is an important topic in digital media forensics. Most detection methods use binary classification to determine the probability of a query being manipulated. Another important topic is locating manipulated regions (i.e., performing segmentation), which are mostly created by three commonly used attacks: removal, copy-move, and splicing. We have designed a convolutional neural network that uses the multi-task learning approach to simultaneously detect manipulated images and videos and locate the manipulated regions for each query. Information gained by performing one task is shared with the other task and thereby enhance the performance of both tasks. A semi-supervised learning approach is used to improve the network’s generability. The network includes an encoder and a Y-shaped decoder. Activation of the encoded features is used for the binary classification. The output of one branch of the decoder is used for segmenting the manipulated regions while that of the other branch is used for reconstructing the input, which helps improve overall performance. Experiments using the FaceForensics and FaceForensics++ databases demonstrated the networks effectiveness against facial reenactment attacks and face swapping attacks as well as its ability to deal with the mismatch condition for previously seen attacks. Moreover, fine-tuning using just a small amount of data enables the network to deal with unseen attacks.

1. Introduction

A major concern in digital image forensics is the deepfake phenomenon [1], a worrisome example of the societal threat posed by computer-generated spoofing videos. Anyone who shares video clips or pictures of him or herself on the Internet may become a victim of a spoof-video attack. Several available methods can be used to translate head and facial movements in real time [30, 14] or create videos from photographs [4, 9]. Moreover, thanks to advances in speech synthesis and voice conversion [19], an attacker can also clone a person’s voice (only a few minutes of speech are needed) and synchronize it with the visual component to create an audiovisual spoof [29, 9]. These methods may become widely available in the near future, enabling anyone to produce deepfake material.

Several countermeasures have been proposed for the visual domain. Most of them were evaluated using only one or a few databases, including the CGvsPhoto database [25], the Deepfakes databases [2, 16, 17], and the FaceForensics/FaceForensics++ databases [26, 27]. Cozzolino et al. addressed the transferability problem of several state-of-the-art spoofing detectors [11] and developed an autoencoder-like architecture that supports generalization and can be easily adapted to a new domain with simple fine-tuning.

Another major concern in digital image forensics is locating manipulated regions. The shapes of the segmentation
mas...
3. Proposed Method

3.1. Overview

Unlike other single-target methods [22, 11, 7], our proposed method outputs both the probability of an input being spoofed and segmentation maps of the manipulated regions in each frame of the input, as diagrammed in Figure 2. Video inputs are treated as a set of frames. We focused on facial images in this work, so the face areas are extracted in the pre-processing phase. In theory, the proposed method can deal with various sizes of input images. However, to maintain simplicity in training, we resize cropped images to 256 × 256 pixels before feeding them into the autoencoder. We resize cropped images to maintain simplicity in training, we resize cropped images to 256 × 256 pixels before feeding them into the autoencoder. However, to maintain simplicity in training, we resize cropped images to 256 × 256 pixels before feeding them into the autoencoder.

3.2. Y-shaped Autoencoder

The partitioning of the latent features (motivated by Cozzolino et al.’s work [11]) and the Y-shaped design of the decoder enables the autoencoder to share valuable information between the classification, segmentation, and reconstruction tasks and thereby improve overall performance by reducing loss. There are three types of loss: activation loss \( \mathcal{L}_{act} \), segmentation loss \( \mathcal{L}_{seg} \), and reconstruction loss \( \mathcal{L}_{rec} \).

Given label \( y_i \in \{0, 1\} \), activation loss measures the accuracy of partitioning in the latent space on the basis of the activation of the two halves of the encoded features:

\[
\mathcal{L}_{act} = \frac{1}{N} \sum_i |a_{i,1} - y_i| + |a_{i,0} - (1 - y_i)|, \quad (1)
\]

where \( N \) is the number of samples, \( a_{i,0} \) and \( a_{i,1} \) are the activation values and defined as the \( L_1 \) norms of the corresponding halves of the latent features, \( h_{i,0} \) and \( h_{i,1} \) (given \( K \) is the number of features of \( \{h_{i,0}|h_{i,1}\} \)):

\[
a_{i,c} = \frac{1}{2K} \|h_{i,c}\|_1, \quad c \in \{0, 1\}. \quad (2)
\]

