Hugo Prieto Tárrega

Feasibility of reconstruction attacks on Deep Neural Networks when parts of the model are exposed to attackers

Computer Security Engineering and Artificial Intelligence

Final Master Project

Directed by
Dr. Marc Sánchez Artigas

UNIVERSITAT ROVIRA i VIRGILI

Tarragona, 2025

Abstract

Protecting deep neural network (DNN) inference with Trusted Execution Environments (TEEs) is often limited by resource constraints, resulting in only partial model protection and leaving some layers exposed. This thesis investigates the feasibility of reconstruction attacks under such conditions, where an adversary exploits exposed intermediate representations to recover original input data. Using a ResNet-50 image recognition model trained by EMBL for off-sample image recognition as a case study, we evaluate how partial TEE protection might be undermined.

We implement both black-box and white-box attack strategies: a black-box approach leveraging generative inversion techniques and a white-box approach that trains inverted neural network models for each targeted "victim" layer. We assess their effectiveness by measuring the visual similarity of reconstructed images to the original training samples, the computational overhead, measured as training time, required for the inversion models, and the fidelity of classification on both in-sample and off-sample data compared to the original ResNet-50. Our methodology-driven evaluation highlights the security risks introduced by partially protected DNNs and aims to inform the design of more robust privacy-preserving techniques in deep learning.

Resum

La protecció de la inferència de xarxes neuronals profundes (DNN) mitjançant Entorns d'Execució Confiables (TEE) sovint es veu limitada per restriccions de recursos, cosa que comporta una protecció parcial del model i l'exposició d'algunes capes. Aquest treball de fi de màster investiga la viabilitat dels atacs de reconstrucció en aquestes condicions, on un adversari pot aprofitar les representacions intermèdies exposades per recuperar les dades d'entrada originals. Com a cas d'estudi, s'utilitza un model de reconeixement d'imatges ResNet-50 entrenat per l'EMBL per a la classificació d'imatges off-sample, per avaluar com es pot veure compromesa la protecció parcial amb TEE.

S'implementen estratègies d'atac tant de tipus black-box com white-box: en el primer cas, emprant tècniques d'inversió generativa; en el segon, entrenant models neuronals invertits per a cada capa "víctima" del model atacat. L'eficàcia dels atacs s'avalua mitjançant la similitud visual entre les imatges reconstruïdes i les mostres originals del conjunt d'entrenament, el cost computacional mesurat en temps d'entrenament dels models d'inversió, i la fidelitat en la classificació tant de dades in-sample com off-sample en comparació amb el ResNet-50 original. Aquesta avaluació basada en la metodologia posa en relleu els riscos de seguretat derivats de la protecció parcial de DNNs i pretén contribuir al disseny de tècniques de preservació de la privacitat més robustes en l'àmbit de l'aprenentatge profund.

Contents

List of code snippets

List of Figures

List of Tables

1 Introduction

1.1 Problem Context

Privacy preservation in deep neural network (`DNN`) inference has become a critical concern as machine learning services handle increasingly sensitive data. In `Machine Learning as a Service (MLaaS)` scenarios, users send private inputs, such as personal images, medical records, or other confidential data, to cloud-hosted models for prediction. This is especially common in **biomedical and healthcare domains**, where DNNs analyze sensitive patient information such as medical images or genomic data, to assist in diagnosis or research. Ensuring the **confidentiality** of such inputs and any derived data is critical, not only to maintain user trust, but also to meet strict regulatory requirements like privacy laws and medical data regulations. In essence, as DNN-powered services become ubiquitous in sensitive applications, protecting user data privacy during inference is both a fundamental necessity and a challenging technical problem.

One promising approach to safeguard data during DNN processing is the use of **Trusted Execution Environments (TEEs)**. TEEs like `Intel SGX`, `ARM TrustZone`, and others provide **hardware-isolated enclaves** where code and data can execute shielded from the rest of the system. When a DNN model runs inside a TEE, the idea is that both the model's parameters and the user's input remain encrypted or inaccessible to the outside world, even if the underlying system, OS or hypervisor, is compromised. In theory, this *confidential computing* model allows a cloud server to perform neural network inference on sensitive data without ever exposing that data or the model's inner workings to cloud administrators or attackers. For example, an enclave can decrypt and process a medical image entirely within protected memory, outputting only the final prediction to the user. This concept has attracted significant attention as it directly addresses the privacy issue in MLaaS by establishing a trusted region of computation within an untrusted infrastructure.

1.2 Motivation

Deep neural networks (DNNs) are increasingly deployed in cloud and edge computing environments to provide AI-driven services on resource-limited devices. Offloading intensive computations to powerful cloud/edge servers enables complex tasks, like image recognition or natural language processing, to run on smartphones, IoT gadgets, and other endpoints with limited hardware. However, this convenience comes with serious security and **privacy risks** when external servers handle sensitive data or computations, since studies have shown that those intermediate *feature maps* can still carry sensitive information. Adversaries may exploit this leakage to infer or reconstruct confidential user inputs from the data flowing through the network.

One approach to mitigate these threats is to use **Trusted Execution Environments (TEEs)**, which isolate computations from the host system to protect sensitive data and models during inference on untrusted infrastructure. In principle, running an entire DNN inside a TEE can keep inputs and model parameters confidential. In

practice, however, fully enclave-protected inference incurs significant performance overhead. Therefore, real-world deployments often adopt partially protected models. In these designs, only certain *critical* layers of the neural network execute inside the TEE. This setup leverages the TEE's security for the most sensitive computations, but uses conventional high-performance processors, such as GPUs, for the rest of the model to meet latency and throughput requirements.

This partial-protection approach still leaves a vulnerability. The inference pipeline is split between trusted and untrusted parts, so some intermediate outputs (the feature maps from the TEE-protected portion) must be revealed to the outside environment to complete the computation. The privacy problem addressed in this thesis is that an adversary who can observe these intermediate features, and possibly also knows the model's architecture, could potentially reconstruct the original input data despite the enclave's presence. In our threat model, the attacker cannot directly access the raw input or fully breach the TEE; instead, they exploit the exposed feature maps, leveraging knowledge of the model to invert its computations and recover private inputs. Such *reconstruction attacks* are a form of inference attack that aim to recover sensitive user information from a model's internal outputs.

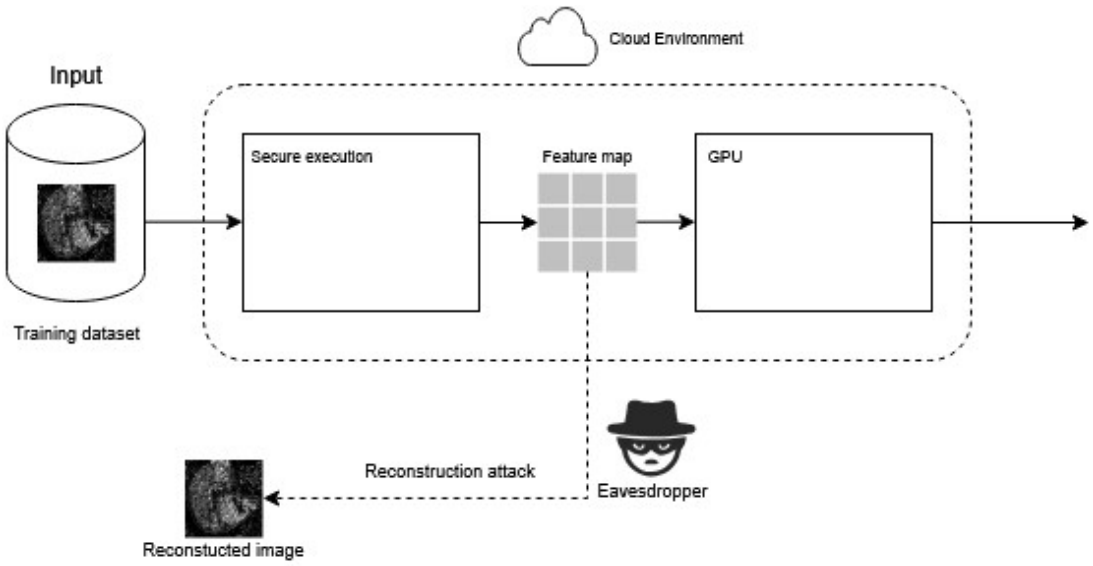


Figure 1: Reconstruction attack context

A further challenge in this context is the widespread use of **transfer learning** in AI development. Models are often not trained from scratch; instead, developers fine-tune **pre-trained networks**, such as ResNet, to adapt them to specific tasks. This means that the early layers of a model deployed in a cloud-edge scenario might closely resemble those of a well-known public model. Consequently, an attacker can exploit prior knowledge of common feature representations or even deploy generative models trained on public data to assist in reconstructing the input. In essence, transfer learning can inadvertently make it **easier to invert intermediate features**, since the adversary may possess a similar feature extractor or have access to the same general data distribution as the model's pre-training.

This master's thesis is carried out within the framework of **CloudSkin**, a €3.4 million European Union project that brings together the Universitat Rovira i Virgili (**URV**) and the European Molecular Biology Laboratory (**EMBL**) among other partners. The work specifically focuses on a CloudSkin use case involving EMBL's **Metaspace** platform; an open, cloud-based platform for sharing and annotating spatial metabolomics data. Within this context, the thesis explores research approaches to protect highly sensitive biomedical data, ensuring that such information can be processed and analyzed securely in distributed computing environments.

The thesis is developed in collaboration with **Cloudlab**, URV's research group specializing in the modeling, design, implementation, and analysis of parallel and distributed systems and algorithms in the Cloud-Edge continuum. Cloudlab has been instrumental by providing expert guidance, a strong research environment, substantial computing resources, and additional funding to scale experiments using external computing power for training and testing.

1.3 Objectives

The core objective of this thesis is to determine whether **reconstruction attacks** can successfully recover original training images from a **partially protected** `ResNet-50` model, thereby evaluating the **privacy risks** of such partial protection. In particular, we aim to apply and compare multiple model inversion techniques, including a state-of-the-art generative inversion method in the **black-box setting** (the `Ginver` approach, notable for requiring no auxiliary dataset [1]) as well as a custom **architecture-informed** inversion method, to attempt the reconstruction of the model's training data.

We will then assess the **feasibility** of these attacks in practical terms, analyzing both their **computational requirements** (time and resources) and the **monetary cost** of execution (given that experiments are run on rented `cloud machines`). Another key objective is to evaluate the **fidelity of reconstructed images** by measuring how closely they resemble the original training inputs, using image similarity metrics such as `Mean Squared Error (MSE)` and `Structural Similarity Index (SSIM)` [8].

Additionally, we will examine whether the reconstructed images can be recognized by the original model as valid "*on-sample*" inputs, for example by checking if a reconstructed image is classified correctly (on vs. off-sample) and analyzing the results with `confusion matrices`. By meeting these objectives, the research will gauge the severity of **privacy leakage** when only partial model protection is in place (since exposing intermediate features can enable input reconstruction) and will thereby contribute insights toward developing more **resilient privacy-preserving techniques** in `deep learning`.

Key Objectives

- Investigate the feasibility of **reconstruction attacks** on a `partially protected ResNet-50` model trained for off-sample image recognition.
- Apply and compare two attack strategies:
  - **Black-box approach:** An inversion attack where the adversary does not use any knowledge of the victim layer's internal structure. Instead, a generic inverse architecture (the **Ginver** style generic method) is applied uniformly to any target layer's output, treating the layer as an opaque module. [1].
  - **White-box approach:** An inversion attack that leverages architectural knowledge of the target network. The adversary constructs a layer-specific `mirrored` decoder for each victim layer, exploiting the known operations of that layer to improve reconstruction.

  In both scenarios, our attack network uses as input the feature map that comes out of the last layer of the target network found within the TEE.
- Evaluate the **computational cost and time** required to train inversion models on cloud infrastructure like AWS.
- Analyze the **monetary cost** of reconstruction attacks based on cloud resource usage.
- Assess the **quality of reconstructed images** to the original training data using metrics like `MSE` and `SSIM` [8].
- Examine whether reconstructed inputs can be classified as *on-sample* by the original model using `confusion matrices` and `ROC curves`.
- Quantify the **privacy leakage** associated with exposing intermediate feature maps and contribute insights for more **privacy-resilient model protection**.

1.4 Problem Statement

Although from a theoretical point of view, the combined use of TEEs and MLaaS is a solution to the security problems of handling sensitive data in insecure environments, such as running in the cloud, it has certain **practical limitations**. Real-world DNN models are **resource-intensive**, often requiring hundreds of megabytes of memory and substantial computational power, especially for deep convolutional networks or transformers. `Intel SGX` enclaves, for instance, have severe memory constraints; only a limited amount of memory can be used securely before incurring costly paging to unprotected memory. Executing an entire deep model inside such an enclave can be orders of magnitude slower due to these memory limitations and the lack of hardware acceleration. In fact, modern DNN inference typically relies on GPUs or TPUs for speed, but these accelerators cannot directly operate on enclave-protected data, since current GPUs do not support working inside SGX enclaves. The net effect is that a naive solution, running the full neural network inside the TEE, often fails to meet the latency and performance requirements of real-time services. Researchers have observed that purely enclave-based DNN inference can be many times slower than normal execution, which is impractical for deployment. This performance gap has driven

the community to explore **partial protection strategies**, where only a portion of the model or computation is secured by the TEE while the rest runs in the open to leverage accelerators and avoid TEE bottlenecks [12].

