Reverse Engineering of Deceptions

on Machine- and Human-Centric Attacks

Suggested Citation: Yuguang Yao, Guo Xiao, Vishal Asnani, Yifan Gong, Jiancheng Liu, Xue Lin, Xiaoming Liu and Sijia Liu (2018), "Reverse Engineering of Deceptions", : Vol. xx, No. xx, pp 1–18. DOI: 10.1561/XXXXXXXXX.

Yuguang Yao
Michigan State University
yaoyugua@msu.edu

Xiao Guo
Michigan State University
guoxia11@msu.edu

Vishal Asnani
Michigan State University
asnanivi@msu.edu

Yifan Gong
Northeastern University
gong.yifa@northeastern.edu

Jiancheng Liu
Michigan State University
liujia45@msu.edu

Xue Lin
Northeastern University
xue.lin@northeastern.edu

Xiaoming Liu
Michigan State University
liuxm@msu.edu

Sijia Liu
Michigan State University
liusiji5@msu.edu

now
the essence of knowledge
Boston — Delft

Contents

Reverse Engineering of Deceptions

Yuguang Yao[1], Guo Xiao[2], Vishal Asnani[3], Yifan Gong[4], Jiancheng Liu[5], Xue Lin[6], Xiaoming Liu[7] and Sijia Liu[8]

[1]*Michigan State University; yaoyugua@msu.edu*
[2]*Michigan State University; guoxia11@msu.edu*
[3]*Michigan State University; asnanivi@msu.edu*
[4]*Northeastern University; gong.yifa@northeastern.edu*
[5]*Michigan State University; liujia45@msu.edu*
[6]*Northeastern University; xue.lin@msu.edu*
[7]*Michigan State University; liuxm@msu.edu*
[8]*Michigan State University; liusiji5@msu.edu*

ABSTRACT

This work presents a comprehensive exploration of Reverse Engineering of Deceptions (RED) in the field of adversarial machine learning. It delves into the intricacies of machine- and human-centric attacks, providing a holistic understanding of how adversarial strategies can be reverse-engineered to safeguard AI systems. For machine-centric attacks, we cover reverse engineering methods for pixel-level perturbations, adversarial saliency maps, and victim model information in adversarial examples. In the realm of human-centric attacks, the focus shifts to generative model information inference and manipulation localization from generated images. Through this work, we offer a forward-looking perspective on the challenges and opportunities associated with RED. In addition, we provide foundational and practical insights in the realms of AI security and trustworthy computer vision.

Yuguang Yao, Guo Xiao, Vishal Asnani, Yifan Gong, Jiancheng Liu, Xue Lin, Xiaoming Liu and Sijia Liu (2018), "Reverse Engineering of Deceptions", : Vol. xx, No. xx, pp 1–18. DOI: 10.1561/XXXXXXXXX.

1

Introduction

In the domain of trustworthy computer vision (CV) and adversarial machine learning (ML), the emergence of Reverse Engineering of Deceptions (**RED**) marks a pivotal evolution. This article is poised to grant readers a profound understanding of RED, a novel and dynamic field at the intersection of AI security and CV (DARPA, 2021). The existing body of research in the field has exhaustively explored *machine-centric* deceptions, such as adversarial attacks aimed at misleading *ML models* (Goodfellow *et al.*, 2014b; Madry *et al.*, 2017), and *human-centric* deceptions, particularly the utilization of generative models to fool *human decision-making* (Creswell *et al.*, 2018; Dhariwal and Nichol, 2021). In the above context, RED introduces an innovative adversarial learning paradigm with the ambitious goal of deciphering and cataloging the intricacies of attacks targeted at both machines and humans.

The concept of RED is not merely an academic exercise; it is a crucial response to the increasing sophistication of adversarial tactics in CV. This burgeoning field seeks to automate the process of recovering and indexing attack 'fingerprints' embedded in adversarial instances. The core question that RED endeavors to answer is: Given an attack, whether machine-centric or human-centric, can we reverse-engineer

the adversary's underlying knowledge and the specifics of their attack toolchains? This question extends beyond the realm of traditional adversarial detection and defense techniques, delving into the deeper layers of adversary intentions, methodologies, and the nuances of model generation.

RED for 'machine-centric' attacks. Recent years have witnessed a rapid expansion in RED research. As for adversarial attacks designed to fool discriminative models, *i.e.*, machine-centric attacks, RED aims not only to defend against these attacks but also to infer the adversary's knowledge, including their identity, objectives, and the details of the attack perturbations. Recent works in this area, such as those by (Nicholson and Emanuele, 2023; Wang *et al.*, 2023; Maini *et al.*, 2021; Zhou and Patel, 2022; Guo *et al.*, 2023c; Moayeri and Feizi, 2021), have focused on reverse-engineering the type of attack generation methods and the associated hyperparameters, like perturbation radius and step number. There is also a growing interest in estimating or attributing adversarial perturbations used in constructing adversarial images (Gong *et al.*, 2022; Goebel *et al.*, 2021; Souri *et al.*, 2021; Thaker *et al.*, 2022), an endeavor closely related to adversarial purification techniques (Srinivasan *et al.*, 2021; Shi *et al.*, 2021; Yoon *et al.*, 2021; Nie *et al.*, 2022) which aim to mitigate the impact of such attacks on model predictions. We note that RED is distinct from research focused on reverse engineering model hyperparameters in a black-box setting (Oh *et al.*, 2019; Wang and Gong, 2018), which typically involves estimating model attributes from the model's prediction logits. By contrast, in the realm of RED against adversarial attacks, the victim model attribute is unknown, and the only available information is the dataset of attack instances.

RED for 'human-centric' attacks. Generative Models (GMs) nowadays generate visually compelling images. However, they also introduce the risk of *human-centric attacks*, leading to the inadvertent spread of misinformation and threats to the trustworthiness of social media. To counteract these negative impacts, two recent research directions aim to reverse engineering deception — model parsing of generative models and manipulation localization. Firstly, model parsing (Asnani *et al.*, 2023b;

Guo *et al.*, 2023a) involves extracting GM hyperparameters used in creating falsified images. Unlike previous model parsing works (Tramèr *et al.*, 2016; Oh *et al.*, 2019; Hua *et al.*, 2018; Batina *et al.*, 2019), which often required additional prior knowledge to predict training information or model hyperparameters, (Asnani *et al.*, 2023b) employs a clustering-based approach to estimate mean and standard deviation across different GMs. In contrast, (Guo *et al.*, 2023a) introduces a novel framework based on Graph Convolution Networks to learn dependencies among these 37 hyperparameters. Secondly, manipulation localization is a well-established computer vision research topic that identifies tampered regions to deduce crucial information about deception. Existing work has predominantly focused on manipulation in either the image editing (Wu *et al.*, 2019; Hu *et al.*, 2020; Zhou *et al.*, 2018; Mayer and Stamm, 2018; Chen *et al.*, 2021; Wang *et al.*, 2022; Zhou *et al.*, 2020) or digital domain (Dang *et al.*, 2020; Zhao *et al.*, 2021; Huang *et al.*, 2022). In contrast, we introduce two manipulation localization algorithms (Asnani *et al.*, 2023a; Guo *et al.*, 2023b) in this work, which are capable of handling both domains simultaneously.

Objective and impact of this tutorial. We aim to present an all-encompassing exploration of RED, from its algorithmic underpinnings to its burgeoning applications, complemented by practical implementations. Delving into various formulations of RED, this article will unravel both the challenges and opportunities inherent in this field. The significance of RED becomes particularly salient in high-stakes applications, such as biometrics, autonomous driving, and healthcare, where the defense against and diagnosis of attacks are paramount. The implications of RED could extend beyond the boundaries of academic research, impacting the real-world deployment of machine intelligence.

Furthermore, the pressing need for security and trustworthiness in future CV models underscores the importance of our work. As the popularity of adversarial ML surges, it becomes increasingly crucial to ensure that research progress aligns with the demand for robust and reliable AI systems. By investigating how one can reverse-engineer threat models from adversarial instances, such as adversarial examples and images synthesized by generative models, our article offers new

perspectives and insights.

Organization. The remainder of this article is structured as follows: Chapters 2 and 3 will offer insights into the RED in machine-centric adversarial images and their potential implications for model parsing of adversarial attacks (*i.e.*, inferring details of a victim model used for attack generation). Chapters 4 and 5 will delve into the RED in the human-centric attack, focusing on two research topics: model parsing of generative models and manipulation localization. Model parsing of generative models involves predicting hyperparameters used in the generative model, given the generated image. In parallel, manipulation localization predicts a segmented mask to identify the manipulated region, and this segmented mask serves to reverse engineer crucial information about the malicious manipulation method. Finally, in Chapter 6, we will explore the broader impact of RED on other pertinent domains and offer our concluding remarks.

2

Reverse Engineering of Adversarial Examples

Chapter overview. In this chapter, we will detail on the definition over the reverse engineer of deceptions (RED) on machine-centric adversarial attacks. Prior to introducing the concept of RED, we revisit the vulnerability of neural network-based image classifiers to adversarial examples, *i.e.*, images slightly altered by an adversary to deceive this classifier. The field of adversarial ML has extensively studied generating and defending against such adversarial attacks, yet the process of reverse-engineering adversarial perturbations from these images remains largely uncharted territory. This untapped area of research heralds the novel adversarial learning paradigm, RED, which—if successful—promises the ability to estimate and nullify adversarial perturbations, potentially restoring the original, untainted images.

The pursuit of RED on adversarial attacks, however, is fraught with challenges. Adversarial perturbations, often subtle and carefully crafted, are notoriously difficult to discern and reverse using a straightforward RED objective. For instance, a conventional image denoising approach might focus excessively on minimizing reconstruction error, neglecting the preservation of classification characteristics inherent to the adversarial perturbations. To address these challenges, this chapter will

formalize the RED problem and elucidate a set of underlying principles critical for designing an effective RED strategy. Our findings indicate that ensuring prediction alignment and incorporating appropriate data augmentation—specifically spatial transformations—are paramount for a robust and generalizable RED method.

Building on these principles, we will introduce a novel framework that synergizes class-discriminative denoising with RED, designated as the Class-Discriminative Denoising based RED (CDD-RED). This framework is rigorously tested across various evaluation metrics, including pixel-level, prediction-level, and attribution-level alignments, and against a multitude of attack generation methods, and even adaptive attacks. Our experiments validate the effectiveness of CDD-RED, showcasing its proficiency in tackling the multifaceted challenges posed by adversarial examples.

2.1 Background and RED Formulation

This section commences by addressing the threat scenario at the core of our focus: adversarial attacks on images. Subsequently, we outline the problem of RED and elucidate the challenges it presents.

2.1.1 Threat Model

We focus on ℓ_p attacks, where the *adversary's goal* is to generate imperceptible input perturbations to fool a well-trained image classifier. Formally, let $\mathbf{x}$ denote a benign image, and $\boldsymbol{\delta}$ be an additive perturbation variable. Given a victim classifier f and a perturbation strength tolerance ϵ (in terms of, *e.g.*, ℓ_∞-norm constraint $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$), the desired *attack generation algorithm* $\mathcal{A}$ then seeks the optimal $\boldsymbol{\delta}$ subject to the perturbation constraints. Such an attack generation process is denoted by $\boldsymbol{\delta} = \mathcal{A}(\mathbf{x}, f, \epsilon)$, resulting in an adversarial example $\mathbf{x}' = \mathbf{x} + \boldsymbol{\delta}$. Here $\mathcal{A}$ can be fulfilled by different attack methods, e.g., FGSM (Goodfellow *et al.*, 2014b), CW (Carlini and Wagner, 2017), PGD (Madry *et al.*, 2017), and AutoAttack (Croce and Hein, 2020).

2.1.2 Problem Statement

Different from conventional defenses to detect or reject adversarial instances (Pang *et al.*, 2020; Liao *et al.*, 2018; Shafahi *et al.*, 2020; Niu *et al.*, 2020), RED aims to address the following question.

(Problem of RED for adversarial examples) With an adversarial image as input, is it feasible to reconstruct the adversarial perturbations $\boldsymbol{\delta}$, and deduce the adversary's goals and understanding, *e.g.*, the true image class concealed by the adversary and the region of the image most affected by adversarial tactics?

Formally, our goal is to reconstruct $\boldsymbol{\delta}$ from a given adversarial example $\mathbf{x}'$, assuming prior knowledge of the victim model f or its approximation $\hat{f}$, particularly when f is not directly accessible. The RED operation is expressed as $\boldsymbol{\delta} = \mathcal{R}(\mathbf{x}', \hat{f})$, which also accommodates the white-box scenario ($\hat{f} = f$) as a particular instance. Our approach involves developing a parametric model $\mathcal{D}_{\boldsymbol{\theta}}$ (for instance, a denoising neural network, which is our primary focus) as a proxy for $\mathcal{R}$. This model is trained on a dataset comprising pairs of adversarial and benign examples, denoted as $\Omega = (\mathbf{x}', \mathbf{x})$. Utilizing $\mathcal{D}_{\boldsymbol{\theta}}$, RED aims to yield a **benign example estimate** $\mathbf{x}_{\text{RED}}$ and an **adversarial example estimate** $\mathbf{x}'_{\text{RED}}$, as described in the following equation:

$$\mathbf{x}_{\text{RED}} = \mathcal{D}_{\boldsymbol{\theta}}(\mathbf{x}'), \quad \mathbf{x}'_{\text{RED}} = \underbrace{\mathbf{x}' - \mathbf{x}_{\text{RED}}}_{\text{perturbation estimate}} + \mathbf{x}, \tag{2.1}$$

where a **perturbation estimate** is given by subtracting the RED's output with its input, *i.e.*, $\mathbf{x}' - \mathcal{D}_{\boldsymbol{\theta}}(\mathbf{x}')$.

We emphasize that RED introduces a novel defensive strategy aimed at 'analyzing' the intricacies of the perturbation in an existing adversarial example, utilizing a retrospective, forensic approach. This contrasts with adversarial detection (AD). **Fig.2.1** visually contrasts RED and AD. While AD is also developed retrospectively, its objective is to ascertain whether an input is an adversarial example for a target model, relying on specific statistics related to model features or logits. Moreover, AD could serve as a preliminary step in the RED process, where it identifies 'detected' adversarial examples for a more detailed RED analysis.

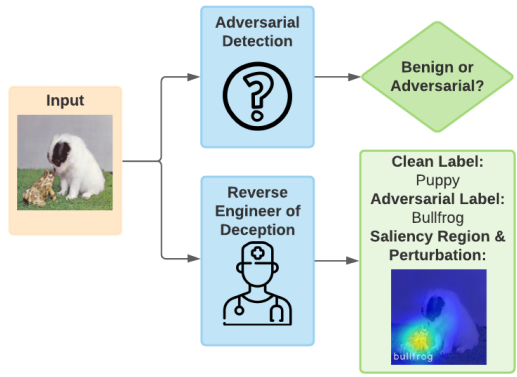


Figure 2.1: Overview of RED versus AD.

In our experiments, it will also be demonstrated that RED's outputs can inform the development of adversarial detection methods. Thus, RED and AD function as synergistic components within a comprehensive loop.

Research challenge. In this study, we define the RED model $\mathcal{D}_{\boldsymbol{\theta}}$ as a denoising network. Nevertheless, devising an appropriate denoiser for RED is a challenging task. Broadly speaking, there are two principal challenges involved. Initially, in contrast to traditional image denoising methods (Zhang *et al.*, 2017), crafting a denoiser for RED requires consideration of the impact from victim models and the data characteristics of adversary-benign pairs. Additionally, simply minimizing reconstruction error may not be effective, as the adversarial perturbation is delicately engineered (Niu *et al.*, 2020). Consequently, either under-denoising or over-denoising can significantly undermine the effectiveness of RED.

2.2 Evaluation Metrics and Denoising-Only Baseline

Given that RED diverges from current defensive methods, we establish new performance metrics for RED, spanning from pixel-level reconstruction error to attribution-level identification of adversary saliency regions. Following this, we utilize these newly developed performance metrics

to illustrate the limitations of a standard image denoiser in adequately serving the needs of RED.

2.2.1 RED Evaluation Metrics

For a trained RED model $\mathcal{D}_{\boldsymbol{\theta}}$, its effectiveness will be gauged using a test dataset $(\mathbf{x}', \mathbf{x}) \in \mathcal{D}_{\text{test}}$. In this context, $\mathbf{x}'$ serves as the test input for the RED model, while $\mathbf{x}$ is the associated ground-truth benign example for reference. The benign example estimate $\mathbf{x}_{\text{RED}}$ and the adversarial example estimate $\mathbf{x}'_{\text{RED}}$ are derived as per (2.1). The evaluation of RED encompasses several dimensions: ① pixel-level reconstruction error, ② prediction-level inference alignment, and ③ attribution-level adversary saliency assessment.

① **Pixel-level**: The reconstruction error is measured by $d(\mathbf{x}, \mathbf{x}_{\text{RED}}) = \mathbb{E}_{(\mathbf{x}', \mathbf{x}) \in \mathcal{D}_{\text{test}}}[\|\mathbf{x}_{\text{RED}} - \mathbf{x}\|_2]$.

② **Prediction-level**: This involves assessing the Prediction alignment (PA) between the benign example and its estimate $(\mathbf{x}_{\text{RED}}, \mathbf{x})$ and between the adversarial example and its estimate $(\mathbf{x}'_{\text{RED}}, \mathbf{x}')$, yielding

$$\text{PA}_{\text{benign}} = \frac{\text{card}(\{(\mathbf{x}_{\text{RED}}, \mathbf{x}) \,|\, F(\mathbf{x}_{\text{RED}}) = F(\mathbf{x})\})}{\text{card}(\mathcal{D}_{\text{test}})} \tag{2.2}$$

$$\text{PA}_{\text{adv}} = \frac{\text{card}(\{(\mathbf{x}'_{\text{RED}}, \mathbf{x}') \,|\, F(\mathbf{x}'_{\text{RED}}) = F(\mathbf{x}')\})}{\text{card}(\mathcal{D}_{\text{test}})} \tag{2.3}$$

where $\text{card}(\cdot)$ is the set cardinality function and F signifies the prediction label from the victim model f.

③ **Attribution-level**: This evaluates Input attribution alignment (**IAA**) for both the benign pair $(\mathbf{x}_{\text{RED}}, \mathbf{x})$ and the adversarial pair $(\mathbf{x}'_{\text{RED}}, \mathbf{x}')$. We apply GradCAM (Selvaraju *et al.*, 2020) for attributing class predictions back to input saliency regions. The choice of GradCAM is inspired by (Boopathy *et al.*, 2020) to identify a class-discriminative localization map **against adversaries**. The underlying concept of IAA is that even subtle adversarial alterations at the pixel level can cause significant discrepancies in input attributions with respect to the authentic label y and the adversary's intended label y' (Boopathy *et al.*, 2020; Xu *et al.*, 2019). Hence, a precise RED model should nullify adversarial attribution effects via $\mathbf{x}_{\text{RED}}$, and estimate the adversarial

intent by identifying the saliency region in $\mathbf{x}'_{\text{RED}}$ (refer to Fig. 2.1 for an illustration).

2.2.2 Denoising-Only (DO) Baseline

We also demonstrate why a standard image denoiser, often considered the pixel-level denoising, falls short in addressing the RED challenge. This limitation prompts us to reassess the approach to denoising specifically from the RED perspective. Initially, we develop the denoising network by focusing on minimizing the reconstruction error:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \ell_{\text{denoise}}(\boldsymbol{\theta}; \Omega) := \mathbb{E}_{(\mathbf{x}', \mathbf{x}) \in \Omega} \|\mathcal{D}_{\boldsymbol{\theta}}(\mathbf{x}') - \mathbf{x}\|_1, \tag{2.4}$$

where a Mean Absolute Error (MAE)-type loss is used for denoising (Liao *et al.*, 2018), and Ω is the training dataset.

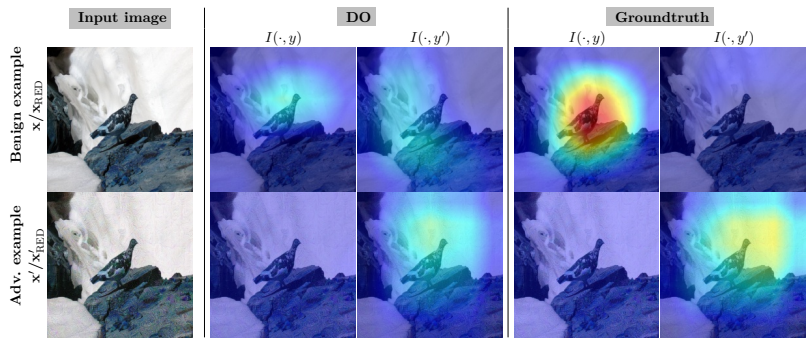


Figure 2.2: IAA of DO compared with ground-truth.

We then assess the effectiveness of DO using the non-adversarial prediction alignment metric $\text{PA}_{\text{benign}}$ and IAA. Our findings show that $\text{PA}_{\text{benign}}$ for DO stands at 42.8%. Furthermore, **Fig. 2.2** illustrates the IAA results of DO in relation to a specific input example. It becomes evident that DO falls short in accurately reconstructing the adversarial saliency regions when compared to the actual adversarial perturbations. This indicates that DO-driven RED is deficient in its reconstruction capabilities at both the prediction and attribution levels. Additionally, another basic strategy might involve applying an adversarial attack back to $\mathbf{x}'$, but this approach is dependent on extra assumptions and may not effectively recapture the original perturbations.

2.3 Proposed Solution: Class-Discriminative Denoising for RED

In this section, we introduce a novel approach known as Class-Discriminative Denoising based RED, abbreviated as CDD-RED, with an overview provided in **Fig. 2.3**. CDD-RED comprises two primary elements. Firstly, we implement a PA regularization to ensure prediction-level consistency for both the estimated benign example $\mathbf{x}_{\text{RED}}$ and the adversarial example $\mathbf{x}'_{\text{RED}}$, in comparison to their actual counterparts $\mathbf{x}$ and $\mathbf{x}'$. Secondly, we introduce a data augmentation method aimed at enhancing the generalization capabilities of RED while maintaining its class-discriminative proficiency.

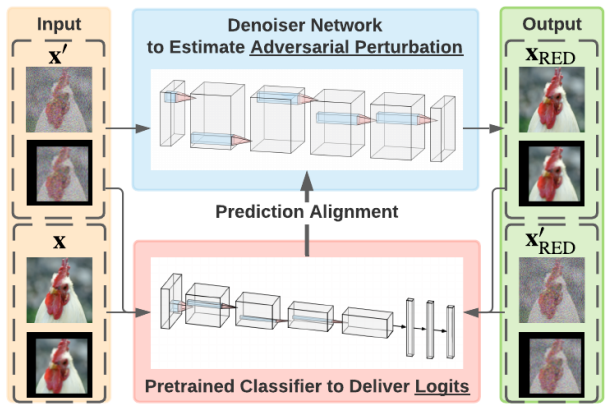


Figure 2.3: CDD-RED overview.

2.3.1 Benign and Adversarial Prediction Alignment

To precisely deduce the adversarial perturbation from an adversarial example, insights gained from the DO method highlight the importance of maintaining the class-discriminative accuracy of RED estimates, ensuring they match the original predictions as seen in $\mathbf{x}_{\text{RED}}$ versus $\mathbf{x}$, and $\mathbf{x}'_{\text{RED}}$ versus $\mathbf{x}'$. Motivated by this, the training goal for CDD-RED should not only focus on reducing the reconstruction error as outlined in (2.4) but also on enhancing PA, essentially replicating the class-discriminative feature of the original data. To fulfill this objective, we supplement the denoiser $\mathcal{D}_{\boldsymbol{\theta}}$ with a known classifier $\hat{f}$ to generate

predictions for the estimated benign and adversarial examples (refer to Fig. 2.3), that is, $\mathbf{x}_{\text{RED}}$ and $\mathbf{x}'_{\text{RED}}$ as defined in (2.1). By comparing $\hat{f}(\mathbf{x}_{\text{RED}})$ with $\hat{f}(\mathbf{x})$, and $\hat{f}(\mathbf{x}'_{\text{RED}})$ with $\hat{f}(\mathbf{x}')$, we aim to bolster PA by minimizing the prediction discrepancy between true and estimated examples:

$$\ell_{\text{PA}}(\boldsymbol{\theta}; \Omega) = \mathbb{E}_{(\mathbf{x}', \mathbf{x}) \in \Omega}[\ell_{\text{PA}}(\boldsymbol{\theta}; \mathbf{x}', \mathbf{x})], \tag{2.5}$$

$$\ell_{\text{PA}}(\boldsymbol{\theta}; \mathbf{x}', \mathbf{x}) := \underbrace{\text{CE}(\hat{f}(\mathbf{x}_{\text{RED}}), \hat{f}(\mathbf{x}))}_{\text{PA for benign prediction}} + \underbrace{\text{CE}(\hat{f}(\mathbf{x}'_{\text{RED}}), \hat{f}(\mathbf{x}'))}_{\text{PA for adversarial prediction}}, \tag{2.6}$$

with CE representing the cross-entropy loss. It is advantageous to combine the denoising loss (2.4) with the PA regularization (2.6) to boost the class-discriminative capabilities, resulting in $\ell_{\text{denoise}} + \lambda \ell_{\text{PA}}$, where $\lambda > 0$ is a tuning parameter. To tackle this, we plan to introduce a data augmentation technique to augment the denoising efficiency in addition to the benefits of the PA regularization.

2.3.2 Data Augmentation for RED

The decision to integrate data transformations into CDD-RED is based on two considerations. Firstly, data transformation can direct the attention of RED towards the most significant attack indicators, as adversarial examples often exhibit sensitivity to input modifications (Luo *et al.*, 2015; Athalye *et al.*, 2018; Xie *et al.*, 2019; Li *et al.*, 2020b; Fan *et al.*, 2021). Secondly, identifying transformation-stable benign/adversarial instances could improve the effectiveness of PA and IAA.

However, selecting the most suitable data augmentation techniques is a complex task. For instance, pixel-altering transformations like Gaussian blurring and colorization might impede the ability to reconstruct the original adversary-benignity pair $(\mathbf{x}', \mathbf{x})$. Consequently, our focus shifts to spatial image transformations, such as rotation, translation, cropping & padding, cutout, and CutMix (Yun *et al.*, 2019), which preserve the original perturbations in a linear manner. In **Fig. 2.4**, we examine RED's performance, in terms of pixel-level reconstruction error and prediction-level alignment accuracy, across various spatial transformations. Notably, CutMix and cropping & padding demonstrate the ability to enhance both metrics, making them suitable augmenta-

tion choices for RED enhancement. Additionally, our empirical studies suggest that a combination of these two transformations can lead to further performance improvements.

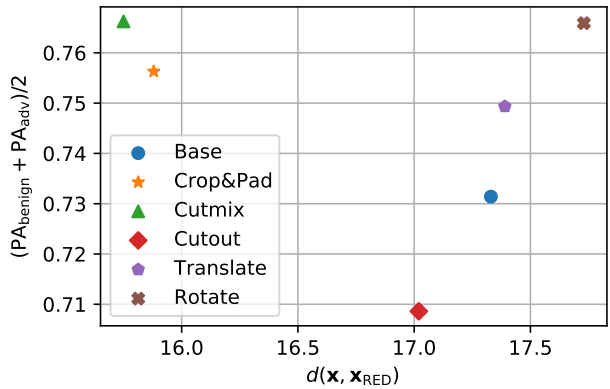


Figure 2.4: The influence of different data augmentations. 'Base' refers to the base training without augmentation.