This ensures that, given an input \( x_i \) of class \( c \), the corresponding half of the latent features \( h_{i,c} \) is activated (\( a_{i,c} > 0 \)). The other half, \( h_{i,1-c} \), remains quiesced (\( a_{i,1-c} = 0 \)). To force the two decoders, \( D_{seg} \) and \( D_{rec} \), to learn the right decoding schemes, we set the off-class part to zero before feeding it to the decoders (\( a_{i,1-c} := 0 \)).
We utilize cross-entropy loss as the segmentation loss to measure the agreement between the segmentation mask ($s_i = D_{seg}(\{h_{i,0}, h_{i,1}\})$) and the ground-truth mask ($m_i$) corresponding to input $x_i$:

$$L_{seg} = \frac{1}{N} \sum_{i} |m_i \log(s_i) + (1 - m_i) \log(1 - s_i)|_1.$$  

(3)

The reconstruction loss uses the $L_2$ distance to measure the difference between the reconstructed image ($\hat{x} = D_{rec}(\{h_{i,0}, h_{i,1}\})$) and the original one ($x_i$). For N samples, the reconstruction loss is

$$L_{rec} = \frac{1}{N} \sum_{i} \|x_i - \hat{x}_i\|_2.$$  

(4)

The total loss is the weighted sum of the three activation losses:

$$L = \gamma_{act}L_{act} + \gamma_{seg}L_{seg} + \gamma_{rec}L_{rec}.$$  

(5)

Unlike Cozzolino et al. [11], we set the three weights equal to each other (equal to 1). This is because the classification task and the segmentation task are equally important, and the reconstruction task plays an important role in the segmentation task. We experimentally compared the effects of the different settings (described below).

3.3. Implementation

The Y-shaped autoencoder was implemented as shown in Figure 3. It is a fully connected CNN using $3 \times 3$ convolutional windows (for the encoder) and $3 \times 3$ deconvolutional windows (for the decoder) with a stride of 1 interspersed with a stride of 2. Following each convolutional layer is a batch normalization layer [13] and a rectified linear unit (ReLU) [20]. The selection block allows only the true half of the latent features ($h_{i,y_i}$) to pass by and zeros out the other half ($h_{i,1-y_i}$). Therefore, the decoders ($D_{seg}, D_{rec}$) are forced to decode only the true half of the latent features. The dimension of the embedding is 128, which has been shown to be optimal [11]. For the segmentation branch ($D_{seg}$), a softmax activation function at the end is used to output segmentation maps. For the reconstruction branch ($D_{rec}$), a hyperbolic tangent function (tanh) is used to shape the output into the range $[-1, 1]$. For simplicity, we directly feed normalized images into the autoencoder without converting them into residual images [11]. Further work will focus on investigating the benefits of using residual images in the classification and segmentation tasks.

Following Cozzolino et al.’s work [11], we trained the network using the ADAM optimizer [15] with a learning rate of 0.001, a batch size of 64, betas of 0.9 and 0.999, and epsilon equal to $10^{-8}$.

4. Experiments

4.1. Databases

We evaluated our proposed network using two databases: FaceForensics [26] and FaceForensics++ [27]. The FaceForensics database contains 1004 real videos collected from YouTube and their corresponding manipulated versions, which are divided into two sub-datasets:

- Source-to-Target Reenactment dataset containing 1004 fake videos created using the Face2Face method [30]: in each input pair for reenactment, the source video (the attacker) and the target video (the victim) are different.

- Self-Reenactment dataset containing another 1004 fake videos created again using the Face2Face method; in each input pair for reenactment, the source and target videos are the same. Although this dataset is not meaningful from the attacker’s perspective, it does present a more challenging benchmark than does the Source-to-Target Reenactment dataset.

Each dataset was split into 704 videos for training, 150 for validation, and 150 for testing. The database also provided segmentation masks corresponding to manipulated videos. Three levels of compression based on the H.264 codec\(^2\) were used: no compression, light compression (quantization = 23), and strong compression (quantization = 40).

