Chapter 13
Capsule-Forensics Networks for Deepfake Detection

Huy H. Nguyen, Junichi Yamagishi, and Isao Echizen

Abstract Several sophisticated convolutional neural network (CNN) architectures have been devised that have achieved impressive results in various domains. One downside of this success is the advent of attacks using deepfakes, a family of tools that enable anyone to use a personal computer to easily create fake videos of someone from a short video found online. Several detectors have been introduced to deal with such attacks. To achieve state-of-the-art performance, CNN-based detectors have usually been upgraded by increasing their depth and/or their width, adding more internal connections, or fusing several features or predicted probabilities from multiple CNNs. As a result, CNN-based detectors have become bigger, consume more memory and computation power, and require more training data. Moreover, there is concern about their generalizability to deal with unseen manipulation methods. In this chapter, we argue that our forensic-oriented capsule network overcomes these limitations and is more suitable than conventional CNNs to detect deepfakes. The superiority of our "Capsule-Forensics" network is due to the use of a pretrained feature extractor, statistical pooling layers, and a dynamic routing algorithm. This design enables the Capsule-Forensics network to outperform a CNN with a similar design and to be from 5 to 11 times smaller than a CNN with similar performance.

H. H. Nguyen (✉) · J. Yamagishi
The Graduate University for Advanced Studies, SOKENDAI and The National Institute of Informatics, Tokyo, Japan
e-mail: nhhuy@nii.ac.jp

J. Yamagishi
e-mail: jyamagis@nii.ac.jp

I. Echizen
The National Institute of Informatics, The University of Tokyo,
and The Graduate University for Advanced Studies, SOKENDAI, Tokyo, Japan
e-mail: iechizen@nii.ac.jp

C. Rathgeb et al. (eds.), *Handbook of Digital Face Manipulation and Detection*,
Advances in Computer Vision and Pattern Recognition,
https://doi.org/10.1007/978-3-030-87664-7_13

13.1 Introduction

Ever since the invention of photography, people have been interested in manipulating photographs, mainly to correct problems in the photos or to enhance them. Technology has advanced far beyond these basic manipulations and can now be used to change the identities of the subjects or alter their emotions. The advent of deep learning has enabled high-quality manipulated images and videos to be easily created. Moreover, the popularity of social media has enabled massive amounts of data, including personal information, news reports, images, and videos, to be created and shared. The consequence is that people with malicious intent can easily make use of these advanced technologies and data to create fake images and videos and then publish them widely on social networks.

The requirements for manipulating or synthesizing videos were dramatically simplified when it became possible to create forged videos from only a short video [22, 46] or even from a single ID photo [7] of the target subject. Suwajanakorn et al.'s mapping method [42] has enhanced the ability of manipulators to learn the mapping between speech and lip motion. State-of-the-art natural speech synthesizers can be used with Suwajanakorn's method to create a fake video of any person speaking anything. Deepfakes [3] exemplify this threat—an attacker with a personal computer and an appropriate tool can create videos of a person impersonating any other person. Deepfake videos have been posted on YouTube with the challenge being to spot them. In this chapter, we use the term "deepfake" to refer to this family of manipulation techniques, not to a particular one. Several examples of high-quality computer-generated images and deepfake ones are shown in Fig. 13.1.

Several countermeasures have been developed to detect fake images and videos. Automatic feature extraction using convolutional neural networks (CNNs) has dra-

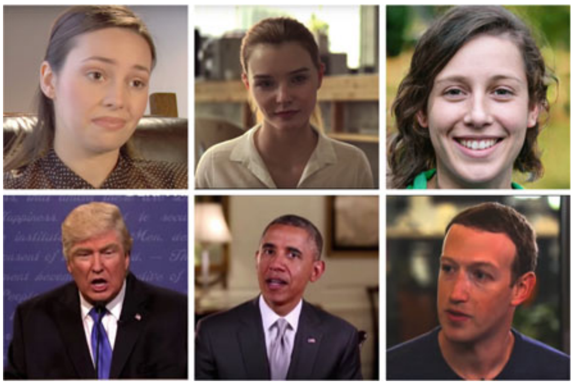

Fig. 13.1 Example computer-generated and deepfake images. Images in top row are fully computer-generated (from Digital Emily Project [6], from Dexter Studios [2], and was generated using Style-GAN [21], respectively). Images in bottom row, left to right, were manipulated using deepfake [3], Face2Face [46], and Neural Textures [45] methods, respectively

matically improved detection performance [4, 36, 38]. Several methods are image-based [4, 36, 54] while others work only on videos [5, 27, 38] or on video with voice [24]. Although some video-based methods perform better than image-based ones, they are only applicable to particular kinds of attacks. For example, some of them [5, 27] may fail if the quality of the eye area is sufficiently good or the synchronization between the video and audio parts is sufficiently natural [25]. In this chapter, we limit our scope to image-based methods since our aim is to build a general detector that can work with both generated/manipulated images and videos and does not rely on any particular kind of attack.

Conventionally, the performance of a CNN can be improved by increasing its depth [16], its width [52], and/or the number of inner connections [19]. Another solution is to use multiple CNNs as is done in Zhou et al.'s two-stream network [54] or to use feature aggregation (feature fusion) or output fusion (ensemble). The fusion approach has been used in several competitions [13, 29]. This approach not only improves network performance on seen data but also improves network performance on unseen data. This has resulted in CNNs and groups of CNNs becoming bigger and thus consuming more memory and computation power. Moreover, they may need more training data, which are not always available when new attacks emerge. Rather than making the network bigger, we took a different approach: redesign it to make it more efficient in memory usage, detection accuracy, and generalization.

We previously reported "Capsule-Forensics" [32], a proof-of-concept capsule network [39] designed especially for detecting manipulated images and videos. In this work, we focused on explaining the theoretical aspect of Capsule-Forensics, which was not fully discussed in our previous work [32]. We hypothesized that the special design of the network makes it better able to detect deepfakes than a corresponding CNN while keeping the network smaller. This special design includes:

- A feature extractor, which is part of a pretrained image classification CNN, prevents the network from overfitting and improves its performance on both seen and unseen attacks.
- A statistical pooling layer, which is used in each primary capsule of the network, greatly reduces the number of parameters compared with the original capsule network while improving performance on deepfake detection.
- A dynamic routing algorithm produces better fusion than the traditional feature aggregation approach.

To sum up, our contribution is three-fold:

1. We provide a theoretical explanation of the Capsule-Forensics network on deepfake detection by verifying our hypothesis that its special design is the reason it performs better than the corresponding CNN version.
2. We visualize the activation of each primary capsule as well as the routing weights and thereby clarify which kind of information these capsules learn and how they agree on the final decision of the entire network. This is a step toward explainability of the Capsule-Forensics network.

3. We introduce small deepfake detection benchmarks that focuses on detection performance, number of parameters, and inference time for both seen and unseen data.

The rest of this chapter is structured as follows. We first describe work related to deepfakes, deepfake detection, and the challenges in deepfake detection. We also give some background on capsule networks. Next, we describe the Capsule-Forensics network. We also visualize the features the Capsule-Forensics network learns to understand the differences between it and a conventional capsule network, which learns the hierarchical relationships between object parts. Then, we describe several experiments we performed to test our hypothesis that the special design of the network makes it better able to detect deepfakes than a corresponding CNN while keeping the network smaller. Finally, we conclude by discussing the meaning of our results and mentioning future work.

13.2 Related Work

13.2.1 Deepfake Generation

Recent achievements demonstrate that deepfakes can reach a photo-realistic level. Thies et al. demonstrated that expression transfer for facial reenactment can be performed in real time [46]. Kim et al. demonstrated the transfer of a head pose along with facial movements from an actor to another person [22]. Similarly, Tripathy et al. devised a lightweight face reenactment method using a generative adversarial network (GAN) [47]. Nirkin et al. presented a face swapping method that does not require training on new faces [33], unlike the early deepfake methods [3]. Thies et al. combined the traditional graphics pipeline with learnable components to deal with imperfect 3D contents [45].

Work on deepfakes has gone beyond only the visual part. Suwajanakorn et al. presented a method for learning the mapping between speech and lip movements in which speech can also be synthesized, enabling creation of a full-function spoof video [42]. Fried et al. demonstrated that speech can be easily modified in any video in accordance with the intention of the manipulator while maintaining a seamless audio-visual flow [15]. Averbuch-Elor et al. addressed a different problem—converting still portraits into motion pictures expressing various emotions [7]. This work greatly simplified the requirements for attackers: simply acquire a picture of the victim (usually a profile picture on a social network or an ID photo). Zakharov et al. followed up by improving the quality of videos generated using only a few input images [53]. Vougioukas et al. raised the bar by introducing a method for animating a facial image from an audio track containing speech [48].

