DeepFake-o-meter: An Open Platform for DeepFake Detection

Yuezun Li[1][†], Cong Zhang[2], Pu Sun[2], Lipeng Ke[3], Yan Ju[3], Honggang Qi[2], and Siwei Lyu[3]

[1]Ocean University of China, China
[2]University of Chinese Academy of Sciences, China
[3]University at Buffalo, State University of New York, USA

Abstract—In recent years, the advent of deep learning-based techniques and the significant reduction in the cost of computation resulted in the feasibility of creating realistic videos of human faces, commonly known as DeepFakes. The availability of open-source tools to create DeepFakes poses as a threat to the trustworthiness of the online media. In this work, we develop an open-source online platform, known as DeepFake-o-meter, that integrates state-of-the-art DeepFake detection methods and provide a convenient interface for the users. We describe the design and function of DeepFake-o-meter in this work.

Index Terms—Multimedia Forensics, DeepFake Detection, Software Engineering

I. INTRODUCTION

The buzzword `DeepFakes` has been frequently featured in the news and social media to refer to realistic impersonating images, videos, and audios that are generated using AI algorithms. Although fabrication and manipulation of digital media are not a new phenomenon [1], powerful AI technology, in particular, deep neural networks (DNNs), and the unprecedented computing power have made it easier than ever to create sophisticated and compelling fakes. Left unchecked, DeepFakes can escalate the scale and danger of disinformation, and fundamentally erode our trust in digital media.

The mounting concerns over the negative impacts of Deep-Fakes have spawned an increasing interest in DeepFake detection. In less than three years, there have been numerous new detection methods of DeepFakes. However, differences in training datasets, hardware, and learning architectures across research publications make rigorous comparisons of different detection algorithms challenging. At the same time, the cumbersome process of downloading, configuring, and installing of individual detection algorithms deny the access of the state-of-the-art DeepFake detection methods to most users. To this end, we have developed an online DeepFake detection platform. It serves three purposes.

- For developers of DeepFake detection algorithms, it provides an API architecture to wrap individual algorithms and run on a third-party remote server.
- For researchers, it is an evaluation/benchmarking platform to compare multiple algorithms on the same input.
- For users, it provides a convenient portal to use multiple state-of-the-art detection algorithms.

† The work was done when the author was a post-doc at University at Buffalo.

Currently we have incorporated 10+ state-of-the-art DeepFake image and video detection methods, and will keep adding more capacities.

In this work, we describe the design and the underlying mechanism of DeepFake-o-meter in details. We start with the overall architecture of the system, which is composed of a web-based front-end that interacts with the user, and an on-server back-end to perform analyses on the input videos. The separation of front-end and back-end is to ensure the security of the user-uploaded data, as well as to accommodate the long running time and the short response time to the users. We further provide an overview of the DeepFake detection algorithms that have been integrated into the current DeepFake-o-meter system. All these algorithms are recent and represent the state-of-the-art in DeepFake detection (two of the algorithms are from the top-performers of the Global DeepFake Detection Challenge). DeepFake-o-meter is designed to be an open-architecture, which can be augmented by incorporating more detection methods over time. We describe the API structures that are needed for third-party developers of DeepFake detection algorithms to have their method integrated into DeepFake-o-meter.

II. PLATFORM DESIGN

This section describes the architecture design of deepfake-o-meter platform. Our platform is composed by three components: *Front-end*, *Back-end* and *Data synchronizing*. The front-end is the website portal to interact with users. The back-end is the core component of this platform, which calls corresponding detection methods to analyze the submitted videos and the data synchronizing is the protocol for exchanging the interested data between front-end and back-end. The overview of the platform architecture is illustrated in Fig.1.