The FaceForensics++ database is an enhanced version of the FaceForensics database and includes the Face2Face dataset plus the FaceSwap\(^3\) dataset (graphics-based manipulation) and the DeepFakes\(^4\) dataset (deep-learning-based manipulation) [27]. It contains 1,000 real videos and 3,000 manipulated videos (1,000 in each dataset). Each dataset was split into 720 videos for training, 140 for validation, and 140 for testing. The same three levels of compression based on the H.264 codec were used with the same quantization values.

For simplicity, we used only videos with light compression (quantization = 23). Images were extracted from videos using Cozzolino et al.’s settings [11]: 200 frames of each training video were used for training, and 10 frames of each validation and testing video were used for validation and testing, respectively. There is no detailed description of the rules for frame selection, so we selected the first (200 or 10) frames of each video and cropped the facial areas. For all databases, we applied normalization with mean $\approx (0.485, 0.456, 0.406)$ and standard deviation $\approx (0.229, 0.224, 0.225)$; these values have been widely used.

\(^2\)http://www.h264encoder.com/
\(^3\)https://github.com/MarekKowalski/FaceSwap/
\(^4\)https://github.com/deepfakes/faceswap/
in the ImageNet Large Scale Visual Recognition Challenge [28]. We did not apply any data augmentation to the trained datasets.

The training and testing datasets were designed as shown in Table 1. For the Training, Test 1, and Test 2 datasets, the Face2Face method [26] was used to create manipulated videos. Images in Test 2 were harder to detect than those in Test 1 since the source and target videos used for reenactment were the same, meaning that the reenacted video frames had better quality. Therefore, we call Test 1 and Test 2 the match and mismatch conditions for a seen attack. Test 3 used the Deepfake attack method while Test 4 used the FaceSwap attack method adopted from the FaceForensics++ database [27]. These both attack methods were not used to create the training set, therefore they were considered as unseen attacks. For the classification task, we calculated the accuracy and equal error rate (EER) of each method. For the segmentation task, we used pixel-wise accuracy between ground-truth masks and segmentation masks. The FT_Res, FT, and Deeper_FT method could not perform the segmentation task. All the results were at the image level.

Table 2. Settings for autoencoder.

<table>
<thead>
<tr>
<th>No.</th>
<th>Method</th>
<th>Depth</th>
<th>Seg. weight</th>
<th>Rec. weight</th>
<th>Rec. Loss</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FT_Res</td>
<td>Shallower</td>
<td>0.1</td>
<td>0.1</td>
<td>L1</td>
<td>Re-implementation of ForensicsTransfer [11] using residual images as input</td>
</tr>
<tr>
<td>2</td>
<td>FT</td>
<td>Shallower</td>
<td>0.1</td>
<td>0.1</td>
<td>L1</td>
<td>Re-implementation of ForensicsTransfer [11] using normal images as input</td>
</tr>
<tr>
<td>3</td>
<td>Deeper_FT</td>
<td>Deeper</td>
<td>0.1</td>
<td>0.1</td>
<td>L1</td>
<td>Proposed deeper version of FT (Proposed_Old method without segmentation branch)</td>
</tr>
<tr>
<td>4</td>
<td>Proposed_Old</td>
<td>Deeper</td>
<td>0.1</td>
<td>0.1</td>
<td>L1</td>
<td>Proposed method using ForensicsTransfer settings</td>
</tr>
<tr>
<td>5</td>
<td>No_Recon</td>
<td>Deeper</td>
<td>1</td>
<td>1</td>
<td>L2</td>
<td>Proposed method without reconstruction branch</td>
</tr>
<tr>
<td>6</td>
<td>Proposed_New</td>
<td>Deeper</td>
<td>1</td>
<td>1</td>
<td>L2</td>
<td>Complete proposed method with new settings</td>
</tr>
</tbody>
</table>

4.2. Training Y-shaped Autoencoder

To evaluate the contributions of each component in the Y-shaped autoencoder, we designed the settings as shown in Table 2. The FT_Res and FT methods are re-implementation of Cozzolino et al.’s method with and without using residual images [11]. They can also be understood as the Y-shaped autoencoder without the segmentation branch. The Deeper_FT method is a deeper version of FT, which has the same depth as the proposed method. The Proposed_Old method is the proposed method using weighting settings from Cozzolino et al.’s work [11], the No_Recon method is the version of the proposed method without the reconstruction branch, and the Proposed_New method is the complete proposed method with the Y-shaped autoencoder using equal losses for the three tasks and the mean squared error for reconstruction loss.