13.2.2 Deepfake Detection

The handcrafted steganalysis-based method developed by Fridrich and Kodovsky [14] was used in early efforts to detect manipulated images. Noise residuals extracted using handcrafted linear and nonlinear high-pass filters are fed into an ensemble classifier. This approach was later implemented in a CNN by Cozzolino et al. [12]. Transfer learning is a common choice when a CNN pretrained on the ImageNet dataset [37] is used [31, 36]. Nguyen et al. [31] used part of a pretrained VGG-19 network [41] as the feature extractor for their modular network while Rössler et al. finetuned the XceptionNet network [11] on a deepfake dataset. Afchar et al. utilized inception modules [43] to build a lightweight network [4] while Wang et al. utilized a dilated residual network [49]. Bayar and Stamm presented a new convolutional layer that helps a CNN adaptively learn manipulation detection features [10]. Zhou et al. proposed using a two-stream network in which one stream takes RGB input and the other takes steganalysis features and uses a triplet loss [54].

Videos provide more information than images for detection, especially when they contain sound. Li et al. used eye blinking as a feature to detect deepfakes [27] while Agarwal et al. used facial expressions and movements [5]. Sabir et al. used a recurrent neural network to additionally learn the temporal information [38]. Korshunov and Marcel used several approaches for lip-syncing and dubbing detection to detect fake videos [24].

In addition to binary classification, another major branch in digital media forensics is locating manipulated regions in images. Besides "pure" segmentation-based approaches [9, 30, 55], binary classification approaches are also applicable by using a sliding window to locate manipulated regions [31, 36]. From a different viewpoint, Li et al. introduced a method called face X-ray to detect the blending boundary between real and fake regions [26]. They noted that blending methods have not been advancing as rapidly as manipulation methods; therefore, focusing on blending methods makes the detector more robust against unseen manipulations.

Several standardized datasets have been constructed to support deepfake detection, including the FaceForensics++ dataset [36], the Google Deepfake Detection (DFD) dataset [1], the DeepFakeTIMIT dataset [25], the Celeb-DF dataset [28], the Deepfake Detection Challenge dataset [13], and the DeeperForensics dataset [20]. We focused on the FaceForensics++ and Google DFD datasets as they cover several well-known attacks, including Face2Face [46], FaceSwap [36], deepfake [3], and Neural Textures [45] attacks (examples are shown in Fig. 13.1). We focused on the image domain and treated videos as a set of separable frames.

13.2.3 Challenges in Deepfake Detection

There are several challenges in deepfake detection. Since deepfakes have altered faces, most deepfake detection methods need to first detect and crop the face. The success of this step depends on the performance of the face detection method. Most state-of-the-art deepfake datasets have annotated face regions, so researchers may assume that cropped faces are available without considering the face detector's performance. Another challenge is the generalizability of the detector when an advanced deepfake technique is introduced. Moreover, a large amount of appropriate training data may not be available when a new attack appears, so detectors using large networks may be difficult to train. Another challenge is gaining user trust by convincing them to accept the detection results. This requires visualizing the learned features and/or focused regions of the detectors.

The performance of general CNNs can usually be improved by increasing their depth, their width, and/or the number of inner connections. Multiple CNNs are commonly used for deepfake detection, especially in competitions [13, 29]. Fusion is often used in the multiple-CNN approach, including feature aggregation (feature fusion) and output fusion (ensemble). Consequently, these networks get bigger with more parameters, consuming more memory and computation power. Since a larger number of parameters usually requires more training data, dealing with new attacks is difficult. Our Capsule-Forensics network was designed to overcome these limitations.

13.2.4 Capsule Networks

"Capsule network" is not a new term as it was first introduced in 2011 by Hinton et al. [17]. They argued that CNNs have limited ability to learn the hierarchical relationships between object parts and introduced a more robust architecture comprising several "capsules." However, they initially faced the same problem affecting CNNs—limited hardware performance—and the lack of effective algorithms, which prevented practical application of capsule networks. CNNs thus remained dominant in this research field.

These problems were overcome when the dynamic routing algorithm [39] and its variant—the expectation-maximization routing algorithm [18]—were introduced. These breakthroughs enabled capsule networks to achieve better performance and outperform CNNs on object classification tasks [8, 18, 39, 50, 51]. The agreements between low- and high-level capsules, which encode the hierarchical relationships between objects and their parts with pose information, enable a capsule network to preserve more information than a CNN while using only a fraction of the data used by a CNN.

13.3 Capsule-Forensics

13.3.1 Why Capsule-Forensics?

To overcome the weakness of conventional CNNs, we adapted the capsule network concept [39], which was originally designed for computer vision tasks, to make it well suited for deepfake detection. We named our adapted network "Capsule-Forensics." Its design takes advantage of transfer learning by using part of a pretrained CNN (trained on the ImageNet dataset [37]) as the feature extractor. This helps the network achieve high performance and have better generalizability. The feature aggregation used in conventional CNNs was replaced with a modified version of the dynamic routing algorithm. The use of a statistical pooling layer in each primary capsule reduces the number of parameters while improving performance. The next two sections describe the processing flow and architecture. We performed several experiments to verify the novelty of this design. The results are presented and discussed in the Evaluation section.

13.3.2 Overview

The Capsule-Forensics based method comprises three processing units, as illustrated in Fig. 13.2. The task performed in the pre-processing unit depends on the input. If the input is video, the first step is to separate the frames. A face detection algorithm is used to crop the facial area(s). The cropped face(s) are sent to the Capsule-Forensics unit for classification. The detection result(s) are sent to the post-processing unit, which works in accordance with the pre-processing one. If the input is an image, nothing is done here. If the input is video, the scores of all frames are averaged. This average score is the final output.

13.3.3 Architecture

The Capsule-Forensics network includes a feature extractor, several primary capsules, and two output capsules ("real" and "fake"), as illustrated in Fig. 13.3. For

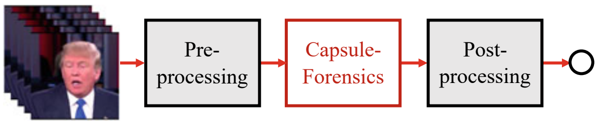

Fig. 13.2 Capsule-Forensics unit processing

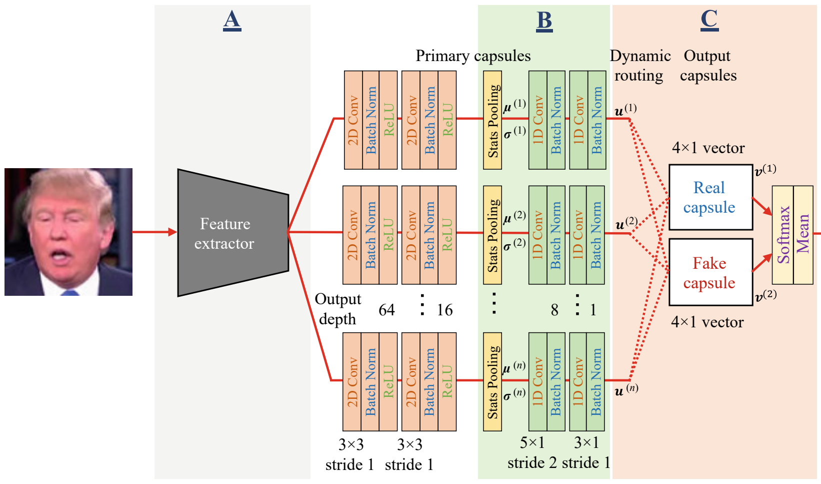

Fig. 13.3 Capsule-Forensics architecture. Blocks A, B, and C contain tunable hyperparameters

simplification, we use the same architecture for all primary capsules. Since we use random weight initialization, their behaviors are not the same after training. The number of primary capsules is a hyperparameter.

Each primary capsule has three parts: a 2D convolutional part, a statistical pooling layer, and a 1D convolutional part. The statistical pooling layer has been proven to be effective in detecting computer-generated images [31, 35] by learning the statistical differences between the real and computer-generated images. For deepfakes, when a part of a face image is swapped, the swapped face region may have different textures and color patterns. The blending region between the swapped face region and the remaining original face region may also contain artifacts. Thus, the statistics such as mean and variance of each filter are useful for differentiating the swapped region from the original one. Moreover, they help reduce the number of parameters by omitting features that are not useful for deepfake detection.

The mean and variance of each filter are calculated in the statistical pooling layer.