Denoting $\mathcal{T}$ as a set of transformations that includes cropping & padding and CutMix operations, and incorporating the denoising loss (2.4), PA regularization (2.6), and data transformations $\mathcal{T}$, we thus formulate the overall training objective for CDD-RED as follows:

$$\min_{\boldsymbol{\theta}} \underbrace{\mathbb{E}_{(\mathbf{x}',\mathbf{x})\in\Omega,t\sim\mathcal{T}}\|\mathcal{D}_{\boldsymbol{\theta}}(t(\mathbf{x}')) - t(\mathbf{x})\|_1}_{\ell_{\text{denoise}}\ (2.4)\text{ with data augmentations}} + \underbrace{\lambda\mathbb{E}_{(\mathbf{x}',\mathbf{x})\in\Omega,t\sim\check{\mathcal{T}}}[\ell_{\text{PA}}(\boldsymbol{\theta}; t(\mathbf{x}'), t(\mathbf{x}))]}_{\ell_{\text{PA}}\ (2.6)\text{ with data augmentation via }\check{\mathcal{T}}}, \tag{2.7}$$

where $\check{\mathcal{T}}$ denotes a properly-selected subset of $\mathcal{T}$, and $\lambda > 0$ is a regularization parameter. In the PA regularizer (2.7), we need to avoid the scenario of over-transformation where data augmentation alters the classifier's original decision. This suggests $\check{\mathcal{T}} = \{t \in \mathcal{T} \,|\, \hat{F}(t(\mathbf{x})) = \hat{F}(\mathbf{x}), \hat{F}(t(\mathbf{x}')) = \hat{F}(\mathbf{x}')\ \}$, where $\hat{F}$ represents the prediction label of the pre-trained classifier $\hat{f}$, *i.e.*, $\hat{F}(\cdot) = \text{argmax}(\hat{f}(\cdot))$.

2.4 Experiments

We demonstrate the effectiveness of our proposed method in 4 aspects: **a)** reconstruction error of adversarial perturbation, *i.e.*, $d(\mathbf{x}, \mathbf{x}_{\text{RED}})$, **b)**

class-discriminative ability of the benign and adversarial example estimate, *i.e.*, $\text{PA}_{\text{benign}}$ and PA_{adv} by victim models, **c)** adversary saliency region recovery, *i.e.*, attribution alignment, and **d)** RED evaluation over unseen attack types and adaptive attacks.

Attack datasets. To train and test RED models, we generate adversarial images on the ImageNet dataset (Deng *et al.*, 2009). We consider 3 **attack methods** including PGD (Madry *et al.*, 2017), FGSM (Goodfellow *et al.*, 2014b), and CW attack (Carlini and Wagner, 2017), applied to 5 **models** including pre-trained ResNet18 (Res18), ResNet50 (Res50) (He *et al.*, 2016), VGG16, VGG19, and InceptionV3 (IncV3) (Szegedy *et al.*, 2015). Furthermore, to evaluate the RED performance on unseen perturbation types during training, additional 2K adversarial examples generated by **AutoAttack** (Croce and Hein, 2020) and 1K adversarial examples generated by **Feature Attack** (Sabour *et al.*, 2015) are included as the unseen testing dataset. AutoAttack is applied on VGG19, Res50 and **two new victim models**, i.e., Alexnet and Robust ResNet50 (R-Res50), via fast adversarial training (Wong *et al.*, 2020) while Feature Attack is applied on VGG19 and Alexnet. The rational behind considering Feature Attack is that feature adversary has been recognized as an effective way to circumvent adversarial detection (Tramer *et al.*, 2020). Thus, it provides a supplement on detection-aware attacks.

RED model training and evaluation. During the training of the RED denoisers, VGG19 (Simonyan and Zisserman, 2015) is chosen as the pretrained classifier $\hat{f}$ for PA regularization. Although different victim models were used for generating adversarial examples, we will show that the inference guided by VGG19 is able to accurately estimate the true image class and the intent of the adversary. In terms of the architecture of $\mathcal{D}_{\boldsymbol{\theta}}$, DnCNN (Zhang *et al.*, 2017) is adopted. The RED problem is solved using an Adam optimizer (Kingma and Ba, 2015) with the initial learning rate of 10^{-4}, which decays 10 times for every 140 training epochs. In (2.7), the regularization parameter λ is set as 0.025. The transformations for data augmentation include CutMix and cropping & padding. The maximum number of training epochs is set as 300.

Baselines. We compare CDD-RED with two baseline approaches: **a)** the conventional denoising-only (DO) approach with the objective function (2.4); **b)** The state-of-the-art Denoised Smoothing (DS) (Salman *et al.*, 2020) approach that considers both the reconstruction error and the PA for benign examples in the objective function. Both methods are tuned to their best configurations.

2.4.1 Experiment Results: Reconstruction Error and PA

Table 2.1 presents the comparison of CDD-RED with the baseline denoising approaches in terms of $d(\mathbf{x}, \mathbf{x}_{\text{RED}})$, $d(f(\mathbf{x}), f(\mathbf{x}_{\text{RED}}))$, $d(f(\mathbf{x}'), f(\mathbf{x}'_{\text{RED}}))$, $\text{PA}_{\text{benign}}$, and PA_{adv} on the testing dataset. As we can see, our approach (CDD-RED) improves the class-discriminative ability from benign perspective by 42.91% and adversarial perspective by 8.46% with a slightly larger reconstruction error compared with the DO approach.

Table 2.1: The performance comparison among DO, DS and CDD-RED on the testing dataset.

	DO	DS	CDD-RED
$d(\mathbf{x}, \mathbf{x}_{\text{RED}})$	9.32	19.19	13.04
$d(f(\mathbf{x}), f(\mathbf{x}_{\text{RED}}))$	47.81	37.21	37.07
$d(f(\mathbf{x}'), f(\mathbf{x}'_{\text{RED}}))$	115.09	150.02	78.21
$\text{PA}_{\text{benign}}$	42.80%	86.64%	85.71%
PA_{adv}	71.97%	72.47%	80.43%

In contrast to DS, CDD-RED achieves similar $\text{PA}_{\text{benign}}$ but improved pixel-level denoising error and PA_{adv}. Furthermore, CDD-RED achieves the best logit-level reconstruction error for both $f(\mathbf{x}_{\text{RED}})$ and $f(\mathbf{x}'_{\text{RED}})$ among the three approaches. This implies that $\mathbf{x}_{\text{RED}}$ rendered by CDD-RED can achieve highly similar prediction to the true benign example $\mathbf{x}$, and the perturbation estimate $\mathbf{x}' - \mathbf{x}_{\text{RED}}$ yields a similar misclassification effect to the ground-truth perturbation.

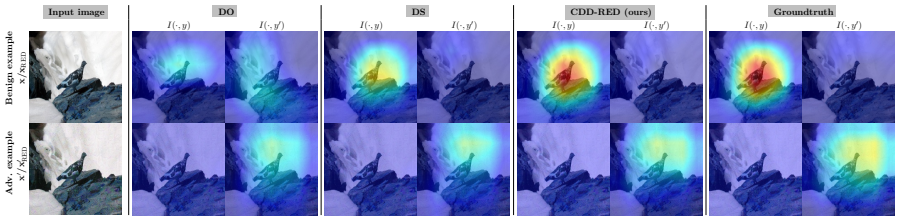
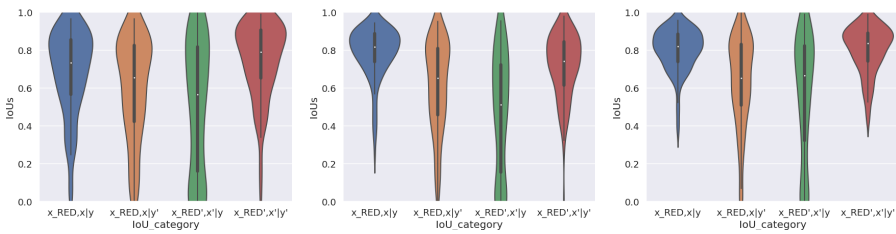


Figure 2.5: Interpretation (I) of benign ($\mathbf{x}/\mathbf{x}_{\text{RED}}$) and adversarial ($\mathbf{x}'/\mathbf{x}'_{\text{RED}}$) image w.r.t. the true label y='ptarmigan' and the adversary targeted label y'='shower curtain'. We compare three methods of RED training, DO, DS, and CDD-RED as our method, to the ground-truth interpretation. Given an RED method, the first column is $I(\mathbf{x}_{\text{RED}}, y)$ versus $I(\mathbf{x}'_{\text{RED}}, y)$, the second column is $I(\mathbf{x}_{\text{RED}}, y')$ versus $I(\mathbf{x}'_{\text{RED}}, y')$, and all maps under each RED method are normalized w.r.t. their largest value. For the ground-truth, the first column is $I(\mathbf{x}, y)$ versus $I(\mathbf{x}', y)$, the second column is $I(\mathbf{x}, y')$ versus $I(\mathbf{x}', y')$.



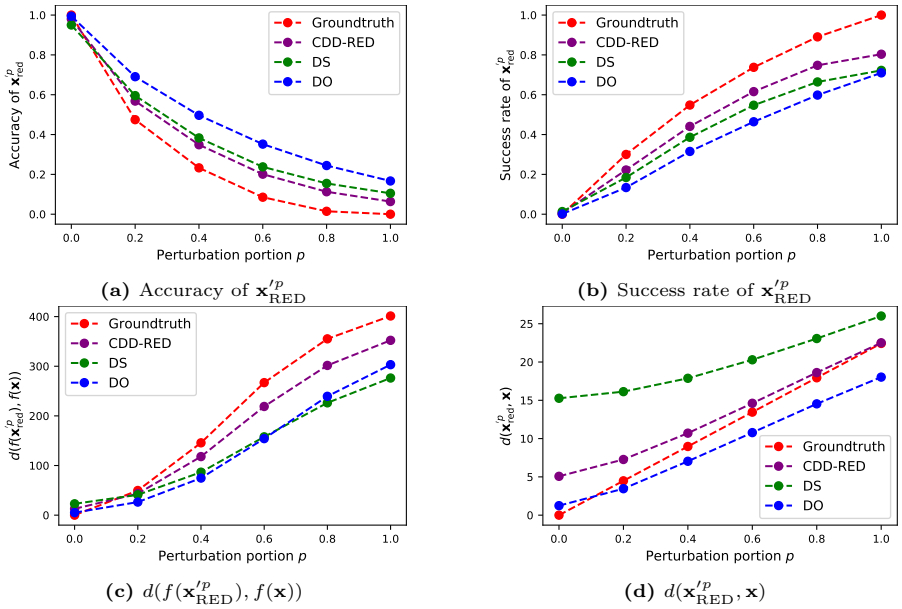
(a) Denoising Only **(b)** Denoised Smoothing **(c)** CDD-RED (ours)

Figure 2.6: IoU distributions of the attribution alignment by three RED methods. Higher IoU is better. For each subfigure, the four IoU scores represent $\text{IoU}(\mathbf{x}_{\text{RED}}, \mathbf{x}, y)$, $\text{IoU}(\mathbf{x}_{\text{RED}}, \mathbf{x}, y')$, $\text{IoU}(\mathbf{x}'_{\text{RED}}, \mathbf{x}', y)$, and $\text{IoU}(\mathbf{x}'_{\text{RED}}, \mathbf{x}', y')$.

2.4.2 Experiment Results: Attribution Alignment

Besides evaluating RED performance through pixel-level and prediction-level alignments, we also explore attribution alignment. **Fig. 2.5** displays attribution maps created by GradCAM, considering $I(\mathbf{x}, y)$, $I(\mathbf{x}', y)$, $I(\mathbf{x}, y')$, and $I(\mathbf{x}', y')$, where $\mathbf{x}'$ is the modified version of $\mathbf{x}$, and y' represents the adversarial target label. The sequence from left to right includes the attribution maps for DO, DS, CDD-RED (our approach), and the ground-truth. Notably, CDD-RED demonstrates an attribution alignment more closely resembling the ground-truth, particularly evident in the comparison between $I(\mathbf{x}_{\text{RED}}, y)$ and $I(\mathbf{x}, y)$.

At a broader dataset level, **Fig. 2.6** presents the distribution of attri-



(a) Accuracy of $\mathbf{x}'^{p}_{\text{RED}}$ **(b)** Success rate of $\mathbf{x}'^{p}_{\text{RED}}$

(c) $d(f(\mathbf{x}'^{p}_{\text{RED}}), f(\mathbf{x}))$ **(d)** $d(\mathbf{x}'^{p}_{\text{RED}}, \mathbf{x})$

Figure 2.7: Reverse engineer partially-perturbed data under different interpolation portion p.

bution IoU scores. In this comparison, it's noticeable that CDD-RED's IoU distribution, relative to DO and DS, is more densely concentrated in higher-value regions. This pattern suggests a closer alignment with the adversary's attribution map. This characteristic highlights a compelling application of our proposed RED method: it can effectively recover the adversary's focal saliency regions, specifically the class-discriminative image areas targeted by the adversary.

2.4.3 Experiment Results: RED Against Unforeseen Attack Types

Our experiments focused on recovering from unforeseen attack types consist of two main components: **a)** the creation of partially perturbed data using linear interpolation and **b)** introducing previously unseen attack types such as AutoAttack, Feature Attack, and Adaptive Attack.

To generate partially perturbed data, we linearly added a fraction p from the perturbation $\mathbf{x}' - \mathbf{x}$ to the original benign example $\mathbf{x}$. This process is represented as $\mathbf{x}'^{p} = \mathbf{x} + p(\mathbf{x}' - \mathbf{x})$. Subsequently, the inter-

polated $\mathbf{x}'^p$ samples were utilized as inputs for an RED model. Our primary objective was to assess the effectiveness of the proposed RED method in recovering from partial perturbations, even in cases where the attacks were not entirely successful.

Fig. 2.7 (a) and (b) illustrate the alignment of predictions with respect to y and y' for the estimated adversarial examples $\mathbf{x}'^p_{\text{RED}} = \mathbf{x}'^p - \mathcal{D}_{\boldsymbol{\theta}}(\mathbf{x}'^p) + \mathbf{x}$, generated by different RED models. Fig. 2.7 (c) displays the logit distance between the predictions of partially perturbed adversarial examples and the predictions of benign examples. In contrast, Fig. 2.7 (d) demonstrates the pixel-wise distance between $\mathbf{x}'^p_{\text{RED}}$ and the benign examples.

A smaller discrepancy between the ground-truth curve (depicted in red) and the curve representing the estimated adversarial examples $\mathbf{x}'^p_{\text{RED}}$ indicates superior performance. Fig. 2.7 (a) and (b) reveal that CDD-RED provides the closest approximation to the ground truth in terms of prediction accuracy and attack success rate. This is further confirmed by the distance in prediction logits displayed in Fig. 2.7 (c). Fig. 2.7 (d) indicates that DS tends to overestimate the additive perturbation, while CDD-RED maintains the perturbation estimation closest to the ground truth. Although DO performs better than CDD-RED when $p < 40\%$, it falls short in providing a more precise adversarial perturbation according to other performance metrics. For instance, in Fig. 2.7 (b) at $p = 20\%$, the $\mathbf{x}'^p_{\text{RED}}$ generated by DO achieves a lower successful attack rate compared to CDD-RED and the ground truth.

Furthermore, regarding benign examples with $p = 0\%$ perturbations, it's worth noting that the RED denoiser, despite not being trained on benign example pairs $(\mathbf{x}, \mathbf{x})$, maintains its ability to effectively recover benign examples. CDD-RED demonstrates proficiency in handling scenarios where both adversarial and benign examples are present. This means that even if a benign example, which is misclassified as adversarial, is erroneously introduced into the RED framework, our method can successfully reconstruct the original perturbation, closely approximating the ground truth.

Table 2.2 presents an overview of RED's performance when dealing with previously unseen attack types, including AutoAttack, Feature Attack, and Adaptive Attack. For both AutoAttack and Feature Attack,

Table 2.2: The $d(\mathbf{x}, \mathbf{x}_{\text{RED}})$, $\text{PA}_{\text{benign}}$, and PA_{adv} performance of the denoisers on the unforeseen perturbation type AutoAttack, Feature Attack, and Adaptive Attack.

		DO	DS	CDD-RED
$d(\mathbf{x}, \mathbf{x}_{\text{RED}})$	AutoAttack	6.41	16.64	8.81
	Feature Attack	5.51	16.14	7.99
	Adaptive Attack	9.76	16.21	12.24
$\text{PA}_{\text{benign}}$	AutoAttack	84.69%	92.64%	94.58%
	Feature Attack	82.90%	90.75%	93.25%
	Adaptive Attack	33.20%	27.27%	36.29%
PA_{adv}	AutoAttack	85.53%	83.30%	88.39%
	Feature Attack	26.97%	35.84%	63.48%
	Adaptive Attack	51.21%	55.41%	57.11%

CDD-RED exhibits superior performance compared to DO and DS, as measured by PA from both benign and adversarial perspectives. Notably, CDD-RED significantly enhances PA_{adv} for Feature Attack, surpassing DO and DS by 36.51% and 27.64%, respectively.

Regarding the adaptive attack (Tramer *et al.*, 2020), we assume that the attacker possesses knowledge of the RED model, specifically $D_{\boldsymbol{\theta}}$. In this scenario, the attacker can employ the PGD attack method to create successful prediction-evasion attacks, even after undergoing the RED operation. To generate such attacks, we employ PGD methods within the ℓ_∞-ball, with a perturbation radius of $\epsilon = 20/255$. Table 2.2 reveals that the Adaptive Attack is considerably more potent than Feature Attack and AutoAttack, resulting in larger reconstruction errors and lower PA. However, CDD-RED still outperforms DO and DS in terms of $\text{PA}_{\text{benign}}$ and PA_{adv}. Moreover, compared to DS, it achieves a more favorable balance between denoising error $d(\mathbf{x}, \mathbf{x}_{\text{RED}})$ and other performance metrics.

In summary, CDD-RED demonstrates the ability to achieve high PA even when confronted with previously unseen attacks, showcasing its capacity to generalize and estimate not only new adversarial examples generated from the same attack method but also novel attack types.

2.5 Conclusion

Studying RED problem of adversarial examples, we aim to recover vital attack components such as adversarial perturbations and adversary saliency regions. It's worth noting that RED has not received thorough exploration in prior research efforts. Our proposal takes significant strides in not only formalizing the RED problem but also crafting a comprehensive pipeline that encompasses not only a solution but also an exhaustive set of evaluation metrics.

We have also identified a series of RED principles that span from the fine-grained pixel level to the more abstract attribution level, all essential in the endeavor to reverse-engineer adversarial attacks. Our approach centers around an effective denoiser-assisted RED technique, which seamlessly integrates class-discrimination and data augmentation into an image denoising network. Through a multitude of rigorous experiments, our approach demonstrates its superiority over existing baseline methods. It showcases a remarkable ability to generalize effectively, even when confronted with previously unseen attack types.

In next chapter, we will extend our RED focus from machine-centric adversarial instances themselves to the victim model they aim at. We will dive into the feasibility of the model parsing via these adversarial examples.

3

Model Parsing via Adversarial Examples

Chapter overview. In this chapter, we will delve deeper into the reverse-engineering potential of machine-centric attacks, this time examining it from a 'model parsing' standpoint. Our objective is to demonstrate that adversarial perturbations can do more than just being estimated and attributed, as discussed in Chapter 2. They can also inadvertently divulge information about the victim model, a process we refer to as 'model parsing.' This entails the inference of the victim model's parameters that the adversary employed in the attack.

Adversarial attack methods stand out for their ability to craft subtle image perturbations that lead even the most advanced deep neural networks (DNNs) astray. Yet, the underlying 'arcana'—the intricate DNN-related details concealed within these attacks—has largely eluded examination. This uncharted inquiry delves into whether we can decode data-agnostic information about the 'victim model' (VM)—the particular ML model or DNN targeted by these attacks—from the adversarial examples themselves. This process, termed 'model parsing of adversarial attacks', aims to unravel the VM's hidden characteristics such as architecture type, kernel size, activation function, and weight sparsity, from the adversarial instances crafted to deceive it. In the upcoming

chapter, we will also investigate the challenge of model parsing in the context of human-centric attacks produced by generative models.

In this chapter, we employ supervised learning to attribute the correct classes of VM's model attributes to a given adversarial instance. We assemble a diverse dataset comprising adversarial attacks from seven distinct types, generated by 135 different victim models with varied configurations. Through this dataset, we explore the ability of a model parsing network (MPN) to accurately deduce VM attributes from unseen adversarial attacks. Extensive experimentation underpins our inquiry into the practicality of VM parsing from adversarial attacks, along with the exploration of factors that impact parsing performance, such as the challenges of generalization in out-of-distribution evaluations. We extend our analysis to transfer attacks, aiming to uncover the source VM attributes and illuminate a potential link between model parsing capability and the transferability of adversarial attacks. Through this chapter, we also intend to highlight the nuanced interplay between attack generation and its victim model characteristics.

3.1 Background and Problem Setup

In this section, we will detail the preliminaries and problem setups of model parsing for adversarial examples. We will first introduce the adversarial attacks we study on and how they are generated. Then, we will introduce what is model parsing and illustrate the problem setup.

3.1.1 Adversarial Attack Generation and Victim Models

We first detail the adversarial attack types that we cover in this chapter and show their dependence on **VM** (**victim model**), *i.e.,* the ML model from which adversarial examples are generated. Meanwhile, we consider different types of model attributes that involve in model parsing.

We focus on ℓ_p attacks, where the adversary aims to generate imperceptible input perturbations to fool an image classifier (Goodfellow *et al.*, 2014b). Let $\mathbf{x}$ and $\boldsymbol{\theta}$ denote a benign image and the parameters of VM. The adversarial attack (*a.k.a*, adversarial example) is defined via the linear perturbation model $\mathbf{x}' = \mathbf{x} + \boldsymbol{\delta}$, where $\boldsymbol{\delta} = \mathcal{A}(\mathbf{x}, \boldsymbol{\theta}, \epsilon)$ denotes

adversarial perturbations, and $\mathcal{A}$ refers to an attack generation method relying on $\mathbf{x}$, $\boldsymbol{\theta}$, and the attack strength ϵ (*i.e.*, the perturbation radius of ℓ_p attacks).

In this study, we examine seven distinct adversarial attack methods, each varying in their reliance on the victim model's parameters ($\boldsymbol{\theta}$). This includes four input gradient-based white-box attacks that have complete access to $\boldsymbol{\theta}$, namely `FGSM` (Goodfellow *et al.*, 2014b), `PGD` (Madry *et al.*, 2017), `CW` (Carlini and Wagner, 2017), and `AutoAttack` (or `AA`) (Croce and Hein, 2020). Additionally, we consider three query-based black-box attacks, which are `ZO-signSGD` (Liu *et al.*, 2019b), `NES` (Ilyas *et al.*, 2018), and `SquareAttack` (or `Square`) (Andriushchenko *et al.*, 2020), that do not require direct access to $\boldsymbol{\theta}$.

✦ `FGSM` (fast gradient sign method) (Goodfellow *et al.*, 2014b): This attack method is given by $\boldsymbol{\delta} = \mathbf{x} - \epsilon \times \text{sign}(\nabla_{\mathbf{x}} \ell_{\text{atk}}(\mathbf{x}; \boldsymbol{\theta}))$, where $\text{sign}(\cdot)$ is the entry-wise sign operation, and $\nabla_{\mathbf{x}} \ell_{\text{atk}}$ is the input gradient of an attack loss $\ell_{\text{atk}}(\mathbf{x}; \boldsymbol{\theta})$ evaluated at $\mathbf{x}$ under $\boldsymbol{\theta}$.

✦ `PGD` (projected gradient descent) (Madry *et al.*, 2017): This extends `FGSM` via an iterative algorithm. Formally, the K-step *PGD ℓ_∞ attack* is given by $\boldsymbol{\delta} = \boldsymbol{\delta}_K$, where $\boldsymbol{\delta}_k = \mathcal{P}_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon}(\boldsymbol{\delta}_{k-1} - \alpha \times \text{sign}(\nabla_{\mathbf{x}} \ell_{\text{atk}}(\mathbf{x}; \boldsymbol{\theta})))$ for $k = 1, \ldots, K$, $\mathcal{P}_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon}$ is the projection operation onto the ℓ_∞-norm constraint $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$, and α is the attack step size. By replacing the ℓ_∞ norm with the ℓ_2 norm, we similarly obtain the *PGD ℓ_2 attack* (Madry *et al.*, 2017).

✦ `CW` (Carlini-Wager) attack (Carlini and Wagner, 2017): Similar to `PGD`, `CW` calls iterative optimization for attack generation. Yet, `CW` formulates attack generation as an ℓ_p-norm regularized optimization problem, with the regularization parameter $c = 1$ by default. For example, the choice of $c = 1$ in `CW` ℓ_2 attack could lead to a variety of perturbation strengths with the average value around $\epsilon = 0.33$ on the `CIFAR-10` dataset. Moreover, `CW` adopts a hinge loss to ensure the misclassification margin. We will focus on `CW` ℓ_2 attack.

✦ `AutoAttack` (or `AA`) (Croce and Hein, 2020): This is an ensemble attack that uses `AutoPGD`, an adaptive version of `PGD`, as the primary means of attack. The loss of `AutoPGD` is given by the difference of logits ratio (DLR) rather than CE or CW loss.

✦ `ZO-signSGD` (Liu *et al.*, 2019b) and `NES` (Ilyas *et al.*, 2018): These

are zeroth-order optimization (ZOO)-based black-box attacks. Unlike the white-box attacks that rely on gradient-based methods, these black-box attacks interact with the victim model ($\boldsymbol{\theta}$) solely by submitting inputs and obtaining the corresponding predictions. ZOO exploits these input-output pairs to approximate input gradients for crafting adversarial perturbations. Yet, `ZO-signSGD` and `NES` call different gradient estimators in ZOO (Liu *et al.*, 2020).

Table 3.1: Summary of adversarial attack types focused in this work. Here GD refers to gradient descent, and WB and BB refer to white-box and black-box dependence on the victim model, respectively.

Attacks	Generation method	Loss	ℓ_p norm	Strength ϵ	Dependence on $\boldsymbol{\theta}$
`FGSM`	one-step GD	CE	ℓ_∞	$\{4, 8, 12, 16\}/255$	WB, gradient-based
`PGD`	multi-step GD	CE	ℓ_∞ ℓ_2	$\{4, 8, 12, 16\}/255$ 0.25, 0.5, 0.75, 1	WB, gradient-based
`CW`	multi-step GD	CW	ℓ_2	soft regularization $c \in \{0.1, 1, 10\}$	WB, gradient-based
`AutoAttack` or `AA`	attack ensemble	CE / DLR	ℓ_∞ ℓ_2	$\{4, 8, 12, 16\}/255$ 0.25, 0.5, 0.75, 1	WB, gradient-based + BB, query-based
`SquareAttack` or `Square`	random search	CE	ℓ_∞ ℓ_2	$\{4, 8, 12, 16\}/255$ 0.25, 0.5, 0.75, 1	BB, query-based
`NES`	ZOO	CE	ℓ_∞	$\{4, 8, 12, 16\}/255$	BB, query-based
`ZO-signSGD`	ZOO	CE	ℓ_∞	$\{4, 8, 12, 16\}/255$	BB, query-based

While there are various forms of ℓ_p attacks available, we have chosen to focus on the adversarial attacks listed in **Table 3.1**. This selection is based on their diversity, including different optimization approaches, attack losses, ℓ_p norms, and levels of dependency on the VM ($\boldsymbol{\theta}$).