Since shallower networks take longer to converge than deeper ones, we trained the shallower ones with 100 epochs and the deeper ones with 50 epochs. For each method, the training stage with the highest accuracy for the classification task and a reasonable segmentation loss (if available) was used to perform all the tests described in this section.
4.3. Dealing with Seen Attacks

The results for the match and mismatch conditions for seen attacks are respectively shown in Tables 3 (Test 1) and 4 (Test 2). The deeper networks (the last four) had substantially better classification performance than the shallower ones (the first two) proposed by Cozzolino et al. [11]. Among the four deeper networks, there were no substantial differences in their performances on the classification task. For the segmentation task, the No_Recon and Proposed_New methods, which used the new weighting settings, had higher accuracy than the Proposed_Old method, which used the old weighting settings.

Table 3. Results for Test 1 (image level).

<table>
<thead>
<tr>
<th>Method</th>
<th>Classification</th>
<th>Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc (%)</td>
<td>EER (%)</td>
</tr>
<tr>
<td>FT_Res</td>
<td>82.30</td>
<td>14.53</td>
</tr>
<tr>
<td>FT</td>
<td>88.43</td>
<td>11.60</td>
</tr>
<tr>
<td>Deeper_FT</td>
<td>93.63</td>
<td>7.20</td>
</tr>
<tr>
<td>Proposed_Old</td>
<td>92.60</td>
<td>7.40</td>
</tr>
<tr>
<td>No_Recon</td>
<td>93.40</td>
<td>7.07</td>
</tr>
<tr>
<td>Proposed_New</td>
<td>92.77</td>
<td>8.18</td>
</tr>
</tbody>
</table>

Table 4. Results for Test 2 (image level).

<table>
<thead>
<tr>
<th>Method</th>
<th>Classification</th>
<th>Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc (%)</td>
<td>EER (%)</td>
</tr>
<tr>
<td>FT_Res</td>
<td>82.33</td>
<td>15.07</td>
</tr>
<tr>
<td>FT</td>
<td>87.33</td>
<td>12.03</td>
</tr>
<tr>
<td>Deeper_FT</td>
<td>92.70</td>
<td>7.80</td>
</tr>
<tr>
<td>Proposed_Old</td>
<td>91.83</td>
<td>8.53</td>
</tr>
<tr>
<td>No_Recon</td>
<td>92.83</td>
<td>8.29</td>
</tr>
<tr>
<td>Proposed_New</td>
<td>92.50</td>
<td>8.07</td>
</tr>
</tbody>
</table>

The performances of all methods was slightly degraded when dealing with the mismatch condition for seen attacks. The FT_Res and Proposed_New methods had the best adaptation ability, as indicated by the lower degradation in their scores. This indicates the importance of using residual images (for the FT_Res method) and of using the reconstruction branch (for the Y-shaped autoencoder with new weighting settings: Proposed_New method). The reconstruction branch also helped the Proposed_New method achieve the highest score on the segmentation task.

4.4. Dealing with Unseen Attacks

4.4.1 Evaluation using pre-trained model

When encountering unseen attacks, all six methods had substantially lower accuracies and higher EERs, a shown in Tables 5 (Test 3) and 6 (Test 4). In Test 3, the shallower methods had better adaptation ability, especially the FT_Res method, which uses residual images. The deeper methods, which had a greater chance of being over-fitted, had nearly random classification results. In Test 4, although all methods suffered from nearly random classification accuracies, their better EERs indicated that the decision thresholds had been moved.

A particularly interesting finding was in the segmentation results. Although degraded, the segmentation accuracies were still high, especially in Test 4, in which FaceSwap copied the facial area from the source faces to the target ones using a computer-graphics method. When dealing with unseen attacks, this segmentation information could thus be an important clue in addition to the classification results for judging the authenticity of the queried images and videos.

Table 5. Results for Test 3 (image level).