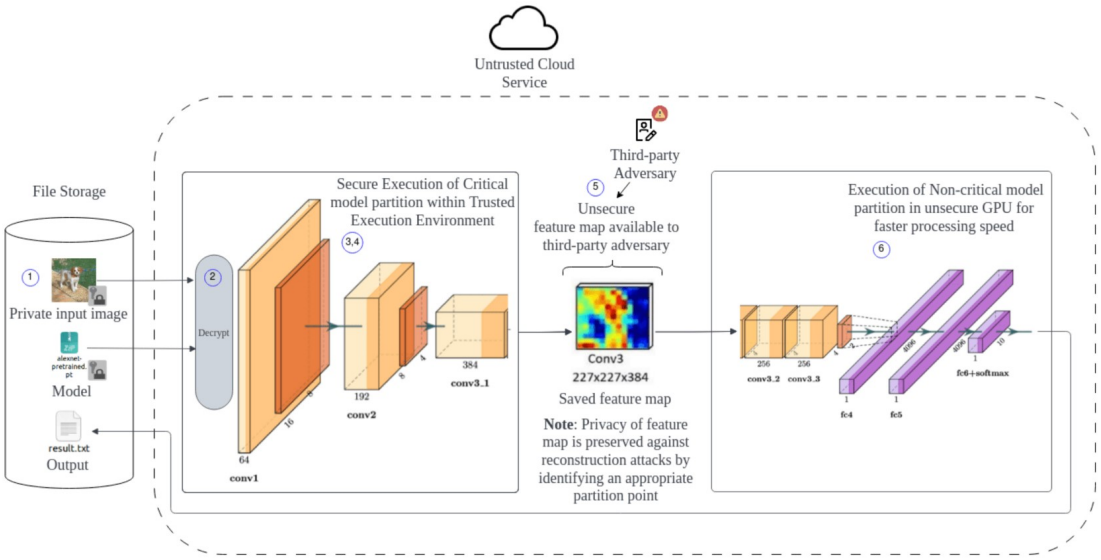


Figure 2: Model partitioning with TEEs

Such **partitioned execution** of DNNs aims to balance security and efficiency. In a typical partially protected setup, the model is split into two parts: a **trusted part** that runs inside the enclave, and an **untrusted part** that runs on the regular host, often using GPU acceleration. For instance, the initial layers of a neural network might execute inside SGX or directly on the end device to process the raw input securely, and then the intermediate result, the *feature map* or activation output from the enclave, is passed to the remaining layers running outside the enclave on the GPU. This division is designed so that the most sensitive computation, often the handling of raw input data, remains protected, while the heavy computation of matrix multiplications and convolutions can be offloaded to faster hardware. Several **state of the art systems** employ this approach. For example, `Slalom` and `Origami Inference` partition neural network inference between an enclave and an external device. These solutions demonstrate that judiciously splitting the model can dramatically improve performance, sometimes achieving an order of magnitude speedup over enclave only execution and make private ML inference more practical. However, this improved efficiency comes at a price: **parts of the model's execution are no longer fully protected**, which means some data is exposed in plaintext outside the TEE.

The **exposure of intermediate representations**, such as feature vectors or activation maps, in a partially protected model presents a serious privacy vulnerability. Even though this intermediate data is a transformed version of the original input, it can still carry a lot of information about that input. An adversary who gains access to the feature maps that leave the enclave, for example, a malicious cloud administrator or any compromised software on the host, could attempt to **reconstruct the original input** from these features. This form of attack is commonly known as a **model inversion** or **reconstruction attack**. The adversary leverages knowledge of the DNN

model and the leaked intermediate features to reverse engineer an approximation of the input data. Alarmingly, **recent research has shown** that high fidelity reconstructions are often possible, especially when the exposed features come from early or mid-layer outputs of the network [8]. The intuition is that early network layers, like the first few convolutional layers of a vision model, retain *low-level structures*, such as edges, textures or simple shapes of the input image, while later layers capture more *abstract semantics*. Thus, if an attacker can intercept the output of an early layer that has been left unencrypted, they can exploit it to reveal visual or textual content that should have remained confidential.

State of the art attack techniques use advanced *generative models* to accomplish this inversion. Notably, **conditional Generative Adversarial Networks (cGANs)** have been employed as powerful reconstructors: the attacker trains a generative network that takes the intermediate feature map as input and learns to produce a plausible original input that would yield those features when fed through the remaining model. Using such techniques, researchers have demonstrated the recovery of recognizable images from internal feature vectors. For example, in a collaborative inference setup, (a form of split model execution between client and server, a method called `Ginver` showed that a cGAN-based attacker could accurately recover private images from intermediate features shared with a server [1]. These findings underscore that the **threat is not merely theoretical**; unless the model is partitioned at a very deep layer or the features are somehow sanitized, an attacker with access to the "partial results" of a neural network can invert those results to obtain sensitive input data. In the context of a biomedical application, this could mean reconstructing a patient's MRI scan or DNA profile from what was assumed to be a safely transformed intermediate, defeating the purpose of using the enclave.

Because of these risks, the **current threat models** for TEE-backed ML inference explicitly consider an *"honest but curious"* adversary who can observe any data outside the enclave. The server hosting the enclave might faithfully execute the protocol, thus being "honest" in operation, but is "curious" in that it will analyze any exposed computations to glean private information. Under this threat model, **intermediate feature exposure is essentially a new attack surface**. Reconstruction attacks via model inversion are a primary concern, as described, but they are not the only potential issue, an attacker might also attempt to infer specific attributes of the input, for example inferring someone's race or health condition from features, or even perform **membership inference** to tell if a certain data point was part of the model's training set [8]. However, inversion attacks are particularly devastating since they directly compromise input privacy by producing a version of the actual data. **Defenses that rely solely on "the data is partial or encrypted" are insufficient** if that partial data remains rich enough to reconstruct the original. Thus, simply assuming later layers' features are "safe" to reveal can be dangerous without thorough verification.

In summary, while **Trusted Execution Environments** provide a crucial tool for protecting DNN computations, their practical use in large-scale machine learning is limited by performance and resource constraints. This has led to architectures where **only part of a model is protected**, inevitably exposing intermediate results to

less secure environments. The **problem context** here is that this necessary compromise opens the door to **privacy breaches**: the exposed features may leak sensitive information through reconstruction attacks and related inference techniques. Consequently, there is a **pressing need for research** that addresses this gap; how can we enjoy the benefits of TEEs (strong data confidentiality during computation) without inadvertently giving adversaries an avenue to steal private data from intermediate representations? The rationale for this thesis is grounded in this challenge. It is imperative to understand the extent of information leakage from DNN feature maps and to develop methods to mitigate it, so that we can achieve both **high performance and robust privacy**. By thoroughly exploring the interplay between enclave based protections and model inversion threats, the thesis aims to advance the state of the art in **privacy-preserving deep learning**, ensuring that sensitive data in MLaaS and biomedical applications remains truly confidential throughout the inference process.

1.5 Scope and limitations

1.5.1 Scope

This work addresses the problem domain of **input reconstruction attacks** on `partially protected neural networks`. In particular, it focuses on scenarios where a neural network's inference is split between a **secure environment** and an **unsecured one**, leaving the model only partially protected. The analysis is centered on a `ResNet-50` convolutional neural network provided by **EMBL**, which was trained for a specialized *off-sample image classification* task. We evaluate **reconstruction attacks** on this model under both **black-box** conditions, where the adversary lacks any internal details and relies only on the observed layer output, treating it as an opaque module) [1], and a **white-box** scenario, where the adversary also has full knowledge of the model's architecture.

All attacks are applied in a controlled experimental setting using this known `ResNet50` architecture and a specific dataset supplied by `EMBL` for the off-sample classification problem.

For evaluation metrics, we employ `Structural Similarity Index (SSIM)` and `Mean Squared Error (MSE)` to quantify the fidelity of reconstructed inputs. We also measure the **computational cost** and **inference time** of the attacks, and examine the model's classification performance on data with same distribution as training dataset, with `on sample vs off sample` original classification problem. The implementation is carried out using the `PyTorch` deep learning framework and executed on `Amazon Web Services (AWS)` infrastructure, and we perform `grid search` to tune key hyperparameters of the attack methods.

The overarching objective of this work is to assess the **practical viability** of these reconstruction attacks and the **privacy implications** of performing only partial neural network inference inside a `Trusted Execution Environment` from a theoretical standpoint. In essence, we investigate whether using a **hardware-based TEE** (such as Intel SGX) to secure part of the model's inference can effectively protect against input reconstruction, or if sensitive information can still leak to an adversary [12].

1.5.2 Limitations

Despite its contributions, this study has several **limitations** that must be acknowledged. **Generality of results** is limited, the findings are strictly valid for the `ResNet-50` model and the specific dataset provided by `EMBL`, and may not directly generalize to other neural network architectures or data domains. The **adversarial scenarios** considered are also constrained: we analyze only straightforward *model inversion (input reconstruction)* attacks in white-box and black-box settings, without exploring more **adaptive adversaries** who could tailor their strategies in response to defenses. Likewise, other categories of privacy attacks, such as *membership inference* or *attribute inference* attacks, are beyond the scope of this work, meaning our evaluation does not cover those potential threats.

There are also **practical and infrastructural limitations** to note. The experiments were bounded by resource constraints, for instance, training and extensive experimentation had to be curtailed due to the **computational cost** of using `AWS` cloud instances, which may have limited the thoroughness or scale of our evaluations. Additionally, the dataset used in this research is **closed/proprietary** (not publicly available), which poses challenges for **reproducibility**. As a result, external researchers cannot exactly replicate our experiments without access to the same data, potentially hindering independent verification of the results.

1.6 Access to the project

Source code, evaluation artifacts and datasets can be accessed via the following stable URL:

Feasibility of reconstruction attacks on Deep Neural Networks

2 State of the art

2.1 Related work

Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures – *Matt Fredrikson, Somesh Jha, Thomas Ristenpart (2015, ACM CCS).* This foundational paper introduced **model inversion attacks**, showing that an attacker with `black-box access` (only model outputs and confidence scores) can reconstruct sensitive features of training data. Notably, they demonstrated extracting recognizable face images from a face recognition model by leveraging the model's confidence values [4], highlighting a new privacy threat in `ML-as-a-service` settings.

Adversarial Neural Network Inversion via Auxiliary Knowledge Alignment – *Ziqi Yang, Ee-Chien Chang, Zhenkai Liang (2019, arXiv preprint & ACM CCS).* This work targets `black-box` inversion in an adversarial setting by training a secondary *inversion network* to act as the inverse of the target model. The inversion model is trained on an auxiliary public dataset and partial prediction outputs of the target. The approach showed that even without access to the original training data, one can accurately recover input images by treating the target `DNN` as an encoder and learning a decoder that reconstructs inputs from its outputs [14].

Model Inversion Attacks Against Collaborative Inference – *Zecheng He, Tianwei Zhang, Ruby B. Lee (2019, ACSAC).* Focusing on `split neural networks`, this paper revealed that when a `DNN` is partitioned between a client and server, a malicious participant can **reconstruct the client's input** from intermediate features [6]. They show that even without access to the other party's model layers, the attacker's half leaks enough information to recover high-fidelity inputs.

Deep Leakage from Gradients – *Ligeng Zhu, Zhijian Liu, Song Han (2019, NeurIPS).* This influential work demonstrated that sharing gradient updates in collaborative learning can lead to **input reconstruction**. The authors introduced `DLG`, which optimizes dummy inputs to match shared gradients, successfully recovering training data from one gradient snapshot [18].

Inverting Gradients – How Easy Is It to Break Privacy in Federated Learning? – *Jonas Geiping et al. (2020, NeurIPS).* Building on `DLG`, this work improved inversion fidelity via a `gradient matching loss` and adversarial regularizations [5]. They showed that `white-box access` to gradients in federated learning, even when averaged across clients, can compromise privacy.

The Secret Revealer: Generative Model-Inversion Attacks Against Deep Neural Networks – *Yuheng Zhang et al. (2020, CVPR).* This paper pioneered the use of `GANs` for model inversion. It proposed `GMI` (Generative Model Inversion): training a `GAN` on public data and optimizing in its latent space to match the target model's output [16]. This method increased inversion accuracy significantly and showed that even high-performing models are vulnerable.

Knowledge-Enriched Distributional Model Inversion Attacks (KED-MI) *– Si Chen et al. (2021, ICCV).* This paper introduced `KED-MI`, a conditional `GAN` trained using *soft labels* from the target model. By modeling the class distribution explicitly, it enhanced the diversity and fidelity of reconstructions [3], achieving a 150% improvement over prior state-of-the-art.