A. Front-end

In order to interact with users, we develop a website to instruct users to submit their interested videos. Fig.2 shows the illustration of the front-end interface. The steps for users to submit videos are as following:

1) Uploading a video either from local machine or using a video URL. Note the maximum video size is constraint to 50 MB in order to maintain a stable and quicker response;
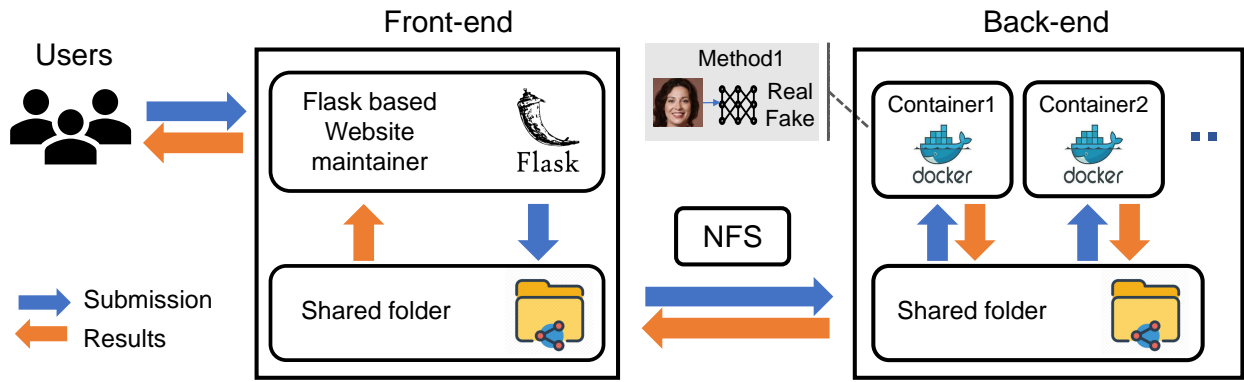2) Selecting the desired deepfake detection methods;

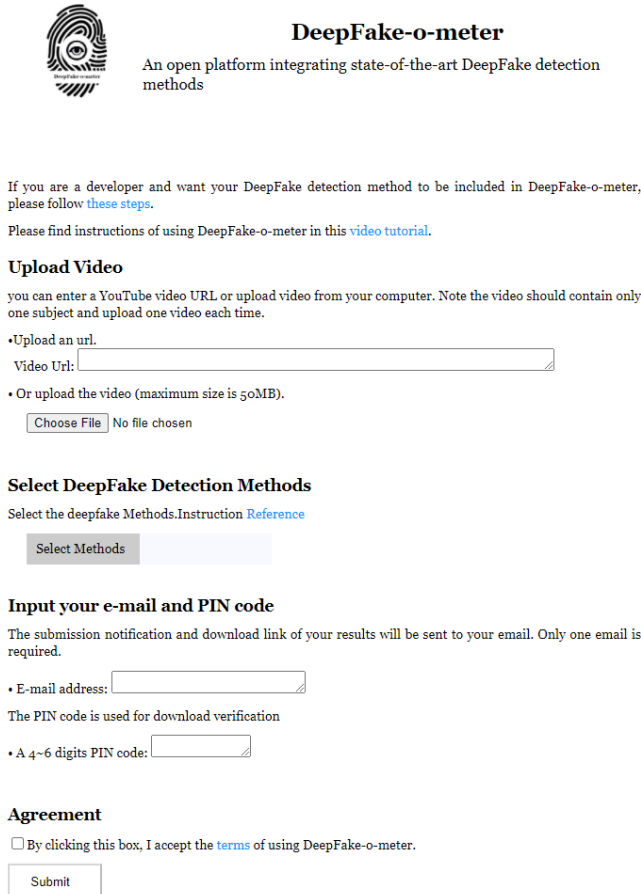Fig. 1. The overview of the platform architecture.

Fig. 2. The illustration of the front-end interface.

3) Inputting user's email address and a 4-6 digits PIN code. Note all the subsequent responses including notification and analyzed results will be sent into the provided email address. The PIN code is used for verification for the analyzed results downloading;

4) The submitted video with other information will sent to the back-end after clicking the submit button.

To construct the front-end, we utilize a python based package *Flask*[1] as the website maintainer. Flask is a lightweight Web Server Gateway Interface (WSGI) framework, depending on the Jinja template engine and the Werkzeug WSGI toolkit. It is a widely used third-party python library for developing web applications. Flask can take over the routing between different web pages and also the service logic after submission, such as email or PIN code validation and packaging the submission under certain requirements.

B. Back-end

The back-end is a computation server mainly for performing deepfake detection methods. In this section, we will describe the key design, returned results and integrated deepfake detection methods respectively.