- Mean:

$$\mu_k = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} I_{kij}$$

- Variance:

$$\sigma_k^2 = \frac{1}{H \times W - 1} \sum_{i=1}^{H} \sum_{j=1}^{W} (I_{kij} - \mu_k)^2,$$

where k is the layer index, H and W are, respectively, the height and width of the filter, and I is a two-dimensional filter array.

The output of the statistical layer goes through the following 1D convolutional part. Then it is dynamically routed to the output capsules. The final result is calculated on the basis of the activation of the output capsules. The algorithm is discussed in detail in the next section. For binary classification, there are two output capsules, as shown in Fig. 13.3. Multi-class classification could be performed by adding more output capsules, as discussed in Sect. 13.4.3.

The Capsule-Forensics source code has been published at https://github.com/nii-yamagishilab/Capsule-Forensics-v2.

13.3.4 Dynamic Routing Algorithm

Different manipulation methods use different face regions, generating models, and blending algorithms. Therefore, each primary capsule extracts different features depending on the manipulation method, and they may work better on a particular manipulation than on others. Furthermore, since the weights of the primary capsules are initialized differently in training, the capsules learn different features for the same input. These features need to be fused correctly to predict whether the input is real or fake. For a capsule network, this fusion is done dynamically using a dynamic routing algorithm. The "agreement" between all primary capsules is calculated and routed to the appropriate output capsule (real or fake for binary classification). An example of the routing weight vectors is visualized in Fig. 13.4. Since the primary capsules may make different judgments and some of them may be wrong, this algorithm is designed to find a consensus. The output probabilities are determined on the basis of the activations of the output capsules.

Let us call the output vector of each primary capsule $\mathbf{u}^{(i)} \in \mathbb{R}^k$ and each output vector capsule $\mathbf{v}^{(j)} \in \mathbb{R}^l$. There are m primary capsules and n output capsules. $\mathbf{W}^{(i)} \in \mathbb{R}^{l \times k}$ is the matrix used to route an $\mathbf{u}^{(i)}$ to all $\mathbf{v}^{(j)}$, and r is the number of iterations. The dynamic routing algorithm is shown in Algorithm 1. A simple example is presented in the Appendix.

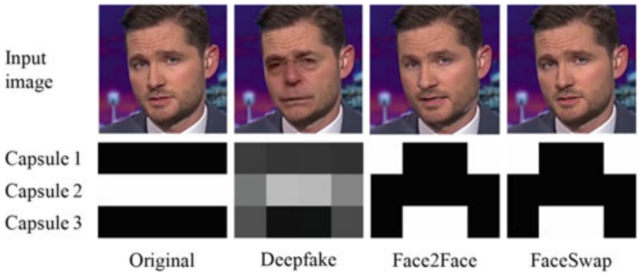

Fig. 13.4 Visualization of the routing matrix $\mathbf{C}^{(2)\top}$ used to route the outputs of three primary capsules to fake output capsule. Face2Face and FaceSwap methods are graphical based, so their routing weights are similar. Deepfake method is deep learning based, so its routing weights are different from the two graphical-based manipulation methods

Algorithm 1 Dynamic routing between capsules.

```
procedure ROUTING($\mathbf{u}^{(i)}, \mathbf{W}^{(i)}, r$)
    $\widehat{\mathbf{W}}^{(i)} \leftarrow \mathbf{W}^{(i)} + \text{rand}(\text{size}(\mathbf{W}^{(i)}))$
    $\widehat{\mathbf{u}}^{(i)} \leftarrow \widehat{\mathbf{W}}^{(i)}\text{squash}(\mathbf{u}^{(i)})$                    $\triangleright\ \widehat{\mathbf{u}}^{(i)} \in \mathbb{R}^{l}$
    $\widehat{\mathbf{u}}^{(i)} \leftarrow \text{dropout}(\widehat{\mathbf{u}}^{(i)})$
    for all output capsules $j$ do
        $\mathbf{B}^{(j)} \leftarrow 0$                    $\triangleright\ \mathbf{B}^{(j)} \in \mathbb{R}^{l\times m}$
    for $r$ iterations do
        for all output capsules $j$ and all vector elements do
            $(c^{(j)}_{\_,1}, c^{(j)}_{\_,2}, \ldots, c^{(j)}_{\_,m}) \leftarrow \text{softmax}(b^{(j)}_{\_,1}, b^{(j)}_{\_,2}, \ldots, b^{(j)}_{\_,m})$
        for all output capsules $j$ do $\mathbf{s}^{(j)} \leftarrow \sum_i^m \mathbf{c}^{(j)}_{:,i} \odot \widehat{\mathbf{u}}^{(i)}$
        for all output capsules $j$ do $\mathbf{v}^{(j)} \leftarrow \text{squash}(\mathbf{s}^{(j)})$
        for all input capsules $i$ and output capsules $j$ do
            $\mathbf{B}^{(j)} \leftarrow \mathbf{B}^{(j)} + \left[\widehat{\mathbf{u}}^{(1)}\ \widehat{\mathbf{u}}^{(2)}\ \ldots\ \widehat{\mathbf{u}}^{(m)}\right] \odot \mathbf{v}^{(j)}$
    return $\mathbf{v}^{(j)}$
```

We slightly improved the algorithm of Sabour et al. [39] by introducing two regularizations: adding random noise to the routing matrix and adding a dropout operation. They are used **only during training** to reduce overfitting. Their effectiveness is discussed in the Evaluation section. Furthermore, a squash function (Eq. 13.1) is applied to $\mathbf{u}^{(i)}$ before routing to normalize it, which helps stabilize the training process. The squash function is used to scale the vector magnitude to unit length.

$$squash(\mathbf{u}) = \frac{\|\mathbf{u}\|_2^2}{1 + \|\mathbf{u}\|_2^2}\frac{\mathbf{u}}{\|\mathbf{u}\|_2} \tag{13.1}$$

In practice, to stabilize the training process, the random noise should be sampled from a normal distribution ($\mathcal{N}(0, 0.01)$), the dropout ratio should not be greater than 0.05 (we used 0.05 in all experiments), and two iterations ($r = 2$) should be used in the dynamic routing algorithm. The two regularizations are used along with random weight initialization to increase the level of randomness, which helps the primary capsules learn with different parameters.

To calculate predicted label $\widehat{y}$, we apply the softmax function to each dimension of the output capsule vectors to achieve stronger polarization rather than simply using the length of the output capsules [39]. The final results are the means of all softmax outputs:

$$\widehat{\mathbf{y}} = \frac{1}{l}\sum_i^l \text{softmax}(v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(n)}), \tag{13.2}$$

where $\widehat{\mathbf{y}}$ is the predicted probabilities vector. Since there is no reconstruction in the Capsule-Forensics method, we simply use the cross-entropy loss function and the Adam optimizer [23] to optimize the network.

13.3.5 Visualization

To illustrate how Capsule-Forensics works, we used a Capsule-Forensics network with three primary capsules trained on the FaceForensics++ database [36]. For visualization, we applied and modified an open-source tool [34] implementing the guided back propagation algorithm [40]. To visualize each primary capsule in this way, we chose the latent features extracted before the statistical pooling layers since they still had the 2D structure.

The activations of each capsule and of the whole network are illustrated in Fig. 13.5. The differences in activation among capsules and between each capsule and the whole network are also shown. The regions of interest mainly include the eyes, nose, mouth region, and facial contours. Some capsules missed some of these regions, and some failed to detect the manipulated input (i.e., the third capsule in Fig. 13.6). Nevertheless, the final results mostly focused on the important regions detected by all capsules due to agreement driven by the dynamic routing algorithm between the other two capsules. A CNN using only the third primary capsule would fail to detect the manipulated input.

The behavior of the Capsule-Forensics network for the deepfake detection problem differs from that of the original capsule network for the inverse graphics problem, in which the focus is on the spatial hierarchies between simple and complex objects [17, 18, 39]. In the deepfake detection problem, abnormal appearances are the key features, so each primary capsule is designed to capture them and communicate its findings to the other capsules. This behavior is similar to that of jurors during a trial, and the consensus judgment is the final detection result.

