Supplementary Material for
UCF: Uncovering Common Features for Generalizable Deepfake Detection

1. Overview

This supplementary material provides additional details regarding our implementation settings, visualizations, and experimental results, including:

- Detailed network architecture (see Sec. 2).
- Detailed implementation settings (see Sec. 3).
- Embedding visualizations (see Sec. 4).
- Additional experiments (see Sec. 5).

2. The Network Architecture

Encoder The encoder proposed in the manuscript has a fingerprint encoder and a content encoder to extract the fingerprint and content features, respectively. Note that both the fingerprint encoder and content encoder are based on Xception [8], with model parameters initialized by pre-training on ImageNet. We also try other backbones and show that our framework can be applied to various existing CNN architectures (Tab. 7 in the manuscript and Tab. 5). Both the fingerprint encoder and content encoder utilize MLP layers for performing multi-task classification. The details of the encoder are shown in Fig. 1.

3. Detailed Implementation Settings

In line with FF++[8], we adopt the official data splits and use 740 videos for training, 140 videos for validation, and 140 videos for testing. To ensure a balanced training dataset, we maintain a 1:1 ratio of real and fake images by randomly selecting a fake image from forgery videos and a real image from genuine videos. To facilitate the disentanglement of common and specific forgery features, we propose an augmentation strategy that involves swapping features within the same labels for both common and specific features. We provide further details of this strategy in Alg. 1. Our proposed approach is implemented in PyTorch, and we utilize one Nvidia A100 GPU per experiment.

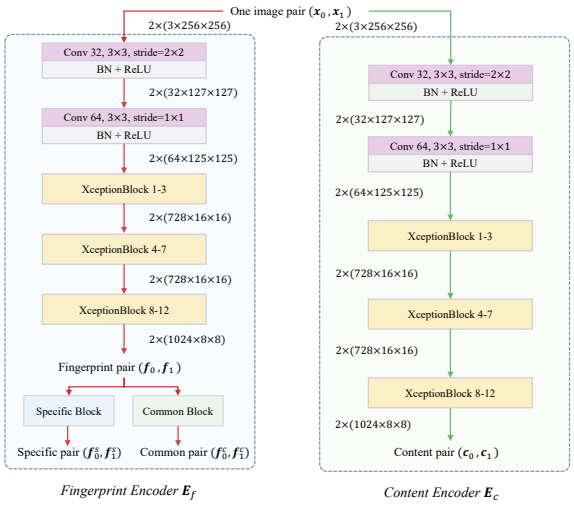

Figure 1: The architecture of the proposed encoder. For both the fingerprint and content encoders, the input image pair will be first passed through two convolutional layers and then fed into three "XceptionBlock". Different from the original Xception [8], we drop the exit flow, which includes two separable convolutional layers to adjust the number of channels. Instead, we design two additional blocks in the fingerprint encoder, *i.e.,* the "Specific Block" and the "Common Block". These blocks are implemented using three convolutional layers to extract specific and common fingerprint features (See Fig. 2). The extracted specific and common fingerprint features are then fed through their respective heads (See Fig. 2), which consist of three MLP layers, for multi-task classification.

4. Additional Embedding Visualizations

Our proposed approach involves four distinct features, *i.e.,* whole forgery, specific forgery, common forgery, and content features (Sec. 3 in the manuscript). To validate our approach, we perform a t-SNE visualization [10] (Fig. 2 in the manuscript), demonstrating that the baseline Xception model [8] and our specific module only capture method-specific artifacts, while our common module captures the common features across different forgeries. We also con-

Training	Method	CelebDF			DFD			DFDC		
		AUC ↑	AP ↑	EER ↓	AUC ↑	AP ↑	EER ↓	AUC ↑	AP ↑	EER ↓
FF++	Xception [8]	0.672	0.748	0.367	0.727	0.889	0.398	0.651	0.822	0.443
	Liang *et al.* [6]	0.706	0.752	0.353	0.829	0.926	0.211	0.700	0.878	0.392
	Ours	**0.824**	**0.847**	**0.287**	**0.945**	**0.973**	**0.104**	**0.805**	**0.895**	**0.301**

Table 1: Comparisons of generalization ability with disentanglement-based methods in terms of other metrics. The metrics are AUC, AP (Average Precision), and EER (Equal Error Rate). For AUC and AP, the higher the better. For EER, the lower the better. Our results demonstrate that our method performs better than both Liang *et al.* [6] and Xception [8] under different datasets with different evaluation metrics, and the best results are highlighted in bold font.