1) Key design: Once the user submits video from the front-end, the back-end starts to call corresponding detection methods for the submitted video. However, different detection methods depend on different environment settings and different detection methods are designed using different programming styles. Therefore, we design an unified framework to integrate the mainstream deepfake detection methods. Specifically, our framework has two major designs, *Container* and *Coding structure* to handle the diversity of environment and programming styles respectively.

Container. We know virtual machines are the first generation tools to solve the environmental conflict problem on a single machine. However, due to the heavy resources occupation, redundant operation and slow startup, virtual machines are replaced by Containers, which can isolate the process without creating a simulate operating system. Docker[2] is the most popular container solution currently, allowing developers to package their applications and dependent environment into a

---

[1]https://flask.palletsprojects.com/en/1.1.x/
[2]https://www.docker.com/

portable container, which can be run on any other machines. To freely run each method, we independently create docker image for each detection method.

Coding structure In order to maintain each method efficiently, we design a coding structure that can provide an interface for each method to follow. Specifically, we design a base class containing four basic functions, named *run*, *crop_face*, *preproc*, *postproc*, *get_softlabel* and *get_hardlabel*.

- *run*: This is the entrance function to process an input image. The input argument is an image and output is the detection score. Given the input image, this function will internally call the functions *crop_face*, *preproc*, *postproc*, *get_softlabel*, *get_hardlabel* in sequel.
- *crop_face*: Since many methods require to extract the face area from the input image before prediction, this function provides an interface to wrap up the face extraction process. This function is optional.
- *preproc*: After face extraction, many methods apply preprocessing operations to the input face, such as changing the channel order or color space. Therefore, the preprocessing operations can be put in here. This function is also optional.
- *get_softlabel*: This function takes as input the prepossessed face and outputs the confidence score (soft label). Less score denotes the face is faker. The details of calling specific detection methods are wrapped here.
- *get_hardlabel*: Based on the soft label, this function assigns the input to real or fake label.

The code structure will also be exposed to the researchers who would like to integrate new methods into platform. The researchers can follow the structure to split their codes into different functions accordingly.

2) Returned results: The formatting of returned results is also an important point. For better visualization to users, we curve the score of each face along with the corresponding frame and save the prediction of each frame to a video. Besides the visualization, we also sort the score along all frames and calculate the Area Under Curve (AUC) score. The results will be zipped together and sent back to the front-end for user to download. Fig.3 illustrates several examples of the returned results. The left part is submitted video and right part plots the corresponding score. Note our platform supports to run several methods at the same time, thus the bottom two examples contains multiple curves.

3) DeepFake detection methods: Our platform integrates the following deepfake detection methods into this platform.

1) **MesoNet** [2] is a self-designed CNN model that focuses on the mesoscopic properties of images. They provide two variants of MesoNet, namely, *Meso4* and *MesoInception4*. Meso4 uses conventional convolutional layers, while MesoInception4 is based on the more sophisticated Inception modules [3]. We integrate *MesoInception4* into the platform.

2) **FWA** [4] is based on ResNet-50 [5] which detects DeepFake videos by exposing the face warping artifacts

due to the resizing and interpolation operations.

3) **VA** [6] targets the visual artifacts in the face organs such as eyes, teeth and facial contours of the synthesized faces. Two variants of this method are provided: VA-MLP and VA-LogReg. VA-MLP is based on a crafted CNN, and VA-LogReg uses a simpler logistic regression model. We integrate VA-MLP into the platform

4) **Xception** [7] comes with FaceForensics++ dataset. It corresponds to a DeepFake detection method based on the XceptionNet model [8]. This method provides three variants: *Xception-raw*, *Xception-c23* and *Xception-c40*. *Xception-raw* are trained on raw videos, while *Xception-c23* and *Xception-c40* are trained on compressed videos with different degrees, respectively. We integrate *Xception-c23* into the platform.

5) **ClassNSeg** [9] is another CNN based DeepFake detection method that is formulated to a multi-task learning problem to imultaneously detect forgery images and segment manipulated areas.

6) **Capsule** [10] employs the VGG19 [11] capsule structure [12] as the backbone architecture for DeepFake classification.