Variational Model Inversion Attacks *– Kuan-Chieh Wang et al. (2021, NeurIPS).* This work reformulates model inversion as a `variational inference` problem. It introduces `VMI`, which uses a probabilistic decoder trained on public data and optimizes a variational lower bound, balancing fidelity and diversity in reconstructions [13].

Exploiting Explanations for Model Inversion Attacks *– Xuejun Zhao et al. (2021, ICCV).* This work revealed that `explainability tools` (e.g., `saliency maps, attention`) can amplify inversion attacks. Their `XAI-aware` inversion models used Grad-CAM and attention transfer to improve input reconstructions [17].

Re-Thinking Model Inversion Attacks Against Deep Neural Networks *– Ngoc-Bao Nguyen et al. (2023, CVPR).* This paper improved inversion attack fidelity by refining `identity loss` and introducing `model augmentation` to avoid overfitting. It achieved over 90% accuracy on CelebA, showing SOTA performance [9].

Label-Only Model Inversion Attacks via Knowledge Transfer (LOKT) *– Ngoc-Bao Nguyen et al. (2023, NeurIPS).* `LOKT` handles the strict `label-only` setting by generating labeled pseudo-samples with an `ACGAN` trained via queries to the target. The surrogate model then enables white-box inversion [10].

Ginver: Generative Model Inversion Attacks Against Collaborative Inference *– Yupeng Yin et al. (2023, WWW).* `Ginver` targets `split inference` and reconstructs inputs directly from intermediate activations in a single forward pass using a trained `c-GAN` [15]. The defnition of the white-box and black-box scenarios differ from those defined on this project.

Model Inversion Attacks Through Target-Specific Conditional Diffusion Models *– Ouxiang Li et al. (2024, arXiv).* `Diff-MI` employs `diffusion models` as priors for inversion. By conditioning a diffusion model on the target model's class and refining predictions iteratively, this method achieved SOTA reconstruction fidelity and diversity [7].

2.2 Main Contributions

In contrast to most existing **Model Inversion Attack (MIA)** studies that use standard public benchmarks, such as `CIFAR-10` or `FaceScrub`, this thesis evaluates inversion attacks on a **proprietary industrial dataset of pharmaceutical images**,

such as those from AstraZeneca. These data are **sensitive and domain-specific**, involving drug compound or medical imaging information, and are *not publicly available*. Demonstrating inversion attacks on such datasets highlights practical **privacy risks** in real-world industrial contexts. By working with **real confidential data**, this research reveals challenges and adversarial behaviors, including non-trivial data distributions and feature complexity, that are absent in synthetic or toy datasets. This contribution bridges the gap between prior academic evaluations and **realistic deployment scenarios** in healthcare and drug development, underscoring the importance of testing privacy attacks on **genuine confidential data** [4, 16].

In addition, we implement and compare two complementary **attack strategies**. First, we reproduce a **Ginver-style black-box generative inversion attack**. In this approach, a neural *inverse model* is trained to map intermediate-layer outputs back to input images, enabling **one-shot reconstruction** without per-sample optimization [15]. Second, we design a **custom white-box reconstruction method** that leverages knowledge of the full ResNet-50 architecture. Concretely, we attach an *inverted* or mirror decoder network to the protected model: given the known architecture of the partial ResNet-50, the decoder is trained to invert its feature outputs into images; similar in spirit to shadow networks and decoders in prior work [9, 17]. By evaluating both black-box (generative) and white-box (decoder-based) inversion under identical conditions, we isolate the impact of **model knowledge** on attack effectiveness and show how generative priors compare with direct reconstruction.

Finally, we conduct a **detailed empirical study** of attack runtime and computational cost as a function of the layer-splitting point. While past works [6] [12]) observed that deeper splits, keeping more layers on the client, worsen reconstruction quality, they generally did not quantify the **performance overheads**. In this thesis, we **systematically measure the latency and compute resources** required for inversion attacks at each potential split layer of ResNet-50. This analysis reveals practical **trade-offs**: for instance, deeper local segments produce larger intermediate features and require longer forward/backward passes, increasing attack latency; in contrast, shallower splits reduce computation but may leak more information. These results extend prior privacy-efficiency studies, such as PATROL [12], by explicitly quantifying the **attack-side cost** of inverting different layers. The findings provide actionable insights for both defenders and attackers regarding the **computational consequences of model partitioning**.

3 Technical background

3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning architectures designed to process data with a grid like topology, such as images or audio spectrograms. CNNs are on of the most important approaches in `image processing` and computer vision tasks because of their ability to automatically learn spatial hierarchies of features from raw pixel data [16].

The main purpose of CNNs is to efficiently extract and combine local features to form abstract representations with great performance in tasks such as `image classification`, `object detection`, and `semantic segmentation`. Unlike traditional fully connected neural networks, CNNs exploit spatial locality through convolutional operations, reducing the number of parameters and improving generalization.

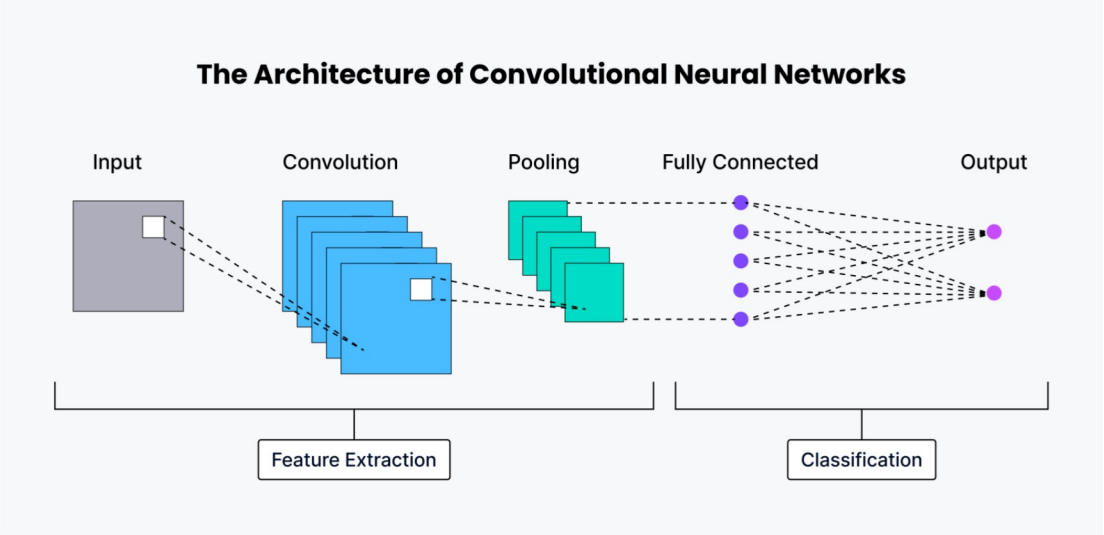


Figure 3: The architecture of CNN

CNN architectures are composed of several core components:

- **Convolutional layers**: Apply learnable `filters`, kernels, that slide over the input, computing dot products to extract local patterns. Hyperparameters such as `stride` and `padding` are used to control the movement and boundary of the filters.
- **Activation functions**: Introduce non-linearity into the network, enabling it to approximate complex functions.
- **Pooling layers**: Perform downsampling operations, such as `max pooling` or `average pooling`, to reduce the spatial dimensions of feature maps while retaining most of the information.
- **Fully connected layers**: Located toward the output of the network, these layers

integrate the learned features to perform final predictions, such as assigning class probabilities.

A key property of CNNs is their **hierarchical feature learning**. Early convolutional layers tend to detect `low-level features`, such as edges, corners, and textures, which are often visually interpretable. Deeper layers progressively learn `high-level semantic features`, including object parts and full object representations [17]. Early-layer features often retain substantial visual similarity to the original input, making them susceptible to `model inversion` or `reconstruction attacks` [15].

The training process of a CNN involves **forward propagation**, where the input passes through all layers to generate predictions, followed by **loss computation**, for exaple using cross-entropy for classification tasks. Then, through **backpropagation**, gradients are calculated and propagated backward to update parameters via optimization algorithms such as `stochastic gradient descent (SGD)`.

3.2 ResNet-50

ResNet-50 stands for **Residual Network** with `50 layers`. The key innovation behind ResNet-50 is the **residual learning** paradigm. The architecture represented a major breakthrough in deep learning, enabling the successful training of very deep convolutional networks without suffering from the severe **degradation problem** that typically occurs when increasing network depth. Traditional deep networks often face the *vanishing* or *exploding gradients* problem, which difficults the effective training of many stacked layers. ResNet addresses this by introducing **skip connections** that allow the input of a layer to be directly added to its output:

$$y = F(x, W) + x \tag{1}$$

where $F(x, W)$ represents the **residual mapping** learned by the network, x is the input, and y is the output after the skip connection.

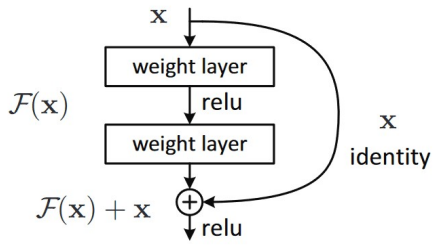


Figure 4: Skip connection

ResNet-50 is a **50-layer deep CNN** composed of convolutional layers, batch normalization layers, `ReLU` activation functions, and pooling layers. Its fundamental building block is the **bottleneck residual block**, which contains three convolutional layers in the sequence: 1×1 (for dimensionality reduction), 3×3 (for spatial processing), and 1×1 (for dimensionality restoration). The overall structure is organized as follows:

- **Conv1** + `MaxPool`

- **Conv Block 1**: 3 bottleneck blocks
- **Conv Block 2**: 4 bottleneck blocks
- **Conv Block 3**: 6 bottleneck blocks
- **Conv Block 4**: 3 bottleneck blocks
- **Global Average Pooling** + `Fully Connected Layer`

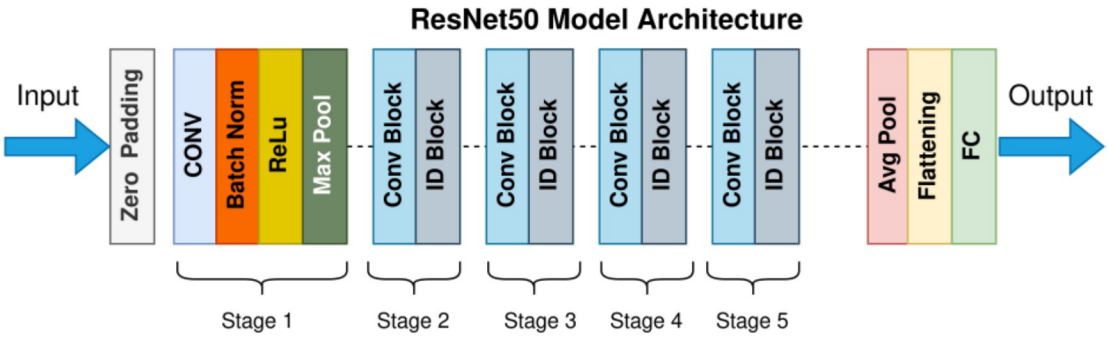


Figure 5: Resnet50 model architecture

ResNet-50 contains approximately **25 million parameters**, making it a balanced choice between model complexity and computational feasibility. Its robustness and strong representational capacity have made it a popular approach in academic research and industrial applications.

3.3 Feature Maps

A **feature map** is the output produced by a `convolutional layer` or another *intermediate layer* in a neural network. It is a data structure that encodes patterns from the input at a given processing stage. A feature map is a **multi-dimensional array**, also called tensor, where each channel corresponds to the response of a specific learned `filter` [16].

Feature maps are generated by applying a set of learnable **filters**, or `kernels`, to the input or to the output of the previous layer through the `convolution` operation:

$$F_i(x, y) = (I * K_i)(x, y) \tag{2}$$

where I is the input, K_i is the i-th convolutional kernel, and $*$ denotes the convolution operation. Each filter K_i produces one channel of the feature map F_i, capturing a specific pattern such as an edge, texture, or shape. The raw convolution outputs are typically passed through an `activation function`, such as `ReLU`, and may undergo `normalization` before being forwarded to the next layer [17].

Feature maps evolve in their semantic content as depth increases [15]:

- **Early layers**: detect low-level features such as edges, corners, and color gradients.

- **Middle layers**: capture more complex patterns, including textures, shapes, and object parts.
- **Deep layers**: encode high-level semantic information, such as entire objects or abstract category-specific patterns.

3.4 Edge Computing

Edge computing refers to performing computation *closer to the data source*, for example, on local devices such as smartphones, IoT nodes, or embedded gateways, instead of sending all data to a centralized cloud server. This paradigm can reduce **latency**, lower **bandwidth usage** and improve **privacy** by limiting the transmission of sensitive raw data [15]. The edge device processes raw data locally and sends only **intermediate results** upstream, reducing both data volume and exposure.