13.4 Evaluation

We conducted several experiments to test the detection performance of the Capsule-Forensics network. After describing the datasets and metrics we used (Sect. 13.4.1 and 13.4.2), we discuss the effectiveness of the improvements introduced in this chapter in comparison with our previous work [32]: larger input size, more primary capsules, and dropout in the dynamic routing algorithm (Sect. 13.4.3). We then compare several candidate feature extractors (Sect. 13.4.4) and evaluate the effectiveness of the statistical pooling layer used in each primary capsule (Sect. 13.4.5). Finally, we compare the detection performance of the improved Capsule-Forensics network with that of a CNN on both seen and unseen attacks (Sect. 13.4.6 and 13.4.7, respectively). For the CNNs, we used the corresponding version of the Capsule-Forensics network using feature aggregation instead of the dynamic routing algorithm, the multi-task learning network [30], the XceptionNet version used in FaceForensics++ work [36], and the EfficientNet network [44]. Among them, the multi-task learning network is a generative classifier while the rest are discriminative classifiers. For the multi-task learning network, in addition to ground-truth labels, segmentation masks

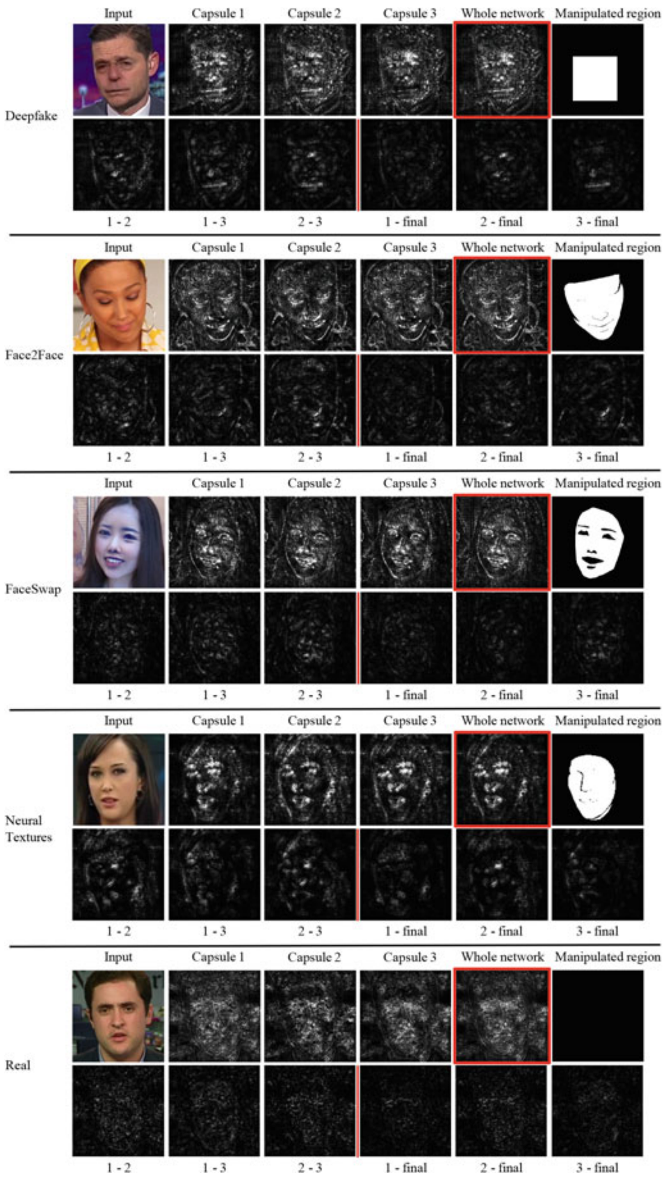

Fig. 13.5 Activation of three capsules and entire Capsule-Forensics network (columns 2, 3, 4, and 5, respectively) on images created using deepfake [3] (row 1), Face2Face [46] (row 3), FaceSwap [36] (row 5), and Neural Textures [45] (row 7) methods and on a real image. Column 6 shows the manipulated regions corresponding to the manipulated images in column 1. The first three columns of rows 2, 4, 6, 8, and 10 show the differences between the activations of capsules 1 and 2, 1 and 3, and 2 and 3 on the corresponding row above, respectively. The three last columns in order show the differences between the activations of capsules 1, 2, and 3 and the activation of the whole network

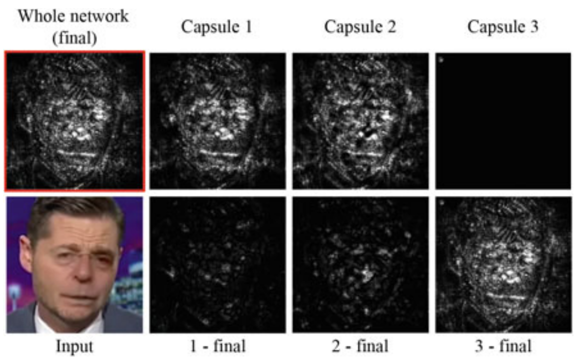

Fig. 13.6 Example case in which one capsule did not work correctly. First row shows activation of whole network and of three capsules. Second row from left to right shows input image and differences between activation of each capsule and of whole network. Although capsule 3 failed to detect manipulated image, final result was correct due to agreement between other two capsules

of the manipulated regions are needed for training. When testing, since segmenting manipulated regions is beyond the scope of this work, we used only its encoder part to perform binary classification. For XceptionNet, we modified its fully connected layer and trained it in two phases. For EfficientNet [44], which recently received a high score in the Deepfake Detection Challenge, we used the B4 version (denoted as EfficientNet-B4) which requires an input size of 380 $\times$ 380 pixels. The larger versions (B5, B6, and B7) require larger inputs and have more parameters, making it impossible to train them on a single-GPU machine.

For simplicity, we used only multi-class classification to compare the original setting in our previous work [32] with the new setting in this work. For the remaining experiments, we tested only binary classification. Except for the one discussed in Sect. 13.4.7, all the evaluations were for performance on seen attacks.

13.4.1 Datasets

We used videos from the FaceForensics++ dataset [36], supplemented with the Google DFD dataset [1]. We used all three levels of compression (none, moderate, and high) and mixed them together to make multiple compression datasets for our experiments. For training, we used version 1 of the FaceForensics++ dataset including original videos and three corresponding manipulated videos created by deepfake [3], Face2Face [46], and FaceSwap [36] methods. For testing, two scenarios were used: seen attacks and unseen attacks. For seen attacks, we used a test set from version 1 of the FaceForensics++ dataset. For unseen attacks, we used test videos created using Neural Textures [45] (unseen method), which was added in

Table 13.1 Configuration of training, validation, and test sets from FaceForensics++ dataset version 1 (for seen attacks) [36]

Type	Training set	Validation set	Test set
Real	720×3 vids $72,000 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs
Deepfake	720×3 vids $72,000 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs
Face2Face	720×3 vids $72,000 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs
FaceSwap	720×3 vids $72,000 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs

Table 13.2 Configuration of test sets for unseen attacks created using Neural Textures method [45] and Google DFD dataset [1]

Type	Neural textures (unseen method)	Google DFD dataset (unseen data)
Real	0 vids 0 imgs	140×3 vids $1,400 \times 3$ imgs
Fakes	358×3 vids $3,580 \times 3$ imgs	$3,065 \times 3$ vids $30,650 \times 3$ imgs

version 2 of the FaceForensics++ dataset, and the entire Google DFD dataset [1] (unseen data).

We took the first 100 frames of the input video for the training set and the first 10 frames for the validation and test sets. FaceForensics++ dataset version 1 (for seen attacks) was divided into a training set, a validation set, and a test set, as shown in Table 13.1. The test sets for unseen attacks are shown in Table 13.2.

13.4.2 Metrics

We used four metrics in our evaluation:

- Classification accuracy $= \frac{TP+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$, where TP, TN, FP, and FN are true positive, true negative, false positive, and false negative, respectively.
- Equal error rate (EER): common value when false positive rate (FPR) equals false negative rate (FNR). FPR $= \frac{\text{FP}}{N}$ (number of false positives divided by number of negatives). FNR $= \frac{\text{FN}}{P}$ (number of false negatives divided by number of positives).
- Half total error rate (HTER): HTER $= \frac{\text{FPR}+\text{FNR}}{2}$.
- Attack presentation classification error rate (APCER): "proportion of attack presentations using the same PAI species incorrectly classified as bona fide presentations in a specific scenario."[1]

[1] ISO/IEC 30107-3 definition. Accessed at https://www.iso.org/obp/ui/#iso:std:iso-iec:19989:-1:ed-1:v1:en:term:3.1.

The thresholds used to determine whether the classification outputs were real or fake were selected on the basis of the EERs calculated for the development sets.

13.4.3 Effect of Improvements

In the first experiment, we measured the effectiveness of the improvements introduced here: larger input size, more primary capsules, and dropout in the dynamic routing algorithm. Since Capsule-Forensics is not limited to binary classification, we also evaluated its multi-class classification ability by changing the number of output capsules, from "Real" and "Fake" capsules to "Real," "Deepfake," "Face2Face," and "FaceSwap" capsules. This modification is obvious and did not require substantial changes to the network architecture.