7) **DSP-FWA** is a further improved method based on FWA, which incorporates a spatial pyramid pooling (SPP) module [13] to better tackle the variations of face resolutions.

8) **CNNDetection** [14] utilizes a standard image classifier trained on only ProGAN [15] , finding it generalizes surprisingly well to unseen architectures, datasets, and training methods.

9) **Upconv** [16] argues that common up-sampling methods (upconvolution or transposed convolution) lack the ability to reproduce spectral distributions of natural training data correctly. They take $2D$ amplitude spectrum as feature and utilize a basic SVM classifier.

10) **WM** ensembles two WS-DAN [17] models (with EfficientNet-b3 [18] and Xception [8] feature extractors, respectively) and a Xception classifier to produce per-face predictions.

11) **Selim** utilizes state-of-the-art encoder, EfficientNet B7, pretrained with ImageNet [19] and noisy student [20], and uses a heuristic way to select 32 frames for each video to average predictions.

The summary of each detection method with code repositories is given in Table I.

C. Data Synchronizing

This section describes the scheme of data synchronizing between front-end and back-end. To enable data sharing between two machines, we utilize the Network File System (NFS) technology. NFS is a distributed file system protocol that can mount remote directories from client to the server. NFS provides a simple and quick way to visit remote systems through the network. For our platform, we need to set up two shared folders. The first one aims to synchronize the data, *i.e.*, user submitted videos and other information such as email
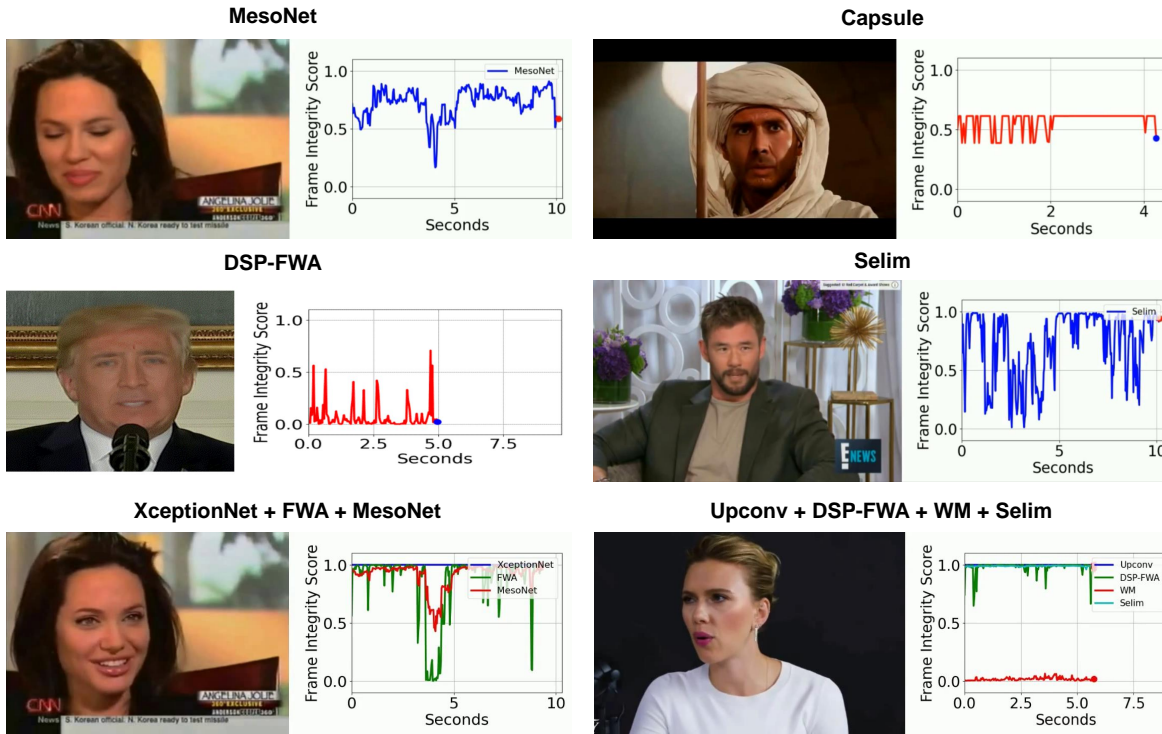
MesoNet **Capsule**

DSP-FWA **Selim**

XceptionNet + FWA + MesoNet **Upconv + DSP-FWA + WM + Selim**

Fig. 3. Examples of generated results.