In the context of `deep learning`, edge computation often takes the form of **split neural network inference** [6]. Early layers of the model are executed at the edge, while later layers run on the cloud server. This approach offers several advantages:

- **Reduced transmission size**: only intermediate feature maps are sent instead of the full raw input.
- **Lower inference latency**: computations are distributed and network transfer is minimized.
- **Resilience to connectivity issues**: partial local processing can still provide useful outputs when the network is unavailable.

However, there is a significant drawback: **intermediate features can still leak sensitive information**. As demonstrated in `model inversion` and `reconstruction attacks` [15, 17], these feature maps may retain substantial details about the original input, making them a potential target for adversaries.

3.5 Trusted Execution Environments

A **Trusted Execution Environment (TEE)** is a *secure area* within a main processor that ensures the **confidentiality** and **integrity** of the code and data loaded inside it. The TEE is **isolated** from the rest of the system, including the operating system, hypervisor, and any other applications [6]. This isolation is enforced at the hardware level, providing a trustworthy context for executing sensitive computations. Main examples of TEE implementations are **Intel SGX**, **ARM TrustZone** and **AMD SEV**.

The core **security properties** of a TEE are:

- **Confidentiality**: Code and data within the TEE cannot be accessed or read by any process outside the secure enclave.
- **Integrity**: Code and data inside the TEE cannot be modified or tampered with by external entities.

- **Attestation**: Remote parties can verify the authenticity of the code running inside the TEE, ensuring that it has not been altered.

Operating with a TEE involves loading code and data into a **secure enclave**, protecting sensitive information with **encrypted memory regions**, enforcing isolation through **hardware-level access controls**, and using **attestation mechanisms** to allow remote verification of the enclave's content and execution state.

In `deep learning`, TEEs are used to protect **privacy-sensitive parts** of a model, such as weights, activations, or raw input data [15, 17]. In **partial protection** scenarios, only a subset of layers is executed inside the TEE, while the remaining layers run in an untrusted environment. This setup reduces computational overhead but introduces a **privacy risk**: intermediate `feature maps` leaving the enclave can still be exploited by adversaries through `model inversion` or `reconstruction attacks`.

3.6 Model Inversion Attacks

A **Model Inversion Attack (MIA)** is a type of privacy attack in which an adversary tries to **recover information about a model's training data** by exploiting some form of access to the model [4, 15]. The ultimate goal may vary: in some cases, the attacker aims for the **reconstruction of full inputs**, such as recovering an image of a person's face from a facial recognition system; in others, the aim is **attribute inference**, where specific sensitive properties, such as medical conditions, are deduced from the model's outputs.

Depending on the adversary's access, MIAs can be classified into the following categories:

- **White-box MIA**: The attacker has complete knowledge of the model architecture and parameters.
- **Black-box MIA**: The attacker can only query the model and observe its outputs without internal details.
- **Label-only MIA**: The attacker only sees the predicted class label, without access to probability scores.

There are few mechanisms to perform a Model Inversion Attack. **Optimization-based approaches** iteratively adjust a candidate input to match the target model's outputs, often by minimizing a loss function that measures the difference between the model's predictions for the candidate and the target outputs. **Generative approaches** employ models such as `GANs` or diffusion-based generators trained to map features back to plausible input samples, enabling realistic reconstructions. **Decoder-based approaches** attach a trained inverse network to the target's intermediate layers, reconstructing inputs directly from `feature maps` [15, 17].

MIAs are possible because **deep neural networks** tend to encode significant information about their training data. **Intermediate feature maps**, especially from

early layers, often preserve low-level structures such as edges and textures that can be visually informative. Similarly, **prediction probabilities** can leak semantic or class-specific features, as high-confidence outputs are typically associated with input patterns closely matching the training data. Consequently, the mapping from the input space to the feature space is often *partially reversible*, allowing adversaries to exploit it for privacy breaches.

3.7 Reconstruction Attacks

A **Reconstruction Attack** is a type of privacy attack in which the adversary's objective is to **recreate the full input data** from some form of information leaked by the model. It is a **specific subtype** of `Model Inversion Attack (MIA)`, with a focus on **pixel level** reconstruction for images or **token level** reconstruction for text data [15, 17].

While Model Inversion Attacks seeks for a broader set of goals, including **attribute inference**, **partial reconstruction** or **estimating the distribution** of the training data, reconstruction attacks focus on recovery of the original input. This distinction is crucial when assessing privacy risks, as high fidelity reconstructions can directly expose sensitive information in a more explicit and impactful way. Reconstruction attacks exploit multiple **sources of leaked information**. **Intermediate feature maps**, which often appear in `split` or `collaborative inference` settings, can retain substantial visual or structural similarity to the input. In `federated learning`, **gradients** shared for model updates have been shown to enable near exact recovery of training examples [18]. Even the **model outputs**, such as class probabilities or logits, can leak enough semantic information to facilitate reconstruction.

Several **techniques** are used to perform reconstruction attacks. **Direct optimization** methods search for an input x' that minimizes the difference between the model's output or intermediate representation for x' and the leaked target values. **Generative models**, such as `GANs`, `diffusion models`, or `VAEs`, can be trained to map features back to realistic inputs, often improving the visual plausibility of reconstructions. **Inverted architectures** attach a decoder network specifically trained to reverse certain layers of the victim model, producing reconstructions directly from the leaked representations.

These attacks are effective because **early feature maps** in deep neural networks preserve spatial structures and local details from the original input. Even **compressed intermediate representations** can carry sufficient information for reconstruction if they retain key statistical and semantic properties. Furthermore, deep models often maintain **redundancy in their internal representations** to improve robustness and generalization, but this same redundancy can be exploited by adversaries to recover private data.

3.8 Generative Adversarial Networks

A **Generative Adversarial Networks (GANs)** is a type of **generative model** that learns to produce data samples that mimic the distribution of a given dataset. The framework consists of two neural networks: a **Generator** and a **Discriminator**, which are trained together in an *adversarial* setup.

The **Generator** (G) takes as input a random vector z drawn from a prior distribution, such as Gaussian, and generates a synthetic sample $G(z)$. The **Discriminator** (D) takes an input, either a real sample from the dataset or a generated sample, and outputs the probability that this sample is real. The two networks are trained simultaneously in a minimax game with the following objective function:

$$\min_G \max_D \; \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{3}$$

This formulation encourages D to correctly distinguish between real and fake samples, while G aims to produce outputs that D classifies as real. The **training process** follows an *adversarial learning* scheme. The discriminator D improves its ability to differentiate between real and generated samples, while the generator G improves its capacity to produce realistic outputs that can fool D. Training continues until G generates samples that are indistinguishable from real data according to D.

GANs have a wide range of **applications**, including `image synthesis`, such as human faces, artwork, medical images, `data augmentation`, `super-resolution`, and `domain adaptation`.

3.9 Generative Model Inversion Attacks Against Collaborative Inference

In a collaborative inference setup (split neural network between a client device and a server), the **intermediate feature maps** exchanged can unintentionally encode a significant amount of information about the original input. Even though the **raw data never leaves the client**, several works have shown that a malicious server can exploit the intermediate activations to reconstruct the client's input, an attack known as model inversion in collaborative inference [15]. Under this context, the **Generative Model Inversion Attacks Against Collaborative Inference (Ginver)** has been proposed.

Ginver's attack train a dedicated **neural network to learn the inversion mapping from feature space** back to input space, rather than solving a new optimization for each input. Typically, the adversary trains a generator network G that takes as input the victim model's intermediate feature, the output of the client-side sub-model $f_{\theta_A}(x)$, and outputs a reconstructed $\hat{x}$. The goal is for G to approximate the inverse of f_{θ_A}, such that $G(f_{\theta_A}(x)) \approx x$. The training can be done in a conditional GAN (c-GAN) framework: G is trained to produce realistic-looking inputs conditioned on the intermediate features, while a discriminator D tries to distinguish between G's outputs and real inputs. The objective typically combines a reconstruction loss (ensuring $f_{\theta_A}(\hat{x})$ matches the original feature) with an adversarial loss to encourage plausible outputs. Once trained, the generator G can invert new feature maps in one forward pass, making the attack fast and scalable compared to optimization-based inversion [15].

An important consideration is the adversary's knowledge of the victim model. In the literature, generative inversion attacks are evaluated under two settings [15]:

- **First scenario**: The adversary has access to the function of the victim's model f_{θ_A}. This allows direct use of f_{θ_A} in training G (backpropagating through f_{θ_A} to minimize $|G(f_{\theta_A}(x)) - x|^2)$). The paper[15] describes an attack where Mallory trains G using intercepted feature maps from the protected model, achieving high-fidelity reconstructions without per-input re-optimization.

Algorithm 1: Ginver's training algorithm

function `TRAIN` *(f_{θ_A}, $\mathcal{F}$, λ, ϵ, T, k)*:

/* f_{θ_A} is the victim's model (front part) */
/* $\mathcal{F}$ is a set of intermediate features */
/* λ is the equilibrium coefficient in Equation (4) */
/* ϵ is the learning rate */
/* T is the number of iterations */
/* k is the batch size */
/* θ_G represents the parameters of generator G */
initialize G, $t \leftarrow 0$

while $t < T$ **do**

Randomly sample $f_{\theta_A}(x_1), f_{\theta_A}(x_2), \ldots, f_{\theta_A}(x_k)$ from $\mathcal{F}$

$$L \leftarrow \frac{1}{k} \sum_{i=1}^{k} \left\| f_{\theta_A}\big(G\big(f_{\theta_A}(x_i)\big)\big) - f_{\theta_A}(x_i) \right\|_2$$

$$L \leftarrow L + \frac{1}{k} \sum_{i=1}^{k} \mathrm{TV}\big(G\big(f_{\theta_A}(x_i)\big)\big)$$

$$\theta_G \leftarrow \theta_G - \epsilon \frac{\partial L}{\partial \theta_G}$$

$t \leftarrow t + 1$

return G

- **Second scenario**: The adversary does *not* have access to f_{θ_A} local model. In this harder scenario, training G is done without direct gradients from f_{θ_A}. This scenario has not been explored in our project.

Relation to our attack

Our work builds directly on the **first scenario** described in Ginver, where the adversary has access to the intermediate outputs of the victim model and can exploit them to train a generator G. This scenario aligns with the conditions of our project: part of the model is executed privately inside a **Trusted Execution Environment (TEE)**, while the subsequent layers are exposed, making it possible to intercept and use the victim's intermediate representations for training the attack model. Such access enables us to evaluate the feasibility of inversion even when raw inputs remain protected by the enclave. Beyond replicating the original Ginver approach, we extend the methodology by presenting a **white-box variant**, where in addition to using the intercepted feature maps, we also leverage full knowledge of the ResNet-50 architecture. In this setting,

the adversary constructs a mirrored inversion network specifically tailored to each attacked layer, thereby exploring whether architectural knowledge can further improve the quality of reconstructions compared to the generic generative method proposed by Yin et al. [15].

3.10 Similarity Metrics

Similarity metrics are quantitative measures that assess how alike two data samples are. In the context of `image reconstruction`, they are used to compare the reconstructed image against the original ground-truth image. These metrics can be **distance-based**, where lower values indicate greater similarity, or **similarity-based**, where higher values indicate greater similarity.

Two standard metrics are **Mean Squared Error (MSE)** and **Structural Similarity Index Measure (SSIM)**. The **Mean Squared Error (MSE)** measures the average squared difference between corresponding pixel values of two images. Formally, it is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2 \tag{4}$$

where x_i and y_i denote the pixel values of the reconstructed and original images, respectively, and n is the total number of pixels. Lower MSE values indicate a closer match between the two images. However, MSE is sensitive to small pixel-level changes and does not always align with human perception of image quality.

Structural Similarity Index Measure (SSIM) evaluates the similarity between two images by considering three components: luminance, contrast, and structural information. The SSIM value ranges from -1 to 1, with 1 indicating perfect similarity. SSIM correlates better with human visual perception compared to MSE and is therefore widely used for perceptual quality assessment.

Despite their usefulness, these metrics have **limitations**. MSE may fail to capture perceptual similarity, meaning that two images with the same MSE can look very different to a human observer. SSIM, while more perceptually aligned, can still be fooled by certain types of distortions or image artifacts. For this reason, similarity metrics are often complemented with **visual inspection** and **task-specific metrics**, such as the classification accuracy of reconstructed samples when fed back into the original model.

4 Methodology

4.1 Overview of the Methodology

This experimental study aims to determine reconstruction attacks can successfully recover original training images from a partially protected ResNet-50 model and evaluate the feasibility of these attacks in practical terms. The project covers everything from **dataset acquisition** to final **evaluation of results** and its interpretation. We begin with preliminary experiments on the `MNIST` dataset to validate our approach on a simple and well understood benchmark. These initial trials of attack give us some insight into the feasibility of the Ginver approach. Once the MNIST prototype has been developed, we then conduct our main analysis on a deeper model: a `ResNet-50` network trained on a sensitive pharmaceutical dataset.