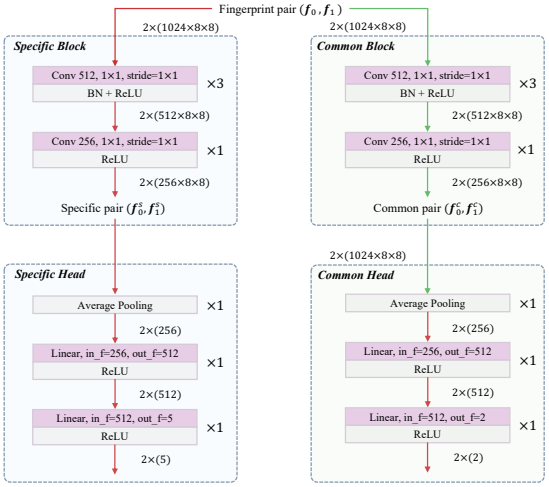

Figure 2: The architecture of the "Specific Block" and "Common Block" in the proposed encoder and the "Specific Head" and "Common Head" in our framework. The encoder consists of two blocks, namely the "Specific Block" and "Common Block", while the framework consists of two heads, namely the "Specific Head" and "Common Head". As we only utilize the forgery features for detection, without considering the content, we input the fingerprint pair into both the "Specific Block" and "Common Block", as well as the "Specific Head" and "Common Head", to obtain the prediction outputs and learn method-specific patterns and common features, respectively. It should be noted that this figure only considers the scenario where we train on the FF++ [8], resulting in outputs of shapes 5 and 2 for the specific and common heads, respectively.

duct ablation studies in the manuscript (Tab. 6) to compare binary classification results with different forgery features.

In this section, we conduct additional t-SNE visualizations to show the embedding visualizations of the whole forgery and content features in our framework, as shown in Fig. 3. The results demonstrate that the whole and specific forgery features learn method-specific artifacts, while our common features can capture the common features across different forgeries. Additionally, the content features do not differentiate between real and each forgery technology, which is expected.

5. Additional Experiments

5.1. Comparison with Disentanglement Methods

In our main script, we identify another disentanglement-based detector, *i.e.,* Liang *et al.* [6]. Since the code of their framework is not available as an open-source resource and the original paper only utilizes one manipulation forgery method for training (*i.e.,* Deepfake [2] in FF++ [8]), we carefully reimplement their framework by following the settings of the original paper. Results in the manuscript (Tab. 3) show that our model outperforms Liang *et al.* on all testing datasets, demonstrating the efficacy of uncovering common features. In this section, we provide additional experiment results of our method with Liang *et al.* [6] and Xception [8] by using different evaluation metrics, including AUC, AP (Average Precision), and EER (Equal Error Rate). Note that both Liang *et al.* and our framework utilize Xception as the backbone. Following our main script (Tab. 3), we use FF++ [8] as the training data and use CelebDF [5], DFD [1], and DFDC [3] as the testing data. The result is shown in Tab. 1. From the result, we can see that our proposed method outperforms both Liang *et al.* and Xception on all testing datasets with all evaluation metrics.

5.2. More ablation studies

This section presents additional ablation studies aimed at investigating the impact of different parameter settings on the performance of the model. Specifically, we focus on the weight values of the specific loss, contrastive loss, and the margin in the contrastive loss. We first search for the optimal parameters for the specific loss, then fix the weight of the specific loss and proceed to search for the optimal parameters for the contrastive loss. Finally, we search for the optimal margin value with the specific and contrastive loss weights fixed.

Comparison with binary classification results with different weights of the specific loss. In our manuscript, the specific loss is a crucial component of our overall objective function (Eq. (10)) as it helps to balance the influence of