3.1.2 Problem Setup

It's evident from the aforementioned attach generation methods that adversarial attacks reveal information about the VM ($\boldsymbol{\theta}$), although the extent of their dependence varies. This observation raises an intriguing question: Can we deduce the *attributes* of $\boldsymbol{\theta}$ from these attack instances, such as adversarial perturbations or the perturbed images? The specific model attributes we focus on include the types of model architectures and more detailed aspects like the type of activation function. We refer to this challenge as **model parsing for adversarial attacks**, as

elaborated below:

(Problem statement) Is it possible to infer victim model information from adversarial attacks? And what factors will influence such model parsing effectiveness?

To the best of our knowledge, determining the feasibility of model parsing from instances of adversarial attacks remains an unresolved issue. The challenges lie in two key areas. **Firstly**, from the **model perspective**, VM has an indirect connection with adversarial attacks, such as through local gradient information or model queries. Consequently, it's unclear to what extent VM information is imprinted in adversarial attacks and how this affects the potential for successful model parsing. **Secondly**, from the **attack perspective**, the variety of adversarial attacks (referenced in Table 3.1) complicates the development of a universal solution for model parsing. Motivated by these considerations, we aim to take an initial substantial step towards exploring the feasibility of model parsing from adversarial attacks and identifying which factors might impact model parsing efficacy. Insights gained from this endeavor could deepen our understanding of the threat model we face and potentially lead to innovative approaches in crafting adversarial defenses and robust models.

Model attributes and setup. We define VMs in adversarial attacks as convolutional neural network (CNN)-based image classifiers. Specifically, we consider five CNN architecture types (`AT`s): ResNet9, ResNet18, ResNet20, VGG11, and VGG13. For a given `AT`, CNN models are further configured by varying the kernel size (`KS`), activation function (`AF`), and weight sparsity (`WS`). Hence, a specific VM ($\boldsymbol{\theta}$) is determined by a valued quadruple (`AT`, `KS`, `AF`, `WS`). While additional attributes could be explored, we prioritize `KS` and `AF` as they are fundamental to CNN construction. Furthermore, we include `WS` as it pertains to sparse models, typically achieved through pruning (*i.e.*, removing redundant model weights) (Han *et al.*, 2015; Frankle and Carbin, 2018). Comprehensive examination of all possible model attributes is reserved for future research. **Table 3.2** presents a summary of the focused model attributes and their respective values, used to define VM instances. With a specified

VM, we generate adversarial attacks following the methods listed in Table 3.1. Unless stated otherwise, our empirical studies primarily utilize `CIFAR-10`, but additional experiments on other datasets.

Table 3.2: Summary of model attributes of interest. Each attribute value corresponds to an attribute class in model parsing.

Model attributes	Code	Classes per attribute
Architecture type	`AT`	ResNet9, ResNet18 ResNet20, VGG11, VGG13
Kernel size	`KS`	3, 5, 7
Activation function	`AF`	ReLU, tanh, ELU
Weight sparsity	`WS`	0%, 37.5%, 62.5%

3.2 Proposal: Multi-Task Classification of Model Attributes

In this section, we solve the model parsing problem as a multi-task supervised classification task applied over the dataset of adversarial attacks. We will show that the well-trained model parsing network could exhibit the great ability of generalization on test-time adversarial data. We will also show that data-model factors that may influence such generalization.

3.2.1 Model Parsing Network and Training

We approach the model parsing problem through the lens of a supervised attribute recognition task. In this context, we construct a parametric model, named the model parsing network (**MPN**). This network is designed to accept adversarial attacks as input and to predict the values of the model attributes (interpreted as 'classes' in Table 3.2). While the concept of supervised learning is straightforward, the development of MPN involves intricate considerations, particularly in terms of the input data type, the backbone network architecture, and the evaluation metrics to be employed.

Initially, we establish a dataset for model parsing by aggregating instances of adversarial attacks targeted at various victim models.

Considering that adversarial attacks are designed to bypass model predictions after training, we utilize the test set from a standard image dataset (for instance, `CIFAR-10`) to create this adversarial dataset, adopting an 80/20 split for both training and assessing (MPN). As delineated in Sec. 3.1, the MPN training dataset is represented as $\mathcal{D}_{\text{tr}} = \{(\mathbf{z}(\mathcal{A}, \mathbf{x}, \boldsymbol{\theta}), y(\boldsymbol{\theta})) \,|\, \mathbf{x} \in \mathcal{I}_{\text{tr}}, \boldsymbol{\theta} \in \Theta\}$, where $\mathbf{z}$ indicates the feature of attack data (like adversarial perturbations $\boldsymbol{\delta}$ or the adversarial example $\mathbf{x}'$) dependent on the chosen attack method $\mathcal{A}$, the initial image $\mathbf{x}$, and the VM (victim model) $\boldsymbol{\theta}$. The term $y(\boldsymbol{\theta})$ represents the actual model attribute label linked to $\boldsymbol{\theta}$. Differentiating from MPN's test data, $\mathcal{I}_{\text{tr}}$ is defined as the set of original images for training MPN, and Θ is the set of model architectures employed for creating attack data in $\mathcal{D}_{\text{tr}}$. For ease of presentation, the training set of MPN is stated as $\mathcal{D}_{\text{tr}} = \{(\mathbf{z}, y)\}$, omitting the dependence on other factors.

We now detail the design of MPN, which is parameterized by $\boldsymbol{\phi}$. Our aim is to keep the architecture of MPN straightforward and distinct from the VM $\boldsymbol{\theta}$. This approach is twofold: Firstly, our goal is to explore the practicality of model parsing from adversarial attacks while maintaining a basic structure for the attribution network ($\boldsymbol{\phi}$). Secondly, we aim to minimize the 'model attribute bias' in $\boldsymbol{\phi}$ when deducing attributes of VM from adversarial attacks. Guided by these considerations, MPN is defined using two uncomplicated networks: (1) a Multilayer Perceptron (`MLP`) with two hidden layers, each comprising 128 units (totaling 0.41M parameters) (LeCun *et al.*, 2015), and (2) a basic 4-layer CNN (`ConvNet-4`) with 64 output channels per layer, succeeded by a single fully-connected layer with 128 hidden units and the final attribution prediction head (totaling 0.15M parameters) (Vinyals *et al.*, 2016). As will be evident later, `ConvNet-4` generally surpasses `MLP` in model parsing accuracy, leading to the selection of `ConvNet-4` as the default MPN model.

Given the datamodel setup, we next tackle the recognition problem of VM's attributes (`AT`, `KS`, `AF`, `WS`) via a multi-head multi-class classifier. We dissect MPN into two parts $\boldsymbol{\phi} = [\boldsymbol{\phi}_{\text{rep}}, \boldsymbol{\phi}_{\text{atr}}]$, where $\boldsymbol{\phi}_{\text{rep}}$ is for data representation acquisition, and $\boldsymbol{\phi}_{\text{attr}}$ corresponds to the attribute-specific prediction head (*i.e.*, the last fully-connected layer in our design).

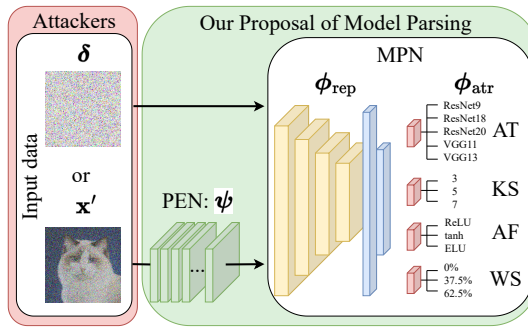


Figure 3.1: The schematic overview of our proposal. Input data (marked in red) is directly generated by the attackers in the wild while our proposal (marked in green) focuses on parsing the victim model attributes from adversarial input data. Here PEN (perturbation estimation network) will be introduced later as a pre-processing step to transform adversarial examples into perturbation-alike input data.

Eventually, four prediction heads $\{\boldsymbol{\phi}_{\text{atr}}^{(i)}\}_{i=1}^{4}$ will share $\boldsymbol{\phi}_{\text{rep}}$ for model attribute recognition; see **Fig. 3.1** for a schematic overview of our proposal. The MPN training problem is then cast as

$$\underset{\boldsymbol{\phi}_{\text{rep}},\{\boldsymbol{\phi}_{\text{atr}}^{(i)}\}_{i=1}^{4}}{\text{minimize}} \ \mathbb{E}_{(\mathbf{z},y)\in\mathcal{D}_{\text{tr}}} \sum_{i=1}^{4} [\ell_{\text{CE}}(h(\mathbf{z};\boldsymbol{\phi}_{\text{rep}},\boldsymbol{\phi}_{\text{atr}}^{(i)}),y_i)], \tag{3.1}$$

where $h(\mathbf{z};\boldsymbol{\phi}_{\text{rep}},\boldsymbol{\phi}_{\text{atr}}^{(i)})$ denotes the MPN prediction at input example $\mathbf{z}$ using the predictive model consisting of $\boldsymbol{\phi}_{\text{rep}}$ and $\boldsymbol{\phi}_{\text{atr}}^{(i)}$ for the i-th attribute classification, y_i is the ground-truth label of the i-th attribute associated with the input data $\mathbf{z}$, and ℓ_{CE} is the cross-entropy (CE) loss characterizing the error between the prediction and the true label.

3.2.2 Evaluation of Model Parsing via Adversarial Examples

Similar to training, we denote by $\mathcal{D}_{\text{test}} = \{(\mathbf{z}(\mathcal{A},\mathbf{x},\boldsymbol{\theta}),y(\boldsymbol{\theta}))\,|\,\mathbf{x}\in\mathcal{I}_{\text{test}},\boldsymbol{\theta}\in\Theta\}$ the test attack set for evaluating the performance of MPN. Here the set of benign images $\mathcal{I}_{\text{test}}$ is different from $\mathcal{I}_{\text{tr}}$, thus adversarial attacks in $\mathcal{D}_{\text{test}}$ are new to $\mathcal{D}_{\text{tr}}$. To mimic the standard evaluation pipeline of supervised learning, we propose the following evaluation metrics.

(1) In-distribution generalization: The MPN testing dataset $\mathcal{D}_{\text{test}}$ follows the attack methods ($\mathcal{A}$) and the VM specifications (Θ) *same as*

$\mathcal{D}_{\text{tr}}$ but corresponding to different original benign images (*i.e.*, $\mathcal{I}_{\text{test}} \neq \mathcal{I}_{\text{tr}}$). The purpose of such an in-distribution evaluation is to examine if the trained MPN can infer model attributes encoded in new attack data given already-seen attack methods.

(2) Out-of-distribution (OOD) generalization: In addition to new test-time images, there exist *attack/model distribution shifts* in $\mathcal{D}_{\text{test}}$ due to using *new* attack methods or model architectures, leading to *unseen* attack methods ($\mathcal{A}$) and victim models (Θ) different from the settings in $\mathcal{D}_{\text{tr}}$.

In the rest of the chapter, both in-distribution and OOD generalization capabilities will be assessed. Unless specified otherwise, the generalization of MPN stands for the *in-distribution generalization*.

3.2.3 Input Data Format for MPN

As revisited from Sec. 3.1, an adversarial example, formulated as $\mathbf{x}' = \mathbf{x} + \boldsymbol{\delta}$, is associated with $\boldsymbol{\theta}$ via $\boldsymbol{\delta}$. Therefore, it might be more advantageous for MPN to utilize *adversarial perturbations* ($\boldsymbol{\delta}$) as the attack data feature ($\mathbf{z}$) instead of the less direct adversarial example $\mathbf{x}'$. **Fig. 3.2** provides empirical evidence supporting this notion by evaluating the generalization ability of MPN when trained on adversarial perturbations versus adversarial examples, considering the two model configurations of MPN, namely, `MLP` and `ConvNet-4`. The efficacy of MPN, trained and evaluated on various attack types, is showcased. The results indicate that employing adversarial perturbations ($\boldsymbol{\delta}$) consistently enhances the accuracy in classifying VM attributes at test time, relative to using adversarial examples ($\mathbf{x}'$). Furthermore, `ConvNet-4` significantly surpasses `MLP` in performance.

Although Fig. 3.2 shows the promise of the generalization ability of MPN when trained and tested on adversarial perturbations, it may raise another practical question of how to obtain adversarial perturbations from adversarial examples if the latter is the only attack source accessible to MPN. To overcome this difficulty, we propose a *perturbation estimator network* (**PEN**) that can be jointly learned with MPN. Once PEN is prepended to the MPN model, the resulting end-to-end pipeline can achieve model parsing using adversarial examples as inputs (see Fig. 3.1).

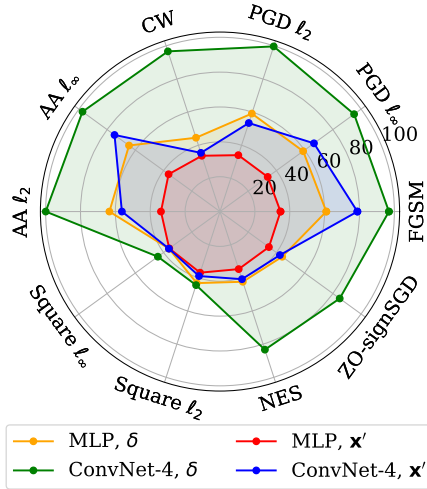


Figure 3.2: The in-distribution generalization of MPN using different formats of input data (adversarial perturbations $\boldsymbol{\delta}$ vs. adversarial examples $\mathbf{x}'$) and parsing networks (`ConvNet-4` vs. `MLP`). The generalization performance is measured by the averaged testing accuracy of attribute-specific classifiers. The attack data are generated from 10 attack methods given in Table 3.1, with ℓ_∞ attack strength $\epsilon = 8/255$ and ℓ_2 attack strength $\epsilon = 0.5$ on `CIFAR-10`. The VM architecture is fixed to ResNet-9, and VM instances are generated by varying other model attributes in Table 3.2.

We use a denoising network, DnCNN (Zhang *et al.*, 2017), to model PEN with parameters $\boldsymbol{\psi}$. PEN obtains perturbation estimates by minimizing the denoising objective function using the true adversarial perturbations as supervision. Extended from (3.1), we then have

$$\begin{aligned} \underset{\boldsymbol{\psi},\boldsymbol{\phi}_{\text{rep}},\{\boldsymbol{\phi}_{\text{atr}}^{(i)}\}_{i=1}^4}{\text{minimize}} \quad & \beta \mathbb{E}_{(\mathbf{x},\mathbf{x}')\in\mathcal{D}_{\text{tr}}}[\ell_{\text{MAE}}(g_{\boldsymbol{\psi}}(\mathbf{x}'), \mathbf{x}' - \mathbf{x})] \\ & + \mathbb{E}_{(\mathbf{x}',y)\in\mathcal{D}_{\text{tr}}} \textstyle\sum_{i=1}^4 [\ell_{\text{CE}}(h(g_{\boldsymbol{\psi}}(\mathbf{x}'); \boldsymbol{\phi}_{\text{rep}}, \boldsymbol{\phi}_{\text{atr}}^{(i)}), y_i)], \end{aligned} \tag{3.2}$$

where $g_{\boldsymbol{\psi}}(\mathbf{x}')$ is the output of PEN given $\mathbf{x}'$ as the input, ℓ_{MAE} is the mean-absolute-error (MAE) loss characterizing the perturbation estimation error, and $\beta > 0$ is a regularization parameter. Compared with (3.1), MPN takes the perturbation estimate $g_{\boldsymbol{\psi}}(\mathbf{x}')$ for VM attribute classification.

3.3 Experiments

In this section, we first introduce our experiment setup and implementation. We then show the effectiveness of our proposed model parsing

method through three different aspects: 1) the in-distribution generalization; 2) the out-of-distribution (OOD) generalization; 3) the evaluation of transfer attacks.

3.3.1 Experiment Setup and Implementation

We utilize image classification datasets such as `CIFAR-10`, `CIFAR-100`, and `Tiny-ImageNet` for the development of victim models, which serve as the basis for generating various attacks. These VM instances are subsequently employed to construct datasets for MPN (model parsing network), as elaborated in Sec. 3.2. Both the types of attacks and the configurations of the victim models are comprehensively listed in Table 3.1 and Table 3.2. Consequently, we have compiled a dataset encompassing adversarial attacks of 7 distinct types, derived from 135 different VMs. These VMs are diversified by 5 architectural types, 3 kernel size configurations, 3 activation function variations, and 3 levels of weight sparsity.

To solve problem (3.1), we train the MPN using the SGD (stochastic gradient descent) optimizer with cosine annealing learning rate schedule and an initial learning rate of 0.1. The training epoch number and the batch sizes are given by 100 and 256, respectively. To solve problem (3.2), we first train MPN according to (3.1), and then fine-tune a pre-trained DnCNN model (Gong *et al.*, 2022) (taking only the denoising objective into consideration) for 20 epochs. Starting from these initial models, we jointly optimize MPN and PEN by minimizing problem (3.2) with $\beta = 1$ over 50 epochs. To evaluate the effectiveness of MPN, we consider both in-distribution and OOD generalization assessment. The generalization performance is measured by testing accuracy averaged over attribute-wise predictions, namely, $\sum_i (N_i \mathrm{TA}(i)) / \sum_i N_i$, where N_i is the number of classes of the model attribute i, and $\mathrm{TA}(i)$ is the testing accuracy of the classifier associated with the attribute i (Fig. 3.1).

3.3.2 In-distribution Generalization of MPN

Table 3.3 presents the in-distribution generalization performance of MPN trained using different input data formats (*i.e.*, adversarial examples $\mathbf{x}'$, PEN-estimated adversarial perturbations $\boldsymbol{\delta}_{\mathrm{PEN}}$, and true

adversarial perturbations $\boldsymbol{\delta}$) given each attack type in Table 3.1. Here the choice of `AT` (architecture type) is fixed to ResNet9, but adversarial attacks on `CIFAR-10` are generated from VMs configured by different values of `KS`, `AF`, and `WS` (see Table 3.2). As we can see, the generalization of MPN varies against the attack type even if model parsing is conducted from the ideal adversarial perturbations ($\boldsymbol{\delta}$). We also note that model parsing from white-box adversarial attacks (*i.e.*, `FGSM`, `PGD`, and `AA`) is easier than that from black-box attacks (*i.e.*, `ZO-signSGD`, `NES`, and `Square`). For example, the worst-case performance of MPN is achieved when training/testing on `Square` attacks. This is not surprising, since `Square` is based on random search and has the least dependence on VM attributes. In addition, we find that MPN using estimated perturbations ($\boldsymbol{\delta}_{\text{PEN}}$) substantially outperforms the one trained on adversarial examples ($\mathbf{x}'$). This justifies the effectiveness of our proposed PEN solution for MPN.

Table 3.3: The in-distribution testing accuracy (%) of MPN trained using different input data formats (adversarial examples $\mathbf{x}'$, PEN-estimated adversarial perturbations $\boldsymbol{\delta}_{\text{PEN}}$, and true adversarial perturbations $\boldsymbol{\delta}$) across different attack types on `CIFAR-10`, with ℓ_∞ attack strength $\epsilon = 8/255$, ℓ_2 attack strength $\epsilon = 0.5$, and `CW` attack strength $c = 1$.

Input data \ Attack type	FGSM	PGD ℓ_∞	PGD ℓ_2	CW	AA ℓ_∞	AA ℓ_2	Square ℓ_∞	Square ℓ_2	NES	ZO-signSGD
$\mathbf{x}'$	78.80	66.62	53.42	35.42	74.78	56.26	38.92	36.21	40.80	42.48
$\boldsymbol{\delta}_{\text{PEN}}$	94.15	83.20	82.58	64.46	91.09	86.89	44.14	42.30	58.85	61.20
$\boldsymbol{\delta}$	96.89	95.07	99.64	96.66	97.48	99.95	44.37	44.05	83.33	84.87

Extended from Table 3.3, **Fig. 3.3** shows the generalization performance of MPN when evaluated using attack data with different attack strengths. We observe that in-distribution generalization (corresponding to the same attack strength for the train-time and test-time attacks) is easier to achieve than OOD generalization (different attack strengths at test time and train time). Another observation is that a smaller gap between the train-time attack strength and the test-time strength leads to better generalization performance.

Table 3.3 and Fig. 3.3 concentrate on the model parsing of adversarial attacks with a specific focus on the ResNet9 architecture implemented on `CIFAR-10`. This examination, however, encompasses various combi-

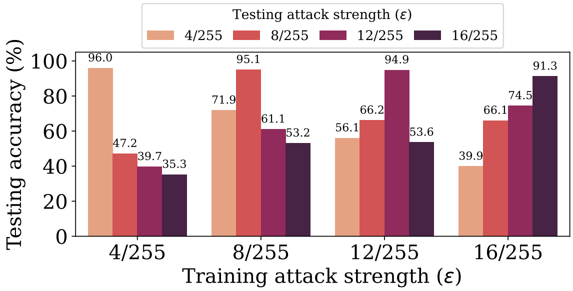


Figure 3.3: Testing accuracies (%) of MPN when trained on adversarial perturbations generated by PGD ℓ_∞ using different attack strengths (ϵ) and evaluated using different attack strengths as well. Other setups are consistent with in Table 3.3.

Table 3.4: In-distribution generalization performance (testing accuracy, %) of MPN given different choices of VMs and datasets, attack types/strengths, and MPN input data formats ($\mathbf{x}'$, $\boldsymbol{\delta}_{\mathrm{PEN}}$, and $\boldsymbol{\delta}$).

Attack type	**Attack strength**	**Dataset and victim model:** CIFAR-10 ResNet9			CIFAR-10 ResNet18			CIFAR-10 ResNet20			CIFAR-10 VGG11			CIFAR-10 VGG13			CIFAR-100 ResNet9			Tiny-ImageNet ResNet18		
		$\mathbf{x}'$	$\boldsymbol{\delta}_{\mathrm{PEN}}$	$\boldsymbol{\delta}$	$\mathbf{x}'$	$\boldsymbol{\delta}_{\mathrm{PEN}}$	$\boldsymbol{\delta}$	$\mathbf{x}'$	$\boldsymbol{\delta}_{\mathrm{PEN}}$	$\boldsymbol{\delta}$	$\mathbf{x}'$	$\boldsymbol{\delta}_{\mathrm{PEN}}$	$\boldsymbol{\delta}$	$\mathbf{x}'$	$\boldsymbol{\delta}_{\mathrm{PEN}}$	$\boldsymbol{\delta}$	$\mathbf{x}'$	$\boldsymbol{\delta}_{\mathrm{PEN}}$	$\boldsymbol{\delta}$	$\mathbf{x}'$	$\boldsymbol{\delta}_{\mathrm{PEN}}$	$\boldsymbol{\delta}$
FGSM	$\epsilon = 4/255$	60.13	85.25	96.82	60.00	86.92	97.66	62.41	88.91	97.64	47.42	73.40	91.75	66.28	90.02	98.57	57.99	82.22	94.86	37.23	84.27	97.04
	$\epsilon = 8/255$	78.80	94.15	96.89	80.44	95.49	97.61	82.29	95.90	97.72	63.13	86.76	92.41	84.92	96.91	98.66	75.58	91.65	94.96	70.29	91.17	97.05
	$\epsilon = 12/255$	86.49	95.96	96.94	88.03	96.89	97.68	88.71	97.13	97.81	73.71	90.19	92.66	91.21	98.10	98.71	82.27	94.01	95.55	76.00	93.45	97.02
	$\epsilon = 16/255$	90.16	96.43	96.94	91.71	97.34	97.68	91.84	97.47	97.79	79.51	91.28	92.60	94.22	98.44	98.73	86.50	94.04	94.74	79.63	94.35	96.87
PGD ℓ_∞	$\epsilon = 4/255$	50.54	76.43	96.02	56.94	79.45	96.96	55.01	80.05	97.49	39.33	66.38	91.84	57.12	81.18	98.29	42.27	72.62	92.65	35.48	76.56	97.18
	$\epsilon = 8/255$	66.62	83.20	95.07	73.29	87.29	95.38	67.49	86.19	96.18	56.62	81.14	92.78	69.16	88.46	97.22	59.71	79.55	90.43	61.85	82.90	96.05
	$\epsilon = 12/255$	76.65	89.73	94.91	81.73	91.67	95.55	76.41	90.16	95.67	70.56	88.92	94.13	78.67	92.93	97.26	70.86	85.31	91.28	73.82	88.80	96.38
	$\epsilon = 16/255$	75.58	86.95	91.28	82.46	90.19	93.19	76.58	87.79	92.50	72.13	87.23	91.85	78.28	90.20	94.66	71.29	82.35	86.84	73.19	85.02	93.54
PGD ℓ_2	$\epsilon = 0.25$	36.75	62.20	99.66	46.35	70.17	99.74	48.24	77.22	99.75	36.47	45.17	98.52	35.81	70.62	99.85	35.92	61.91	99.29	35.55	35.68	99.68
	$\epsilon = 0.5$	53.42	82.58	99.64	60.89	84.70	99.56	61.62	89.11	99.61	41.56	66.58	98.68	57.83	87.64	99.83	48.89	79.26	99.01	35.52	54.56	99.71
	$\epsilon = 0.75$	62.66	89.04	99.48	71.01	89.89	99.22	70.76	92.06	99.36	47.02	78.12	98.52	72.76	92.32	99.74	59.19	85.14	98.61	35.56	81.33	99.71
	$\epsilon = 1$	71.65	91.73	99.26	77.09	92.09	98.94	76.84	92.82	98.96	54.20	84.30	98.41	79.93	93.96	99.57	66.97	87.63	97.89	43.48	88.81	99.64
CW	$c = 0.1$	33.77	55.60	96.71	47.77	63.26	96.11	33.56	63.11	94.10	33.73	48.90	94.37	33.68	65.48	96.95	34.41	46.47	92.55	35.96	35.77	95.52
	$c = 1$	35.42	64.46	96.66	45.75	65.25	97.45	33.74	62.71	97.08	33.89	55.61	91.29	36.12	68.66	98.58	34.25	55.18	93.25	35.54	35.29	89.35
	$c = 10$	36.38	64.45	96.64	45.83	65.32	97.41	33.83	63.52	97.11	38.29	56.83	91.33	38.51	68.28	98.62	34.25	55.89	93.18	35.45	53.18	94.20

nations of model attributes, resulting in multiple ResNet9-type VM instances for generating attacks. Additionally, **Table 3.4** highlights the in-distribution generalization of MPN across a range of victim model architectures and datasets. The findings align with those presented in Table 3.3, revealing two key insights: (1) Utilizing actual adversarial perturbations ($\boldsymbol{\delta}$) and perturbations estimated by PEN ($\boldsymbol{\delta}_{\mathrm{PEN}}$) tends to enhance model parsing accuracy. (2) The task of inferring model attributes is comparatively more straightforward when utilizing white-box, gradient-based adversarial perturbations, as evidenced by their testing accuracy surpassing 90%. It is also observed that when adversarial examples ($\mathbf{x}'$) or PEN-estimated adversarial perturbations ($\boldsymbol{\delta}_{\mathrm{PEN}}$) are employed for model parsing, the accuracy improves with the increase in the strength of the adversarial attacks.