On this project, we explore model inversion techniques under two different scenarios: **white-box** and **black-box** scenario. In the white-box setting, we assume full access to the target model's architecture and intermediate feature maps from various layers of the `ResNet-50` that can be directly obtained and used for inversion [8]. On this approach, we have designed an adversarial neural network with the inverted architecture, since we know which are the layers that form the target network. In contrast, the black-box setting assumes the attacker must treat the target layer as an opaque module, relying only on its externally observable output, in other words, the exposed feature vector without insight into the layer's architecture. For the black-box attacks, we rely on strategies that have been proposed in prior works [15]. By applying both white-box and black-box inversion techniques on intermediate features extracted from different layers of the ResNet-50, we can assess how the amount of accessible information and how the depth of the attacked layer affects the success of the inversion.

For each chosen layer, we train a dedicated inversion model to reconstruct the original input image from that layer's features. Each inversion model learns to map high dimensional feature vectors back to the input pixel space by minimizing a reconstruction loss. We evaluate the performance of these inversion models using quantitative metrics such as Mean Squared Error (`MSE`) and Structural Similarity Index (`SSIM`), which measure how closely the reconstructed images resemble the original inputs. Furthermore, to gauge the semantic validity of the reconstructions, we feed the recovered images into the original classifier network to check if they yield the correct predictions. The classification accuracy of the reconstructed images thus serves as an additional indicator of attack success, revealing whether the inversion has merely produced vague approximations or truly meaningful reconstructions of the sensitive data.

In summary, our methodology proceeds through the following stages: **dataset acquisition**, **model selection & training**, **attack implementation** (designing and applying white-box/black-box inversion on selected layers), **inversion model training** and **evaluation** of the results. This workflow provides an overview of the experimental pipeline, from the selection of data and models to the implementation of inversion attacks and the assessment of their outcomes, setting the stage for more detailed technical discussions in subsequent sections.

4.2 Datasets Used

4.2.1 MNIST dataset

Used as an initial test subject to validate the inversion attack pipeline in a simple, low complexity setting before moving to more complex models and sensitive data [8, 15]).

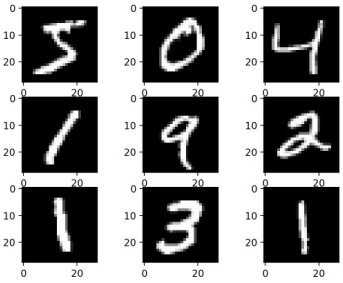


Figure 6: MNIST sample examples

Type	Handwritten digit images (grayscale)
Size	70,000 total images (60k train, 10k test)
Resolution	28×28 pixels
Classes	10 (digits 0–9)

Table 1: Characteristics of the MNIST dataset.

- Easy to train, with minimal preprocessing and widely recognized benchmark in ML literature.
- Facilitates quick experimentation and hyperparameter tuning for inversion models; training is feasible on a personal laptop (no need for cloud resources such as `AWS`).
- Serves as a proof-of-concept environment for implementing the attack methodology prior to tackling more complex and sensitive datasets.

4.2.2 EMBL Dataset

This custom dataset, provided by the European Molecular Biology Laboratory (EMBL), contains pharmaceutical and biomedical images, off-sample mass spectrometry images as described in [11]) for a binary classification task distinguishing *off-sample* vs. *on-sample* inputs.

- Main dataset for our ResNet-50 experiments, representing a real-world industrial scenario with proprietary pharmaceutical data, such as AstraZeneca, with correspondingly high privacy stakes.

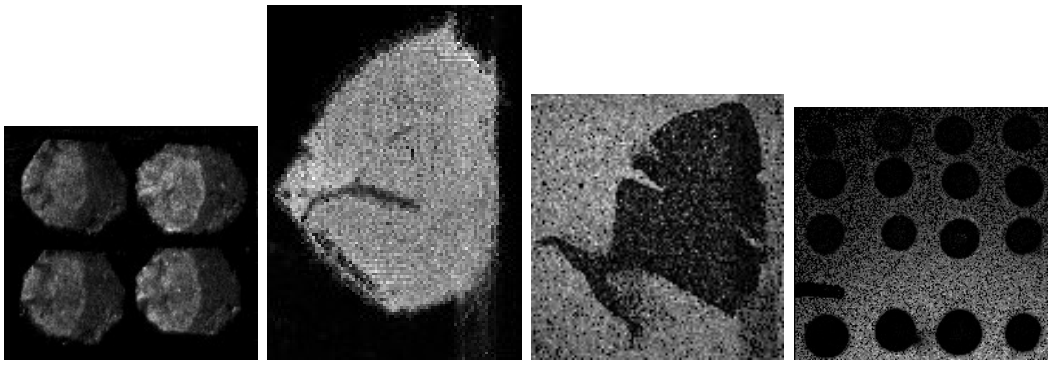


Figure 7: EMBL dataset sample examples

Type	Domain-specific pharmaceutical/biomedical images
Labels	Off-sample vs. on-sample (binary classification)
Resolution	$\sim$150 $\times$ 150 pixels, multi-channel (variable dimensions)
Size	23,238 images (proprietary dataset, not publicly available)

Table 2: Characteristics of the EMBL dataset.

- The complexity and variability of these data make inversion attacks more challenging and any reconstructed results more meaningful for evaluation.
- The non-public, sensitive nature of the data highlights the practical privacy risks associated with inversion attacks in an industry setting.

4.3 Initial Experiments with MNIST

As a first step, we conducted preliminary experiments on the `MNIST` handwritten digits dataset to validate the overall inversion pipeline under a controlled, low-complexity setting. The aim was to:

i. Verify that feature-map–based reconstruction could be trained on modest hardware, saving time and money.

ii. Build a proof of concept of the inversion attack (feature extraction $\rightarrow$ inversion model training $\rightarrow$ evaluation).

iii. Make easier the transition to a deeper architecture (ResNet-50) and sensitive real-world data on later iterations.

For the inversion attack, we focus on reconstructing an input image $\hat{x}$ from an intermediate representation h produced by the classifier's early layers. Denote the front part of the classifier by f_{θ_A} and an input by x; the exposed feature map is $h = f_{\theta_A}(x)$. An inversion network g_ϕ is trained to map h back to pixel space:

$$\hat{x} \;=\; g_\phi(h) \quad \text{with} \quad h = f_{\theta_A}(x).$$

The decoder parameters ϕ are optimized to **minimize a reconstruction loss**

between the output $\hat{x}$ and the original x. We adopt a pixel-space objective:

$$\mathcal{L}_{\text{rec}}(\phi) \;=\; \frac{1}{N}\sum_{i=1}^{N} \left\| g_\phi\big(f_{\theta_A}(x_i)\big) - x_i \right\|_2^2, \tag{5}$$

in other words, **mean squared error (MSE)**, which is standard for image-space reconstruction baselines; perceptual or structural terms, such as SSIM, could be incorporated in later experiments when moving to more complex data.

In this stage, we deliberately *do not* report performance numbers; since we seek to confirm the feasibility and stability of the approach. This feasibility step is consistent with prior work that motivates attacking intermediate representations in split/cloud-edge settings [8, 12, 15].

Network architecture used

The MNIST classifier is a small `PyTorch` CNN implemented in `Model.py` (class accesible in the github of the project). Concretely, it consists of two convolutional blocks followed by fully connected layers and dropout:

- `conv1`: `nn.Conv2d(1, 32, kernel_size=3, stride=1)`.
- `conv2`: `nn.Conv2d(32, 64, kernel_size=3, stride=1)`.
- `dropout1`, `dropout2`: `nn.Dropout(0.25)` and `nn.Dropout(0.5)`.
- `fc1`: `nn.Linear(9216, 128)`; `fc2`: `nn.Linear(128, 10)`.

The forward pass applies convolutions and non-linearities, with spatial downsampling (via pooling in the forward routine) producing a $64 \times 12 \times 12$ feature map (flattened to 9216) before the fully connected layers. This compact architecture ensures fast training and clear inspection of intermediate activations.

Whitebox scenario

In the **white-box** variant (architecture-informed), g_ϕ mirrors the known operations of the victim layer, transposing convolutions aligned to the feature shape, to better exploit structural knowledge.

Following `Model.py`, we used a minimal decoder (class `AdversarialNet`) composed of two transposed-convolution layers:

- `nn.ConvTranspose2d(64, 32, kernel_size=3, stride=1)`
- `nn.ConvTranspose2d(32, 1, kernel_size=3, stride=1)`

This design consumes the 64-channel feature map h, after `conv2`/pooling, and upsamples back to a single-channel image.

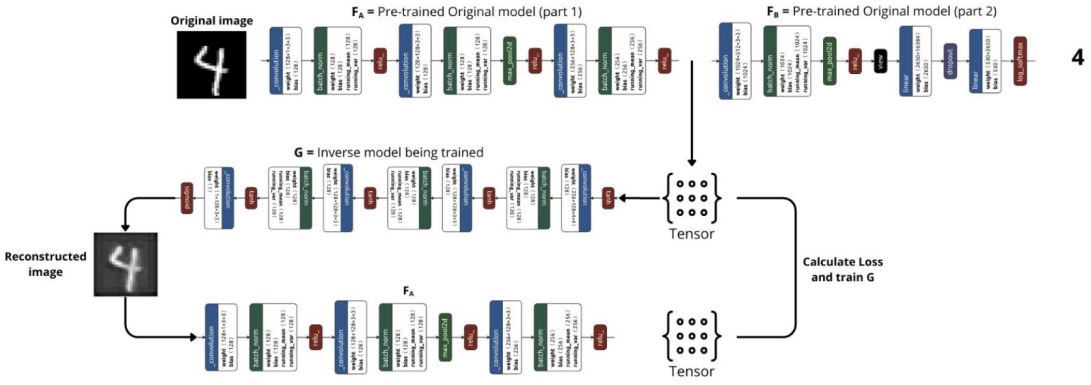


Figure 8: MNIST white-box scenario training phase

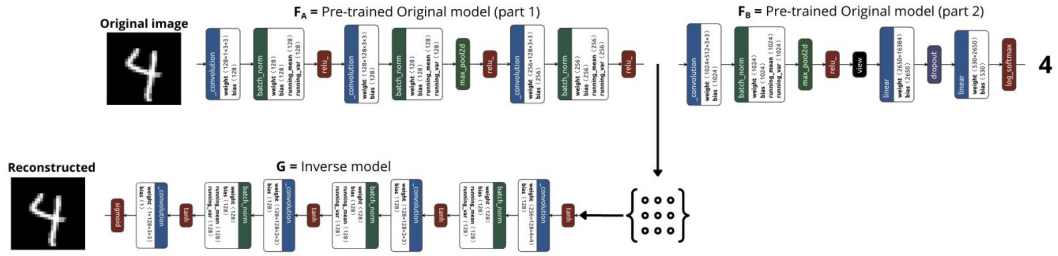


Figure 9: MNIST white-box scenario exploiting phase

Blackbox scenario

In the **black-box setting**, the same generic decoder (generic inverse) is used across candidate layers. g_ϕ is implemented as a lightweight decoder that does not exploit the internals of f_{θ_A} (treating the layer as opaque) [15]. This is based on Ginver's approach, already described on section 3.9.

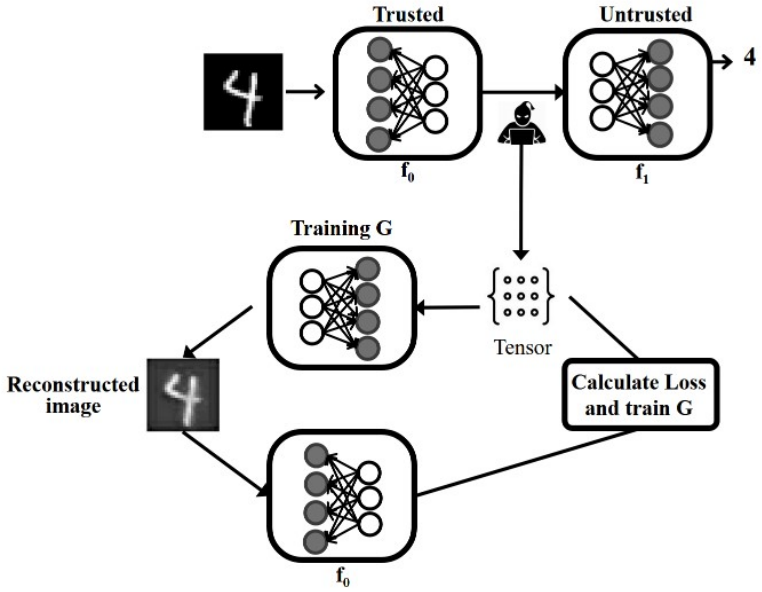


Figure 10: MNIST black-box scenario

Obtained results

The MNIST experiments served their purpose as a feasibility check and informed the subsequent methodology on ResNet-50 and sensitive data:

- *Pipeline validation:* We confirmed that training an inversion model from intermediate features is stable and converges on commodity hardware, validating the training/evaluation loop prior to scaling.
- *Attack modes:* Implementing both **black-box** (generic inverse) and **white-box** (mirrored decoder) variants on MNIST clarified the practical difference between exploiting vs. not exploiting layer internals [15].
- *Readiness for real data:* With the pipeline verified, we proceed to ResNet-50 and domain-specific, privacy-sensitive data where the stakes and difficulty are higher [8, 11].
- *Good quality results*: The similarity metrics, MSE and SSIM, show that the images reconstructed using the attack show a high degree of similarity to the original images.