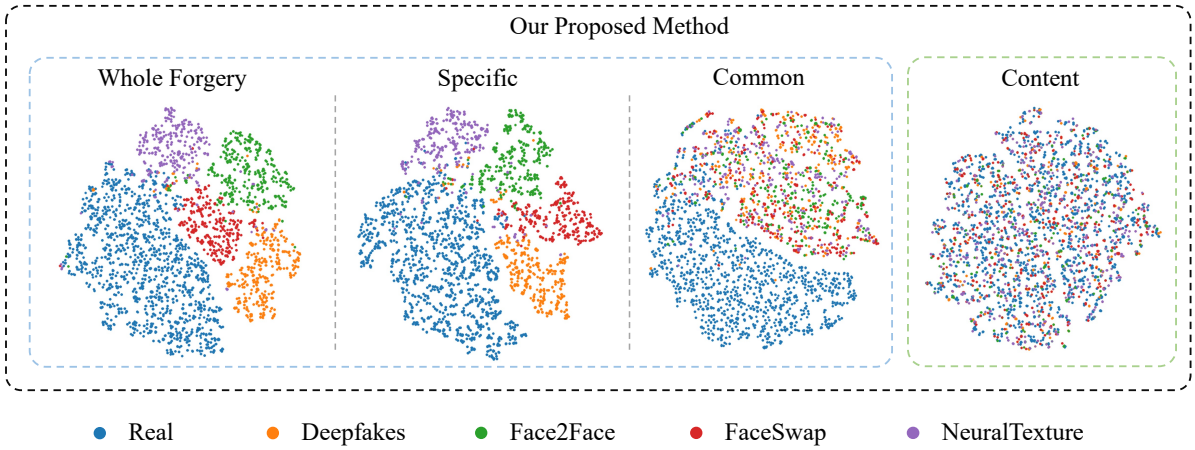

Figure 3: The t-SNE [10] visualization of features extracted from our framework on FF++ [8]. In the visualization, images generated by the four methods locate separately in the latent space, which reveals that the whole forgery actually learns method-specific features, consistent with our forgery-specific module. This observation explains that both the whole and specific forgery features can mainly recognize specific types of forgeries and thus fail to generalize well to a broader range of forgeries. Additionally, as expected, the common module of our method captures the common forgery features across different methods, while the content module captures only forgery-irrelevant features.

Table 2: Testing AUC results obtained for different weights of the specific loss on various datasets are shown. The best result is highlighted in bold font. "Avg." represents the average AUC for cross-datasets. Other hyper-parameters λ_3 and α are set to be 0.05 and 3 in this setting.

Training	Method	Testing AUC			
		CelebDF	DFD	DFDC	Avg.
FF++	λ_1=0.01	0.806	0.936	0.787	0.843
	λ_1=0.1	**0.824**	**0.945**	**0.805**	**0.858**
	λ_1=1	0.806	0.927	0.791	0.841
	λ_1=10	0.768	0.934	0.772	0.825

Table 3: Testing AUC results obtained for different weights of the contrastive loss on various datasets are shown. The best result is highlighted in bold font "Avg." represents the average AUC for cross-datasets. Other hyper-parameters λ_1 and α are set to be 0.1 and 3 in this setting.

Training	Method	Testing AUC			
		CelebDF	DFD	DFDC	Avg.
FF++	λ_3=0.01	0.823	0.944	0.793	0.853
	λ_3=0.05	**0.824**	**0.945**	**0.805**	**0.858**
	λ_3=0.1	0.802	0.933	0.788	0.841
	λ_3=0.5	0.806	0.924	0.780	0.837

specific features on the final result. To evaluate the generalization performance of the model under various weight values (*i.e.,* $\lambda_1 = 0.01, 0.1, 1, 10$), we conducted ablation studies to evaluate the impact of the different weights of the specific loss (λ_1). The results, as shown in Tab. 2, suggest the existence of a performance peak for different choices of λ_1. The table reveals that when λ_1 is either too large ($\lambda_1 = 10$) or too small ($\lambda_1 = 0.01$), the model's performance is limited. Our experiments indicate that the optimal choice for λ_1 is 0.1.

Comparison with binary classification results with different weights of the contrastive loss. The contrastive regularization loss is also an essential component of our overall objective function (Eq. (10)). We conduct ablation studies to evaluate the impact of different weights of the contrastive loss (λ_3) on the model's generalization performance under various weight values (*i.e.,* $\lambda_3 = 0.01, 0.05, 0.1, 0.5$). The results, presented in Tab. 3, indicate that the optimal choice for λ_3 is 0.05.

Margin in the contrastive loss Margin is a hyper-parameter in our contrastive regularization loss in the manuscript (Eq. (5)). It determines the minimum distance between the anchor-positive pair and the anchor-negative pair. Specifically, if the difference between the distances of anchor-positive and anchor-negative pairs is less than α, then the loss value is set to 0. Otherwise, the loss value is proportional to the difference between the distances. In our case, it controls the separation between the common and specific forgery representations of the real and fake samples, ensuring that they are distinct from each other. We

Table 4: Testing AUC results obtained for different values of the margin α in the contrastive loss on various datasets are shown. The best result is highlighted in bold font. "Avg." represents the average AUC for cross-datasets. Other hyper-parameters λ_1 and λ_3 are set to be 0.1 and 0.05 in this setting.

Training	Method	Testing AUC			
		CelebDF	DFD	DFDC	Avg.
FF++	margin=1	0.801	**0.955**	0.792	0.849
	margin=2	0.799	0.936	0.786	0.840
	margin=3	**0.824**	0.945	**0.805**	**0.858**
	margin=4	0.789	0.923	0.782	0.831

Table 5: Testing AUC results obtained for different backbones applied in our framework. The best result is highlighted in bold font. "Avg." represents the average AUC for cross-datasets.