TABLE I
SUMMARY OF INTEGRATED DEEPFAKE DETECTION METHODS. SEE TEXTS FOR MORE DETAILS.

Methods	Repositories	Release Date
MesoNet [2]	https://github.com/DariusAf/MesoNet	2018.09
FWA [4]	https://github.com/danmohaha/CVPRW2019_Face_Artifacts	2018.11
VA [6]	https://github.com/FalkoMatern/Exploiting-Visual-Artifacts	2019.01
Xception [7]	https://github.com/ondyari/FaceForensics	2019.01
ClassNSeg [9]	https://github.com/nii-yamagishilab/ClassNSeg	2019.06
Capsule [10]	https://github.com/nii-yamagishilab/Capsule-Forensics-v2	2019.10
CNNDetection	https://github.com/peterwang512/CNNDetection	2019.12
DSP-FWA	https://github.com/danmohaha/DSP-FWA	2019.11
Upconv	https://github.com/cc-hpc-itwm/UpConv	2020.03
WM	https://github.com/cuihaoleo/kaggle-dfdc	2020.07
Selim	https://github.com/selimsef/dfdc_deepfake_challenge	2020.07

address, from font-end to the back-end. The second one is used to share the detection results of user's submitted videos from the back-end to the front-end, see Fig.1.

III. CONCLUSION

In this work, we describe an open platform, known as DeepFake-o-meter, for DeepFake detection. This platform is composed by front-end and back-end. The front-end is a web application to interact with users and back-end perform corresponding detection methods on submitted videos. The platform integrates more than 10 state-of-the-art detection methods and it also provides interfaces for researchers to incorporate their method into the platform.

For future works, we will continue integrate more DeepFake detection methods into the platform. Furthermore, we will study the use of multi-GPU platform to accelerate the analysis process. We will also augment the APIs so as to accommodate more general detection methods for other media formats (still images and audio signals).

REFERENCES

[1] H. Farid, *Digital Image Forensics*. MIT Press, 2012.
[2] D. Afchar, V. Nozick, J. Yamagishi, and I. Echizen, "Mesonet: a compact facial video forgery detection network," in *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2018.
[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[4] Y. Li and S. Lyu, "Exposing deepfake videos by detecting face warping artifacts," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2019.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[6] F. Matern, C. Riess, and M. Stamminger, "Exploiting visual artifacts to expose deepfakes and face manipulations," in *IEEE Winter Applications of Computer Vision Workshops (WACVW)*, 2019.

[7] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner, "FaceForensics++: Learning to detect manipulated facial images," in *ICCV*, 2019.

[8] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *CVPR*, 2017.

[9] H. H. Nguyen, F. Fang, J. Yamagishi, and I. Echizen, "Multi-task learning for detecting and segmenting manipulated facial images and videos," in *IEEE International Conference on Biometrics: Theory, Applications and Systems (BTAS)*, 2019.

[10] H. H. Nguyen, J. Yamagishi, and I. Echizen, "Use of a capsule network to detect fake images and videos," *arXiv preprint arXiv:1910.12467*, 2019.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[12] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *NeurIPS*, 2017.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence (TPAMI)*, 2015.

[14] S.-Y. Wang, O. Wang, R. Zhang, A. Owens, and A. A. Efros, "Cnn-generated images are surprisingly easy to spot... for now," in *CVPR*, 2020.

[15] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation," in *ICLR*, 2018.

[16] R. Durall, M. Keuper, and J. Keuper, "Watch your up-convolution: Cnn based generative deep neural networks are failing to reproduce spectral distributions," in *CVPR*, 2020.

[17] T. Hu, H. Qi, Q. Huang, and Y. Lu, "See better before looking closer: Weakly supervised data augmentation network for fine-grained visual classification," *arXiv preprint arXiv:1901.09891*, 2019.

[18] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*, 2019.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR*, 2009.

[20] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, "Self-training with noisy student improves imagenet classification," in *CVPR*, 2020.