<table>
<thead>
<tr>
<th>Method</th>
<th>Classification</th>
<th>Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc (%)</td>
<td>EER (%)</td>
</tr>
<tr>
<td>FT_Res</td>
<td>64.75</td>
<td>30.71</td>
</tr>
<tr>
<td>FT</td>
<td>62.61</td>
<td>37.43</td>
</tr>
<tr>
<td>Deeper_FT</td>
<td>51.21</td>
<td>42.71</td>
</tr>
<tr>
<td>Proposed_Old</td>
<td>53.75</td>
<td>42.00</td>
</tr>
<tr>
<td>No_Recon</td>
<td>51.96</td>
<td>42.45</td>
</tr>
<tr>
<td>Proposed_New</td>
<td>52.32</td>
<td>42.24</td>
</tr>
</tbody>
</table>

Table 6. Results for Test 4 without fine-tuning (image level).

<table>
<thead>
<tr>
<th>Method</th>
<th>Classification</th>
<th>Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc (%)</td>
<td>EER (%)</td>
</tr>
<tr>
<td>FT_Res</td>
<td>53.50</td>
<td>43.10</td>
</tr>
<tr>
<td>FT</td>
<td>52.29</td>
<td>41.79</td>
</tr>
<tr>
<td>Deeper_FT</td>
<td>53.39</td>
<td>37.00</td>
</tr>
<tr>
<td>Proposed_Old</td>
<td>56.82</td>
<td>36.29</td>
</tr>
<tr>
<td>No_Recon</td>
<td>54.86</td>
<td>35.86</td>
</tr>
<tr>
<td>Proposed_New</td>
<td>54.07</td>
<td>34.04</td>
</tr>
</tbody>
</table>

4.4.2 Fine-tuning using small amount of data

We used the validation set (a small set normally used for selecting hyper-parameters in training that differs from the test set) of the FaceForensics++ - FaceSwap dataset [27] for fine-tuning all the methods. To ensure that the amount of data was small, we used only ten frames for each video. We divided the dataset into two parts: 100 videos of each class for training and 40 of each class for evaluation. We trained them using 50 epochs and selected the best models on the basis of their performance on the evaluation set.

The results after fine-tuning for Test 4 are shown in Table 7. Their classification and segmentation accuracies increased around 25% and 8%, respectively, which are remarkable compared with the small amount of data used. The one exception was the Proposed_Old method – its segmentation accuracy did not improve. The FT_Res method
had better adaptation than the \textit{FT} one, which supports Cozzolino et al.’s claim \cite{11}. The \textit{Proposed\_New} method had the highest transferability against unseen attacks as evidenced by the results in Table 7.

### 5. Conclusion

The proposed convolutional neural network with a Y-shaped autoencoder demonstrated its effectiveness for both classification and segmentation tasks without using a sliding window, as is commonly used by classifiers. Information sharing among the classification, segmentation, and reconstruction tasks improved the network’s overall performance, especially for the mismatch condition for unseen attacks. Moreover, the autoencoder can quickly adapt to deal with unseen attacks by using only a few samples for fine-tuning. Future work will mainly focus on investigating the effect of using residual images \cite{11} on the autoencoder’s performance, processing high-resolution images without re-sizing, improving its ability to deal with unseen attacks, and extending it to the audiovisual domain.

### Acknowledgement

This research was supported by JSPS KAKENHI Grant Number JP16H06302, JP18H04120, and JST CREST Grant Number JPMJCR18A6, Japan.

### References


\begin{table}[!h]
\centering
\begin{tabular}{|l|c|c|c|}
\hline
\textbf{Method} & \textbf{Classification} & \textbf{Segmentation} & \textbf{No} \\
\hline
& \textbf{Acc (\%)} & \textbf{EER (\%)} & \textbf{Acc (\%)} \\
\hline
\textit{FT\_Res} & 80.04 (↑ 26.54) & 17.57 (↑ 25.53) & 0.00 \\
\textit{FT} & 70.89 (↑ 18.60) & 25.36 (↑ 16.23) & 0.00 \\
\textit{Deep\_FT} & 82.00 (↑ 28.61) & 17.33 (↑ 19.67) & 0.00 \\
\textit{Proposed\_Old} & 78.57 (↑ 21.75) & 20.79 (↑ 15.50) & 84.39 (↑ 0.16) \\
\textit{Proposed\_New} & 83.71 (↑ 29.64) & 15.07 (↑ 18.97) & 93.01 (↑ 8.34) \\
\hline
\end{tabular}
\caption{Results for Test 4 after fine-tuning (image level).}
\end{table}