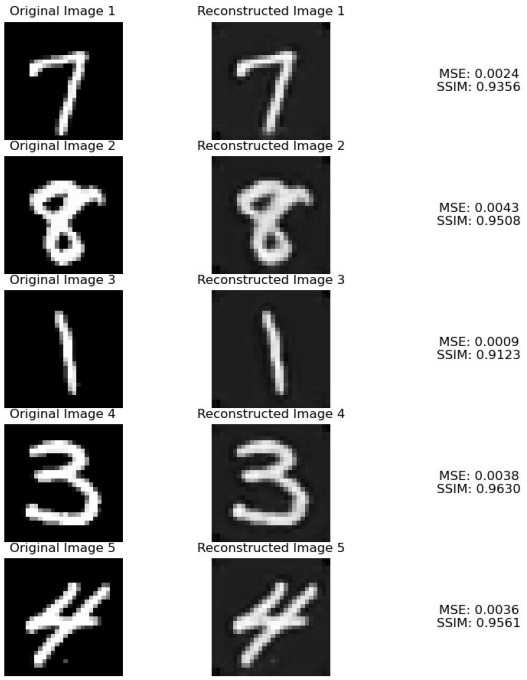


Figure 11: Reconstruction attack results with MNIST

4.4 Model Architectures

ResNet-50 has been used as the victim model's backbone due to its proven efficacy in image classification tasks. ResNet-50 is a 50-layer deep convolutional neural network featuring residual skip connections that mitigate vanishing gradients and enable very deep architectures. This model was selected for its strong feature extraction capabilities, which are crucial for our attack, and because it has been used in prior

off-sample image recognition and inversion studies. In addition, ResNet-50 served as the target network in the original generative inversion attacks (the *Ginver* approach) aimed at reconstructing images from intermediate features, demonstrating that its latent representations can be exploited by adversaries [11, 15]. Its widespread adoption and high performance [8] make ResNet-50 a representative choice for evaluating the efficacy of our attacks.

In order to success in our attack, We modify the standard ResNet-50 architecture to facilitate fine-grained attack insertion points. In particular, we introduce **multiple return points** in the network's `forward()` function to output intermediate feature maps at various depths. These hooks allow the attacker to intercept and obtain the **activations from any chosen layer during inference**. Furthermore, we decompose each residual block of ResNet-50 into individual layers, separating convolutions, batch normalization, ReLU, and skip connections, rather than treating an entire block as a single module. This granular re-organization provides flexibility to partition the model at arbitrary layer boundaries, granting the adversary freedom to select attack points at a per-layer level. Such architectural adjustments mirror the needs of partition-based threat models [12], where partial model exposure, even at deeper layers, can leak information. By enabling returns from and inserts into each convolutional layer, our ResNet-50 modification ensures that we can simulate both shallow and deep interception attacks with precision.

To reconstruct input images from intercepted features, we design **two types of decoder networks** corresponding to different adversarial knowledge scenarios (black-box vs. white-box) [8]. Specifically, we implement:

- **Generic Ginver-style decoder (black-box):** In the first approach, a single unified inversion model is used uniformly across all layer outputs. This decoder is inspired by the generative inversion strategy of Ginver [15], which employs a fixed generative network to map intermediate features back to the input space. The model does not assume detailed knowledge of the layer's internal structure, treating the feature vector as an input to a generic image reconstruction network. This scenario represents a *black-box* attack, where the adversary relies on a one-size-fits-all inversion architecture for any target layer. The code of this decoder can be read on the code appendix section 2.

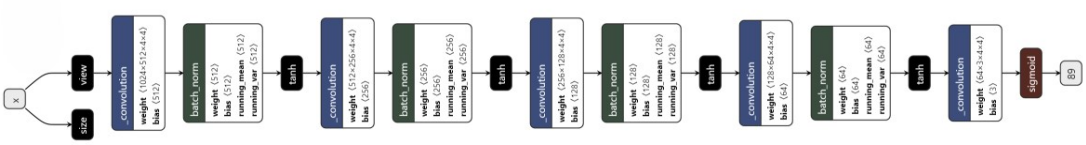


Figure 12: Generic inversion model architecture

- **Layer-specific mirrored decoders (white-box):** In the second approach, we craft a dedicated decoder architecture for each victim layer, effectively mirroring that layer's operations in reverse. In this *white-box* setting, the attacker leverages full knowledge of the layer's architecture and functionality. For example, for a

given convolutional layer's output, we construct a corresponding decoder with transposed convolutions (and appropriate upsampling or activation functions) that invert the forward computations of that layer. Each decoder is thus tailored to the dimensionality and characteristics of its target layer's feature map. By aligning the inversion network with the victim model's layer-wise transformations, this approach can more precisely reconstruct the original input (as also suggested by prior partition-based inversion studies [12]). The trade-off is that a unique decoder must be trained for every layer of interest, which is feasible only when the model architecture is known to the attacker.

```
class ResNetInversion_MaxPool(nn.Module):
    def __init__(self, nc=3, ngf=64):
        super(ResNetInversion_MaxPool, self).__init__()

        self.decoder = nn.Sequential(
            # 1. Invert MaxPool (3×3, stride 2)
            nn.ConvTranspose2d(64, 64, kernel_size=3,
                stride=2, padding=1, output_padding=1),

            # 2. No specific inversion for ReLU,
            # but use ReLU in decoder
            nn.ReLU(inplace=True),

            # 3. Invert BatchNorm - just use another
            #BatchNorm with learned parameters
            nn.BatchNorm2d(64),

            # 4. Invert Conv (7×7, 64, stride 2)
            # transpose convolution
            nn.ConvTranspose2d(64, nc, kernel_size=7,
                stride=2, padding=3, output_padding=1),

            # Final activation to
            # produce image-like outputs
            nn.Tanh()
        )

    def forward(self, x):
        return self.decoder(x)
```

Code 1: Definition of an inverted convolutional block in `PyTorch`.

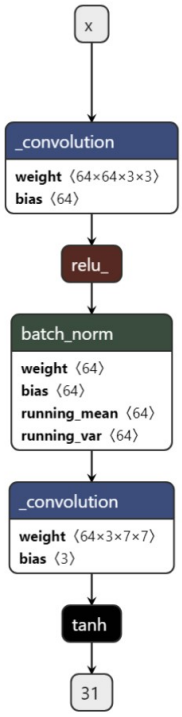


Figure 13: Layer-specific mirrored model architecture for Maxpool layer

4.5 Attack and Inversion Methodology

In our threat model, a malicious cloud server intercepts intermediate features from a victim's deep model and aims to recover the original input image from these features. This is essentially a **representation inversion attack**, also known as a model inversion attack. The attacker leverages the information-rich feature maps produced by the victim's network (the ResNet-50 classifier) to reconstruct a version of the input. Formally, let $\mathbf{h}i = f1 : i(x)$ denote the activation output at some layer i of the victim's model f for input x. The attack seeks to learn an approximate inverse mapping G_i such that $G_i(\mathbf{h}_i) \approx x$. In practice, we implement G_i as a deep generative model that takes $\mathbf{h}_i$ as input and outputs a reconstructed image $\hat{x}$.

Inversion approach

Rather than using only iterative gradient-based optimization as in early attacks, we adopt a learning-based inversion method inspired by recent generative inversion techniques. Specifically, we train a dedicated inversion network to map features back to the input space. This network learns to approximate the **inverse of the victim's feature extractor** $f_{1:i}$. Aided by a discriminator the generator is encouraged to produce realistic images that match the original input's structure and content. This generative attack strategy, similar to Ginver, allows rapid inference of images from features and has been shown to yield high-quality reconstructions even when the attacker does not repeat a costly optimization for each query. We emphasize that the attacker is assumed to know the architecture of the victim model's later layers (a common scenario in collaborative inference settings) and has access to the intermediate activation $\mathbf{h}i$ transmitted for remote inference. This knowledge (white-box of $fi + 1 : \text{end}$) helps

the adversary train G_i to match the distribution of genuine inputs that produce such features.

Chosen split layers

We evaluate the attack at several cut-points in the ResNet-50 architecture to understand how layer depth affects invertibility. ResNet-50 consists of an initial convolution stage followed by four sequential residual blocks with skip connections. **From this point this residual blocks will be defined as Conv block 1, Conv block 2, Conv block 3 and Conv block 4**. You can appreciate these split points in the image 14.

We select representative layers at the boundaries of these major blocks as the points of attack, where the model could be partitioned between client and server. The rationale for focusing on block outputs is that each residual block's output feeds into the next stage and would naturally be the transmission point in split inference. Moreover, due to the skip connections, the output of a block includes a direct additive input from an earlier layer, potentially carrying forward more detailed information useful for reconstruction.

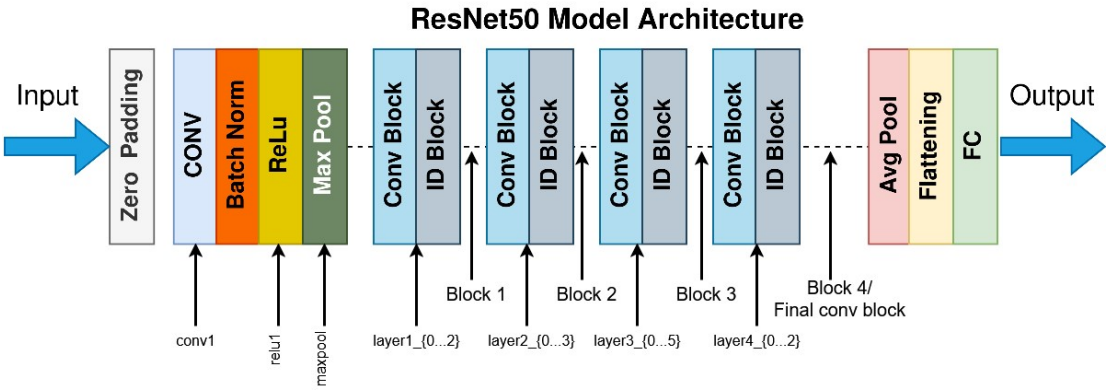


Figure 14: Split points on Resnet50 selected to be attacked

We considered the following cut-points in ResNet-50, corresponding to increasing network depth:

- **After initial conv layer:** The feature map after the first conv + maxpool is still large in spatial size, on the order of 56×56 pixels, and retains mostly **low-level visual features**, like edges or textures. Exposing this early representation poses a worst-case privacy risk, as it is almost a downsampled version of the original image. We expect nearly perfect or **high-fidelity reconstructions** from this shallow layer's features, since very little abstraction has taken place.
- **After first residual block (block1 output):** At this point, the feature map has 256 channels at 56×56 resolution. The network has begun to extract **higher-level features**, but many structural details of the image like shapes and outlines are still present. The skip connection in this block means the output is the sum of the block's convolutional transformation and the original input to the block,

preserving information. Thus, an attacker can still achieve a highly recognizable reconstruction from block1 features. This block represents a typical early split used in practice to offload most computation while unfortunately leaving a **lot of image information in the exposed feature**.

- **After intermediate blocks (block2 / block3):** We also examine inversion after deeper residual blocks. For instance, the block2 block output (512 channels at 28×28) and block3 output (1024 channels at 14×14) contain more abstracted features. Many fine details, such exact textures or small structures, are lost as the feature maps become spatially smaller and more semantically oriented. However, even these mid-level features still encode the overall layout and prominent objects in the image. The attacker's generator can leverage this to produce **partial reconstructions** that capture the general scene or object shape, though some details may be blurred or incorrect. Prior work has shown that moving the cut deeper in a network gradually decreases reconstruction quality, but only modestly until very deep layers.

- **After final residual block (block4 output):** This is the output right before global pooling (2048 channels of size 7×7). It is a highly compressed representation of the image. We expect reconstructions from this block's features to be much lower in quality; perhaps the rough silhouette or dominant colors of the object can be recovered, but fine details, such as facial features or text, would be largely missing. Indeed, by this stage the network has discarded most spatial detail in favor of **high-level classification features**. Attacking this block tests the limits of our inversion model. A successful reconstruction here, even if blurry, still indicates a privacy breach. Conversely, if images are unrecognizable, it suggests the partition might be safe from inversion. Notably, the skip connections present in ResNet's earlier stages have less influence by the final block, yet the **cumulative effect of residual connections** throughout the network can make even deep features more informative than one might assume. Recent studies have found that skip connections significantly improve inversion attacks' success, especially the last-stage skip in ResNet. This insight aligns with our reasoning: the block4 output in ResNet-50 still benefited from a residual addition, making it slightly easier to invert than a strictly feedforward compression of equal depth.

In addition to the five main cut-points described above, we also selected **intermediate layers within each block** as potential split points. Specifically, after each subgroup of convolutional operations inside a bottleneck block (as described on section 3.2), we defined return points in the `forward()` function of our `Model.py` implementation. This design choice allows more fine-grained modularity: instead of considering only the aggregated output of an entire residual block, we can analyze how the effectiveness of inversion attacks evolves at different stages *within the same block*.