As shown in Table 13.3, using larger images improved performance substantially as expected. The effect of random noise was limited. In our previous work [32], most of the training sets were small, so random noise made a substantial contribution. In this work, we used the first 100 frames instead of the first 10 for the training set, so the set was ten times larger. Although the random noise did not result in improvement in all cases, it still played an important role in reducing the HTER when combined with dropout and increased the accuracy of multi-class classification. Increasing the number of primary capsules also helped improve performance. The combination of all three improvements achieved the best performance for both binary and multi-class classification. We refer to this combination as "new setting" in Table 13.3 to distinguish it from the "original setting" (the setting used in our previous work [32]).

13.4.4 Feature Extractor Comparison

The feature extractor is an important part of the Capsule-Forensics network (block **A** in Fig. 13.3). Rather than training a simple CNN from scratch along with the other parts of the network, as is done in the traditional capsule network approach [39], we used part of a pretrained CNN (trained on the ImageNet dataset [37]). We selected three commonly used extractors as candidates:

- VGG-19 [41]: used from the beginning until the third max pooling layer.
- ResNet-50 [16]: used from the beginning until the end of the "conv3_x" layer.
- XceptionNet [11]: used from the beginning until the end of the first block of its "middle flow."

In addition to evaluating these candidates, we evaluated a simple CNN with three convolutional layers as the feature extractor, like the ones used in conventional capsule networks. The CNN was trained along with the other parts of the Capsule-Forensics network. In addition, we also fine tuned the pretrained feature extractors

Table 13.3 Performance of Capsule-Forensics with original [32] and new settings introduced here

Input size	No. of capsules	Random noise	Dropout	Binary classification accuracy (%)	Binary classification HTER (%)	Multi-class classification accuracy (%)
Original setting [32]:						
128 × 128	3	No	No	87.45	15.41	85.89
128 × 128	3	Yes	No	88.57	15.35	87.12
New setting:						
300 × 300	3	No	No	89.88	11.28	87.51
300 × 300	3	Yes	No	90.86	11.29	87.54
300 × 300	10	No	No	91.61	11.52	88.51
300 × 300	10	Yes	No	91.32	12.07	89.98
300 × 300	3	No	Yes	91.33	12.37	89.19
300 × 300	3	Yes	Yes	91.19	11.93	88.44
300 × 300	10	No	Yes	92.17	10.70	90.51
300 × 300	**10**	**Yes**	**Yes**	**92.00**	**10.64**	**91.22**

(indicated by "FT" after their names) to check whether fine-tuning helps improve overall performance. We tested the extractors on both the original and new settings except for the simple CNN. It was tested on only the original setting since training it on the new setting would consume a much greater amount of memory and take much longer. The results are shown in Table 13.4.

All the extractors performed better using the new setting. Fine-tuning did not help much when using the new setting. Besides reducing memory usage and shortening training time, using pretrained feature extractors resulted in better performance than using a CNN extractor trained from scratch. These results support our hypothesis that using a pretrained feature extractor contributes to the superiority of our Capsule-Forensics network.

The ResNet-50 based feature extractor has the smallest number of parameters, making it about ten times smaller than the VGG-19 and XceptionNet ones. The VGG-19 extractor with the new setting achieved the highest classification accuracy and had the lowest HTER. For dealing with seen manipulations, if performance is more important than the number of parameters, VGG-19 is the best choice. Otherwise, ResNet-50 is more suitable.

Table 13.4 Performance (in %) of feature extractors with and without fine-tuning (FT) with both original and new settings

Feature extractor	Training accuracy	Test accuracy	Test HTER	No. of parameters
Original setting [32]:				
Simple CNN	98.97	83.36	25.42	371,712
VGG-19	99.81	88.57	15.35	2,325,568
VGG-19 FT	99.54	90.08	12.49	2,325,568
ResNet-50	99.60	88.21	16.09	225,344
ResNet-50 FT	99.69	87.45	13.60	225,344
XceptionNet	99.58	85.52	19.10	2,720,736
XceptionNet FT	99.45	85.41	18.91	2,720,736
New setting:				
VGG-19	99.83	**92.00**	**10.64**	2,325,568
VGG-19 FT	99.63	90.98	13.40	2,325,568
ResNet-50	99.17	90.59	14.60	**225,344**
ResNet-50 FT	99.69	90.14	14.94	225,344
XceptionNet	99.79	90.42	13.35	2,720,736
XceptionNet FT	99.84	91.39	10.85	2,720,736

13.4.5 Effect of Statistical Pooling Layers

In another experiment, we compared the performance and size of two versions of the Capsule-Forensics network: one using and one not using a statistical pooling layer for each primary capsule (block **B** in Fig. 13.3). Previous work [31, 35] suggested that using a statistical pooling layer is effective for detecting computer-generated images. For the version without statistical pooling layers, we replaced the 1D convolutional layers with 2D ones and added an adaptive average pooling layer at the end of each primary capsule. We hypothesized that the statistical pooling layer helps filter out unnecessary information, i.e., information that is not relevant to deepfake detection. Therefore, using a statistical pooling layer in each primary capsule helps reduce feature size and improve performance. Moreover, reducing the feature size results in a smaller routing matrix, which uses less memory and computation power. We used the VGG-19 feature extractor in this experiment. The results are shown in Table 13.5.

With both the original and new settings, using statistical pooling layers greatly improved classification accuracy and reduced the HTER for the seen test set. Moreover, using them reduced the number of parameters by 400%. These results support our hypothesis that using statistical pooling layers contributes to the superiority of our Capsule-Forensics network. An interesting observation from the results is that the number of parameters was independent of the input size (128×128 in the original setting and 300×300 in the new setting). This is because both the statistical and adaptive average pooling layers were designed to deal with variations in input size.

Table 13.5 Performance (in %) with and without statistical pooling (SP) layer in primary capsules for both original and new settings with VGG-19 feature extractor. (Number of parameters does not include number for feature extractor.)

Settings	Test accuracy	Test HTER	No. of parameters
Original setting [32]:			
With SP layer	88.57	15.35	1,571,070
Without SP layer	83.51	15.78	6,689,280
New setting:			
With SP layer	92.00	10.64	1,571,070
Without SP layer	87.70	11.65	6,689,280

13.4.6 Capsule-Forensics Network Versus CNNs: Seen Attacks

In a third experiment, we compared the performance of the dynamic routing algorithm used in the Capsule-Forensics network with that of traditional feature aggregation (block **C** in Fig. 13.3). The VGG-19 feature extractor was used in both cases. We also evaluated the performance of the multi-task learning network [30], the XceptionNet network, and the EfficientNet-B4 network [44]. It is important to note that this version of XceptionNet differs from the one used in our feature extractor (Sect. 13.4.4), which was pretrained on the ImageNet dataset [37], with only part of it used. Since the training dataset was imbalanced (the number of fake samples was three time the number of real samples), we additionally evaluated the effect of using a weighted softmax function during training. The experiment results are shown in Table 13.6.

The effect of using a weighted softmax function is not clear. Since the dataset was not heavily imbalanced, this result is reasonable. Although having the smallest number of parameters, the multi-task learning network had the worst performance. The dynamic routing algorithm helped the Capsule-Forensics network achieve higher performance, especially with the new setting. The numbers of parameters for the Capsule-Forensics network and the corresponding CNN using feature aggregation were almost the same, whereas the numbers for the EfficientNet-B4 and the XceptionNet networks were about 4.5 to 5.3 times larger. Moreover, the test accuracy of the Capsule-Forensics network and the Efficient-B4 network was almost the same. The large input size of the EfficientNet-B4 network (380×380 vs 300×300) might be the reason for its lower HTER.

In addition to the results on the mixed compression test set shown in Table 13.6, we also broke it down into three compression levels, as shown in Table 13.7. There were no substantial differences between the performances of Capsule-Forensics, XceptionNet, and EfficientNet-B4. Their performances were degraded from no compression to moderate compression to high compression. With their average accuracy about 84%, detecting highly compressed deepfake videos was still challenging when most of the deepfake artifacts were erased by the compression algorithm. Capsule-Forensics and

Table 13.6 Performance (in %) of Capsule-Forensics using dynamic routing algorithm, its corresponding CNN using the traditional feature aggregation approach, and the other baselines on seen attacks. Number of parameters is for entire network, including feature extractor

Settings	Test accuracy	Test HTER	No. of parameters
Original setting [32]:			
Dynamic routing	88.57	15.35	2,796,889
Feature aggregation	86.26	15.15	2,798,059
New setting:			
Dynamic routing	**92.00**	10.64	**3,896,638**
Feature aggregation	91.82	11.51	3,903,328
Multi-task learning [30]	73.08	26.30	148,200
XceptionNet [36]	90.73	9.91	20,811,050
EfficientNet-B4 [44]	**92.82**	**8.67**	17,552,202
Using weighted softmax:			
Dynamic routing	**92.21**	10.91	**3,896,638**
Feature aggregation	91.75	10.68	3,903,328
XceptionNet [36]	91.83	10.14	20,811,050
EfficientNet-B4 [44]	91.49	**8.64**	17,552,202

Table 13.7 Performance (in %) of Capsule-Forensics and other classifiers at three levels of compression on the FaceForensics++ dataset.