3.3.3 OOD Generalization of MPN vs. Unseen Attack Types

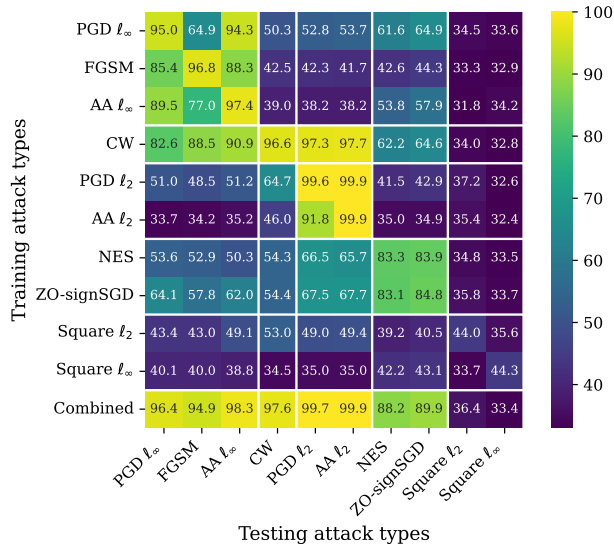


Figure 3.4: Generalization performance (%) matrix of MPN when trained on a row-specific attack type but evaluated on a column-specific attack type. The attack data are given by adversarial perturbations with strength $\epsilon = 8/255$ for ℓ_∞ attacks, $\epsilon = 0.5$ for ℓ_2 attacks, and $c = 1$ for `CW` attack. The VM architecture and the dataset are set to ResNet9 and `CIFAR-10`. The 'combined' row represents MPN training on the collection of four attack types: `PGD` ℓ_∞, `PGD` ℓ_2, `CW`, and `ZO-signSGD`.

In **Fig. 3.4**, we exhibit the generalization matrix of MPN, demonstrating its performance when trained on one type of attack (for instance, the `PGD` ℓ_∞ attack shown in row 1) and then tested on a different attack type (such as the `FGSM` attack indicated in column 2). These adversarial perturbations are generated from an array of ResNet9-based VMs on `CIFAR-10`, each configured with various model attribute settings. The matrix's diagonal entries represent the in-distribution generalization of MPN for each specific attack type, whereas the off-diagonal entries reveal the OOD generalization, highlighting how MPN adapts when the attack type at test time differs from that during training.

First, our analysis reveals that MPN exhibits enhanced cross-attack type generalization when the attacks bear similarities. This observation leads to the identification of *generalization communities*: ℓ_∞ attacks

(including PGD ℓ_∞, FGSM, and AA ℓ_∞), ℓ_2 attacks (comprising CW, PGD ℓ_2, and AA ℓ_2), and zeroth-order optimization (ZOO)-based black-box attacks (such as NES and ZO-signSGD). **Second**, we note that MPN struggles with learning and generalizing from Square attacks, as reflected by the lower test accuracies observed in the last two rows and columns of the matrix. This finding aligns with the data presented in Table 3.3. **Third**, acknowledging the existence of these generalization communities, we explore the potential of data augmentation by amalgamating diverse attack types, including PGD ℓ_∞, PGD ℓ_2, CW, and ZO-signSGD, into the MPN training dataset. We aim to assess whether such augmentation enhances the out-of-distribution (OOD) generalization of MPN. The outcomes of this experiment are detailed in the **'combined' row of Fig. 3.4**. As anticipated, the incorporation of a combination of attack types does indeed improve MPN's generalization capabilities across various attack types, with the notable exception of the random search-based Square attack.

3.3.4 Model Parsing via Transfer Attacks

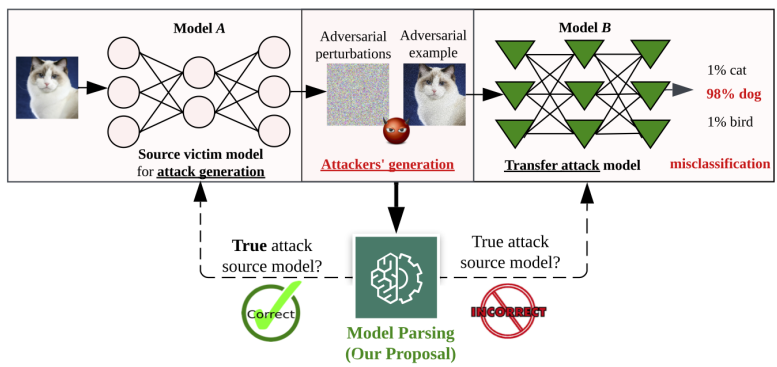
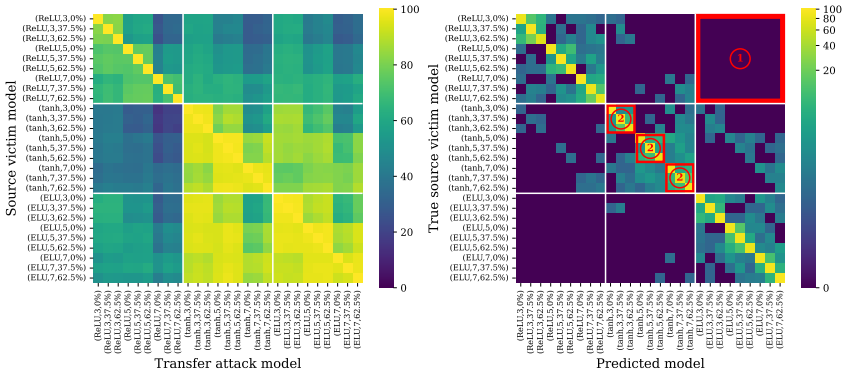


Figure 3.5: Motivating example on model parsing of transfer attack. A successful model parsing module can reveal the true source victim model.

In the context of model parsing as a practical application, we explore the ability of MPN to accurately deduce the source VM attributes from transfer attacks, where these attacks are employed against distinct mod-



(a) Attack successful rate (%) (b) Confusion matrix (%)

Figure 3.6: Model parsing of transfer attacks: Transfer attack success rate matrix (a) and model parsing confusion matrix (b). Given the architecture type ResNet9, the dataset `CIFAR-10`, and the attack type `PGD` ℓ_∞ (with strength $\epsilon = 8/255$), each model attribute combination (`AF`, `KS`, `WS`) defines a model instance to be attacked, transferred, or parsed.

els (illustrated in **Fig. 3.5**). Utilizing ResNet9 as the VM architecture, we manipulate the values of model attributes `KS`, `AF`, and `WS`, resulting in 27 unique ResNet9-type VMs. **Fig. 3.6** displays both the transfer attack success rate (ASR) matrix (Fig. 3.6a) and the model parsing confusion matrix (Fig. 3.6b). The considered attack type is the `PGD` ℓ_∞ attack with a strength of $\epsilon = 8/255$ on `CIFAR-10`, and the adversarial perturbations generated from these various VMs are utilized for training and evaluating MPN.

In **Fig. 3.6a**, the off-diagonal entries indicate the ASRs of transfer attacks originating from row-wise VMs and targeting column-wise models. We observe that adversarial attacks crafted from ReLU-based VMs are generally more challenging to transfer to target models with smooth activation functions, such as ELU or tanh. Conversely, given consistent values of `KS` and `AF`, transferability across models with different weight sparsity levels seems more feasible.

Fig. 3.6b portrays the confusion matrix for MPN trained on attack data derived from all 27 ResNet9-like VMs. Each row in this matrix corresponds to the actual VM used for generating the attack dataset, while each column relates to a predicted model attribute configuration. The diagonal entries represent the correct model parsing accuracy, and

the off-diagonal entries indicate the misclassification rates for incorrectly predicted model attribute configurations. Notably, attacks generated from ReLU-based VMs lead to a lower misclassification rate by MPN for predictions involving ELU or tanh (as highlighted in the marked region ①). However, a higher rate of misclassification is observed when MPN is evaluated on attack data corresponding to varying `WS` values (as indicated in the marked region ②). These findings, coupled with the insights on ASRs of transfer attacks from Fig. 3.6a, imply a linkage between the ease of transferability of attacks and the subsequent model parsing difficulty: *If attacks are challenging (or straightforward) to transfer from a source model to a target model, the inference of source model attributes from these attacks becomes easier (or harder).*

3.4 Conclusion

This chapter has delved into the intricate realm of model parsing for adversarial examples. We have shown that adversarial examples, while primarily designed to mislead and deceive machine learning models, carry with them a wealth of information about the source victim models from which they originate. Through the lens of model parsing, these adversarial perturbations transform from the knowledge-limited deception tools into rich sources of insight, enabling the reverse engineering capability.

Our exploration highlighted that model parsing accuracy is significantly influenced by the nature of the adversarial examples. Specifically, the attributes of the victim models, such as architecture, activation functions, and weight sparsity, play a pivotal role in determining the effectiveness of model parsing. The ability to correctly infer these attributes from adversarial examples opens up new avenues for enhancing the security and trustworthiness of machine learning systems.

Moreover, the study of transfer attacks in the context of model parsing presented intriguing findings. We noted that the transferability of adversarial attacks is closely linked to the feasibility of model parsing. Attacks that are challenging to transfer between models often provide clearer signals for model parsing, enabling more accurate inference of the source model's attributes. Conversely, easily transferable attacks

tend to obscure the distinctive features necessary for effective model parsing.

In conclusion, model parsing for adversarial examples stands as a testament to the multifaceted nature of adversarial machine learning. It underscores the potential to turn adversarial weaknesses into analytical strengths. As we continue to advance in this field, the insights gained from model parsing will undoubtedly contribute to the development of more robust, secure, and transparent AI systems. Future research in this domain holds the promise of further unraveling the complexities of adversarial examples, paving the way for a deeper understanding and enhanced resilience against adversarial threats.

4

Reverse Engineering of Generated Images

Chapter overview. Expanding upon RED techniques designed for machine-centric attacks, this chapter shifts the focus to RED techniques tailored for human-centric attacks. Specifically, we introduce the model parsing of generative models, which infers that hyperparameters utilized in the generative model are responsible for image synthesization. Notably, we explore two types of dependencies that can enhance the overall model parsing performance: dependencies among different generative models and among different hyperparameters used in each generative model. Building upon this insight, two methods–the two-stage model parsing network (Chapter 4.3) and the learnable graph pooling network (Chapter 4.4)–will be proposed in this chapter.

4.1 Motivation and Background

Recent advancements in image generation, particularly with the advent of Generative Adversarial Networks (GANs) as cited in (Goodfellow *et al.*, 2014a), have led to significant improvements in the image quality. Various Generative Models (GMs), including GANs and Variational Autoencoders (VAEs) referenced in (Karras *et al.*, 2019; Choi *et al.*, 2018; Karras *et al.*, 2018; Kingma and Welling, 2014; Burgess *et al.*,

2017; Chen *et al.*, 2018; Dhariwal and Nichol, 2021), now possess the ability to create highly realistic images, often indistinguishable from actual photographs by *humans*. This level of photorealism, however, brings forth concerns regarding the potential misuse of these technologies, such as orchestrating misinformation campaigns, as discussed in (Waldemarsson, 2020; Heath, 2019). Consequently, the field of deepfake detection, referenced in (Rossler *et al.*, 2019; McCloskey and Albright, 2019; Guarnera *et al.*, 2020; Marra *et al.*, 2019b; Dang *et al.*, 2020; Nirkin *et al.*, 2020), has garnered *increased* interest.

In an effort to advance beyond the conventional genuine *versus* fake dichotomy prevalent in deepfake detection, Yu *et al.* (2019) introduced the concept of source model classification for a given generated image. This approach, termed *image attribution*, operates under the assumption of a *closed set* of GMs, which are utilized in both the training and testing phases.

The progression of image attribution towards open-set recognition, or the ability to classify images generated by GMs not encountered during training, is a sought-after goal. One might ponder what further insights *we can gain* beyond merely categorizing a GM as an *unseen* or *new* model. An intriguing question arises: Is it possible to discern the design nuances of these new GMs? Can we determine how their architectures diverge from those of the known GMs included in our training set? Unraveling these aspects is crucial, particularly for defenders aiming to trace the origins of images crafted by malevolent entities or to pinpoint coordinated misinformation campaigns utilizing identical GMs. This pursuit essentially forms the cornerstone of what we consider the grand challenge in the reverse engineering of GMs.

4.2 Problem Statement

Our research aims to advance the field of image attribution for GMs (generative models) by addressing a novel and challenging problem: model parsing. Model parsing involves estimating the hyperparameters of an unseen GM solely from its generated images. This approach is rooted in the observation that different GMs primarily vary in their model hyperparameters, such as network architectures (*e.g.*, the number

of layers/blocks, the type of normalization) and training loss functions. The goal is to map generated images to the embedding space of the model hyperparameters used to create them. While reverse engineering machine learning models based on their input and output (Oh *et al.*, 2019; Tramèr *et al.*, 2016), or by accessing hardware usage during inference (Hua *et al.*, 2018; Batina *et al.*, 2019), has been previously explored, to our knowledge, such an approach has not been applied to GMs using only generated images as input. This research endeavors to fill this gap by developing methods to accurately estimate the hyperparameters of unseen GMs from their generated images, thereby contributing a new dimension to the understanding and analysis of GMs.

4.3 Proposed Method 1: Two-stage Model Parsing Network

Our model comprises two main parts as illustrated in **Fig. 4.1** (at the bottom). The Fingerprint Estimation network (FEN) discerns the subtle, distinct patterns that GMs imprint on the images they create. Our approach to recognizing fingerprints is anchored on fundamental fingerprint characteristics: magnitude, recurrence, spectral domain, and symmetric spectral response. We define varied loss functions that impose these characteristics to ensure that the recognized fingerprints exhibit these crucial attributes. Such guidelines allow us to discern the fingerprints of GMs even in the absence of a ground truth.

The fingerprints we identify are distinctive and become the foundational element for further procedures.

Following this, the next segment of our design is the Parsing Network (PN). This network interprets the extracted fingerprint and forecasts the model's hyperparameters. To capitalize on the resemblances amongst various GMs, we categorize these GMs into multiple groupings, determined by their original hyperparameters. Each GM's average and variance are computed. We use two different parsers: cluster parser and instance parser to predict the mean and deviation of these parameters, which are then combined as the final predictions.

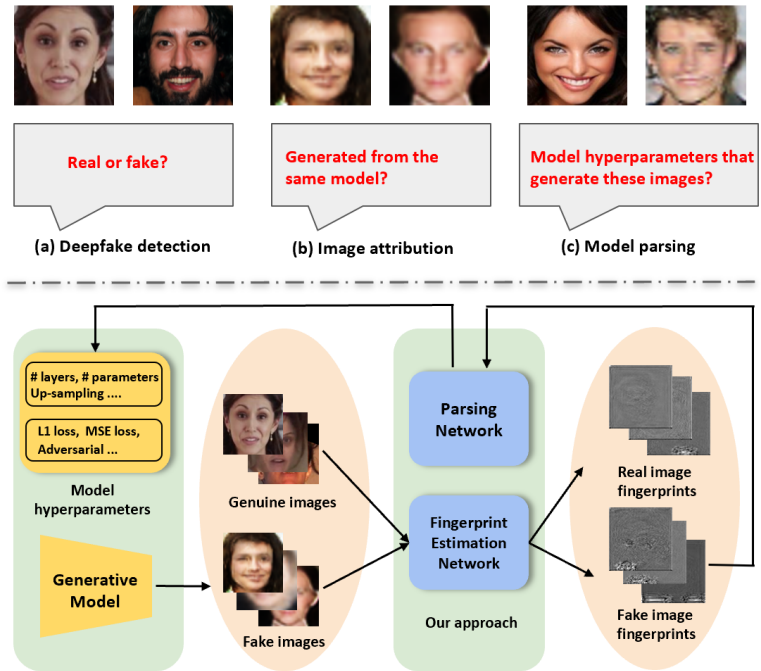


Figure 4.1: Top: Three increasingly difficult tasks: (a) *deepfake detection* classifies an image as genuine or fake; (b) *image attribution* predicts which of a closed set of GMs generated a fake image; and (c) *model parsing*, proposed here, infers hyperparameters of the GM used to generate an image, for those models unseen during training. Bottom: We present a framework for model parsing, which can also be applied to simpler tasks of deepfake detection and image attribution.

4.3.1 Data Collection

We make the first attempt to study the model parsing problem. Given that research thrives on data, curating a dataset specific to this novel research problem becomes paramount. Considering the proliferation of GMs in recent literature (Wang *et al.*, 2021b; Jabbar *et al.*, 2020), our selection criteria for GMs to incorporate in our dataset are based on several factors. To this end, we assemble a list of 116 publicly available GMs, including ProGan (Karras *et al.*, 2018), StyleGAN (Karras *et al.*, 2019), and others. For each GM, we collect $1,000$ generated images. Therefore, our dataset $\mathcal{D}$ comprises of $116,000$ images. These GMs

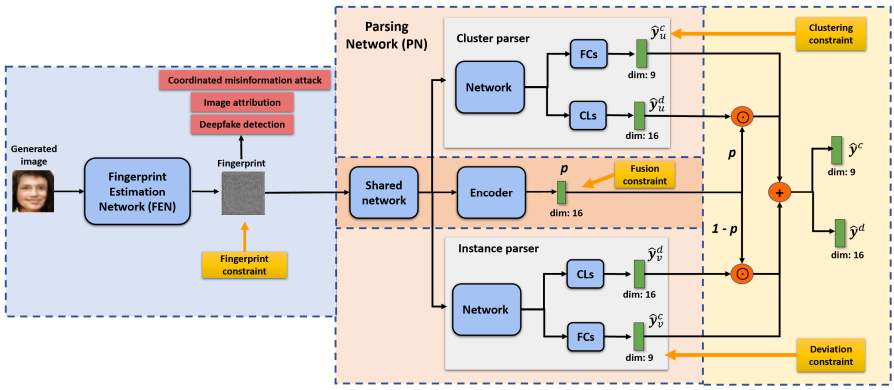


Figure 4.2: Our framework includes two components: 1) the FEN is trained with four objectives for fingerprint estimation; and 2) the PN consists of a shared network, two parsers to estimate mean and deviation for each parameter, an encoder to estimate fusion parameter, fully connected layers (FCs) for continuous type parameters and separate classifiers (CLs) for discrete type parameters in network architecture and loss function prediction. Blue boxes denote trainable components; green boxes denote feature vectors; orange boxes denote loss functions; red boxes denote other tasks our framework can handle; black arrows denote data flow; orange arrows denote loss supervisions. Best viewed in color.

were trained on datasets with various contents, such as CelebA (Liu *et al.*, 2015), MNIST (Deng, 2012), CIFAR10 (Krizhevsky, Hinton, *et al.*, 2009), ImageNet (Deng *et al.*, 2009), facades (Zhu *et al.*, 2017), edges2shoes (Zhu *et al.*, 2017), and apple2oranges (Zhu *et al.*, 2017). The dataset is available here. Additionally, we systematically catalog the hyperparameters for each GM, as detailed in their corresponding scholarly articles. Specifically, our focus encompasses two primary facets: the underlying network architecture and the loss functions during the training phase. We now discuss our framework step by step, with first discussing about the FEN , followed by the PN.

4.3.2 Fingerprint Estimation Network (FEN)

We adopt a network structure similar to the DnCNN model used in (Zhang *et al.*, 2017). As shown in **Fig. 4.2**, the input to FEN is a generated image $\mathbf{X}$, and the output is a fingerprint image $\mathbf{F}$ of the

same size. Drawing inspiration from previous research on tangible fingerprint estimation (Jourabloo *et al.*, 2018; Zhang *et al.*, 2019b; Wang *et al.*, 2020b; Yu *et al.*, 2019; Marra *et al.*, 2019a), we establish four guiding constraints to ensure our deduced fingerprints exhibit the desired characteristics.

Magnitude loss. Fingerprints are often conceptualized as image noise patterns with minimal magnitudes. Hence, our inaugural constraint is designed to regulate the fingerprint image to maintain a subdued magnitude, utilizing an L_2 loss:

$$J_m = ||\mathbf{F}||_2^2. \tag{4.1}$$

Spectrum loss. We advocate for reducing the low-frequency components within a fingerprint image by introducing a low-pass filter in its frequency domain:

$$J_s = ||\mathcal{L}(\mathcal{F}(\mathbf{F}), f)||_2^2, \tag{4.2}$$

where $\mathcal{F}$ denotes the Fourier transform, and $\mathcal{L}$ represents the low-pass filter that targets the central $f \times f$ region of the 2D Fourier spectrum, setting all other values to zero.

Repetitive loss. We amplify the high-frequency data to accentuate this recurring motif:

$$J_r = -max\{\mathcal{H}(\mathcal{F}(\mathbf{F}), f)\}, \tag{4.3}$$

Here, $\mathcal{H}$ acts as a high-pass filter, setting the central $f \times f$ domain of the 2D Fourier spectrum to null.

Energy loss. (Wang *et al.*, 2020b) demonstrated that distinctive patterns are evident in the Fourier spectrum of images crafted by CNN networks. Notably, these patterns exhibit comparable energy along the vertical and horizontal axes of the Fourier spectrum. To encapsulate this insight, we introduce our concluding constraint:

$$J_e = ||\mathcal{F}(\mathbf{F}) - \mathcal{F}(\mathbf{F})^T||_2^2, \tag{4.4}$$

where $\mathcal{F}(\mathbf{F})^T$ symbolizes the transposition of $\mathcal{F}(\mathbf{F})$.

These outlined constraints steer the direction of our fingerprint estimation training. As illustrated in Fig. 4.2, the cumulative fingerprint constraint is expressed as:

$$J_f = \lambda_1 J_m + \lambda_2 J_s + \lambda_3 J_r + \lambda_4 J_e, \tag{4.5}$$

with λ_1, λ_2, λ_3, λ_4 functioning as individual loss weights for each respective term.

4.3.3 Model Parsing Using Fingerprint

As visualized in Fig. 4.2, our framework incorporates two distinct parsers: the cluster parser and the instance parser. These predictions are amalgamated to deduce both the network's architectural design and its associated loss functions. Subsequent sections delve into the mechanics of ground truth computation and a comprehensive elaboration of our framework's intricacies.

Ground truth hyperparamters

Network architecture. Our main goal is not to retrieve all the specific network parameters. We focus on inferring key hyperparameters that describe the network's overall architecture. These are substantially fewer than the actual network parameters and can provide a concise understanding of the architecture. Building upon the foundation laid by previous studies in neural architecture search (Tan *et al.*, 2019; Pham *et al.*, 2018; Liu *et al.*, 2018), we identify a set of 15 essential hyperparameters that encompass various architectural features. For organizational purposes, we divided the network architecture parameters, represented as $\mathbf{y}^n$, into two categories: $\mathbf{y}^{n_c} \in \mathbb{R}^9$ for continuous parameters and $\mathbf{y}^{n_d} \in \mathbb{R}^6$ for discrete ones.

Loss function. Besides the intrinsic architecture of a network, the parameters a model learns during training can significantly influence the fingerprints found on the generated images. These learned parameters are largely shaped by the training data and the loss functions employed during the training process. Consequently, we delve into the potential of predicting these training loss functions using the fingerprints we've estimated. The dataset of 116 GMs makes use of 10 distinct loss functions for training. To represent this information, we devise a ground-truth vector, denoted by $\mathbf{y}^l \in \mathbb{R}^{10}$, where each element holds a binary value. This binary value indicates the presence (or absence) of a particular loss function during the training of a given model.

Our framework aims to parse hyperparameters, which can be broadly categorized into two: continuous and discrete. The former encapsulates the continuous attributes related to network architecture, while the latter encompasses discrete architectural details and parameters linked to the loss function. For a streamlined explanation, we'll segregate these parameters based on their nature - continuous or discrete - throughout this chapter. We represent these two sets of parameters as $\mathbf{y}^c$ for continuous and $\mathbf{y}^d$ for discrete.

Prediction of parameters

Directly inferring the hyperparameters for each Generative Model (GM) independently has proven to produce subpar results in our experiments. Interestingly, several GMs in our dataset exhibit similarities in terms of their network architectures or loss functions. Harnessing these inherent similarities can potentially bolster the accuracy of hyperparameter predictions.

To capitalize on these resemblances, we employ k-means clustering to group the GMs into distinct clusters. Our approach encompasses two levels of prediction: a broader, cluster-level prediction and a more nuanced, GM-specific prediction. These two prediction levels are then amalgamated to yield the final hyperparameter estimation.

Thus, we concatenate the ground truth network architecture parameters $\mathbf{y}^n$ and loss function parameters $\mathbf{y}^l$, denoted as $\mathbf{y}^{nl}$. We use these ground truth vectors to perform k-means clustering to find the optimal k-clusters in the dataset $\mathcal{D} = \{\boldsymbol{C}_1, \boldsymbol{C}_2, ...\boldsymbol{C}_k\}$. Our clustering objective can be written as:

$$\text{argmin}_{\mathcal{D}} \sum_{i=1}^{k} \sum_{\mathbf{y}_j^{nl} \in \boldsymbol{C}_i} ||\mathbf{y}_j^{nl} - \mu_i||^2, \tag{4.6}$$

where μ_i is the mean of the ground truth of the GMs in $\boldsymbol{C}_i$.

To ascertain the cluster ground truth and prediction, we employ distinct strategies tailored to continuous and discrete parameters:

For continous paramters, the prediction error is quantified using the L_2 loss, articulated as:

$$J_u^c = ||\hat{\mathbf{y}}_u^c - \mathbf{y}_u^c||_2^2, \tag{4.7}$$

where $\hat{\mathbf{y}}_u^c$ symbolizes the predicted cluster mean and $\mathbf{y}_u^c$ represents the normalized ground truth of the cluster mean.

For discrete paramters, the loss term for discrete parameters cluster-prediction is defined as:

$$J_u^d = -\sum_{m=1}^{M} \text{sum}(\mathbf{y}_{u_m}^d \odot \log(\mathcal{S}(\hat{\mathbf{y}}_{u_m}^d))), \tag{4.8}$$

where $\mathbf{y}_{u_m}^d$ is the ground-truth one-hot vector for the respective class in the m-th discrete type parameter, $\hat{\mathbf{y}}_{u_m}^d$ are the class logits, $\mathcal{S}$ is the Softmax function that maps the class logits into the range of $[0, 1]$, $\odot$ is the element-wise multiplication, and sum() computes the summation of a vector's elements.

Therefore, the clustering constraint is given by:

$$J_u = \gamma_1 J_u^c + \gamma_2 J_u^d, \tag{4.9}$$

where γ_1 and γ_2 are the loss weights for each term.

For instance parser, we use the same procedure as stated above for cluster prediction, but we change the ground truth vector. To compute the ground truth deviation vector, denoted as $\mathbf{y}_v$, we take into consideration two different types of parameters—continuous and discrete. For continuous parameters, this deviation is characterized by the disparity between an individual GM's ground truth and its corresponding cluster's ground truth. Contrastingly, for discrete parameters, the innate ground truth class for these parameters can serve as the deviation from the cluster's predominant class.