In summary, our attack methodology involves splitting the victim network at various depths and training an inversion model to reconstruct inputs from the exposed features. By comparing these different split points, we can analyze how the information content of features diminishes or persists) as the network goes deeper. This methodological design allows us to quantify the privacy leakage at each layer: if a high-quality

reconstruction is obtained from layer i, then revealing $f_{1:i}(x)$ to an adversary is clearly privacy-threatening. Our approach builds on and reinforces findings from prior works; is expected to find that early layers carry most of the input data and thus must be protected or kept client-side, while suitably deep partitions can obscure enough detail to safeguard the input (often requiring additional measures like Trusted Execution for the initial layers). The next sections detail how we trained the inversion models and evaluate the reconstruction fidelity using various metrics.

4.6 Training Procedure

Before the training phase, the original EMBL dataset was reorganized into distinct **training** and **testing** sets. To automate this process, we implemented the `organize_dataset.py` script, which performs an **80/20 split** of the data. The script scans the raw dataset, identifies `onsample` and `offsample` images, applies random shuffling for reproducibility, and then distributes the images into their respective directories. This ensures a clean separation between training and evaluation data, allowing the experiments to be conducted under reproducible and controlled conditions.

We applied several preprocessing steps to the input feature maps and labels before training. Each input image was zero-padded if needed and then normalized channel-wise using the mean and standard deviation of ImageNet (zero mean, unit variance) to match the pretrained model's input range [11]. **Images were resized** to a fixed resolution (224×224) and converted to **three channels**, by replicating the single-channel image, as required by the ResNet-50 architecture [11]. Data augmentation was employed to increase variability: we applied random crops/pads, horizontal/vertical flips, rotations, zooms, and brightness/contrast adjustments using the fast.ai default settings [11]. The class labels were encoded as one-hot vectors for the network's softmax loss.

Hyperparameters were tuned using a grid search over a predefined range of values. The key hyperparameters and their roles are:

- `learning rate` (η): controls the step size in gradient updates, since too large can cause divergence and too small slows convergence.
- `batch size`: number of training samples processed in one forward/backward pass. Larger batches give more stable gradients but require more memory.
- `optimizer type`: the algorithm for weight updates, such as `SGD` with momentum or `Adam`, which determines how gradients are accumulated and whether adaptive learning rates or momentum are used.
- `dropout rate`: the probability of dropping each hidden unit during training, a form of regularization that prevents co-adaptation of features.
- `weight decay`: the coefficient of L2 regularization on the weights, which penalizes large weights in the loss to reduce overfitting.
- `momentum`: (for SGD) the factor that accelerates convergence by accumulating a velocity of past gradients.

Each of these hyperparameters was varied in the grid search. For instance, different learning rates (like 1e-4, 1e-3 or 1e-2) and batch sizes (like 32, 64 or 128) were tested. The best combination was selected based on validation performance.

Training was performed for a fixed number of epochs with a separate validation set to monitor generalization. In practice, we trained for up to **50–100 epochs**, but employed **early stopping to halt training if the validation loss did not improve for a few consecutive epochs**. In our experiments, we used a 5-fold (grouped) cross-validation scheme [11], ensuring that images from the same source dataset were not split between training and validation folds. Specifically, we reserved one fold for validation and trained on the remaining folds in each round. Early stopping patience was typically set to 3 epochs, meaning training stopped if the validation loss failed to decrease for 3 epochs in a row. Additionally, the above-mentioned regularization, dropout and weight decay, was applied during training to mitigate overfitting, since excessive training can lead to overfitting which is known to increase vulnerability to inference attacks [8].

Overall, the training loop involved feeding minibatches of preprocessed images into the network, computing the loss like cross-entropy, and updating the weights according to the chosen optimizer and learning rate. We monitored the training and validation loss at each epoch. The final model was selected based on the lowest validation loss or highest validation accuracy achieved during training. By combining careful preprocessing, systematic hyperparameter search, and training strategies such as early stopping and regularization, we ensured a robust training process that generalizes well to unseen data.

4.7 Evaluation Metrics

4.7.1 Similarity metrics

We use **MSE** (Mean Squared Error) and **SSIM** (Structural Similarity Index Measure) to quantify image similarity. MSE is defined as the average squared difference between corresponding pixels of a ground-truth image X and a reconstructed image Y:

$$\mathrm{MSE}(X, Y) = \frac{1}{N} \sum_{i=1}^{N} (X_i - Y_i)^2,$$

where N is the number of pixels in the image. SSIM compares local patterns of pixel intensities, taking into account luminance, contrast, and structural information. In one common form, for images X, Y with local means μ_X, μ_Y, variances σ_X^2, σ_Y^2 and covariance σ_{XY}, SSIM is given by

$$\mathrm{SSIM}(X, Y) = \frac{(2\mu_X\mu_Y + C_1)(2\sigma_{XY} + C_2)}{(\mu_X^2 + \mu_Y^2 + C_1)\,(\sigma_X^2 + \sigma_Y^2 + C_2)},$$

where C_1, C_2 are small stabilization constants. A higher SSIM (closer to 1) indicates more similar structure.

Despite their simplicity, **MSE** and **SSIM** have limitations in modeling perceptual similarity. MSE treats all pixel differences equally, so minor distortions that are vi-

sually insignificant can still produce large MSE values. SSIM is more aligned with human perception than MSE, but it still relies on local low-level statistics. In practice, optimizing for low MSE or high SSIM does not always yield images that look better to human observers. This observation can be clearly seen in the image 15, where images with similar MSE values have different SSIM values or images with similar MSE and SSIM values are clearly visually different.

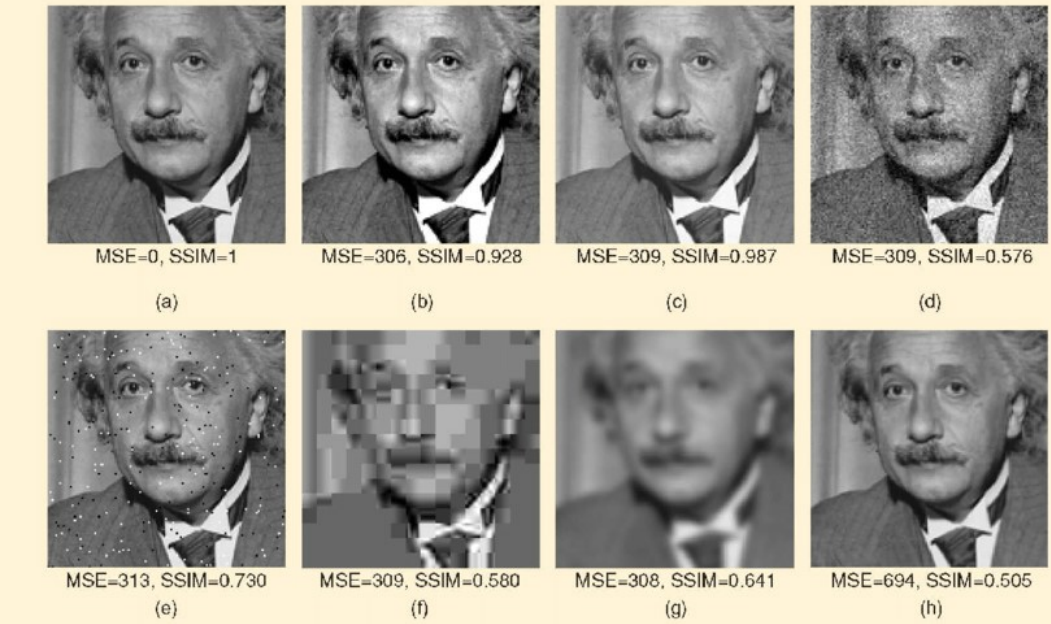


Figure 15: Comparation among MSE, SSIM and visual similarity

To better capture human visual similarity, we also include the **LPIPS** (Learned Perceptual Image Patch Similarity) metric. LPIPS uses a deep network trained on natural images to extract features, and computes the distance between the feature representations of two images. This learned perceptual distance **correlates more strongly with human judgments of image quality** than pixel-wise metrics. In effect, LPIPS measures the weighted ℓ_2 difference of deep activations between X and Y, providing a more semantically meaningful similarity score. LPIPS is computed as follows:

$$\text{LPIPS}(x, y) = \sum_l \frac{1}{H_l W_l} \sum_{h=1}^{H_l} \sum_{w=1}^{W_l} \left\| w_l \odot \left(\hat{f}_l(x)_{hw} - \hat{f}_l(y)_{hw} \right) \right\|_2^2$$

where:

- l indexes the **layer** of the deep feature extractor (typically a pretrained network such as `AlexNet`, `VGG`, or `ResNet`).
- H_l, W_l denote the **height** and **width** of the feature map at layer l.
- $\hat{f}_l(x)$ and $\hat{f}_l(y)$ represent the **unit-normalized feature activations** for the input images x and y at layer l.

- w_l is a learned **per-channel weight vector** applied to the features.
- $\odot$ denotes the **element-wise multiplication**.
- $\|\cdot\|_2^2$ indicates the squared **Euclidean distance**.

4.7.2 Classification accuracy

Finally, we assess semantic fidelity via **classification accuracy**. Each reconstructed image is passed through the original image classifier, and we record the fraction whose predicted label matches the ground truth. A high classification accuracy indicates that the reconstruction preserves the class-discriminative features of the image. We further analyze errors using a **confusion matrix** between true and predicted labels. By definition, a confusion matrix C has entries $C_{i,j}$ equal to the number of samples whose true class is i and predicted class is j. This table highlights which classes are most often confused by the reconstructed images, providing insight into the semantic mistakes of the model.

4.8 Experimental Setup

4.8.1 Hardarware Setup

The experimental evaluation of the proposed inversion attacks was carried out using different hardware configurations, depending on the stage of the work and the computational requirements of each experiment.

Initially, for the **feasibility tests** performed on the `MNIST` dataset, all experiments were executed on a **personal computer**. This was sufficient for training small-scale models and running preliminary inversion attacks. These first trials confirmed the viability of the approach and validated the experimental pipeline before scaling to more complex scenarios. These are the the specifications of the device:

Compute	**Value**
Device Name	DESKTOP-CUCI874
Physical Processor	11th Gen Intel(R) Core(TM) i9-11900H
Clock Speed (GHz)	2.50 GHz
vCPUs	8 (16 threads)
Memory (GiB)	64.0 (63.7 usable)
CPU Architecture	x86_64
System Type	64-bit OS, x64-based processor
GPU	NVIDIA GeForce RTX 3050 Ti Laptop GPU
FLOPs	8.7 TFLOPs
GPU Architecture	Ampere
Video Memory (GiB)	4.0
GPU Compute Capability	8.6
Integrated GPU	Intel(R) UHD Graphics (2.0 GiB VRAM)
FPGA	0

Table 3: Hardware specifications for my personal laptop.

Once the attack proved successful on MNIST and experiments began with `ResNet-50`, it became evident that the personal computer was inadequate for the increased computational demand. For example, performing a complete **grid search** with this setup required more than six days to finish, making it impractical for systematic evaluation.

To address this limitation, the experiments were migrated to the **research group's cluster**. This cluster provided 8 CPU-based compute units, which allowed a degree of parallelization. However, the absence of **GPU acceleration** meant that the expected speedup was limited. Although this configuration slightly reduced training times compared to the personal computer, it did not provide the necessary computational efficiency for large-scale experimentation. We can see the specs of the cluster in the following table:

Server (hostname)	Model	RAM per CPU	Memory (GB)	CPU Sockets	Cores per CPU	Threads per CPU	Total Cores	Total Threads	CPU
controller		32	1	6	12	6	12	Intel Xeon E5-2420 v2	
cloudfunctions			1	4	4	4	4	Intel Xeon E5-2407 0	
proxy2			1	4	4	4	4	Intel Xeon E5-2407 0	
storage0		15	1	4	4	4	4	Intel Xeon E5-2403 0	
storage1		15	1	4	4	4	4	Intel Xeon E5-2403 0	
storage2		15	1	4	4	4	4	Intel Xeon E5-2403 0	
storage3		15	1	4	4	4	4	Intel Xeon E5-2403 0	
storage4			1	4	4	4	4	Intel Xeon E5-2403 v2	
storage5			1	4	4	4	4	Intel Xeon E5-2403 v2	
storage6			1	4	4	4	4	Intel Xeon E5-2403 v2	
compute0	Dell PowerEdge R420		98	2	6	6	12	12	Intel Xeon E5-2420 v2
compute1	Dell PowerEdge R420		82	2	6	6	12	12	Intel Xeon E5-2420 v2
compute2	Dell PowerEdge R420		32	2	6	6	12	12	Intel Xeon E5-2620 v3
compute3	Dell PowerEdge R430		32	2	8	16	16	32	Intel Xeon E5-2620 v4
compute4	Dell PowerEdge R440	16	32	2	12	24	24	48	Intel Xeon Gold 5118
compute5	Dell PowerEdge R440	16	32	2	12	24	24	48	Intel Xeon Gold 5118
compute6	HPE DL365 Gen10+ 8SFF	64	128	2	32	64	64	128	
compute7	HPE DL365 Gen10+ 8SFF	64	128	2	32	64	64	128	

Table 4: Cloud Lab Server Hardware Specifications

Consequently, the final phase of the project relied on **external cloud computing resources**. By contracting GPU-enabled instances from AWS, it was possible to significantly accelerate training and grid search procedures, enabling the completion of experiments that would otherwise have been infeasible within the project timeline. The service hired was the Amazon EC2's G5 instance, referenced as **g5.xlarge**. These are its specs:

Compute	Value
vCPUs	4
Memory (GiB)	16
Memory per vCPU (GiB)	4
Physical Processor	AMD EPYC 7R32
Clock Speed (GHz)	2.8 GHz
CPU Architecture	x86_64
GPU	1
GPU Architecture	NVIDIA A10G
FLOPs (FP32)	31.5 TFLOPs
Video Memory (GiB)	24
GPU Compute Capability	8.6
FPGA	0

Table 5: Hardware specifications for `g5.xlarge`.