Detector	No compression		Moderate compression		High Compression	
	Accuracy	HTER	Accuracy	HTER	Accuracy	HTER
Capsule-Forensics	97.27	3.87	94.62	6.42	84.11	21.64
Multi-task learning [30]	81.12	17.80	69.23	25.94	68.86	35.19
XceptionNet [36]	96.12	4.80	92.82	7.60	83.25	17.33
EfficientNet-B4 [44]	98.37	2.50	95.50	4.88	84.96	18.62

EfficientNet handled the moderately compressed deepfake videos quite well, with only about 3% degradation in accuracy compared with the uncompressed ones.

Using the Capsule-Forensics network can save a large amount of memory and computation power compared with the amounts used by CNNs while maintaining high performance even for compressed videos. This is important for applications integrating a presentation attack detector into an Internet of things or a handheld device that does not have powerful hardware to prevent unauthorized facial authentication. The Capsule-Forensics network demonstrated it effectiveness against this kind of attack [32].

Table 13.8 Performance (in %) of three versions of Capsule-Forensics network, two versions of the corresponding CNN, and other baselines on unseen attacks. Number of parameters is for **entire network**, including feature extractor

Detectors	Neural Textures		Google DFD dataset		No. of parameters
	Accuracy	APCER	Accuracy	HTER	
Capsule-Forensics (VGG-19)	24.33	75.67	44.51	40.29	3,896,638
Capsule-Forensics (ResNet-50)	**37.93**	**62.07**	**64.98**	40.89	**1,796,414**
Capsule-Forensics (XceptionNet FT)	31.38	68.62	55.73	38.30	4,007,673
Feature aggregation (VGG-19)	28.81	71.19	58.09	38.70	3,903,328
Feature aggregation (ResNet-50)	24.00	76.00	62.48	37.70	1,803,104
Multi-task learning [30]	**44.69**	**55.31**	**78.74**	42.21	**148,200**
XceptionNet [36]	26.79	73.21	47.29	40.37	20,811,050
EfficientNet-B4 [44]	31.55	68.45	58.63	**34.23**	17,552,202

13.4.7 Capsule-Forensics Network Versus CNNs: Unseen Attacks

Detecting unseen attacks is a difficult problem in deepfake detection, especially for machine-learning-based detectors. When the data distribution changes, the learned features, and decision boundaries are usually no longer correct. Furthermore, large networks with a large number of parameters tend to memorize the training data, especially when the data amount is small. We expected that the Capsule-Forensics network can be better generalized than large networks thanks to the statistical pooling operation and dynamic routings of the primary capsules. To test this, we performed one last experiment in which we tested the detectors on a challenging unseen manipulation method, Neural Textures [45]. It is unlike any of the methods normally used to create seen datasets. We also tested the detectors on a different large deepfake dataset, the Google DFD dataset. We evaluated three new versions of the Capsule-Forensics network with different feature extractors (VGG-19, ResNet-50 (lightweight) and fine-tuned XceptionNet) and with two versions of a CNN using feature aggregation (with VGG-19 and ResNet-50 feature extractors), the multi-task learning network [30], the XceptionNet network [36], and the EfficientNet-B4 network [44].

As shown in Table 13.8, all the detectors performed poorly on the Neural Textures method, with APCERs greater than 50%. The three best detectors on seen attacks (Capsule-Forensics using VGG-19, XceptionNet, and EfficientNet-B4—which are discriminative classifiers) had the worst performances on this method. The multi-task

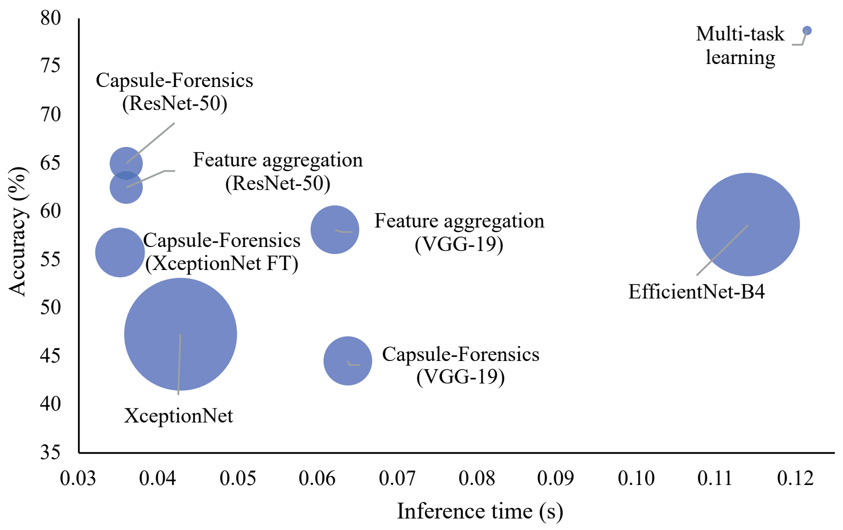

Fig. 13.7 Comparison between several versions of Capsule-Forensics network and CNNs for classification accuracy, inference time, and model size on Google DFD dataset [1]

learning network (which is a generative classifier) achieved the best results, followed by the lightweight Capsule-Forensics network using the ResNet-50 feature extractor. The performances of all detectors were slightly better on the Google DFD dataset. The Capsule-Forensics network using ResNet-50 again had the second highest accuracy, below the multi-task learning network. Since the multi-task learning network was specially designed to deal with unseen attacks, it was able to beat all the other detectors. However, its drawback is poor performance on seen attacks, as seen in the previous section.

Figure 13.7 shows a comparison on the classification accuracy, inference time (for one image), and model size of all detectors on the Google DFD dataset [1]. All tests were done using a NVIDIA DGX Station machine. The Capsule-Forensics network using the ResNet-50 feature extractor and its corresponding CNN using feature aggregation had the second smallest sizes and were the second fastest detectors. They were a bit slower than the Capsule-Forensics network using the XceptionNet feature extractor. Due to the design of the VGG-19 network, detectors using it as the feature extractor have the longest inference times (about twice the shortest times). The XceptionNet-based detector had the largest size but had limited detection accuracy. The EfficientNet-B4-based detector and the multi-task learning detector were the two slowest ones. It is important to note that we measured only the inference time of the encoder part of the multi-task learning detector for the binary classification task. Although it has fewer parameters than the other detectors, some memory-related operations slowed it down.

Although having limited performance on unseen attacks, this experiment demonstrated that the Capsule-Forensics network is better able to detect deepfakes than

CNNs. Between the two versions of the Capsule-Forensics network, if performance on seen attacks is more important, using VGG-19 as the feature extractor is the better choice. If performance on unseen attacks is more important, or a lightweight and fast network is needed, using ResNet-50 as the feature extractor is the better choice.

13.5 Conclusion and Future Work

Our experiments demonstrated that the Capsule-Forensics network is better able to detect deepfakes than conventional CNNs. Its use of a pretrained feature extractor, statistical pooling layers, and a dynamic routing algorithm enables it to achieve better performance with fewer parameters than corresponding CNNs. Furthermore, it has better performance than other discriminative classifiers on unseen manipulations, although further improvement is needed. Visualization of the activation of each capsule enables the learned features to be analyzed. These promising results and the understanding gained from the analysis should lead to further research on and development of capsule networks, not only for digital forensics but also for many other applications.

Future work includes enabling the Capsule-Forensics network to use temporal information to detect fake videos and improving its generalizability (in other words, reducing the gap between discriminative classifiers and generative classifiers). Moreover, deepfake datasets mostly contain images and videos containing only one or two people. In reality, deepfake methods can be applied to a crowd; therefore, deepfake detection in the wild is also an important research direction.

Acknowledgements This work was partially supported by JSPS KAKENHI Grants JP16H06302, JP18H04120, JP21H04907, JP20K23355, and JP21K18023 and by JST CREST Grants JPMJCR18A6 and JPMJCR20D3, including the AIP challenge program, Japan.

13.6 Appendix

This appendix presents a simple example of the dynamic routing algorithm shown in Algorithm 1 with three ($m = 3$) primary capsules $\mathbf{u^{(i)}} \in \mathbb{R}^k$, $i = 1..3$ and two ($n = 2$) output capsules $\mathbf{v^{(j)}} \in \mathbb{R}^l$, $j = 1..2$. All equations are written out in full.