To gauge the accuracy of deviation-level prediction, appropriate loss functions are employed. In the context of continuous parameters, the L_2 loss is leveraged to quantify the prediction error:

$$J_v^c = ||\hat{\mathbf{y}}_v^c - \mathbf{y}_v^c||_2^2, \tag{4.10}$$

Herein, $\hat{\mathbf{y}}_v^c$ represents the predicted deviation, whilst $\mathbf{y}_v^c$ represents the groud-truth deviation for continuous parameters.

The loss component for the deviation prediction of discrete parameters is delineated as:

$$J_v^d = -\sum_{m=1}^{M} \text{sum}(\mathbf{w}_m \odot \mathbf{y}_{v_m}^d \odot \log(\mathcal{S}(\hat{\mathbf{y}}_{v_m}^d))), \tag{4.11}$$

Here, $\mathbf{y}^d_{v_m}$ is the ground-truth one-hot deviation vector affiliated with the m-th classifier. $\mathbf{w}_m$ represents the weight vector associated with all classes of the m-th classifier, and $\hat{\mathbf{y}}^d_{v_m}$ corresponds to the class logits.

Incorporating the findings presented in Fig. 4.2, the cumulative deviation constraint can be presented as:

$$J_v = \gamma_3 J^c_v + \gamma_4 J^d_v. \tag{4.12}$$

Within this equation, γ_3 and γ_4 are the loss weights designated to each respective term.

Continuous and discrete parameter integration

For the continuous parameter space, the process is straightforward. The coarse-level predictions (reflecting broader general trends) and the deviations (capturing finer individual-specific variations) are combined using element-wise addition. The formula is represented as:

$$\hat{\mathbf{y}}^c = \hat{\mathbf{y}}^c_u + \hat{\mathbf{y}}^c_v, \tag{4.13}$$

This ensures that both the global trend and the granular deviations are captured in the final prediction.

However, when it comes to the discrete parameter space, the task is not as straightforward. Our initial approach, which involved an element-wise addition of logits for every classifier from both parsers, did not yield satisfactory results. This motivated us to explore an alternative approach.

To reconcile the outputs from both parsers, we designed an encoder network to predict a fusion parameter, denoted as $\hat{p^d}$, for every classifier. This fusion parameter, constrained between 0 and 1, plays a pivotal role in guiding how much weightage to give to the coarse versus granular level prediction for each classifier.

To elucidate, for the m-th classifier, the fusion parameter p^d_m is supervised by the ground truth and can be mathematically represented as:

$$p^d_m = \begin{cases} 1, & \mathbf{y}^d_{u_m} = \mathbf{y}^d_{v_m} \\ 0, & \mathbf{y}^d_{u_m} \neq \mathbf{y}^d_{v_m}. \end{cases} \tag{4.14}$$

In essence, this approach harmonizes the outputs from the cluster and instance parsers in the discrete parameter space, optimizing the accuracy

of the overall prediction.To train our encoder, we use the ground truth fusion parameter $\mathbf{p}^d$ which is the concatenation for all parameters. The training is done via cross-entropy loss as shown below:

$$J_p = -\sum_{m=1}^{M} (p_m^d \log(\mathcal{G}(\hat{p}_m^d)) + (1 - p_m^d)\log(1 - \mathcal{G}(\hat{p}_m^d))). \tag{4.15}$$

where $\mathcal{G}$ is the Sigmoid function that maps the class logits into the range of $[0, 1]$.

As shown in Fig. 4.2 for discrete parameters, the final prediction is given by:

$$\hat{\mathbf{y}}^d = \hat{\mathbf{p}}^d \odot \hat{\mathbf{y}}_u^d + (\mathbf{1} - \hat{\mathbf{p}}^d) \odot \hat{\mathbf{y}}_v^d. \tag{4.16}$$

The overall loss function for model parsing is given by:

$$J = J_f + J_u + J_v + \gamma_5 J_p. \tag{4.17}$$

where γ_5 is the loss weight for fusion constraint. Our framework is trained end-to-end with fingerprint estimation (Eqn. 4.5) and model parsing (Eqn. 4.17).

4.3.4 Experiments

In pioneering the field of GM parsing, we face the challenge of a lack of existing comparative works. To construct a baseline for our research, we draw *an* analogy with the image attribution task. In image attribution, each model is typically symbolized as a one-hot vector, with uniform inter-model distances in the dimensional space constituted by these vectors. Conversely, in model parsing, we represent each model with a 25-dimensional vector, partitioned into network architectures (15 dimensions) and training loss functions (10 dimensions), leading to non-uniform distances among models in this 25-D space.

To establish a baseline, termed *random ground-truth*, we shuffle each parameter's values or classes across all 116 GMs. This ensures that the assigned ground-truth deviates from the actual ground-truth while retaining the authentic distribution of each parameter, implying that the random ground-truth baseline does not rely on random chance. These randomized vectors mirror the properties of our actual ground-truth vectors regarding non-equal distances, yet they do not correspond to the true model hyperparameters. We train and evaluate our proposed

approach using this randomly shuffled ground-truth. Given the inherent randomness of this baseline, we execute three random shufflings and report the averaged performance.

Additionally, we investigate a baseline where continuous hyperparameters are always predicted as their mean, and discrete hyperparameters as their mode, across the four sets. These mean/mode values, representing central tendencies, might offer adequate performance for model parsing.

In order to verify the effectiveness of our proposed fingerprint estimation constraints, we undertake an ablation study. This involves training our framework exclusively with the model parsing objective, as outlined in Eqn. 4.17. This procedure establishes what we term the *no fingerprint* baseline. Additionally, to emphasize the significance of our clustering and deviation parser, we implement an approach where the network architecture and loss functions are determined using a single parser. This parser directly estimates the parameters, as opposed to calculating a mean and deviation. We denote this approach as the *using one parser* baseline.

Network architecture prediction

The outcomes of our network architecture prediction are summarized in **Table 4.1**. Our method exhibits notably lower L1 error rates in comparison to the random ground-truth baseline for continuous parameters, and superior classification accuracy and F1 scores for discrete parameters. These results suggest a substantial and generalizable link between the generated images and the meaningful architecture hyperparameters and loss function types' embedding space, surpassing a random vector with the same length and distribution. This significant correlation underpins the validity and feasibility of model parsing for GMs.

Moreover, our approach surpasses the mean/mode baseline, indicating that merely predicting the average for continuous parameters falls short. The inferior outcomes when omitting fingerprint estimation objectives highlight their critical role in model parsing. By comparing the results of using just one parser against our full method, the superiority of our approach, especially in estimating mean and deviation, is clearly

Table 4.1: Performance of network architecture prediction. We use L_1 error, p-value, correlation coefficient, coefficient of determination and slope of RANSAC regression line for continuous type parameters. For discrete parameters, we use F1 score and classification accuracy. We also show the standard deviation over all the test samples for L_1 error. The first value is the standard deviation across sets, while the second one is across the samples. The p-value would be estimated for every ours-baseline pair. Our method performs better for both types of variables compared to the three baselines. [KEYS: corr.: correlation, coef.: coefficient, det.: determination]

Method	Continuous type					Discrete type	
	L_1 error ↓	P-value ↓	Corr. coef. ↑	Coef. of det. ↑	Slope ↑	F1 score ↑	Accuracy ↑
Random ground-truth	$0.184 \pm 0.019/0.036$	0.006 ± 0.001	0.261 ± 0.181	0.315 ± 0.095	0.592 ± 0.041	0.529 ± 0.078	0.575 ± 0.097
Mean/mode	$0.164 \pm 0.011/0.016$	0.035 ± 0.005	0.326 ± 0.112	0.467 ± 0.015	0.632 ± 0.024	0.612 ± 0.048	0.604 ± 0.046
No fingerprint	$0.170 \pm 0.035/0.012$	0.017 ± 0.004	0.738 ± 0.014	0.605 ± 0.152	0.892 ± 0.021	0.700 ± 0.032	0.663 ± 0.104
Using one parser	$0.161 \pm 0.028/0.035$	0.032 ± 0.002	0.226 ± 0.030	0.512 ± 0.116	-0.529 ± 0.075	0.607 ± 0.034	0.593 ± 0.104
Ours	$\mathbf{0.149 \pm 0.019/0.014}$	-	$\mathbf{0.744 \pm 0.098}$	$\mathbf{0.612 \pm 0.161}$	$\mathbf{0.921 \pm 0.021}$	$\mathbf{0.718 \pm 0.036}$	$\mathbf{0.706 \pm 0.040}$

demonstrated.

For continuous parameters, we delve deeper into the efficacy of regression predictions by employing three key metrics: the correlation coefficient, the coefficient of determination, and the slope of the RANSAC regression line. These metrics are calculated based on the comparison between the predictions and the ground-truth values. *Additionally, we conduct a t-test for each pair comparing our method with the baseline, where the null hypothesis assumes that the sequence of sample-wise L1 error differences originates from a zero-mean Gaussian distribution. The p-value, calculated for each comparison, averages below* 0.05 *across all four sets. This statistically significant result leads to the rejection of the null hypothesis, affirming the substantial improvement of our method.* Closer to 1, the values of these metrics signify more effective regression. Our method records a slope of 0.921, a correlation coefficient of 0.744, and a coefficient of determination of 0.612, all indicating the robustness of our approach. Furthermore, our method consistently surpasses all baselines across these three metrics.

Loss function prediction

F1 scores and classification accuracy metrics are employed to assess the performance related to loss function parameters. These results are

Table 4.2: F1 score and classification accuracy for loss type prediction. Our method performs better than all the three baselines.

Method	Loss function prediction	
	F1 score ↑	Classification accuracy ↑
Random ground-truth	0.636 ± 0.017	0.716 ± 0.028
Mean/mode	0.751 ± 0.027	0.736 ± 0.056
No fingerprint	0.800 ± 0.116	0.763 ± 0.079
Using one parser	0.687 ± 0.036	0.633 ± 0.052
Ours	$\mathbf{0.813 \pm 0.019}$	$\mathbf{0.792 \pm 0.021}$

detailed in **Table 4.2**. The outcomes for the random ground-truth baseline approximate the level of random guessing. In contrast, our method significantly outshines all baselines, demonstrating superior performance in these metrics.

Practical applications of model parsing

As the inaugural proposers of the model parsing task, it's pertinent to consider *what level of performance is necessary for model parsing to be practically viable in real-world scenarios?* A plausible criterion for practical utility is an error rate below 10%. This benchmark is established by examining two highly similar generative models in our dataset, RSGAN_HALF and RSGAN_QUAR, which differ in merely 2 out of 15 parameters. This comparison suggests that an error margin under 10% is satisfactory, as it remains below the variation between these closely related models. Consequently, for model parsing to be considered effective in practical applications, we anticipate an L1 error of less than 0.1 and an F1 score exceeding 90%. Our method attains an L1 error marginally over 10% (specifically 0.14) and an F1 score around 80%, both within reasonable proximity to these set thresholds.

4.4 Proposed Method 2: Learnable Graph Pooling Network

In the preceding context, we detailed our proposed model parsing method, which harnesses the effectiveness of estimated fingerprints for each image, resulting in remarkable performance in predicting both the

loss function and architecture parameters. However, this approach primarily focuses identifying correlations among generative models (GMs) through a clustering-based method, with less emphasis on learning dependencies among the 37 hyperparameters that need to be parsed. Given that the generative model utilizes a varied set of hyperparameters jointly, and considering inherent dependencies among these hyperparameters, it is actually essential to leverage such hyperparameters dependencies for enhancing the model parsing performance.

To offer an alternative solution for model parsing, we introduce the second model parsing approach, which focuses more on capturing dependencies among various hyperparameters. Specifically, we first formulate model parsing into a graph node classification problem (See Chapter 4.4.1) and then propose a novel model parsing method called Learnable Graph Pooling Network (LGPN), which leverages the effectiveness of Graph Convolution Network (GCN) in capturing the correlation among graph nodes (Kipf and Welling, 2016; Veličković *et al.*, 2017; Fan *et al.*, 2019; Guo *et al.*, 2019; Hsu *et al.*, 2021). The architecture of LGPN is shown in Fig. 4.4, and its details are in Chapter 4.4.2.

4.4.1 Preliminary

First, we outline the process of transforming model parsing into a graph node classification task. Then, we revisit Graph Convolution Network for a comprehensive understanding of its role and relevance in our approach.

Model parsing graph formulation

First, we integrate diffusion models into the RED dataset utilized in the previous work (Asnani *et al.*, 2023b), resulting in an expanded dataset denoted as RED140, comprising 140 diverse Generative Models (GMs)[1]. Following this, we utilize the label co-occurrence pattern among training samples in RED140 to construct a directed graph, as illustrated in

[1]We adhere to the nomenclature introduced in the preceding work (Asnani *et al.*, 2023b), illustrated in the pie chart of Fig. 4.3a, where AA, AR, and NF denote Adversarial Attack models, Auto-Regressive models, and Normalizing Flow models, respectively.

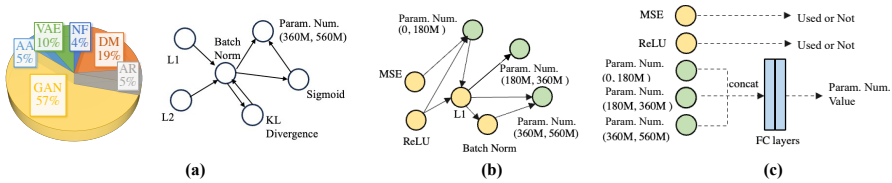


Figure 4.3: (a) We study the co-occurrence pattern among different hyperparameters in 140 different GMs, whose composition is shown as the pie chart, and subsequently construct a directed graph to capture dependencies among these hyperparameters. (b) We define the discrete-value graph node ( ) (*e.g.*, `L1` and `Batch Norm`) for each discrete hyperparameter. For each continuous hyperparameter ( ), we partition its range into n distinct intervals, and each interval is then represented by a graph node. For example, `Parameter Number` has three corresponding continuous-value graph nodes. (c) In the inference, discrete-value graph node features are used to classify if discrete hyperparameters are used in the given GM. Also, we concatenate corresponding continuous-value graph node features and regress the continuous hyperparameter value.

Fig. 4.3a. This directed graph, formed based on the label co-occurrence pattern, highlights the fundamental correlations between different categories and has proven effective in various applications with GCN-based methods (Chen *et al.*, 2019; Ye *et al.*, 2020; Nguyen *et al.*, 2021; Ding *et al.*, 2021; Tirupattur *et al.*, 2021). In our approach, this directed graph is tailored to model parsing, where we define discrete-value and continuous-value graph nodes to represent hyperparameters that require parsing, as depicted in **Fig. 4.3b**.

More formally, we first denote the conditional probability $P(L_j|L_i)$, representing the probability of hyperparameter L_j occurring when hyperparameter L_i is present. We count the occurrence of such a pair in the RED140 to retrieve the matrix $\mathbf{G} \in \mathbb{R}^{C\times C}$ (C is the hyperparameter number), and $\mathbf{G}_{ij}$ denotes the conditional probability of $P(L_j|L_i)$. Subsequently, we employ a fixed threshold τ to filter out edges with low correlations. Consequently, we obtain a directed graph $\mathbf{A} \in \mathbb{R}^{C\times C}$ where each element $\mathbf{A}_{ij}$ is a binary value indicating the presence of an edge between node i and j. In this graph data structure, we can leverage graph nodes and edges to represent hyperparameters and their dependencies, respectively.

We represent each discrete hyperparameter, such as discrete archi-

tecture parameters and objective functions, as a graph node, specifically denoted as a discrete-value graph node, as shown in Fig. 4.3b. For continuous hyperparameters, such as layer number and parameter number, we initially divide the range into n different intervals, with each interval represented by a graph node termed as a continuous-value graph node. We cast the model parsing problem as a graph node classification task. In this setup, the discrete-value graph node feature determines whether a given hyperparameter is utilized in the GM, while the continuous-value node feature indicates the range within which the hyperparameter falls. This formulation facilitates the extraction of effective representations for hyperparameters, consequently enhancing the model parsing performance.

Stack graph convolution layers

Given the directed graph denoted as $\mathbf{A}$, let's revisit the fundamental formulation of the Graph Convolution Network (GCN). In this context, we can express the stacked graph convolution operation as follows:

$$h_i^l = \mathrm{ReLU}(\sum_{j=1}^{N} \mathbf{A}_{i,j} \mathbf{W}^l h_j^{l-1} + \mathbf{b}^l), \tag{4.18}$$

h_i^l denotes the feature of the i-th node in graph $\mathbf{A}$, and $\mathbf{W}^l$ and $\mathbf{b}^l$ represent the associated weight and bias terms. It's worth noting that different nodes in the graph have different degrees, leading to significant differences in magnitudes for the aggregated node features. This variability can hinder the acquisition of an effective graph node representation, as the learning process tends to be biased towards nodes with higher degrees. To this end, we formulate the Eq. 4.18 into:

$$h_i^l = \mathrm{ReLU}(\sum_{j=1}^{N} \hat{\mathbf{A}}_{i,j} \mathbf{W}^l h_j^{l-1} / d_i + \mathbf{b}^l), \tag{4.19}$$

where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\mathbf{I}$ is an $n \times n$ identity matrix, and d_i is degree of node i, namely, $d_i = \sum_{j=1}^{n} \hat{\mathbf{A}}_{ij}$.

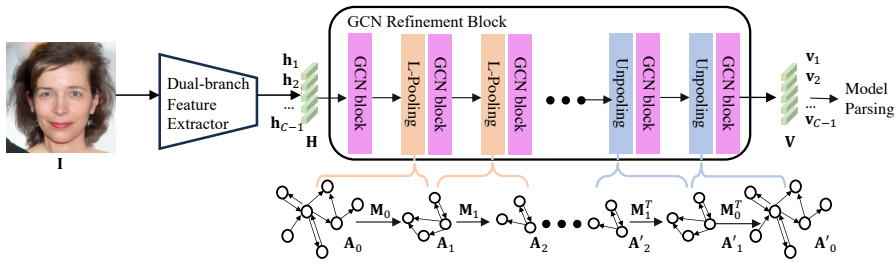


Figure 4.4: Learnable Graph Pooling Network. Initially, we employ the dual-branch feature extractor, illustrated in Fig. 4.5, to extract the feature **H** from the input image **I**. Subsequently, the obtained feature **H** undergoes refinement through the proposed GCN refinement block, which stacks different GCN layers with paired pooling-unpooling layers. The output of the GCN refinement block is the refined feature **V**, utilized for the model parsing task. Our method is trained jointly with three different objective functions.

4.4.2 Learnable Graph Pooling Network

We introduce our proposed LGPN, which contains a dual-branch feature extractor and a Graph Convolution Network (GCN) refinement block, as depicted in **Fig. 4.4**. More formally, given the input image, the dual-branch feature extractor (Fig. 4.5) learns the image representation that is then transformed into a set of graph node features. Graph node features along the pre-defined directed graph are fed into the GCN refinement block (see chapter 4.4.2).

Dual-branch feature extractor

In the field of image forensics, a common practice involves the use of specialized-design feature extractors for tasks such as distinguishing between real and CNN-generated images (Masi *et al.*, 2020; Wang *et al.*, 2020b; Schwarz *et al.*, 2021; Durall *et al.*, 2020) or identifying manipulated regions (Bayar and Stamm, 2018; Dong *et al.*, 2022; Wu *et al.*, 2019; Zhou *et al.*, 2018; Zhou *et al.*, 2017). In the context of these established approaches, our work introduces a simple yet effective feature extractor, which is structured as a dual-branch network (**Fig. 4.5**) — given the input image, we use one branch (*i.e.*, ResNet branch) to propagate the original image information, meanwhile, the other branch,

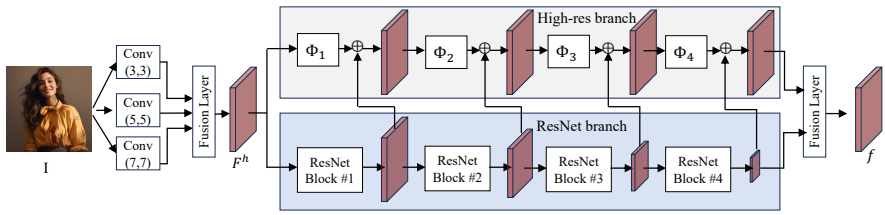


Figure 4.5: The dual-branch feature extractor. First, convolution layers with different kernel sizes extract feature maps of the input image $\mathbf{I}$. A fusion layer concatenates these feature maps and then proceeds the concatenated feature to the ResNet branch and high-res branch. ResNet branch consists of pre-trained ResNet blocks, while the high-res branch maintains the high-resolution representation to help detect generation artifacts. Lastly, output features of two branches are combined for downstream tasks, such as model parsing and CNN-generated image detection.

denoted as *high-res branch*, harnesses the high-resolution representation that helps detect high-frequency generation artifacts originating from various generative models.

Formally, for a given the image $\mathbf{I} \in \mathbb{R}^{3\times W\times H}$, we employ three distinct 2D convolution layers, each utilizing different kernel sizes (*e.g.*, 3×3, 5×5, and 7×7), to extract feature maps from $\mathbf{I}$. Subsequently, we concatenate these feature maps and pass them through a fusion layer, represented by a 1×1 convolution layer, designed for channel dimension reduction. Consequently, we obtain the feature map $\mathbf{F}^h \in \mathbb{R}^{D\times W\times H}$, sharing the same height and width as $\mathbf{I}$.

Subsequently, we feed $\mathbf{F}^h$ through the dual-branch backbone. The ResNet branch consists of four pre-trained ResNet blocks, commonly employed as a standard baseline for generalized CNN-generated image detection (Wang *et al.*, 2020b; Ojha *et al.*, 2023). Intermediate features produced by each ResNet block are upsampled and integrated into the high-res branch, as depicted in Fig. 4.5. The high-res branch is also equipped with four distinct convolution blocks (*e.g.*, Φ_b with $b \in 1\ldots 4$), which avoid operations such as 2D convolution with large strides and pooling layers that spatially downsample feature maps. The incorporation of high-resolution representation follows a similar strategy to previous works (Trinh *et al.*, 2021; Boroumand *et al.*, 2018; Liu *et al.*, 2022; Guo *et al.*, 2023b; Gragnaniello *et al.*, 2021) that also harness such potent representation for various image forensic tasks, whereas our

dual-branch architecture is distinct to their approaches.

Finally, the output feature maps from both the ResNet branch and the high-res branch are concatenated and passed through an `AVGPOOL` layer. The resulting learned representation, $\mathbf{f} \in \mathbb{R}^D$, is designed to capture the generation artifacts present in the input image $\mathbf{I}$. This learned image feature can deduce the crucial information of used GMs and benefits model parsing. We learn C independent linear layers, *i.e.,* $\Theta = \{\theta_{i=0}^{C-1}\}$ to transform $\mathbf{f}$ into a set of graph node features $\mathbf{H} = \{\mathbf{h}_0, \mathbf{h}_1, ..., \mathbf{h}_{(C-1)}\}$, which can form as a tensor, $\mathbf{H} \in \mathbb{R}^{C \times D}$. We use $\mathbf{H}$ to denote graph node features of the directed graph (*i.e.,* graph topology) $\mathbf{A} \in \mathbb{R}^{C \times C}$.

GCN refinement block

The GCN refinement block introduces a learnable pooling-unpooling mechanism that progressively transforms the original graph $\mathbf{A}_0$ into a sequence of coarsened graphs $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n$. Graph convolution is then applied to these graphs at different levels. The pooling operation involves the merging of graph nodes, facilitated by a learned matching matrix $\mathbf{M}$. Moreover, correlation matrices of different graphs, denoted as $\mathbf{A}_l$ [2], are learned using `MLP` layers. Importantly, these correlation matrices are influenced by the generative model (GM) responsible for generating the input image, underscoring the significant impact of the GM on the generation of correlation graphs.

Learnable graph pooling layer. First, $\mathbf{A}_l \in \mathbb{R}^{m \times m}$ and $\mathbf{A}_{l+1} \in \mathbb{R}^{n \times n}$ denote directed graphs at l th and $l+1$ th layers, which have m and n $(m \geq n)$ graph nodes, respectively. We use an assignment matrix $\mathbf{M}_l \in \mathbb{R}^{m \times n}$ to convert $\mathbf{A}_l$ to $\mathbf{A}_{l+1}$ as:

$$\mathbf{A}_{l+1} = {\mathbf{M}_l}^T \mathbf{A}_l \mathbf{M}_l. \tag{4.20}$$

Also, we use $\mathbf{H}_l \in \mathbb{R}^{m \times D}$ and $\mathbf{H}_{l+1} \in \mathbb{R}^{n \times D}$ to denote graph node features of $\mathbf{A}_l$ and $\mathbf{A}_{l+1}$, where each graph node feature is D dimensional. Therefore, we can use $\mathbf{M}_l$ to perform the graph node aggregation via:

$$\mathbf{H}_{l+1} = {\mathbf{M}_l}^T \mathbf{H}_l. \tag{4.21}$$

[2]Here, a graph at the l-th layer and its correlation are denoted as $\mathbf{A}_l$.

For the simplicity, we use f_{GCN} denotes the mapping function that is imposed by a GCN block which has multiple GCN layers. $\mathbf{H}_l^{in}$ and $\mathbf{H}_l^{out}$ are the input and output feature of the l th GCN blocks:

$$\mathbf{H}_l^{out} = f_{GCN}(\mathbf{H}_l^{in}). \tag{4.22}$$

The ideal scenario involves making the pooling operation and correlation among different hyperparameters dependent on the features extracted from the given input image, denoted as f. Assuming the learned feature at the l-th layer is $\mathbf{H}_l^{out}$, we employ two separate sets of weights, namely $\mathbf{W_m}$ and $\mathbf{W_a}$, to learn the matching matrix $\mathbf{M}_l$ and the adjacency matrix $\mathbf{A}_l$ at the l-th layer. This can be expressed as:

$$\mathbf{M}_{l+1} = \text{softmax}(\mathbf{W_m}\mathbf{H}_l^{out}), \tag{4.23}$$

$$\mathbf{A}_{l+1} = \text{sigmoid}(\mathbf{W_a}\mathbf{H}_l^{out}). \tag{4.24}$$

Learnable graph unpooling layer. We perform the graph unpooling operation, which restores and refines the information in the graph to its original resolution for the initial graph node classification task. Illustrated in Fig. 4.4, to eliminate confusion, we use $\mathbf{H}$ and $\mathbf{V}$ to denote the graph node features on the pooling and unpooling branches, respectively. The correlation matrix on the unpooling branch is denoted by $\mathbf{A}'$.

$$\mathbf{A}'_{l-1} = \mathbf{M}_l\mathbf{A}'_l\mathbf{M}_l^T, \tag{4.25}$$

$$\mathbf{V}_{l-1} = \mathbf{M}_l\mathbf{V}_l, \tag{4.26}$$

where $\mathbf{A}'_l$ and $\mathbf{A}'_{l-1}$ are the l th and $l-1$ th layers in the unpooling branch. In the last, we use the refined feature $\mathbf{V}_l$ for the model parsing, as depicted in Fig. 4.4.