Training	Backbone	Testing AUC			
		CelebDF	DFD	DFDC	Avg.
FF++	ResNet34 [4]	0.809	0.923	0.729	0.820
	Efficient-B1 [9]	0.827	0.933	0.800	0.853
	Xception [8]	0.824	0.945	**0.805**	0.858
	ConvNext [7]	**0.869**	**0.946**	0.802	**0.872**

conduct ablation studies to evaluate the performance of different weight values (*i.e.,* $\alpha = 1, 2, 3, 4$). Results in Tab. 4 show that the optimal choice for α is 3.

5.3. Other choices of backbone

In this section, we investigate the choice of backbone on the generalization ability of our proposed framework. In the manuscript (Sec. 4.3), we consider two backbones for our framework (Xception [8] and ConvNeXt [7]) to demonstrate that our proposed framework can largely improve the generalization performance of both Xception and ConvNeXt backbones. To further show that our framework is effective and applicable to different backbone choices, we additionally conduct experiments on two backbones in our proposed framework, *i.e.,* ResNet-34 [4], Efficient-B1 [9]. Results from Tab. 5 show that larger networks tend to result in greater generality. Overall, our proposed framework has the potential to enhance the generalization ability of various existing backbone networks, which could be explored in future research.

Algorithm 1 Pseudo code for swapping common and specific features in a PyTorch style.

```
# spe_label, f_spe: list of types of the specific label
# f_share, f_spe: common and specific feature tensors
# spe_label: label of the specific features
# aug_idx: index for data augmentation
# bs: batch_size

def swap_specific_features(spe_label, f_spe):
    # move torch tensor to a python list in cpu
    spe_label = spe_label.cpu().numpy().tolist()
    # get the input specific label
    index_list = list(range(len(spe_label)))
    # init a dictionary, where its key is the type and value is the index
    spe_dict = defaultdict(list)
    # do for-loop to get specific dict
    for i, one_type in enumerate(spe_label):
        spe_dict[one_type].append(index_list[i])
    # shuffle the value list of each key
    for keys in spe_dict.keys():
        random.shuffle(spe_dict[keys])
    # generate a new index list for the value list
    new_index_list = []
    for one_type in spe_label:
        value = spe_dict[one_type].pop()
        new_index_list.append(value)
    # swap the value_list by new_index_list
    f_spe_new = f_spe[new_index_list]
    return f_spe_new

def swap_common_features(bs, f_share):
    # Swap the real features
    idx_list = list(range(0, bs//2))  # the first half is real
    random.shuffle(idx_list)
    f_share[0: bs//2] = f_share[idx_list]  # swap real common features
    # Swap the real features
    idx_list = list(range(bs//2, bs))  # the second half is forgery
    random.shuffle(idx_list)
    f_share[bs//2: bs] = f_share[idx_list]  # swap forgery common features
    return f_share
```

5.4. Feature Swapping

To achieve improved disentanglement, we introduce a feature-swapping strategy that prioritizes the proximity of features with the same label while encouraging distance between those with different labels. In high-dimensional vector spaces, we believe that common features of different categories should be far apart while those within the same category should be close. Similarly, for specific instances, features of different types should be distant while those with the same type should be proximate. Through the use of this augmentation strategy, we can obtain more distinctive disentangled features and ultimately achieve better generalization performance. To provide further clarity on this strategy, we present the pseudo codes in Alg. 1, which outline the detailed steps. The implementation details for each step are provided in the comments within the code.

References

[1] Deepfakedetection. https://ai.googleblog.com/2019/09/contributing-data-to-deepfakedetection.html Accessed 2021-04-24.

[2] DeepFakes. www.github.com/deepfakes/faceswap Accessed 2021-04-24.

[3] Deepfake detection challenge. https://www.kaggle.com/c/deepfake-detection-challenge Accessed 2021-04-24.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[5] Yuezun Li, Xin Yang, Pu Sun, Honggang Qi, and Siwei Lyu. Celeb-df: A new dataset for deepfake forensics. In *CVPR*, 2020.

[6] Jiahao Liang, Huafeng Shi, and Weihong Deng. Exploring disentangled content information for face forgery detection. In *ECCV*, pages 128–145. Springer, 2022.

[7] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *CVPR*, pages 11976–11986, 2022.

[8] Andreas Rossler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics++: Learning to detect manipulated facial images. In *ICCV*, 2019.

[9] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, pages 6105–6114. PMLR, 2019.

[10] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 2008.