There are three routing matrices corresponding to the three primary capsules, each represented by

$$\widehat{\mathbf{W}}^{(i)} = \mathbf{W}^{(i)} + \mathbf{N} \ \forall i$$

$$= \begin{bmatrix} w_{1,1}^{(i)} & w_{1,2}^{(i)} & \dots & w_{1,k}^{(i)} \\ w_{2,1}^{(i)} & w_{2,2}^{(i)} & \dots & w_{2,k}^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{l,1}^{(i)} & w_{l,2}^{(i)} & \dots & w_{l,k}^{(i)} \end{bmatrix} + \begin{bmatrix} n_{1,1} & n_{1,2} & \dots & n_{1,k} \\ n_{2,1} & n_{2,2} & \dots & {}_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ n_{l,1} & n_{l,2} & \dots & n_{l,k} \end{bmatrix},$$

with $\widehat{\mathbf{W}}^{(i)} \in \mathbb{R}^{l \times k}, \mathbf{W}^{(i)} \in \mathbb{R}^{l \times k}, \mathbf{N}^{(i)} \in \mathbb{R}^{l \times k}, \mathbf{N}^{(i)} \sim \mathcal{N}(0, 0.01)$.

The next steps are to process $\mathbf{u^{(i)}}$ to form $\widehat{\mathbf{u}}^{(i)}$:

$$\widehat{\mathbf{u}}^{(i)} = \begin{bmatrix} \widehat{u}_1^{(i)} \\ \widehat{u}_2^{(i)} \\ \vdots \\ \widehat{u}_l^{(i)} \end{bmatrix} = \widehat{\mathbf{W}}^{(i)} \text{squash}(\mathbf{u}^{(i)}) = \begin{bmatrix} \widehat{w}_{1,1}^{(i)} & \widehat{w}_{1,2}^{(i)} & \dots & \widehat{w}_{1,k}^{(i)} \\ \widehat{w}_{2,1}^{(i)} & \widehat{w}_{2,2}^{(i)} & \dots & \widehat{w}_{2,k}^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ \widehat{w}_{l,1}^{(i)} & \widehat{w}_{l,2}^{(i)} & \dots & \widehat{w}_{l,k}^{(i)} \end{bmatrix} \text{squash}\left(\begin{bmatrix} u_1^{(i)} \\ u_2^{(i)} \\ \vdots \\ u_k^{(i)} \end{bmatrix} \right),$$

with $\mathbf{u}^{(i)} \in \mathbb{R}^k, \widehat{\mathbf{u}}^{(i)} \in \mathbb{R}^l$.

$\widehat{\mathbf{u}}^{(i)} \leftarrow \text{dropout}(\widehat{\mathbf{u}}^{(i)})$.

Then, two matrices $\mathbf{B}^{(1)}, \mathbf{B}^{(2)} \in \mathbb{R}^{l \times 3}$ corresponding to the two output capsules are initialized:

$$\mathbf{B}^{(j)} = \begin{bmatrix} b_{1,1}^{(j)}) & b_{1,2}^{(j)}) & b_{1,3}^{(j)}) \\ b_{2,1}^{(j)}) & b_{2,2}^{(j)}) & b_{2,3}^{(j)}) \\ \vdots & \vdots & \vdots \\ b_{l,1}^{(j)}) & b_{l,2}^{(j)}) & b_{l,3}^{(j)}) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix}.$$

For r iterations do:

$$\mathbf{C}^{(j)} = \begin{bmatrix} c_{1,1}^{(j)} & c_{1,2}^{(j)} & c_{1,3}^{(j)} \\ c_{2,1}^{(j)} & c_{2,2}^{(j)} & c_{2,3}^{(j)} \\ \vdots & \vdots & \vdots \\ c_{l,1}^{(j)} & c_{l,2}^{(j)} & c_{l,3}^{(j)} \end{bmatrix} = \begin{bmatrix} \text{softmax}(b_{1,1}^{(j)}, b_{1,2}^{(j)}, b_{1,3}^{(j)}) \\ \text{softmax}(b_{2,1}^{(j)}, b_{2,2}^{(j)}, b_{2,3}^{(j)}) \\ \vdots \\ \text{softmax}(b_{l,1}^{(j)}, b_{l,2}^{(j)}, b_{l,3}^{(j)}) \end{bmatrix}, \mathbf{C}^{(j)} \in \mathbb{R}^{l \times 3}.$$

$$\mathbf{s}^{(j)} = \begin{bmatrix} s_1^{(j)} \\ s_2^{(j)} \\ \vdots \\ s_l^{(j)} \end{bmatrix} = \begin{bmatrix} c_{1,1}^{(j)} \\ c_{2,1}^{(j)} \\ \vdots \\ c_{l,1}^{(j)} \end{bmatrix} \odot \begin{bmatrix} \widehat{u}_1^{(1)} \\ \widehat{u}_2^{(1)} \\ \vdots \\ \widehat{u}_l^{(1)} \end{bmatrix} + \begin{bmatrix} c_{1,2}^{(j)} \\ c_{2,2}^{(j)} \\ \vdots \\ c_{l,2}^{(j)} \end{bmatrix} \odot \begin{bmatrix} \widehat{u}_1^{(2)} \\ \widehat{u}_2^{(2)} \\ \vdots \\ \widehat{u}_l^{(2)} \end{bmatrix} + \begin{bmatrix} c_{1,3}^{(j)} \\ c_{2,3}^{(j)} \\ \vdots \\ c_{l,3}^{(j)} \end{bmatrix} \odot \begin{bmatrix} \widehat{u}_1^{(3)} \\ \widehat{u}_2^{(3)} \\ \vdots \\ \widehat{u}_l^{(3)} \end{bmatrix}$$

($\odot$ represents element-wise multiplication).

$$\mathbf{v}^{(j)} = \begin{bmatrix} v_1^{(j)} \\ v_2^{(j)} \\ \vdots \\ v_l^{(j)} \end{bmatrix} = \text{squash}(\mathbf{s}^{(j)}).$$

$$\mathbf{B}^{(j)} \leftarrow \mathbf{B}^{(j)} + \left[\widehat{\mathbf{u}}^{(1)}\ \widehat{\mathbf{u}}^{(2)}\ \widehat{\mathbf{u}}^{(3)}\right] \odot \mathbf{v}^{(j)}$$

$$= \begin{bmatrix} b_{1,1}^{(j)} & b_{1,2}^{(j)} & b_{1,3}^{(j)} \\ b_{2,1}^{(j)} & b_{2,2}^{(j)} & b_{2,3}^{(j)} \\ \vdots & \vdots & \vdots \\ b_{l,1}^{(j)} & b_{l,2}^{(j)} & b_{l,3}^{(j)} \end{bmatrix} + \begin{bmatrix} \widehat{u}_1^{(1)} v_1^{(j)} & \widehat{u}_1^{(2)} v_1^{(j)} & \widehat{u}_1^{(3)} v_1^{(j)} \\ \widehat{u}_2^{(1)} v_2^{(j)} & \widehat{u}_2^{(2)} v_2^{(j)} & \widehat{u}_2^{(3)} v_2^{(j)} \\ \vdots & \vdots & \vdots \\ \widehat{u}_l^{(1)} v_l^{(j)} & \widehat{u}_l^{(2)} v_l^{(j)} & \widehat{u}_l^{(3)} v_l^{(j)} \end{bmatrix}.$$

Finally, return $\mathbf{v}^{(j)}$.
Figure 13.4 is a visualization of $\mathbf{C}^{(2)}$, where $l = 4$.