Discussion. This learnable pooling-unpooling mechanism offers three distinct advantages. Firstly, each supernode in the coarsened graph serves as the combination of features from its corresponding child nodes, and graph convolutions on supernodes have a large receptive field for aggregating the features. Secondly, the learnable correlation adapts and models hyperparameter dependencies dynamically based on generation artifacts of the input image feature (*e.g.*, $\mathbf{H}$ or $\mathbf{V}$). Lastly, learned correlation graphs $\mathbf{A}$ vary across different levels, which effectively

addresses the issue of over-smoothing commonly encountered in GCN learning (Li *et al.*, 2019; Min *et al.*, 2020; Chen *et al.*, 2020). Therefore, through this pooling-unpooling mechanism, we are able to learn a correlation between hyperparameters that is dependent on the GM used to generate the input image, thereby enhancing the performance of model parsing.

4.4.3 Training with Multiple Loss Functions

We jointly train our approach by minimizing three distinct losses: the graph node classification loss $\mathcal{L}^{graph}$, generation artifacts isolation loss $\mathcal{L}^{iso}$ and hyperparameter hierarchy constraints $\mathcal{L}^{hier.}$. In this context, $\mathcal{L}^{graph}$ encourages each graph node feature to predict the corresponding hyperparameter label, $\mathcal{L}^{iso}$ strives to project real and fake images into two separated manifolds, aiding LGPN in exclusively parsing hyperparameters for generated images. Furthermore, $\mathcal{L}^{hier}$ enforces hierarchical constraints among different hyperparameters, contributing to the stability of the training process.

Training samples. Given a training sample denoted as $\{\mathbf{I}, \mathbf{y}\}$, in which $\mathbf{I}$ is the input image and $\mathbf{y} = \{y_0, y_1, ..., y_{(C-1)}\}$ is the corresponding annotation for parsed hyperparameters (*e.g.*, loss functions, discrete and continuous architecture parameters). Specifically, y_c is assigned as 1 if the sample is annotated with category c and 0 otherwise, where $c \in \{0, 1, ..., C-1\}$. Empirically, C is set as 55: we use 28 graph nodes to represent discrete hyperparameters (*i.e.*, 10 loss functions and 18 discrete architecture parameters), and other 27 nodes to represent 9 continuous architecture parameters.

Graph node classification loss. Given image $\mathbf{I}$, we convert the refined feature $\mathbf{V}$ into the predicted score vector, denoted as $\mathbf{s} = \{s_0, s_1, ..., s_{(C-1)}\}$. We employ the sigmoid activation to retrieve the corresponding probability vector $\mathbf{p} = \{p_0, p_1, ..., p_{(C-1)}\}$.

$$p_c = \sigma(s_c). \tag{4.27}$$

In general, cross entropy is used as the objective function, so we have:

$$\mathcal{L}^{graph} = \sum_{c=0}^{C-1} (y_c \log p_c + (1 - y_c) \log(1 - p_c)). \tag{4.28}$$

Hyperparameter hierarchy prediction. Different hyperparameters can be grouped together. For example, we group two graph nodes which represent `L1` and `MSE`, respectively, since these two hyperparameters are two pixel-level loss functions; Two nodes representing `ReLu` and `Tanh` can be merged together, because these two hyperparameters are nonlinearity functions. Therefore, we define the hyperparameter hierarchy assignment $\mathbf{M}^s$ to reflect this inherent nature. Suppose, at the layer l th, we minimize the L_2 norm of the difference between the predicted matching matrix $\mathbf{M}_l$ and ${\mathbf{M}_l}^s$.

$$\mathcal{L}^{hier} = \|{\mathbf{M}_l}^s - \mathbf{M}_l\|_2 = \sqrt{\sum_{i=0}\sum_{j=0}({m^s}_{ij} - m_{ij})^2} \tag{4.29}$$

Generation trace isolation loss. We denote the image-level binary label as y^{img} and use p^{img} to represents the probability that $\mathbf{I}$ is a generated image. Then we have:

$$\mathcal{L}^{iso} = \sum_{i=0}^{M-1}(y^{img}\log p^{img} + (1-y^{img})\log(1-p^{img})). \tag{4.30}$$

In summary, our joint training loss function can be expressed as $\mathcal{L}^{all} = \lambda_1\mathcal{L}^{graph} + \lambda_2\mathcal{L}^{hier} + \lambda_3\mathcal{L}^{iso}$, where λ_1 and λ_2 equal 0 when $\mathbf{I}$ is real.

4.4.4 Experimental Result

Each GM in RED116 and RED140 comprises $1,000$ images, resulting in a total of $116,000$ and $140,000$ generated images, respectively. GMs in these two datasets are trained on real image datasets containing various contents such as objects, handwritten digits, and human faces. Consequently, in RED140, we also incorporate real images on which these GMs are trained. We adhere to the protocol outlined in (Asnani *et al.*, 2023b), which involves the creation of 4 test sets. Each of these sets have different categories of GMs, including GAN, VAE, DM, and others. We train our model on the 104 GMs from three test sets to predict 37 hyperparameters. The evaluation is conducted on GMs in one remaining test set. The performance is averaged across four test sets, measured by F1 score and AUC for discrete hyperparameters (loss function and discrete architecture parameters) and L1 error for continuous architecture parameters.

Method	Loss Function		Dis. Archi. Para.		Con. Archi. Para.
	F1 ↑	Acc. ↑	F1 ↑	Acc. ↑	L1 error ↓
Random GT ((Asnani *et al.*, 2023b))	0.636	0.716	0.529	0.575	0.184
FEN ((Asnani *et al.*, 2023b))	0.813	0.792	0.718	0.706	0.149
FEN.* ((Asnani *et al.*, 2023b))	0.801	0.811	0.701	0.708	0.146
LGPN *w/o* GCN	0.778	0.801	0.689	0.701	0.169
LGPN *w/o* pooling	0.790	0.831	0.698	0.720	0.145
LGPN	**0.841**	**0.833**	**0.727**	**0.755**	**0.130**

(a) The model parsing performance on RED116.

Method	Loss Function		Dis. Archi. Para.		Con. Archi. Para.
	F1 ↑	Acc. ↑	F1 ↑	Acc. ↑	L1 error ↓
FEN. ((Asnani *et al.*, 2023b))	0.793	0.807	0.691	0.707	0.156
LGPN *w/o* GCN	0.766	0.778	0.657	0.674	0.159
LGPN *w/o* pooling	0.819	0.823	0.710	0.732	0.122
LGPN	**0.829**	**0.840**	**0.761**	**0.753**	**0.105**

(b) The model parsing performance on RED140.

Table 4.3: We report the performance on parsing different hyperparameters, *Loss Function* reports the averaged prediction performance on 10 objective functions. The averaged prediction performance on 18 discrete architecture parameters and 9 continuous architecture parameters are reported in *Dis. Archi. Para.* and *Con. Archi. Para.*, respectively. [**Bold**: best result; ∗ means our reproduction with the public source code.]

We report the model parsing performance in **Table 4.3**. Our proposed method significantly surpasses FEN and achieves state-of-the-art performance on both datasets. We conduct various ablations on our methods. Initially, we employ the dual-branch feature extractor with fully-connected layers for model parsing, which yields inferior performance compared to FEN (Asnani *et al.*, 2023b) in both RED116 and RED140 datasets. This suggests that a straightforward adaptation to model parsing does not yield satisfactory results. Second, we replace fully-connected layers with the GCN module without pooling, which is used to refine the learned feature for each hyperparameter. As a result, the performance increases, especially on parsing the *loss functions* (*e.g.*, 3.0% and 4.5% higher accuracy in RED116 and RED140, respectively). This observation aligns with our assertion that employing a directed graph with GCN refinement modules aids in capturing dependencies among hyperparameters. Nonetheless, the over-smoothing problem emerges when stacking numerous GCN layers, limiting the overall performance. In our complete LGPN model, this issue is miti-

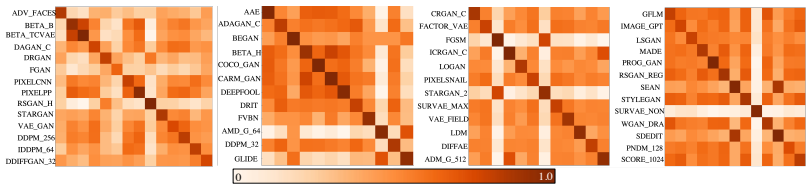


Figure 4.6: Cosine similarity between generated correlation graphs for *unseen* GMs in four test sets. Each element of these matrices is the average cosine similarities of 2,000 pairs of generated correlation graphs **A** from corresponding GMs. These matrices diagonals show two images from the same GM have highly similar correlation graphs **A**.

gated, leading to a 5.1% and 2.9% improvement in F1 score compared to the model without pooling, particularly in parsing the loss function and discrete architecture parameters.

Fig. 4.6 shows that correlation graphs generated from image pairs exhibit significant similarity when both images belong to the same "unseen" GM. This finding demonstrates that our correlation graph largely depends on the GM instead of image content, considering the different contents present in the unseen GMs from each test.

4.5 Conclusion

This chapter expands upon RED techniques, shifting focus from machine-centric to human-centric attacks, and introduces model parsing of generative models (GMs). Specifically, model parsing of GMs aims to infer the hyperparameters used in GMs, given the human-attack samples. We propose two novel methods: the two-stage model parsing network and the learnable graph pooling network, which consider dependencies among various GMs and their specific hyperparameters, respectively. Additionally, this chapter is motivated by the rapid advancements in image generation, especially with GANs and diffusion models, and the rising concerns over their misuse. Furthermore, our research addresses the challenge of model parsing, a process of reverse engineering GMs to estimate their hyperparameters based solely on generated images, a concept largely unexplored in existing literature. This endeavor enhances the understanding and analysis of GMs, adding a new layer to the field of image attribution and deepfake detection.

5

Manipulation Localization of Generated Images

Chapter overview Nowadays, individuals can easily be misled by high-quality and realistic manipulated images (*i.e.*, human-centric attack samples). Also, well-rounded RED techniques should be able to counteract such dissemination of misleading information. Therefore, this chapter delves into the research topic of *manipulation localization*, building on the recent progress in model parsing discussed in the previous chapter. Moreover, manipulation localization has been extensively studied in the Computer Vision community and aligns with the human-centric perspective inherent in the RED paradigm. In this chapter, we present passive and proactive localization schemes in Chapter 5.3 and Chapter 5.4, respectively, which can localize manipulations in image editing and digital domains.

5.1 Motivation and Background

The widespread availability of user-friendly editing software and advancements in manipulation techniques from the image editing domain, such as splicing, copy-move, and removal, has made low-cost image manipulation more accessible. Secondly, in the digital domain, the detection of manipulated facial images and videos has become a critical

concern, especially with the rising prevalence of these manipulated media in social media platforms. This growing trend underscores the urgent need for the development of effective methods that can not only detect manipulated face images but also precisely localize the manipulated regions within them. To this end, there is a pressing need for a unified algorithm capable of localizing manipulation areas in both digital and image editing domains.

In the early stage, numerous existing approaches concentrated on pixel-wise classification to detect manipulated regions in the image-editing domain (Wu *et al.*, 2019; Hu *et al.*, 2020; Zhou *et al.*, 2018; Mayer and Stamm, 2018; Chen *et al.*, 2021; Wang *et al.*, 2022; Zhou *et al.*, 2020). For instance, in a recent development, (Ji *et al.*, 2023) introduced an uncertainty-guided framework to quantify data and model uncertainties, and (Zhou *et al.*, 2023) proposed a framework applying conventional contrastive learning. Moreover, (Sun *et al.*, 2023) learns semantic-agnostic features with the help of auxiliary tasks, while (Zhai *et al.*, 2023) conducted localization in a weakly-supervised manner, requiring only binary image-level labels. Later on, the concept of localizing forgery has also been embraced in the digital facial forgery community (Zhao *et al.*, 2021; Chai *et al.*, 2020; Cozzolino *et al.*, 2018). (Dang *et al.*, 2020) employ the standard ℓ_1 loss to localize regions containing man-made artifacts. Similarly, (Zhao *et al.*, 2021) utilizes binary cross-entropy (BCE) to supervise the 4D volume in the self-consistency branch of their model. Also, (Huang *et al.*, 2022) leverages face parsing as an auxiliary function to aid in localizing edited areas. Being orthogonal to all these prior methods, we present passive and proactive localization schemes in Chapter 5.3 and Chapter 5.4, respectively, which can localize manipulations in both image editing and digital domains.

5.2 Problem Statement

The RED initiative focuses on the reverse engineering or deduction of crucial information related to deception. Aside from the model parsing, this research paradigm also contains manipulation localization, identifying specific regions tampered with in attack samples. The information from these identified regions can be employed to reverse engineer crucial

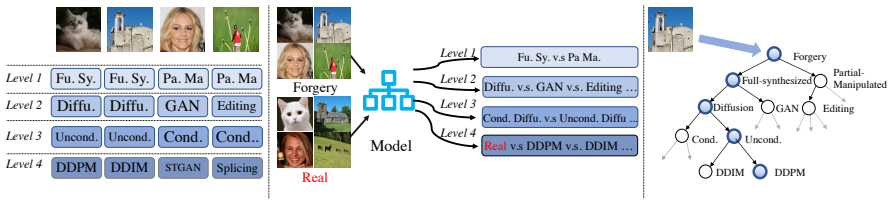


Figure 5.1: (a) We represent the forgery attribute of each manipulated image with multiple labels, at different levels. (b) For an input image, we encourage the algorithm to classify its fine-grained forgery attributes at different levels, *i.e.* a 2-way classification (fully synthesized or partially manipulated) on level 1. (c) We perform the fine-grained classification via the hierarchical nature of different forgery attributes, where each depth l node's classification probability is conditioned on classification probabilities of neighbor nodes at depth $(l-1)$. [Key: Fu. Sy.: Fully Synthesized; Pa. Ma.: Partially manipulated; Diff.: Diffusion model; Cond.: Conditional; Uncond.: Unconditional].

details about the malicious manipulation method.

More formally, given an image $\mathbf{X} \in \mathbb{R}^{W \times H \times 3}$, the proposed manipulation localization method needs to localize the manipulation region at the pixel level, performing binary segmentation and outputting a binary mask $\mathbf{M} \in \mathbb{R}^{W \times H}$, where the $\mathbf{M}_{ij}$ indicates if the ij-th pixel has been manipulated or not. In addition, leveraging the predicted $\mathbf{M}$ is a common practice to enhance the forgery detection task, in which the manipulation localization method maps $\mathbf{X}$ to a binary variable $\mathbf{y}$ that decides whether $\mathbf{X}$ has been forged or is pristine.

5.3 Passive Scheme Manipulation Localization

Developing a unified manipulation localization algorithm for both image editing and digital domains presents a significant challenge due to the substantial differences in images generated by various forgery methods, which exhibit diverse attributes. As depicted in **Fig. 5.1a**, these forgery attributes can range from indicating whether a forged image is entirely synthesized or partially manipulated, to discerning the specific forgery method used, such as a diffusion model generating images from Gaussian noise (Ho *et al.*, 2020) or an image editing process employing techniques like Poisson editing (Pérez *et al.*, 2003). To address this issue, we adopt a multi-label approach to represent the forgery attributes of each image

at different levels. Subsequently, we introduce a hierarchical fine-grained formulation for IFDL, demanding the algorithm to classify the fine-grained forgery attributes of each image across various levels, leveraging the inherent hierarchical structure of different forgery attributes. After that, a novel framework, termed Hierarchical Fine-grained Network, is proposed for image forgery detection and localization.

5.3.1 Hierarchical Fine-grained Formulation

Given an input image, our method conducts fine-grained forgery attribute classification at various levels, as depicted in **Fig. 5.1b**. This hierarchical approach proves advantageous for image-level forgery detection, as the fine-grained classification learns a comprehensive IFDL representation to differentiate individual forgery methods. Additionally, for pixel-level localization, the features from fine-grained classification serve as a valuable prior, enhancing the localization process.

In **Fig. 5.1c**, we exploit the hierarchical dependency between forgery attributes in fine-grained classification. The classification probability of each node is conditioned on the path from the root to that node. For instance, the classification probability at a node representing `DDPM` is conditioned on the classification probabilities of all nodes in the path `Forgery`→ `Fully Synthesis`→`Diffusion`→`Unconditional`→`DDPM`. Our approach differs from prior works (Wu *et al.*, 2019; Marra *et al.*, 2018; Yu *et al.*, 2019; Marra *et al.*, 2019a) that assume a "flat" structure where attributes are considered mutually exclusive. Predicting the entire hierarchical path contributes to understanding forgery attributes from coarse to fine, effectively capturing dependencies among individual forgery attributes.

5.3.2 Hierarchical Fine-grained Network

We propose Hierarchical Fine-grained Network (HiFi-Net), as depicted in **Fig. 5.2**. HiFi-Net has three components: multi-branch feature extractor, localization module and detection module. Each branch of the multi-branch extractor classifies images at one forgery attribute level. The localization module generates the forgery mask with the help of a deep-metric learning based objective. Finally, the classification

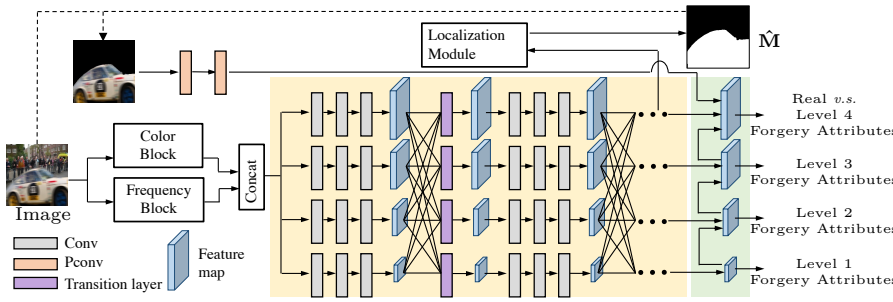


Figure 5.2: Given the input image, we first leverage color and frequency blocks to extract features. The multi-branch feature extractor ( ) learns feature maps of different resolutions, for the fine-grained classification at different levels. The localization module generates the forgery mask, $\hat{\mathbf{M}}$, to identify the manipulation region. After that, we leverage the classification module ( ) for fine-grained classifications at different levels.

module performs the forgery classification, further helping learn IFDL representations.

Multi-branch feature extractor

First, we extract features from the given input image using both color and frequency blocks. The frequency block applies a Laplacian of Gaussian (LoG) (Burt and Adelson, 1987) onto the CNN feature map. This architecture design is similar to the method in (Masi *et al.*, 2020), which exploits image generation artifacts that can exist in both RGB and frequency domain (Wang *et al.*, 2022; Dong *et al.*, 2022; Wang *et al.*, 2020b; Zhang *et al.*, 2019b). Subsequently, we propose a multi-branch feature extractor, and whose branch is denoted as θ_b with $b \in \{1 \dots 4\}$. Each θ_b generates a feature map of a specific resolution, facilitating fine-grained classification at the corresponding level. For example, for the finest level (*i.e.* identifying the individual forgery methods), one needs to model contents at all spatial locations, which requires high-resolution feature map. Conversely, low-resolution feature maps suffice for coarse-level (*i.e.* binary) classification.

Additionally, we note that different forgery methods generate manipulated areas with distinct patterns. *e.g.*, deepfake methods (Rossler *et al.*, 2019; Li *et al.*, 2020a) manipulate the whole inner part of the face,

whereas STGAN (Liu *et al.*, 2019a) changes sparse facial attributes such as mouth and eyes. Therefore, we position the localization module at the end of the highest-resolution branch of the extractor, specifically the branch responsible for classifying specific forgery methods. This design ensures that features obtained for fine-grained classification serve as a prior for localization. Such a configuration is crucial for effectively localizing manipulated regions in images manipulated using both CNNs and traditional image editing methods.

Localization module

The model contains a localization module that is optimized to match the ground-truth binary mask $\mathbf{M}$. The localization module maps feature output from the highest-resolution branch (θ_4), denoted as $\mathbb{F} \in \mathbf{R}^{512\times W\times H}$, to the mask $\hat{\mathbf{M}}$ to localize the forgery. To model the dependency and interactions of pixels on the large spatial area, the localization module employs the self-attention mechanism (Zhang *et al.*, 2019a; Wang *et al.*, 2018). As shown in the localization module architecture in the **Fig. 5.3a**.

We employ 1×1 convolution to form g, ϕ and ψ, which convert input feature $\mathbf{F}$ into $\mathbf{F}_g = g(\mathbf{F})$, $\mathbf{F}_\phi = \phi(\mathbf{F})$ and $\mathbf{F}_\psi = \psi(\mathbf{F})$. Given $\mathbf{F}_\phi$ and $\mathbf{F}_\theta$, we compute the spatial attention matrix $\mathbf{A}_s = \text{softmax}(\mathbf{F}_\phi^T \mathbf{F}_\theta)$. We then use this transformation $\mathbf{A}_s$ to map $\mathbf{F}_g$ into a global feature map $\mathbf{F}' = \mathbf{A}_s \mathbf{F}_g \in \mathbb{R}^{512\times W\times H}$.

Following the approach in (Masi *et al.*, 2020), we employ a metric learning objective function for localization, intending to create a wider margin between genuine and manipulated pixels. This design aims to enhance generalization across diverse manipulation domains by improving separation and adaptability to previously unseen domain data. Specifically, we transform $\mathbf{F}_{\text{img}}$ into a binary mask $\mathbf{M} \in \mathbf{R}^{W\times H}$ that indicates the likelihood of being manipulated at each spatial locations. Unlike the competitive approach (Wu *et al.*, 2019), which utilizes softmax regression as the classifier, our method takes a different approach. We first extract features for each pixel and subsequently characterize the geometric relationships among these acquired features using a radial decision boundary within a hyper-sphere. To achieve this, we compute

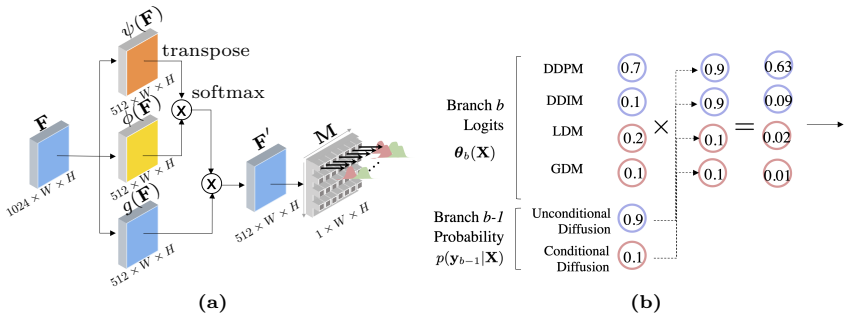


Figure 5.3: (a). The localization module adopts the self-attention mechanism to transfer the feature map $\mathbf{F}$ to the localization mask $\mathbf{M}$. (b). The classification probability output from branch θ_b depends on the predicted probability at branch θ_{b-1}, following the definition of the hierarchical forgery attributes tree.

a reference center, denoted as $\mathbf{c} \in \mathbb{R}^D$, by averaging features from all pixels in the training set's real images, where $D = 18$.

$$\mathcal{L}_{loc} = \frac{1}{HW} \sum_{i}^{H} \sum_{j}^{W} \mathcal{L}(\mathbf{F}'_{ij}, \mathbf{M}_{ij}; \mathbf{c}, \tau), \tag{5.1}$$

where:

$$\mathcal{L} = \begin{cases} \left\| \mathbf{F}'_{ij} - \mathbf{c} \right\|_2 & \text{if } \mathbf{M}_{ij} \text{ real} \\ \max\left(0, \tau - \left\| \mathbf{F}'_{ij} - \mathbf{c} \right\|_2\right) & \text{if } \mathbf{M}_{ij} \text{ forged.} \end{cases}$$

Here τ is a pre-defined margin. The first term in $\mathcal{L}$ enhances the compactness of features for real pixels, while the second term enforces a margin τ to ensure that the distribution of forged pixels is separated from real pixels. It's important to note two distinctions from previous approaches (Ruff *et al.*, 2018; Masi *et al.*, 2020): 1) we use the second term in $\mathcal{L}$ for enforcing separation, unlike (Ruff *et al.*, 2018); 2) compared to (Masi *et al.*, 2020), which operates at the image level with two margins, we tackle the more challenging pixel-level learning with a single margin, reducing the number of hyperparameters for improved simplicity.

Classification module

We aim to capture the hierarchical dependencies among various forgery attributes. Let us consider an input image $\mathbf{X}$. Denoting the output logits and predicted probability of branch θ_b as $\theta_b(\mathbf{X})$ and $p(\mathbf{y}_b|\mathbf{X})$, respectively, we can express this as follows:

$$p(\mathbf{y}_b|\mathbf{X}) \doteq \text{softmax}\left(\theta_b(\mathbf{X}) \odot (1 + p(\mathbf{y}_{b-1}|\mathbf{X}))\right) \tag{5.2}$$

Before computing the probability $p(\mathbf{y}_b|\mathbf{X})$ at branch θ_b, we scale logits $\theta_b(\mathbf{X})$ based on the previous branch probability $p(\mathbf{y}_{b-1}|\mathbf{X})$. Then, we enforce the algorithm to learn hierarchical dependency. Specifically, in Eq. (5.2), we repeat the probability of the coarse level $b-1$ for all the logits output by branch at level b, following the hierarchical structure. **Fig. 5.3b** shows that the logits associated to predicting `DDPM` or `DDIM` are multiplied by probability for the image to be `Unconditional (Diffusion)` in the last level, based on the hierarchical tree structure.

5.3.3 Training and Inference

During training, each branch is optimized for classification at its corresponding level. We employ four classification losses, denoted as $\mathcal{L}^1cls$, $\mathcal{L}^2cls$, $\mathcal{L}^3cls$, and $\mathcal{L}^4cls$ for the four branches. Specifically, at branch b, $\mathcal{L}^b_{cls}$ represents the cross-entropy between $p(\mathbf{y}_b|\mathbf{X})$ and the ground truth categorical label $\hat{\mathbf{y}}_b$. The entire architecture is trained end-to-end, with different learning rates assigned to each layer. The detailed objective function is as follows:

$$\mathcal{L}_{tot} = \begin{cases} \lambda\mathcal{L}_{loc} + \mathcal{L}^1_{cls} + \mathcal{L}^2_{cls} + \mathcal{L}^3_{cls} + \mathcal{L}^4_{cls} & \text{if } \mathbf{X} \text{ is forged} \\ \lambda\mathcal{L}_{loc} + \mathcal{L}^4_{cls} & \text{if } \mathbf{X} \text{ is real} \end{cases}$$

where $\mathbf{X}$ is the input image. When the input image is labeled as "real", we only apply the last branch (θ_4) loss function, otherwise we use all the branches. λ is the hyper-parameter that keeps $\mathcal{L}_{loc}$ on a reasonable magnitude.

In the inference, HiFi-Net generates the forgery mask from the localization module, and predicts forgery attributes at different levels. We use the output probabilities at level 4 for forgery attribute classification. For binary "forged vs. real" classification, we predict as forged if the highest probability falls in any manipulation method at level 4.

