DeepFake Disrupter: The Detector of DeepFake Is My Friend
(Supplementary Material)

Anonymous CVPR submission

Paper ID 7865

In the following, we provide Algorithms Pseudocode, Model Architecture and Training details and Algorithms for the training and evaluation process for the DeepFake Disrupter and additional experiments.

1. Algorithm

Algorithm 1: Training with DeepFake Disrupter

Data: real inputs $x$, target conditions $c$, hyper-parameter $\epsilon$, $C_1$, $C_2$, $C_3$ and number of epochs $E$, batch size $B$

Result: Well trained Perturbation generator $P(\cdot)$

1. Initialize Perturbation Generator $P(\cdot)$ with weight $W_p$;
2. Initialize loss weights to $W_i = 1$, for $i = 1, 2, 3$, where $W_i = C_i$;
3. Loading pre-trained DeepFake Generator $G$ with weight $W_G$;
4. Loading pre-trained DeepFake discriminator $D$ with weight $W_D$;
5. for $epoch = 0$ to $E$ do
6.   for $i = 1$ to $B$ do
7.     Compute perturbation $\eta$ by $\eta = P(x)$;
8.     Compute Adversarial Inputs $\widehat{x} = x + \eta$;
9.     Compute $L$ using Eq.(7);
10.    Update $W_{P(t+1)} \leftarrow W_{P(t)}$ using $\nabla_W L(t)$;
11.    Keep $W_G$ and $W_D$ unchanged after each iteration;
12.    Using GradNorm [1] to update loss item weights $W_i$
13.   end
14. end

Algorithm 2: Evaluation Detection Outcomes

Data: a batch of test data: real images or videos $x$, target conditions $c$, number of testing data: $N$

Result: Precision $p$, Recall $r$ and F-1 score $F$

1. Loading pre-trained models $P(\cdot)$, $G(\cdot)$, $D(\cdot)$;
2. for data in batches do
3.   Compute $\eta = P(x)$;
4.   Compute $x_{fake} = G(x + \eta, c)$;
5.   Compute $x_{real} = x + \eta$;
6. end
7. Pass $N$ real inputs $x$, $N$ fake inputs $x_{fake}$ into DeepFake Detector $D$;
8. Compute $p = \text{True}_\text{Real}/(\text{True}_\text{Real} + \text{False}_\text{Real})$;
9. $r = \text{True}_\text{Fake}/(\text{True}_\text{Fake} + \text{False}_\text{Fake})$;
10. $F1 = 2 \ast (P \ast r)/(p + r)$;
11. Pass $N$ perturbed inputs $x_{real}$ into $D$;
12. Compute $r = \text{True}_\text{Real}/(\text{True}_\text{Real} + \text{False}_\text{Fake})$

2. Model Architecture and Training Details

Perturbation Generator For perturbation generator $P(\cdot)$, we choose U-Net [5]. The U-net architectures can be divided into two sections: The encoding section and the decoding section. We use 2D U-Net for image-based experiment and 3D U-Net for video-based experiment. In the encoding section, we apply contraction blocks consists of 2D or 3D convolution and max-pooling layers to encode the source inputs. In the decoding section, we apply expansion blocks consisting of 2D or 3D transpose convolution as well as normal 2D or 3D convolutions. The center part of U-Net is that each feature map of the encoding section has a shortcut connection with the corresponding feature map in the decoding section.

DeepFake Generator We use StarGAN [2], GANimation [4] and First-Order-Motion Model [8] to illustrate that our
The proposed pipeline can be used for different DeepFake manipulation systems. For StarGAN, we use a pretrained generator model in the open-source implementation used by [7]. This generator is trained on the CelebA dataset with seven domains including black hair, blond hair, brown hair, gender (male/female) and aged (young/old). For GANimation, we use its official open-source pretrained generators trained for 37 epochs on the CelebA dataset for 80 action units (AU) based on the Facial Action Unit Coding System [3]. For First-Order Motion Model, we deploy its official open-source framework. The core part of this framework includes motion estimation and image generation. Firstly, a source image and a driving video will be fed into Motion Module, then the Motion module will use a keypoint detector to extract motion representations, after which generating a dense optical flow and occlusion map mapping from the driving video to the source image. Finally, the generation process will provide quality animations by feeding into the source image and Motion Module’s outputs.