References

1. Contributing data to deepfake detection research. https://ai.googleblog.com/2019/09/contributing-data-to-deepfake-detection.html. Accessed 24 Sept 2019
2. Dexter studio. http://dexterstudios.com/en/. Accessed 01 Sept 2019
3. Terrifying high-tech porn: Creepy 'deepfake' videos are on the rise. https://www.foxnews.com/tech/terrifying-high-tech-porn-creepy-deepfake-videos-are-on-the-rise. Accessed 17 Feb 2018
4. Afchar D, Nozick V, Yamagishi J, Echizen I (2018) MesoNet: a compact facial video forgery detection network. In: International workshop on information forensics and security (WIFS). IEEE
5. Agarwal S, Farid H, Gu Y, He M, Nagano K, Li H (2019) Protecting world leaders against deep fakes. In: Conference on computer vision and pattern recognition workshops (CVPRW), pp 38–45
6. Alexander O, Rogers M, Lambeth W, Chiang JY, Ma WC, Wang CC, Debevec P (2010) The digital emily project: Achieving a photorealistic digital actor. IEEE Comput Graph Appl 30(4):20–31
7. Averbuch-Elor H, Cohen-Or D, Kopf J, Cohen MF (2017) Bringing portraits to life. ACM Trans Graph
8. Bahadori MT (2018) Spectral capsule networks. In: International conference on learning representations (ICLR)
9. Bappy JH, Simons C, Nataraj L, Manjunath B, Roy-Chowdhury AK (2019) Hybrid lstm and encoder-decoder architecture for detection of image forgeries. IEEE Trans Image Process
10. Bayar B, Stamm MC (2016) A deep learning approach to universal image manipulation detection using a new convolutional layer. In: Workshop on information hiding and multimedia security (IH&MMSEC). ACM
11. Chollet F (2017) Xception: deep learning with depthwise separable convolutions. In: Conference on computer vision and pattern recognition (CVPR). IEEE
12. Cozzolino D, Poggi G, Verdoliva L (2017) Recasting residual-based local descriptors as convolutional neural networks: an application to image forgery detection. In: Workshop on information hiding and multimedia security (IH&MMSEC). ACM

13. Dolhansky B, Bitton J, Pflaum B, Lu J, Howes R, Wang M, Ferrer CC (2020) The deepfake detection challenge dataset. arXiv preprint arXiv:2006.07397
14. Fridrich J, Kodovsky J (2012) Rich models for stage analysis of digital images. IEEE Trans Inf Foren Sec
15. Fried O, Tewari A, Zollhöfer M, Finkelstein A, Shechtman E, Goldman DB, Genova K, Jin Z, Theobalt C, Agrawala M (2019) Text-based editing of talking-head video. In: International conference and exhibition on computer graphics and interactive techniques (SIGGRAPH). ACM
16. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Conference on computer vision and pattern recognition (CVPR), pp 770–778
17. Hinton GE, Krizhevsky A, Wang SD (2011) Transforming auto-encoders. In: International conference on artificial neural networks (ICANN). Springer
18. Hinton GE, Sabour S, Frosst N (2018) Matrix capsules with EM routing. In: International conference on learning representations workshop (ICLRW)
19. Huang G, Liu Z, Van Der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks. In: Conference on computer vision and pattern recognition (CVPR), pp 4700–4708
20. Jiang L, Li R, Wu W, Qian C, Loy CC (2020) Deeperforensics-1.0: A large-scale dataset for real-world face forgery detection. In: Conference on computer vision and pattern recognition (CVPR)
21. Karras T, Laine S, Aila T (2019) A style-based generator architecture for generative adversarial networks. In: Conference on computer vision and pattern recognition (CVPR), pp 4401–4410
22. Kim H, Garrido P, Tewari A, Xu W, Thies J, Nießner M, Pérez P, Richardt C, Zollhöfer M, Theobalt C (2018) Deep video portraits. In: International conference and exhibition on computer graphics and interactive techniques (SIGGRAPH). ACM
23. Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. In: International conference on learning representations (ICLR)
24. Korshunov P, Marcel S (2018) Speaker inconsistency detection in tampered video. In: European signal processing conference (EUSIPCO). IEEE, pp 2375–2379
25. Korshunov P, Marcel S (2019) Vulnerability assessment and detection of deepfake videos. In: International conference on biometrics (ICB)
26. Li L, Bao J, Zhang T, Yang H, Chen D, Wen F, Guo B (2020) Face x-ray for more general face forgery detection. In: Conference on computer vision and pattern recognition (CVPR), pp 5001–5010
27. Li Y, Chang MC, Farid H, Lyu S (2018) In ictu oculi: Exposing AI generated fake face videos by detecting eye blinking. arXiv preprint arXiv:1806.02877
28. Li Y, Yang X, Sun P, Qi H, Lyu S (2020) Celeb-df: A large-scale challenging dataset for deepfake forensics. In: Conference on computer vision and pattern recognition (CVPR), pp 3207–3216
29. Liu A, Wan J, Escalera S, Jair Escalante H, Tan Z, Yuan Q, Wang K, Lin C, Guo G, Guyon I et al (2019) Multi-modal face anti-spoofing attack detection challenge at cvpr2019. In: Conference on computer vision and pattern recognition workshops (CVPRW), pp 0–0
30. Nguyen HH, Fang F, Yamagishi J, Echizen I (2019) Multi-task learning for detecting and segmenting manipulated facial images and videos. In: International conference on biometrics: theory, applications and systems (BTAS). IEEE
31. Nguyen HH, Tieu NDT, Nguyen-Son HQ, Nozick V, Yamagishi J, Echizen I (2018) Modular convolutional neural network for discriminating between computer-generated images and photographic images. In: International conference on availability, reliability and security (ARES). ACM
32. Nguyen HH, Yamagishi J, Echizen I (2019) Capsule-forensics: Using capsule networks to detect forged images and videos. In: International conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 2307–2311

33. Nirkin Y, Keller Y, Hassner T (2019) Fsgan: Subject agnostic face swapping and reenactment. In: International conference on computer vision (ICCV). IEEE
34. Ozbulak U (2019) Pytorch cnn visualizations. https://github.com/utkuozbulak/pytorch-cnn-visualizations
35. Rahmouni N, Nozick V, Yamagishi J, Echizen I (2017) Distinguishing computer graphics from natural images using convolution neural networks. In: International workshop on information forensics and security (WIFS). IEEE
36. Rössler A, Cozzolino D, Verdoliva L, Riess C, Thies J, Nießner M (2019) Faceforensics++: learning to detect manipulated facial images. In: International conference on computer vision (ICCV)
37. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC, Fei-Fei L (2015) ImageNet Large scale visual recognition challenge. Int J Comput Vis
38. Sabir E, Cheng J, Jaiswal A, AbdAlmageed W, Masi I, Natarajan P (2019) Recurrent convolutional strategies for face manipulation detection in videos. In: Conference on computer vision and pattern recognition workshops (CVPRW), pp 80–87
39. Sabour S, Frosst N, Hinton GE (2017) Dynamic routing between capsules. In: Conference on Neural Information Processing Systems (NIPS)
40. Selvaraju RR, Cogswell M, Das A, Vedantam R, Parikh D, Batra D (2017) Grad-cam: visual explanations from deep networks via gradient-based localization. In: International conference on computer vision (ICCV). IEEE, pp 618–626
41. Simonyan K, Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. In: International conference on learning representations (ICLR)
42. Suwajanakorn S, Seitz SM, Kemelmacher-Shlizerman I (2017) Synthesizing obama: learning lip sync from audio. ACM Trans Graph
43. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Conference on computer vision and pattern recognition (CVPR), pp 1–9
44. Tan M, Le Q (2019) Efficientnet: rethinking model scaling for convolutional neural networks. In: International conference on machine learning (ICML), pp 6105–6114
45. Thies J, Zollhöfer M, Nießner M (2019) Deferred neural rendering: image synthesis using neural textures. In: Computer graphics and interactive techniques (SIGGRAPH). ACM
46. Thies, J, Zollhofer, M, Stamminger, M, Theobalt, C, Nießner, M (2016) Face2Face: real-time face capture and reenactment of RGB videos. In: Conference on computer vision and pattern recognition (CVPR). IEEE
47. Tripathy S, Kannala J, Rahtu E (2019) Icface: interpretable and controllable face reenactment using gans. arXiv preprint arXiv:1904.01909
48. Vougioukas K, Center SA, Petridis S, Pantic M (2019) End-to-end speech-driven realistic facial animation with temporal gans. In: Conference on computer vision and pattern recognition workshops (CVPRW), pp 37–40
49. Wang SY, Wang O, Owens A, Zhang R, Efros AA (2019) Detecting photoshopped faces by scripting photoshop. In: International conference on computer vision (ICCV). IEEE
50. Xi E, Bing S, Jin Y (2017) Capsule network performance on complex data. arXiv preprint arXiv:1712.03480
51. Xiang C, Zhang L, Tang Y, Zou W, Xu C (2018) Ms-capsnet: a novel multi-scale capsule network. IEEE Signal Process Lett 25(12):1850–1854
52. Zagoruyko S, Komodakis N (2016) Wide residual networks. In: British machine vision conference (BMVC). BMVA
53. Zakharov E, Shysheya A, Burkov E, Lempitsky V (2019) Few-shot adversarial learning of realistic neural talking head models. arXiv preprint arXiv:1905.08233

54. Zhou P, Han X, Morariu VI, Davis LS (2017) Two-stream neural networks for tampered face detection. In: Conference on computer vision and pattern recognition workshop (CVPRW). IEEE
55. Zhou P, Han X, Morariu VI, Davis LS (2018) Learning rich features for image manipulation detection. In: Conference on computer vision and pattern recognition (CVPR), pp 1053–1061