5.3.4 Experimental Results

In this chapter, we present the experimental results of our proposed method for localizing manipulation in image editing and digital domains,

Localization	Col.	Cov.	NI.16	CAS.	IM20	Avg.
	Metric: AUC(%) – Pre-trained					
ManT. ((Wu *et al.*, 2019))	82.4	81.9	79.5	81.7	74.8	80.0
SPAN ((Hu *et al.*, 2020))	93.6	92.2	84.0	79.7	75.0	84.9
PSCC ((Liu *et al.*, 2022))	98.2	84.7	85.5	82.9	80.6	86.3
Ob.Fo. ((Wang *et al.*, 2022))	95.5	92.8	**87.2**	84.3	82.1	88.3
Ours*	98.3	**93.2**	87.0	85.8	82.9	89.4
Ours	**98.4**	92.4	86.9	**86.6**	**83.4**	**89.6**

(a)

Localization	Cov.	CAS.	NI.16	Avg.
	Metric: AUC(%) / F1(%) – Fine-tuned			
SPAN ((Hu *et al.*, 2020))	93.7/55.8	83.8/40.8	96.1/58.2	91.2/51.6
PSCC ((Liu *et al.*, 2022))	94.1/72.3	87.5/55.4	**99.6**/81.9	93.7/69.8
Ob.Fo. ((Wang *et al.*, 2022))	95.7/75.8	88.2/57.9	99.6/82.4	94.5/72.0
Ours	**96.1**/**80.1**	**88.5**/**61.6**	98.9/**85.0**	**94.6**/**75.5**

(b)

Detection	AUC(%)	F1(%)
ManT. ((Wu *et al.*, 2019))	59.9	56.7
SPAN ((Hu *et al.*, 2020))	67.3	63.8
PSCC ((Liu *et al.*, 2022))	99.5	97.1
Ob.Fo. ((Wang *et al.*, 2022))	**99.7**	97.3
Ours	99.5	**97.4**

(c)

Table 5.1: (a) Localization performance of the pre-train model. (b) Localization performance of the fine-tuned model. (c) Detection performance on *CASIA* dataset. All results of prior works are ported from (Wang *et al.*, 2022). [Key: **Best**; Second Best; *Ours** uses the same pre-trained dataset as (Liu *et al.*, 2022), and *ours* is pre-trained on HiFi-IFDL].

respectively.

Image editing performance

Table 5.1 presents IFDL results for the image editing domain. The evaluation is conducted on 5 datasets: *Columbia* (Ng *et al.*, 2009), *Coverage* (Wen *et al.*, 2016), *CASIA* (Dong *et al.*, 2013), *NIST16* (*NIST: Nist nimble 2016 datasets.* 2016), and *IMD*20 (Novozamsky *et al.*, 2020). Following the previous experimental setup of (Wu *et al.*, 2019; Hu *et al.*, 2020; Liu *et al.*, 2022; Dong *et al.*, 2022; Wang *et al.*, 2022), we pre-train the model on our proposed HiFi-IFDL and then fine-tune

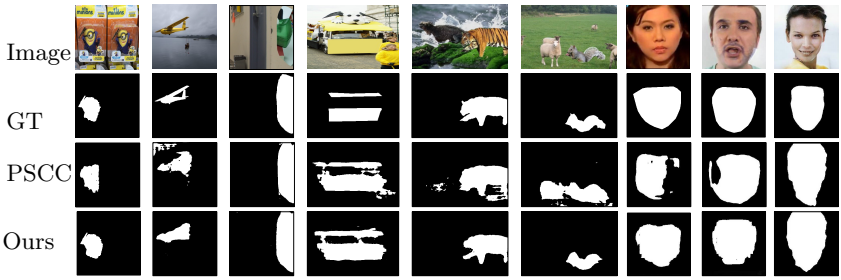


Figure 5.4: Qualitative results on different forged images. The first 6 columns are from image editing methods whereas the last 3 columns are images generated by Faceshifer (Li *et al.*, 2020a) and STGAN (Liu *et al.*, 2019a).

the pre-trained model on the *NIST16*, *Coverage* and *CASIA*. We also report the performance of HiFi-Net pre-trained on the same dataset as (Liu *et al.*, 2022). Table 5.1a reports the pre-trained model performance, in which our method achieves the best average performance. The ObjectFormer (Wang *et al.*, 2022) adopts the powerful transformer-based architecture and solely specializes in forgery detection of the image editing domain, nevertheless its performance are on-par with ours. In the fine-tune stage, our method achieves the best performance on average AUC and F1. Specifically, we only fall behind on *NIST16*, where AUC tends to saturate. We also report the image-level forgery detection results in Table 5.1c, achieving comparable results to ObjectFormer (Wang *et al.*, 2022). We show qualitative results in **Fig. 5.4**, where the manipulated region identified by our method can capture semantically meaningful object shape, such as the shapes of the tiger and squirrel.

DFFD dataset performance

We evaluate our method on the Diverse Fake Face Dataset (DFFD) (Dang *et al.*, 2020), which contains forged images created via different facial forgery methods, and real faces from FFHQ (Karras *et al.*, 2019) and CelebA (Liu *et al.*, 2015). For a fair comparison, we follow the same experiment setup and metrics: IoU and pixel-wise binary classification accuracy (PBCA) for pixel-level localization, and AUC and PBCA for image-level detection. **Table 5.2** reports that our method obtains

IoU (↑) / PBCA (↑)	Real	Fu. Syn.	Par. Man.
Att. ((Dang *et al.*, 2020))	−/**0.998**	0.847/0.847	0.401/0.786
Ours	−/0.978	**0.893**/**0.893**	**0.411**/**0.801**

(a)

IINC (↓) / C.S. (↓)	Real	Fu. Syn.	Par. Man.
Att. ((Dang *et al.*, 2020))	0.015/−	0.077/**0.095**	**0.311**/0.429
Ours	**0.010**/−	**0.060**/0.107	0.323/**0.410**

(b)

Table 5.2: The localization performance: (a) Metrics are IoU and PBCA, the higher the better, (b) Metrics are IINC and Cosine Similarity, the lower the better. [Keys: Fu. Syn.: Fully-synthesized; Par. Man.: Partially-manipulated]

competitive performance on detection and the best localization performance on partial-manipulated images. Specifically, compared to Attention Xception (Dang *et al.*, 2020), our method still achieves the more accurate localization performance on Partial Manipulated and Fully Synthesized images. For the localization performance on real images, our performance is comparable with the Attention Xception (Dang *et al.*, 2020).

5.4 Proactive Scheme Manipulation Localization

The methods previously mentioned all employ a *passive* approach, where the input image, regardless of whether it's authentic or altered, is analyzed without any preliminary modification for detection purposes. On the other hand, a *proactive* strategy has been adopted in certain computer vision tasks, which involves introducing specific signals into the original image. For instance, earlier studies have incorporated a predetermined template into real images.

Driven by the goal of enhancing the generalizability of manipulation detection/localization and inspired by proactive approaches in various tasks, we introduce a *proactive* strategy MaLP tailored for detecting image manipulations and then localize the manipulation areas using three different modules. The process is as follows: at the moment of image capture, our system uses the encryption module to embed a barely noticeable signal, referred to as a *template*, into the image,

effectively encrypting it. Should this encrypted image subsequently undergo manipulation via a Generative Model (GM), our algorithm proficiently differentiates between the original encrypted image and its altered counterpart by identifying the embedded template using a detection module. Further, MaLP specifically focuses on learning and optimizing a template. When this template is integrated into genuine images, it significantly enhances the ability to localize manipulations, should the images be altered subsequently using the localization module. Ideally, this encryption mechanism could be integrated directly into camera hardware, ensuring all captured images are protected immediately. Our methodology sets itself apart from other related proactive endeavors (Ruiz *et al.*, 2020; Yeh *et al.*, 2020; Segalis and Galili, 2020; Wang *et al.*, 2021a) in various aspects, including its primary objective (focused on detection rather than other tasks), the nature of template learning (learnable as opposed to pre-defined), the diversity of templates, and its capacity for generalization.

Developing a proactive approach for manipulation localization presents several hurdles. Firstly, crafting a methodology to learn the template in an *unsupervised* manner is a complex task. Secondly, generating a fakeness map that matches the resolution of the input image poses substantial computational challenges, particularly when determining the authenticity of each individual pixel. Previous studies (Chai *et al.*, 2020; Dang *et al.*, 2020) have attempted to address this by either reducing the image resolution or employing a patch-based strategy, but these methods often yield fakeness maps of lower accuracy and resolution. Finally, it's essential that the templates are versatile enough to accurately localize alterations made by GMs that were not seen during the training phase.

We will now discuss all the three modules of MaLP in details followed by the experiemntal results for manipulation detection and localization.

5.4.1 Method Description

MaLP is divided into three main parts:

1. **Encryption Module**: This module is responsible for encrypting real images to protect them.

2. **Localization Module**: It uses a two-branch architecture to estimate a 'fakeness map'. This map helps in identifying areas in an image that might have been manipulated or altered.

3. **Detection Module**: This component focuses on recognizing whether an image has been encrypted or tampered with. It does this by restoring any hidden patterns in the image and using the classifier from the localization module.

We will now describe each module in some detail.

Encryption module

In the method outlined by (Asnani *et al.*, 2022), a learnable template, chosen randomly from a set of templates, is applied to a real image. The intensity with which this template is applied is regulated by a hyperparameter, labeled as m. This regulation is crucial for ensuring the original quality of the image remains largely unaffected. The encryption process is summarised below:

$$\mathcal{T}(\boldsymbol{I}_j^R) = \boldsymbol{I}_j^R + m \times \boldsymbol{S}_i \text{ where } i = Rand(1, 2, ..., n). \tag{5.3}$$

We select the value of m as 30% for our framework.

The template set is refined with an emphasis on features such as minimal magnitude, the orthogonality of the templates, and the inclusion of high-frequency elements (Asnani *et al.*, 2022). The properties are applied as constraints as follows.

$$J_T = \lambda_1 \times \sum_{i=1}^{n} ||\boldsymbol{S}_i||_2 + \lambda_2 \times \sum_{\substack{i,j=1 \\ i \neq j}}^{n} \text{CS}(\boldsymbol{S}_i, \boldsymbol{S}_j) + \lambda_3 \times ||\mathcal{L}(\mathfrak{F}(\boldsymbol{S}))||_2, \tag{5.4}$$

where CS is the cosine similarity, $\mathcal{L}$ is the low-pass filter, $\mathfrak{F}$ is the fourier transform, λ_1, λ_2, λ_3 are weights for losses of low magnitude, orthogonality and high-frequency content, respectively.

Localization module

In developing the localization module, two key attributes are prioritized: an extensive receptive field for accurate fakeness map estimation, and

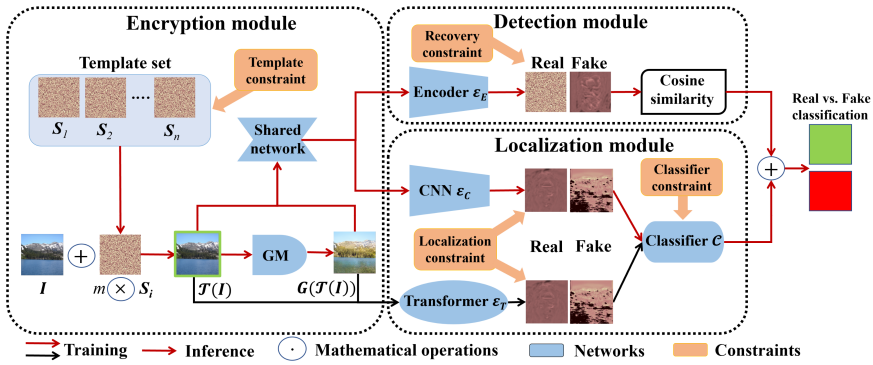


Figure 5.5: The overview of MaLP. It includes three modules: encryption, localization, and detection. We randomly select a template from the template set and add it to the real image as encryption. The GM is used in inference mode to manipulate the encrypted image. The detection module recovers the added template for binary detection. The localization module uses a two-branch architecture to estimate the fakeness map. Lastly, we apply the classifier to the fakeness map to better distinguish them from each other. Best viewed in color.

high efficiency during inference. Networks with larger receptive fields are capable of incorporating widely separated areas in the image to enhance localization accuracy. However, the drawback is that larger receptive fields usually result from deeper network architectures, which can lead to reduced speed in the inference process.

In this design, we utilize a dual-branch structure, comprising a shallow Convolutional Neural Network (CNN) and a Vision Transformer (ViT). This architecture is visually represented in **Fig. 5.5**. The rationale behind this approach is to use the shallow CNN branch to detect local features within the image, while employing the deeper ViT branch to grasp the global features. While the training phase involves both branches to enhance template learning, only the shallow CNN branch is used during inference for greater efficiency. Specifically, the shallow CNN, with its 10 layers, is adept at inference tasks but is limited to local feature detection due to its smaller receptive field. On the other hand, the ViT transformer, through its self-attention mechanism across image patches, excels at capturing global information and effectively estimating the fakeness map, even in widely separated image regions. Both networks are trained jointly.

The fakeness map estimation relies on the ground-truth fakeness map. In the case of fake images, our strategy involves elevating the agreement between the predicted and ground-truth fakeness maps. This is achieved by optimizing both the cosine similarity (CS) and the structural similarity index (SS) between these two maps. On the other hand, for encrypted images, the target fakeness map is ideally zero. To guide the predicted map towards this goal for encrypted images, an L_2 loss, referenced in (Huang *et al.*, 2022), is applied. Furthermore, to amplify the distinction between the fakeness maps of fake and encrypted images, we strive to minimize the cosine similarity between the predicted map of encrypted images and $\boldsymbol{M}_{GT}$. The localization loss is defined as:

$$J_L = \begin{cases} \Big\{\lambda_4 \times ||\mathcal{E}_{C/T}(\boldsymbol{I})||_2^2 + & \text{if } \boldsymbol{I} \in \mathcal{T}(\boldsymbol{I}^R) \\ \lambda_5 \times \mathrm{CS}(\mathcal{E}_{C/T}(\boldsymbol{I}), \boldsymbol{M}_{GT})\Big\} & \\ \Big\{\lambda_6 \times (1 - \mathrm{CS}(\mathcal{E}_{C/T}(\boldsymbol{I}), \boldsymbol{M}_{GT})) + & \text{if } \boldsymbol{I} \in G(\mathcal{T}(\boldsymbol{I}^R)) \\ \lambda_7 \times (1 - SS(\mathcal{E}_{C/T}(\boldsymbol{I}), \boldsymbol{M}_{GT}))\Big\}. & \end{cases} \tag{5.5}$$

Finally, we use a classifier to use the predicted fakeness map as features to output a binary decision of real and fake.

Detection module

We follow (Asnani *et al.*, 2022) to recover the added template from the encrypted images by maximizing the cosine similarity between $\boldsymbol{S}$ and $\boldsymbol{S}_R$. However, we minimize the cosine similarity between ($\boldsymbol{S}_R$) and $\mathcal{S}$ for manipulated images.

$$J_R = \begin{cases} \lambda_8 \times (1 - \mathrm{CS}(\boldsymbol{S}, \boldsymbol{S}_R)) & \text{if } x \in \mathcal{T}(\boldsymbol{I}^R) \\ \lambda_9 \times (\sum_{i=1}^{n} (\mathrm{CS}(\boldsymbol{S}_i, \boldsymbol{S}_R))) & \text{if } x \in G(\mathcal{T}(\boldsymbol{I}^R)). \end{cases} \tag{5.6}$$

For using the fakeness maps, we use the following equation by applying the binary cross entropy loss on the averaged logits as follows:

$$\begin{aligned} J_C = & \lambda_{10} \times -\sum_j \Big\{ y_j . \log\Big[\frac{\mathcal{C}(\boldsymbol{X}_j) + CS(\boldsymbol{S}_R, \boldsymbol{S})}{2}\Big] - \\ & (1 - y_j) . \log\Big[1 - \frac{\mathcal{C}(\boldsymbol{X}_j) + CS(\boldsymbol{S}_R, \boldsymbol{S})}{2}\Big] \Big\}, \end{aligned} \tag{5.7}$$

where y_j is the class label, $\boldsymbol{S}$ and $\boldsymbol{S}_R$ are the added and recovered template respectively.

Table 5.3: Manipulation localization comparison with prior works.

Method	Localization			Detection		
	CS ↑	PSNR ↑	SSIM ↑	Accuracy ↑	EER ↓	AUC ↑
(Dang *et al.*, 2020)	0.6230	6.214	0.2178	0.9975	**0.0050**	0.9975
(Huang *et al.*, 2022)	0.8831	22.890	**0.7876**	0.9945	0.0077	0.9998
MaLP	**0.9394**	**23.020**	0.7312	**0.9991**	0.0072	**1.0**

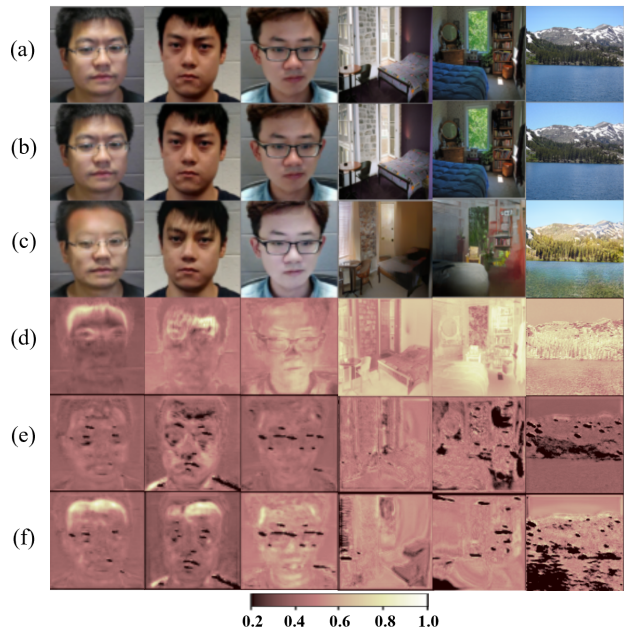


Figure 5.6: Visualization of fakeness maps for faces and generic images showing generalization across unseen attribute modifications and GMs: (a) real image, (b) encrypted image, (c) manipulated image, (d) $\boldsymbol{M}_{GT}$, (e) predicted fakeness map for encrypted images, and (f) predicted fakeness map for manipulated images. The first column shows the manipulation of (seen GM, seen attribute modification) *i.e.*(STGAN, bald). Following two columns show the manipulation of (seen GM, unseen attribute modification) *i.e.*(STGAN, [bangs, pale skin]. The fourth and fifth columns show manipulation of unseen GM, GauGAN for non-face images. The last column shows manipulation by unseen GM, DRIT. We see that the fakeness map of manipulated images is more bright and similar to $\boldsymbol{M}_{GT}$, while the real fakeness map is more close to zero. We use the cmap as "pink" to better visualize the fakeness map. All face images come from SiW-Mv2 data (Guo *et al.*, 2022).

Table 5.4: Comparison of localization performance across unseen GMs and attribute modifications. We train on STGAN bald/smile attribute modification and test on AttGAN/StyleGAN.

Method	Cosine similarity ↑(AttGAN)			Cosine similarity ↑ (StyleGAN)		
	Bald	Black Hair	Eyeglasses	Smile	Age	Gender
(Huang *et al.*, 2022)	0.8141	0.6932	0.6950	0.6176	0.3141	0.6470
MaLP	**0.8201**	**0.7940**	**0.8557**	**0.8159**	**0.8255**	**0.8016**

5.4.2 Experiments

In our study, comparisons are drawn with the methodologies presented in (Huang *et al.*, 2022) and (Dang *et al.*, 2020) concerning manipulation localization, as detailed in **Table 5.3**. The MaLP model demonstrates superior cosine similarity and maintains similar PSNR metrics for localization when contrasted with (Huang *et al.*, 2022). However, a noticeable decrease in SSIM is observed. This decline might be linked to the image quality degradation resulting from the application of our template to real images prior to manipulation. Despite this, the incorporation of the learned template effectively aids in pinpointing manipulated areas, a fact corroborated by the cosine similarity figures, though it adversely affects SSIM and PSNR values. The performance in differentiating real from fake images is also evaluated. Our proactive method shows significant superiority over passive approaches, achieving a perfect score in AUC and near-perfect accuracy. Visual demonstrations of fakeness maps for images altered by previously unseen GMs are depicted in **Fig. 5.6**. Impressively, MaLP is proficient in accurately generating fakeness maps for novel modifications and GMs across a spectrum of datasets, including both facial and generic images.

Generalization across attributes

Adhering to the experimental framework established in (Huang *et al.*, 2022), we assess MaLP's capabilities in dealing with unseen attribute alterations. MaLP is trained using the STGAN, focusing on the bald/smile attributes, and is subsequently evaluated on novel attribute modifications utilizing unseen Generative Models (GMs) like AttGAN and StyleGAN. The results, tabulated in **Table 5.4**, indicate MaLP's enhanced generalizability across various unseen attribute modifications.

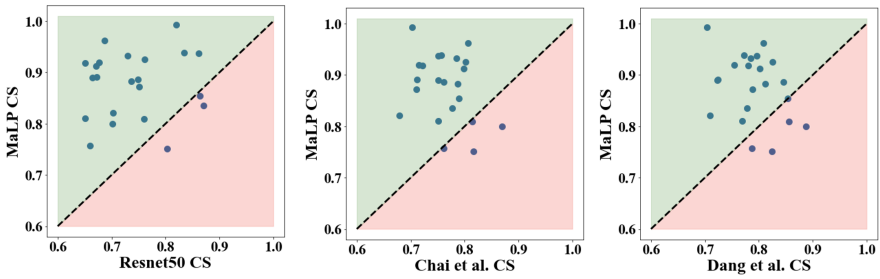


Figure 5.7: Benchmark for manipulation localization across 22 different unseen GMs, showing cosine similarity between ground-truth and predicted fakeness maps.

It's noteworthy that while AttGAN and STGAN share similar high-level structures, StyleGAN does not. This discrepancy is reflected in our findings, where a marked improvement in localization accuracy is observed with StyleGAN as compared to AttGAN. This contrast underscores a key limitation of passive methodologies, which tend to underperform when the GM used for testing does not closely resemble the GM employed during training, a challenge effectively addressed by our MaLP framework.

Generalization across GMs

While (Huang *et al.*, 2022) attempts to demonstrate generalizability across different unseen Generative Models (GMs), its scope is constrained to GMs within the same domain as the dataset used during training. In contrast, we introduce a benchmark designed to assess generalization in future manipulation localization studies, encompassing 22 diverse GMs spanning various domains. Our selection criteria for GMs include public availability and the capability for partial manipulation.

Due to the absence of an open-source implementation for (Huang *et al.*, 2022), we employ a passive approach using a ResNet50 network, as per (He *et al.*, 2016), to approximate the fakeness map, serving as our baseline for comparisons. Our approach is further benchmarked against the methodologies of (Chai *et al.*, 2020) and (Dang *et al.*, 2020). Notably, the fakeness maps generated by (Chai *et al.*, 2020) and (Dang *et al.*, 2020) are at a resolution at least 5× lower than the input images,

owing to their patch-based approach. For an equitable comparison, we upscale their predicted fakeness maps to match the resolution of $\boldsymbol{M}_{GT}$. The comparative analysis of cosine similarity is presented in **Fig. 5.7**. MaLP demonstrates superior performance over all baseline models across nearly all GMs, validating the efficacy of the proactive approach.

5.5 Conclusion

This chapter builds upon the advancements in model parsing to introduce methods for the RED paradigm in the human-centric perspective — manipulation localization. The motivation for this research topic stems from the increased accessibility of image manipulation tools and the need to detect and precisely locate manipulated regions in images, especially facial images on social media. Specifically, the problem statement revolves around developing a manipulation localization technique that performs binary segmentation on images to identify and localize manipulated regions, enhancing the ability to detect and understand image forgeries. We present both passive and proactive schemes for the manipulation localization algorithm, addressing human-centric attacks in both image editing and digital domains. We show in experiments that our two proposed methods are much more generalizable and achieve SoTA performance compared to the prior works. For the future manipulation localization research, we believe it is meaningful to explore the manipulation produced by recent prevalent image editing tools based on the diffusion model, which empirically demonstrates the unprecedented image generation quality.

6

Conclusion and Discussion

In our preceding chapters, we have extensively explored the realm of Reverse Engineering of Deceptions (RED) in the context of both machine-centric and human-centric attacks. With regard to machine-centric attacks, our efforts have successfully reverse-engineered pixel-level perturbations, adversarial saliency maps, and victim model information from adversarial examples. In the domain of human-centric attacks, we have accomplished the reverse engineering of generative model information and manipulation locations in generated images. However, the concept of RED extends far beyond these accomplishments.

In terms of machine-centric attacks, poisoning attacks, which target the training process of machine learning models, represent a significant and parallel track to adversarial attacks. These attacks aim to degrade model performance or implant backdoors, known as backdoor attacks (Gu *et al.*, 2017; Chen *et al.*, 2017). The challenge of reverse engineering such backdoors also falls within the scope of RED. Numerous studies have focused on identifying the backdoor trigger location, the trigger pattern, and the associated target label (Wang *et al.*, 2019; Wang *et al.*, 2020a; Chen *et al.*, 2022).

In addition, model inversion attacks (Fredrikson *et al.*, 2015) and

membership inference attacks (Shokri *et al.*, 2017) also serve as conduits for RED. Concerning human-centric attacks, recent advancements in model inversion techniques have concentrated on extracting training samples through generative models, including large language models (Nasr *et al.*, 2023) and diffusion models (Carlini *et al.*, 2023). Moreover, membership inference attacks, which determine whether a generated sample is part of the training set, have garnered interest within the community. These attacks can be viewed as another branch of RED (Hu and Pang, 2023; Fu *et al.*, 2023).

One important aspect of our research is the study of the model parsing ability of generative models (GMs), aiming to infer the hyperparameters behind image synthesis in these models. This approach is crucial in addressing human-centric attacks, where GMs, despite their ability to generate visually compelling images, also pose risks of misinformation and threats to the trustworthiness of social media.

Another significant aspect of our research is the focus on manipulation localization, a key area in computer vision and correlates with the human-centric attack in the RED paradigm. We aim to identify tampered regions in images to deduce crucial information about the manipulation method used. Prior research predominantly concentrated on manipulation in either the image editing or digital domain (Zhao *et al.*, 2021; Chai *et al.*, 2020; Cozzolino *et al.*, 2018; Dang *et al.*, 2020). However, in contrast to these previous efforts, we introduce passive and proactive algorithms for manipulation localization that are capable of handling both domains simultaneously, demonstrating better generalization across unknown attacks.

With the rapid advancement of generative AI technologies like ChatGPT and Bard, coupled with increasing concerns over security risks posed by potential adversarial attacks, the importance of RED remains paramount. We believe that a comprehensive understanding of RED is crucial for the development of safer AI systems, offering insights into protective measures against emerging threats in this dynamic field.

References

Andriushchenko, M., F. Croce, N. Flammarion, and M. Hein. (2020). "Square attack: a query-efficient black-box adversarial attack via random search". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII*. Springer. 484–501.

Asnani, V., X. Yin, T. Hassner, S. Liu, and X. Liu. (2022). "Proactive Image Manipulation Detection". In: *CVPR*.

Asnani, V., X. Yin, T. Hassner, and X. Liu. (2023a). "Malp: Manipulation localization using a proactive scheme". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12343–12352.

Asnani, V., X. Yin, T. Hassner, and X. Liu. (2023b). "Reverse engineering of generative models: Inferring model hyperparameters from generated images". *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Athalye, A., L. Engstrom, A. Ilyas, and K. Kwok. (2018). "Synthesizing Robust Adversarial Examples". In: *International Conference on Machine Learning (ICML)*.