**DeepFake Detector** As our work is to test the effectiveness of perturbation generator rather than detection power of deepfake detectors, we use commonly used backbone for various SOTA deepfake detectors, namely Xception, Resnet18 and Resnet50 as our detection architectures. All these models are trained on the FaceForensics++ datasets. In terms of image level detectors, for Xception architecture, we choose the open-source pretrained model from [6]. For Resnet18 and Resnet50, we trained 100 epochs to get a classification accuracy at 96% and 98% respectively. In terms of video level discriminators, we choose 3D Xception and 3D resnet18 and trained 100 epochs to get accuracy at 91% and 95% respectively.

**Hyper Parameters** There are several hyper parameters. The first one is $\epsilon$ introduced in Eq. (1) in the main submission to constrain the scale of perturbation. In order to ensure the perturbation to be human imperceptible, we follow baseline method [7] to set $\epsilon = 0.05$. For hyper parameters $C_1$, $C_2$ and $C_3$ that balance different loss items, we initialize them to be 1 at the beginning, and then we follow GradNorm [1] algorithm to adaptively update the loss weight items at each iteration. In the GradNorm algorithm, there is a further hyper parameters $\alpha$ that corresponding to the strength of restoring force, and we set $\alpha = 0.1$ for all our experiments. For detailed explanation of the GradNorm algorithm, please refer to the original work.

### 3. Additional Experiments

**Adaptive Attack** An attacker knowing the detector challenges the defender. But preparing for the worst could perhaps ensure that the worst will not happen. We thus train an enhanced deepfake generator $G'$ by incorporating the detector loss. We use $G'$ and detector $D$ to train disrupter $P$.

We choose StarGAN as our attacker, Xception as the detector. In Table 1, if the attacker knows the detector, [20] will have a much lower precision and F1-score (e.g. 0.52 & 0.67 in Disrupting StarGAN), but the performance of our DeepFake Disrupter can remain similar as before.

<table>
<thead>
<tr>
<th>Disruption Methods</th>
<th>precision</th>
<th>recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disrupting StarGAN [7]</td>
<td>0.64</td>
<td>0.99</td>
<td>0.78</td>
</tr>
<tr>
<td>Disrupting StarGAN [7]-Adaptive</td>
<td>0.52</td>
<td>0.99</td>
<td>0.67</td>
</tr>
<tr>
<td>DeepFake Disrupter(Ours)</td>
<td>0.86</td>
<td>0.99</td>
<td>0.92</td>
</tr>
<tr>
<td>DeepFake Disrupter(Ours-Adaptive)</td>
<td>0.83</td>
<td>0.99</td>
<td>0.89</td>
</tr>
</tbody>
</table>

Table 1. Deepfake detection performance with adaptive attacks.

**Ablation on different test size** In Table 2, our algorithm can achieve superior results under three larger test sizes evaluated by Xception/Resnet18 for StarGAN/GANimation generators.

<table>
<thead>
<tr>
<th>Disruption Methods</th>
<th>Xception</th>
<th>Resnet18</th>
</tr>
</thead>
<tbody>
<tr>
<td>StarGAN [2]</td>
<td>0.72</td>
<td>0.70</td>
</tr>
<tr>
<td>Disrupting StarGAN [7]</td>
<td>0.78</td>
<td>0.77</td>
</tr>
<tr>
<td>DeepFake disrupter (ours)</td>
<td><strong>0.92</strong></td>
<td><strong>0.93</strong></td>
</tr>
<tr>
<td>GANimation [4]</td>
<td>0.74</td>
<td>0.76</td>
</tr>
<tr>
<td>Disrupting GANimation [7]</td>
<td>0.82</td>
<td>0.79</td>
</tr>
<tr>
<td>DeepFake disrupter (ours)</td>
<td><strong>0.89</strong></td>
<td><strong>0.88</strong></td>
</tr>
</tbody>
</table>

Table 2. F1-score under different test image size 100, 500, 1000

### References