Batina, L., S. Bhasin, D. Jap, and S. Picek. (2019). "{CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel". In: *28th USENIX Security Symposium (USENIX Security 19)*. 515–532.

Bayar, B. and M. C. Stamm. (2018). "Constrained convolutional neural networks: A new approach towards general purpose image manipulation detection". *IEEE Transactions on Information Forensics and Security*. 13(11): 2691–2706.

Boopathy, A., S. Liu, G. Zhang, P.-Y. Chen, S. Chang, and L. Daniel. (2020). "Visual Interpretability Alone Helps Adversarial Robustness".

Boroumand, M., M. Chen, and J. Fridrich. (2018). "Deep residual network for steganalysis of digital images". *IEEE Transactions on Information Forensics and Security*. 14(5): 1181–1193.

Burgess, C. P., I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner. (2017). "Understanding disentangling in β-VAE". In: *NeurIPS*.

Burt, P. J. and E. H. Adelson. (1987). "The Laplacian pyramid as a compact image code". In: *Readings in computer vision*. Elsevier. 671–679.

Carlini, N. and D. Wagner. (2017). "Towards evaluating the robustness of neural networks". In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE.

Carlini, N., J. Hayes, M. Nasr, M. Jagielski, V. Sehwag, F. Tramer, B. Balle, D. Ippolito, and E. Wallace. (2023). "Extracting training data from diffusion models". In: *32nd USENIX Security Symposium (USENIX Security 23)*. 5253–5270.

Chai, L., D. Bau, S.-N. Lim, and P. Isola. (2020). "What makes fake images detectable? Understanding properties that generalize". In: *ECCV*.

Chen, D., Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. (2020). "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. No. 04. 3438–3445.

Chen, R. T. Q., X. Li, R. Grosse, and D. Duvenaud. (2018). "Isolating Sources of Disentanglement in Variational Autoencoders". In: *NeurIPS*.

Chen, T., Z. Zhang, Y. Zhang, S. Chang, S. Liu, and Z. Wang. (2022). "Quarantine: Sparsity can uncover the trojan attack trigger for free". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 598–609.

Chen, X., C. Dong, J. Ji, J. Cao, and X. Li. (2021). "Image manipulation detection by multi-view multi-scale supervision". In: *ICCV*.

Chen, X., C. Liu, B. Li, K. Lu, and D. Song. (2017). "Targeted backdoor attacks on deep learning systems using data poisoning". *arXiv preprint arXiv:1712.05526*.

Chen, Z.-M., X.-S. Wei, P. Wang, and Y. Guo. (2019). "Multi-label image recognition with graph convolutional networks". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 5177–5186.

Choi, Y., M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo. (2018). "StarGAN: Unified generative adversarial networks for multi-domain image-to-image translation". In: *CVPR*.

Cozzolino, D., J. Thies, A. Rössler, C. Riess, M. Nießner, and L. Verdoliva. (2018). "Forensictransfer: Weakly-supervised domain adaptation for forgery detection". *arXiv preprint arXiv:1812.02510*.

Creswell, A., T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath. (2018). "Generative adversarial networks: An overview". *IEEE signal processing magazine*. 35(1): 53–65.

Croce, F. and M. Hein. (2020). "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks". In: *International Conference on Machine Learning (ICML)*. PMLR.

Dang, H., F. Liu, J. Stehouwer, X. Liu, and A. K. Jain. (2020). "On the detection of digital face manipulation". In: *CVPR*.

DARPA. (2021). "Reverse Engineering of Deceptions". https://www.darpa.mil/program/reverse-engineering-of-deceptions.

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. (2009). "ImageNet: A large-scale hierarchical image database". In: *CVPR*.

Deng, L. (2012). "The MNIST database of handwritten digit images for machine learning research [best of the web]". *Signal Processing Magazine*. 29(6): 141–142.

Dhariwal, P. and A. Nichol. (2021). "Diffusion models beat gans on image synthesis". In:

Ding, H., H. Zhang, J. Liu, J. Li, Z. Feng, and X. Jiang. (2021). "Interaction via bi-directional graph of semantic region affinity for scene parsing". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 15848–15858.

Dong, C., X. Chen, R. Hu, J. Cao, and X. Li. (2022). "MVSS-Net: Multi-View Multi-Scale Supervised Networks for Image Manipulation Detection". *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Dong, J., W. Wang, and T. Tan. (2013). "Casia image tampering detection evaluation database". In: *2013 IEEE China Summit and International Conference on Signal and Information Processing*. IEEE. 422–426.

Durall, R., M. Keuper, and J. Keuper. (2020). "Watch your up-convolution: Cnn based generative deep neural networks are failing to reproduce spectral distributions". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 7890–7899.

Fan, L., S. Liu, P.-Y. Chen, G. Zhang, and C. Gan. (2021). "When Does Contrastive Learning Preserve Adversarial Robustness from Pretraining to Finetuning?" *Advances in Neural Information Processing Systems*. 34.

Fan, W., Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin. (2019). "Graph neural networks for social recommendation". In: *The world wide web conference*. 417–426.

Frankle, J. and M. Carbin. (2018). "The lottery ticket hypothesis: Finding sparse, trainable neural networks". *arXiv preprint arXiv:1803.03635*.

Fredrikson, M., S. Jha, and T. Ristenpart. (2015). "Model inversion attacks that exploit confidence information and basic countermeasures". In: *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 1322–1333.

Fu, W., H. Wang, C. Gao, G. Liu, Y. Li, and T. Jiang. (2023). "Practical Membership Inference Attacks against Fine-tuned Large Language Models via Self-prompt Calibration". *arXiv preprint arXiv:2311.06062*.

Goebel, M., J. Bunk, S. Chattopadhyay, L. Nataraj, S. Chandrasekaran, and B. Manjunath. (2021). "Attribution of gradient based adversarial attacks for reverse engineering of deceptions". *arXiv preprint arXiv:2103.11002*.

Gong, Y., Y. Yao, Y. Li, Y. Zhang, X. Liu, X. Lin, and S. Liu. (2022). "Reverse engineering of imperceptible adversarial image perturbations". *arXiv preprint arXiv:2203.14145*.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. (2014a). "Generative adversarial nets". In: *NeurIPS*.

Goodfellow, I. J., J. Shlens, and C. Szegedy. (2014b). "Explaining and harnessing adversarial examples". *arXiv preprint arXiv:1412.6572*.

Gragnaniello, D., D. Cozzolino, F. Marra, G. Poggi, and L. Verdoliva. (2021). "Are GAN generated images easy to detect? A critical analysis of the state-of-the-art". In: *2021 IEEE international conference on multimedia and expo (ICME)*. IEEE. 1–6.

Gu, T., B. Dolan-Gavitt, and S. Garg. (2017). "Badnets: Identifying vulnerabilities in the machine learning model supply chain". *arXiv preprint arXiv:1708.06733*.

Guarnera, L., O. Giudice, and S. Battiato. (2020). "DeepFake Detection by Analyzing Convolutional Traces". In: *CVPR Workshops*.

Guo, X., V. Asnani, S. Liu, and X. Liu. (2023a). "Tracing Hyperparameter Dependencies for Model Parsing via Learnable Graph Pooling Network". *arXiv preprint arXiv:2312.02224*.

Guo, X., X. Liu, Z. Ren, S. Grosz, I. Masi, and X. Liu. (2023b). "Hierarchical Fine-Grained Image Forgery Detection and Localization". In: *In Proceeding of IEEE Computer Vision and Pattern Recognition*.

Guo, X., Y. Liu, A. Jain, and X. Liu. (2022). "Multi-domain Learning for Updating Face Anti-spoofing Models". In: *ECCV*.

Guo, Z., Y. Zhang, and W. Lu. (2019). "Attention Guided Graph Convolutional Networks for Relation Extraction". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 241–251.

Guo, Z., K. Han, Y. Ge, W. Ji, and Y. Li. (2023c). "Scalable Attribution of Adversarial Attacks via Multi-Task Learning". *arXiv preprint arXiv:2302.14059*.

Han, S., J. Pool, J. Tran, and W. Dally. (2015). "Learning both weights and connections for efficient neural network". *Advances in neural information processing systems*. 28.

He, K., X. Zhang, S. Ren, and J. Sun. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Heath, V. (2019). "From a Sleazy Reddit Post to a National Security Threat: A Closer Look at the Deepfake Discourse". In: *Disinformation and Digital Democracies in the 21st Century*. The NATO Association of Canada.

Ho, J., A. Jain, and P. Abbeel. (2020). "Denoising diffusion probabilistic models". *Advances in Neural Information Processing Systems*. 33: 6840–6851.

Hsu, I., X. Guo, P. Natarajan, N. Peng, *et al.* (2021). "Discourse-level relation extraction via graph pooling". In: *AAAI DLG Wrokshop*.

Hu, H. and J. Pang. (2023). "Membership inference of diffusion models". *arXiv preprint arXiv:2301.09956*.

Hu, X., Z. Zhang, Z. Jiang, S. Chaudhuri, Z. Yang, and R. Nevatia. (2020). "SPAN: spatial pyramid attention network for image manipulation localization". In: *European Conference on Computer Vision*. Springer. 312–328.

Hua, W., Z. Zhang, and G. E. Suh. (2018). "Reverse engineering convolutional neural networks through side-channel information leaks". In: *Proceedings of the 55th Annual Design Automation Conference*. 1–6.

Huang, Y., F. Juefei-Xu, Q. Guo, Y. Liu, and G. Pu. (2022). "FakeLocator: Robust localization of GAN-based face manipulations". *IEEE Transactions on Information Forensics and Security*. 17: 2657–2672.

Ilyas, A., L. Engstrom, A. Athalye, and J. Lin. (2018). "Black-box Adversarial Attacks with Limited Queries and Information". *arXiv preprint arXiv:1804.08598*.

Jabbar, A., X. Li, and B. Omar. (2020). "A Survey on Generative Adversarial Networks: Variants, Applications, and Training". *arXiv preprint arXiv:2006.05132*.

Ji, K., F. Chen, X. Guo, Y. Xu, J. Wang, and J. Chen. (2023). "Uncertainty-guided Learning for Improving Image Manipulation Detection". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 22456–22465.

Jourabloo, A., Y. Liu, and X. Liu. (2018). "Face de-spoofing: Anti-spoofing via noise modeling". In: *ECCV*.

Karras, T., T. Aila, S. Laine, and J. Lehtinen. (2018). "Progressive growing of GANs for improved quality, stability, and variation". In: *ICLR*.

Karras, T., S. Laine, and T. Aila. (2019). "A style-based generator architecture for generative adversarial networks". In: *CVPR*. 4401–4410.

Kingma, D. P. and M. Welling. (2014). "Auto-Encoding Variational Bayes". In: *ICLR*.

Kingma, D. P. and J. Ba. (2015). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*.

Kipf, T. N. and M. Welling. (2016). "Semi-supervised classification with graph convolutional networks". *arXiv preprint arXiv:1609.02907*.

Krizhevsky, A., G. Hinton, *et al.* (2009). "Learning multiple layers of features from tiny images".

LeCun, Y., Y. Bengio, and G. Hinton. (2015). "Deep learning". *nature*. 521(7553): 436–444.

Li, G., M. Muller, A. Thabet, and B. Ghanem. (2019). "Deepgcns: Can gcns go as deep as cnns?" In: *Proceedings of the IEEE/CVF international conference on computer vision*. 9267–9276.

Li, L., J. Bao, H. Yang, D. Chen, and F. Wen. (2020a). "Faceshifter: Towards high fidelity and occlusion aware face swapping". *CVPR*.

Li, Q., Y. Guo, and H. Chen. (2020b). "Practical no-box adversarial attacks against DNNs". *Advances in Neural Information Processing Systems (NeurIPS)*.

Liao, F., M. Liang, Y. Dong, T. Pang, X. Hu, and J. Zhu. (2018). "Defense against Adversarial Attacks Using High-Level Representation Guided Denoiser". *arXiv:1712.02976 [cs]*. May. (Accessed on 05/26/2021).

Liu, C., B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. (2018). "Progressive neural architecture search". In: *ECCV*.

Liu, M., Y. Ding, M. Xia, X. Liu, E. Ding, W. Zuo, and S. Wen. (2019a). "Stgan: A unified selective transfer network for arbitrary image attribute editing". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 3673–3682.

Liu, S., P.-Y. Chen, X. Chen, and M. Hong. (2019b). "signSGD via Zeroth-Order Oracle". In: *International Conference on Learning Representations*.

Liu, S., P.-Y. Chen, B. Kailkhura, G. Zhang, A. O. Hero III, and P. K. Varshney. (2020). "A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications". *IEEE Signal Processing Magazine*. 37(5): 43–54.

Liu, X., Y. Liu, J. Chen, and X. Liu. (2022). "PSCC-Net: Progressive spatio-channel correlation network for image manipulation detection and localization". *IEEE Transactions on Circuits and Systems for Video Technology*.

Liu, Z., P. Luo, X. Wang, and X. Tang. (2015). "Deep learning face attributes in the wild". In: *Proceedings of the IEEE international conference on computer vision*. 3730–3738.

Luo, Y., X. Boix, G. Roig, T. Poggio, and Q. Zhao. (2015). "Foveation-based mechanisms alleviate adversarial examples". *arXiv preprint arXiv:1511.06292*.

Madry, A., A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. (2017). "Towards deep learning models resistant to adversarial attacks". *arXiv preprint arXiv:1706.06083*.

Maini, P., X. Chen, B. Li, and D. Song. (2021). "Perturbation Type Categorization for Multiple ℓ_p Bounded Adversarial Robustness". URL: https://openreview.net/forum?id=Oe2XI-Aft-k.

Marra, F., D. Gragnaniello, D. Cozzolino, and L. Verdoliva. (2018). "Detection of gan-generated fake images over social networks". In: *2018 IEEE conference on multimedia information processing and retrieval (MIPR)*. IEEE. 384–389.

Marra, F., D. Gragnaniello, L. Verdoliva, and G. Poggi. (2019a). "Do gans leave artificial fingerprints?" In: *IEEE conference on multimedia information processing and retrieval (MIPR)*.

Marra, F., C. Saltori, G. Boato, and L. Verdoliva. (2019b). "Incremental learning for the detection and classification of GAN-generated images". In: *WIFS*.

Masi, I., A. Killekar, R. M. Mascarenhas, S. P. Gurudatt, and W. AbdAlmageed. (2020). "Two-branch recurrent network for isolating deepfakes in videos". In: *ECCV*. Springer.

Mayer, O. and M. C. Stamm. (2018). "Learned forensic source similarity for unknown camera models". In: *ICASSP*.

McCloskey, S. and M. Albright. (2019). "Detecting GAN-generated imagery using saturation cues". In: *ICIP*.

Min, Y., F. Wenkel, and G. Wolf. (2020). "Scattering gcn: Overcoming oversmoothness in graph convolutional networks". *Advances in neural information processing systems*. 33: 14498–14508.

Moayeri, M. and S. Feizi. (2021). "Sample efficient detection and classification of adversarial attacks via self-supervised embeddings". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 7677–7686.

Nasr, M., N. Carlini, J. Hayase, M. Jagielski, A. F. Cooper, D. Ippolito, C. A. Choquette-Choo, E. Wallace, F. Tramèr, and K. Lee. (2023). "Scalable Extraction of Training Data from (Production) Language Models". *arXiv preprint arXiv:2311.17035*.

Ng, T.-T., J. Hsu, and S.-F. Chang. (2009). "Columbia image splicing detection evaluation dataset". *DVMM lab. Columbia Univ CalPhotos Digit Libr*.

Nguyen, B. X., B. D. Nguyen, T. Do, E. Tjiputra, Q. D. Tran, and A. Nguyen. (2021). "Graph-based person signature for person re-identifications". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 3492–3501.

Nicholson, D. A. and V. Emanuele. (2023). "Reverse engineering adversarial attacks with fingerprints from adversarial examples". *arXiv preprint arXiv:2301.13869*.

Nie, W., B. Guo, Y. Huang, C. Xiao, A. Vahdat, and A. Anandkumar. (2022). "Diffusion models for adversarial purification". *arXiv preprint arXiv:2205.07460*.

Nirkin, Y., L. Wolf, Y. Keller, and T. Hassner. (2020). "DeepFake detection based on the discrepancy between the face and its context". *arXiv preprint arXiv:2008.12262*.

NIST: Nist nimble 2016 datasets. (2016). URL: https://www.nist.gov/itl/iad/mig/,.

Niu, Z., Z. Chen, L. Li, Y. Yang, B. Li, and J. Yi. (2020). "On the Limitations of Denoising Strategies as Adversarial Defenses". *arXiv:2012.09384 [cs]*.

Novozamsky, A., B. Mahdian, and S. Saic. (2020). "IMD2020: A Large-Scale Annotated Dataset Tailored for Detecting Manipulated Images". In: *2020 IEEE Winter Applications of Computer Vision Workshops (WACVW)*. 71–80.

Oh, S. J., B. Schiele, and M. Fritz. (2019). "Towards reverse-engineering black-box neural networks". *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*: 121–144.

Ojha, U., Y. Li, and Y. J. Lee. (2023). "Towards universal fake image detectors that generalize across generative models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 24480–24489.

Pang, R., X. Zhang, S. Ji, X. Luo, and T. Wang. (2020). "AdvMind: Inferring Adversary Intent of Black-Box Attacks". In: *the International Conference on Knowledge Discovery & Data Mining (KDD)*.

Pérez, P., M. Gangnet, and A. Blake. (2003). "Poisson image editing". In: *ACM SIGGRAPH 2003 Papers*. 313–318.

Pham, H., M. Guan, B. Zoph, Q. Le, and J. Dean. (2018). "Efficient neural architecture search via parameters sharing". In: *ICML*.

Rossler, A., D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner. (2019). "Faceforensics++: Learning to detect manipulated facial images". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 1–11.

Ruff, L., R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft. (2018). "Deep one-class classification". In: *ICML*. 4393–4402.

Ruiz, N., S. A. Bargal, and S. Sclaroff. (2020). "Disrupting deepfakes: Adversarial attacks against conditional image translation networks and facial manipulation systems". In: *ECCV*.

Sabour, S., Y. Cao, F. Faghri, and D. J. Fleet. (2015). "Adversarial manipulation of deep representations". *arXiv preprint arXiv:1511.05122*.

Salman, H., M. Sun, G. Yang, A. Kapoor, and J. Z. Kolter. (2020). "Denoised smoothing: A provable defense for pretrained classifiers". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Schwarz, K., Y. Liao, and A. Geiger. (2021). "On the frequency bias of generative models". *Advances in Neural Information Processing Systems*. 34: 18126–18136.

Segalis, E. and E. Galili. (2020). "OGAN: Disrupting Deepfakes with an Adversarial Attack that Survives Training". *arXiv preprint arXiv:2006.12247*.

Selvaraju, R. R., M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. (2020). "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization". *International Journal of Computer Vision*.

Shafahi, A., W. R. Huang, C. Studer, S. Feizi, and T. Goldstein. (2020). "Are adversarial examples inevitable?" *arXiv:1809.02104 [cs, stat]*. Feb. (Accessed on 05/26/2021).

Shi, C., C. Holtz, and G. Mishne. (2021). "Online adversarial purification based on self-supervision". *arXiv preprint arXiv:2101.09387*.

Shokri, R., M. Stronati, C. Song, and V. Shmatikov. (2017). "Membership inference attacks against machine learning models". In: *2017 IEEE symposium on security and privacy (SP)*. IEEE. 3–18.

Simonyan, K. and A. Zisserman. (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations (ICLR)*.

Souri, H., P. Khorramshahi, C. P. Lau, M. Goldblum, and R. Chellappa. (2021). "Identification of Attack-Specific Signatures in Adversarial Examples". *arXiv preprint arXiv:2110.06802*.

Srinivasan, V., C. Rohrer, A. Marban, K.-R. Müller, W. Samek, and S. Nakajima. (2021). "Robustifying models against adversarial attacks by langevin dynamics". *Neural Networks*. 137: 1–17.

Sun, Z., H. Jiang, D. Wang, X. Li, and J. Cao. (2023). "SAFL-Net: Semantic-Agnostic Feature Learning Network with Auxiliary Plugins for Image Manipulation Detection". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 22424–22433.

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. (2015). "Rethinking the Inception Architecture for Computer Vision". *CoRR*.

Tan, M., B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. (2019). "MnasNet: Platform-aware neural architecture search for mobile". In: *CVPR*.

Thaker, D., P. Giampouras, and R. Vidal. (2022). "Reverse Engineering ℓ_p attacks: A block-sparse optimization approach with recovery guarantees". In: *International Conference on Machine Learning*. PMLR. 21253–21271.

Tirupattur, P., K. Duarte, Y. S. Rawat, and M. Shah. (2021). "Modeling multi-label action dependencies for temporal action localization". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1460–1470.

Tramer, F., N. Carlini, W. Brendel, and A. Madry. (2020). "On adaptive attacks to adversarial example defenses". *arXiv preprint arXiv:2002.08347*.

Tramèr, F., F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. (2016). "Stealing machine learning models via prediction {APIs}". In: *25th USENIX security symposium (USENIX Security 16)*. 601–618.

Trinh, L., M. Tsang, S. Rambhatla, and Y. Liu. (2021). "Interpretable and Trustworthy Deepfake Detection via Dynamic Prototypes". In: *WACV*. 1973–1983.

Veličković, P., G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. (2017). "Graph attention networks". *arXiv preprint arXiv:1710.10903*.

Vinyals, O., C. Blundell, T. Lillicrap, D. Wierstra, *et al.* (2016). "Matching networks for one shot learning". *Advances in neural information processing systems*. 29.

Waldemarsson, C. (2020). *Disinformation, Deepfakes & Democracy; The European response to election interference in the digital age*. The Alliance of Democracies Foundation.

Wang, B. and N. Z. Gong. (2018). "Stealing hyperparameters in machine learning". In: *2018 IEEE symposium on security and privacy (SP)*. IEEE. 36–52.

Wang, B., Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. (2019). "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 707–723.

Wang, J., Z. Wu, J. Chen, X. Han, A. Shrivastava, S.-N. Lim, and Y.-G. Jiang. (2022). "Objectformer for image manipulation detection and localization". In: *CVPR*. 2364–2373.

Wang, R., G. Zhang, S. Liu, P.-Y. Chen, J. Xiong, and M. Wang. (2020a). "Practical detection of trojan neural networks: Data-limited and data-free cases". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII 16*. Springer. 222–238.

Wang, R., F. Juefei-Xu, M. Luo, Y. Liu, and L. Wang. (2021a). "FakeTagger: Robust Safeguards against DeepFake Dissemination via Provenance Tracking". In: *ACMM*.

Wang, S.-Y., O. Wang, R. Zhang, A. Owens, and A. A. Efros. (2020b). "CNN-generated images are surprisingly easy to spot... for now". In: *CVPR*. 8695–8704.

Wang, X., R. Girshick, A. Gupta, and K. He. (2018). "Non-local neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7794–7803.

Wang, X., Y. Li, C.-J. Hsieh, and T. C. M. Lee. (2023). "CAN MACHINE TELL THE DISTORTION DIFFERENCE? A REVERSE ENGINEERING STUDY OF ADVERSARIAL ATTACKS". URL: https://openreview.net/forum?id=NdFKHCFxXjS.

Wang, Z., Q. She, and T. E. Ward. (2021b). "Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy". *ACM Computing Surveys*. 54(2).

Wen, B., Y. Zhu, R. Subramanian, T.-T. Ng, X. Shen, and S. Winkler. (2016). "COVERAGE—A novel database for copy-move forgery detection". In: *2016 IEEE international conference on image processing (ICIP)*. IEEE. 161–165.

Wong, E., L. Rice, and J. Z. Kolter. (2020). "Fast is better than free: Revisiting adversarial training". In: *International Conference on Learning Representations (ICLR)*.

Wu, Y., W. AbdAlmageed, and P. Natarajan. (2019). "Mantra-net: Manipulation tracing network for detection and localization of image forgeries with anomalous features". In: *CVPR*. 9543–9552.

Xie, C., Z. Zhang, Y. Zhou, S. Bai, J. Wang, Z. Ren, and A. L. Yuille. (2019). "Improving transferability of adversarial examples with input diversity". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Xu, K., S. Liu, P. Zhao, P.-Y. Chen, H. Zhang, Q. Fan, D. Erdogmus, Y. Wang, and X. Lin. (2019). "Structured Adversarial Attack: Towards General Implementation and Better Interpretability". In: *International Conference on Learning Representations (ICLR)*.

Ye, J., J. He, X. Peng, W. Wu, and Y. Qiao. (2020). "Attention-driven dynamic graph convolutional network for multi-label image recognition". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXI 16*. Springer. 649–665.

Yeh, C.-Y., H.-W. Chen, S.-L. Tsai, and S.-D. Wang. (2020). "Disrupting image-translation-based deepfake algorithms with adversarial attacks". In: *WACVW*.

Yoon, J., S. J. Hwang, and J. Lee. (2021). "Adversarial purification with score-based generative models". In: *International Conference on Machine Learning*. PMLR. 12062–12072.

Yu, N., L. S. Davis, and M. Fritz. (2019). "Attributing fake images to GANs: Learning and analyzing GAN fingerprints". In: *ICCV*. 7556–7566.

Yun, S., D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. (2019). "Cutmix: Regularization strategy to train strong classifiers with localizable features". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 6023–6032.

Zhai, Y., T. Luan, D. Doermann, and J. Yuan. (2023). "Towards Generic Image Manipulation Detection with Weakly-Supervised Self-Consistency Learning". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 22390–22400.

Zhang, H., I. Goodfellow, D. Metaxas, and A. Odena. (2019a). "Self-attention generative adversarial networks". In: *International conference on machine learning*. PMLR. 7354–7363.

Zhang, K., W. Zuo, Y. Chen, D. Meng, and L. Zhang. (2017). "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising". *IEEE transactions on image processing*. 26(7): 3142–3155.

Zhang, X., S. Karaman, and S.-F. Chang. (2019b). "Detecting and simulating artifacts in gan fake images". In: *2019 IEEE international workshop on information forensics and security (WIFS)*. IEEE. 1–6.

Zhao, T., X. Xu, M. Xu, H. Ding, Y. Xiong, and W. Xia. (2021). "Learning self-consistency for deepfake detection". In: *CVPR*.

Zhou, J., X. Ma, X. Du, A. Y. Alhammadi, and W. Feng. (2023). "Pre-training-free Image Manipulation Localization through Non-Mutually Exclusive Contrastive Learning". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 22346–22356.

Zhou, M. and V. M. Patel. (2022). "On Trace of PGD-Like Adversarial Attacks". *arXiv preprint arXiv:2205.09586*.

Zhou, P., B.-C. Chen, X. Han, M. Najibi, A. Shrivastava, S.-N. Lim, and L. Davis. (2020). "Generate, segment, and refine: Towards generic manipulation segmentation". In: *AAAI*.

Zhou, P., X. Han, V. I. Morariu, and L. S. Davis. (2017). "Two-stream neural networks for tampered face detection". In: *2017 IEEE conference on computer vision and pattern recognition workshops (CVPRW)*. IEEE. 1831–1839.

Zhou, P., X. Han, V. I. Morariu, and L. S. Davis. (2018). "Learning rich features for image manipulation detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1053–1061.

Zhu, J.-Y., T. Park, P. Isola, and A. A. Efros. (2017). "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks". In: *ICCV*.