4.8.2 Software Setup

The experimental framework was implemented with:

- **Python**: version 3.12.4, with Conda for package management and environment isolation.
- **GPU Support**: CUDA toolkit version 12.5 for GPU acceleration enabling efficient deep learning computation.

The following core libraries have been used:

- **PyTorch**: Deep learning framework for model implementation and training.
- **Torchvision**: Computer vision utilities and pre-trained models.
- **NumPy and SciPy**: Numerical computing libraries.
- **Scikit-image**: Image processing and metrics computation (SSIM).
- **LPIPS**: Learned perceptual image patch similarity metric.
- **Matplotlib**: Visualization and result plotting.
- **Pandas**: Data manipulation and metrics analysis.

Reproducibility Setup: The code and testing have been developed with the following features to consistently reproduce test cases:

- **Version Control**: Git repository structure with systematic branching for code management and experimental tracking.
- **Environment Isolation**: Docker containers ensuring consistent execution environments across different hardware configurations.

- **Dependency Management**: Fixed package versions specified in `requirements.txt` preventing version conflicts and ensuring reproducible installations.
- **Random Seed Control**: Deterministic training and evaluation procedures through systematic seed initialization for PyTorch, NumPy, and random number generators.

Code Organization:

The experimental framework was structured using a modular architecture facilitating systematic experimentation:

- `attack.py`: Complete training pipeline for inversion model development across different ResNet-50 layers.
- `Model.py`: Neural network architecture definitions including ResNet modifications and custom inversion generators.
- `generate_metrics.py`: Comprehensive evaluation metrics computation including MSE, SSIM, LPIPS, and classification accuracy.
- `result_attack.py`: Visualization framework and result analysis tools for generating comparative plots and statistical summaries.
- `check_cuda.py`: Hardware detection and GPU utilization utilities for optimal resource allocation.

Specialized Tools:

- **Netron**: Neural network architecture visualization for model inspection and diagram generation
- **Playwright**: Automated browser control for programmatic diagram generation and screenshot capture
- **Docker**: Containerization framework for reproducible experiments using custom `Dockerfile` configurations

4.9 Results Visualization and Analysis

We perform both qualitative and quantitative evaluation of the reconstructed images to gain insights into model performance. Qualitatively, we examine example images by displaying original inputs alongside model reconstructions. This **qualitative inspection** is automated via Python scripts. For instance, the script `result_attack.py` runs the model under various settings to obtain reconstructed outputs. Another script `generate_graphs.py` then takes these outputs to overlay or compare images systematically (similar to approaches in Yin et al. [15]). In this way, visual comparisons highlight artifacts or distortions introduced by the model and help identify failure modes.

The following visualization types are generated to summarize the results:

- **Reconstructed images**: Sample original images and their corresponding reconstructions are shown side-by-side (or in montages) to qualitatively assess visual fidelity and artifact patterns.
- **Metric line plots**: We plot quantitative metrics (MSE, SSIM, LPIPS and classification accuracy) as line graphs, often across model layers or training epochs, to capture performance trends. For example, plotting SSIM vs. layer index reveals how structural quality changes with depth.
- **Boxplots of metrics**: For each layer, we plot the distribution of values across all reconstructed samples using boxplots. Boxplots reveal the *variability* of metrics (MSE, SSIM, LPIPS and classification accuracy), showing medians, interquartile ranges, and outliers. This provides a clearer view of consistency and dispersion in attack performance across layers.
- **Confusion matrices**: Color-coded matrices summarize classification or attack outcomes by comparing predicted labels to ground truth. Each cell (i, j) shows the count of instances of true class i predicted as class j, highlighting systematic errors or biases.

Quantitative evaluation uses several complementary metrics. The mean-squared error (MSE) captures the average pixel-wise reconstruction error. The Structural Similarity Index (SSIM) measures perceptual similarity by accounting for luminance, contrast, and structural information between images; higher SSIM indicates images are more perceptually alike. LPIPS (Learned Perceptual Image Patch Similarity) measures distance in a deep feature space, extracted from a pretrained network, and correlates with human judgments of image similarity. Together, SSIM and LPIPS capture perceptual aspects of image quality beyond raw pixel error. We compute these metrics for each image and plot them (using `generate_graphs.py`) as a function of model layer or other variables. This multilayer plotting is inspired by recent studies such as Yin et al. [15], which use per-layer metric curves to diagnose model behavior.

Confusion matrices provide a complementary view for classification results or model attacks. We use the script `generate_conf_matrix.py` to compute and plot these matrices from the model's predicted labels. Confusion matrix analysis (**classification analysis**) intuitively shows which classes are often confused or whether an attack succeeds (on sample vs off sample). This form of visualization is widely recommended for diagnosing model performance; for example, ML-Doctor [8] and Ovchinnikova et al. [11] both employ confusion heatmaps to evaluate image-based classifiers. Such matrices make it easy to spot class-specific biases or failure patterns.

In summary, the results are visualized through a combination of qualitative image comparisons, quantitative metric plots, and confusion matrices. This multi-faceted approach, supported by the automated scripts `result_attack.py`, `generate_graphs.py`, and `generate_conf_matrix.py`, provides a thorough analysis of reconstruction quality and classification accuracy, aligning with best practices in recent literature [8, 11, 15].

4.10 Limitations and Considerations

The methodology presented here has several limitations and considerations that were addressed during the study. We discuss major challenges and our mitigation strategies in the following items:

- **Computational resources:** The experiments were limited by the **lack of high-performance hardware**, since there is no GPU on the laptop or the CloudSking cluste, resulting in extremely long training times, on the order of hours or even days, for each `grid_search` run. To overcome this, we **migrated training to AWS GPU instances** (`g5.xlarge`), which significantly reduced execution time. For example, a single full grid search required total training time $T \approx 48{,}324\,\mathrm{s}$ ($\approx 13.4\,\mathrm{h}$), which on the CPU-only machines would have been tens of hours longer. This reduction in time came at a financial cost, as described below. On images 16 and 17 you can appreciate how much time takes to train an attack model per targeted layer.

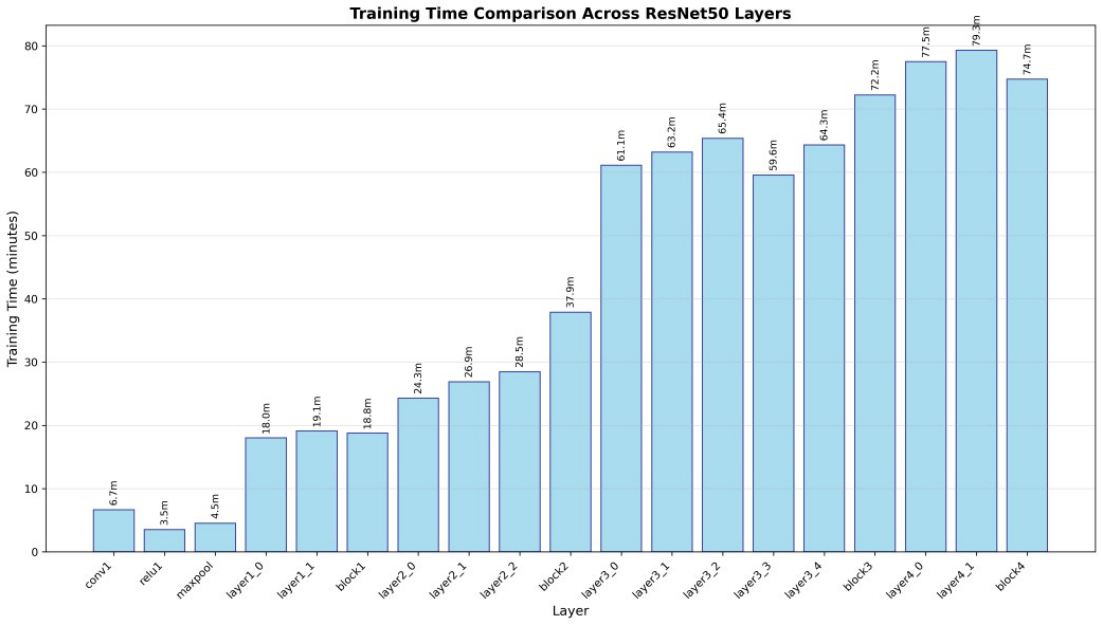


Figure 16: Training time comparison across attacked layers in minutes

- **Overfitting risk:** Given the model complexity and dataset size, we observed tendencies to overfit during training. We addressed this by using **early stopping** (with *patience* parameter) and extensive hyperparameter tuning, carefully searching learning rates, weight decay (TV loss), and batch sizes via grid search to find a model that generalizes well. However, each additional trial of patience and grid search further increased computation time.

- **Financial cost:** Using cloud GPUs incurs a non-trivial monetary cost, which affects the feasibility of experiments. The AWS `g5.xlarge` instance costs about**$1.006 per hour**. Thus, a single grid search (which took roughly 13.4 hours) cost approximately

$$C \approx 1.006\ \$/\mathrm{h} \times 13.4\ \mathrm{h} \approx \$13.50.$$

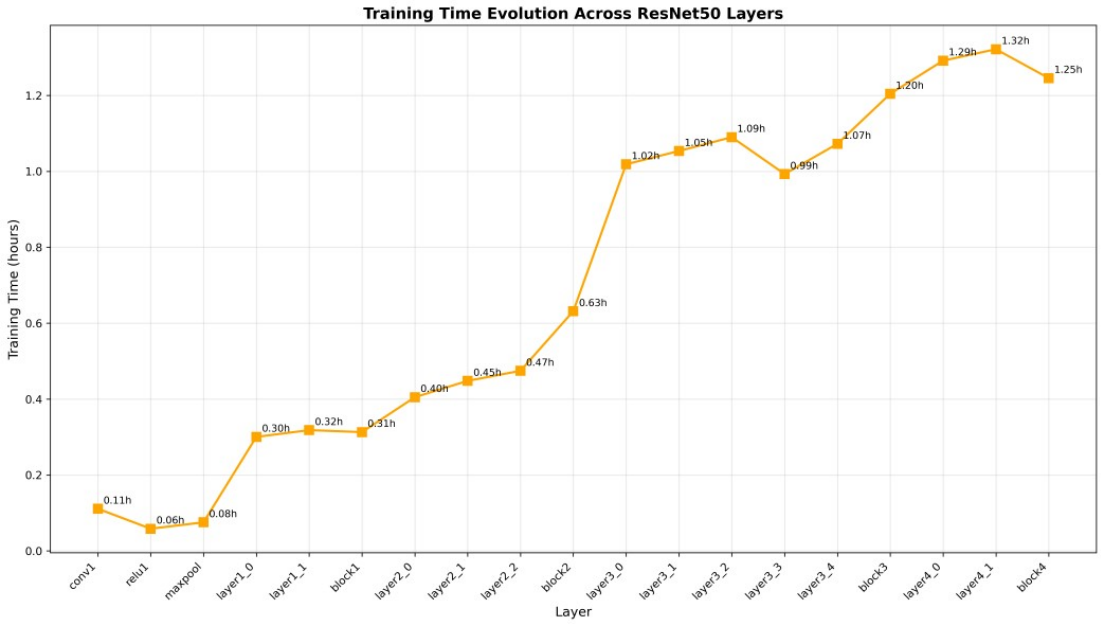


Figure 17: Training time comparison across attacked layers in hours

We performed on the order of 5 to 10 such grid searches for tuning, implying a total expenditure of roughly **$67–$135**. This level of cost limits the reproducibility and scalability of the approach without substantial funding.

5 Results

In the following subsections, the metrics related with similarity between reconstructed images and originals and related with accuracy of classification on/off sample problem are shown. The aim is to understand the **evolution of the performance of the attack across de different layers** of the targeted Resnet50. In addition, to similarity we evaluate the capacity of the reconstruction attack to generate images that will be classified as the same class than the original one (on sample vs off sample) by the original Resnet50 classifier net. metrics All the raw data can be consulted on table 6 and table 7.

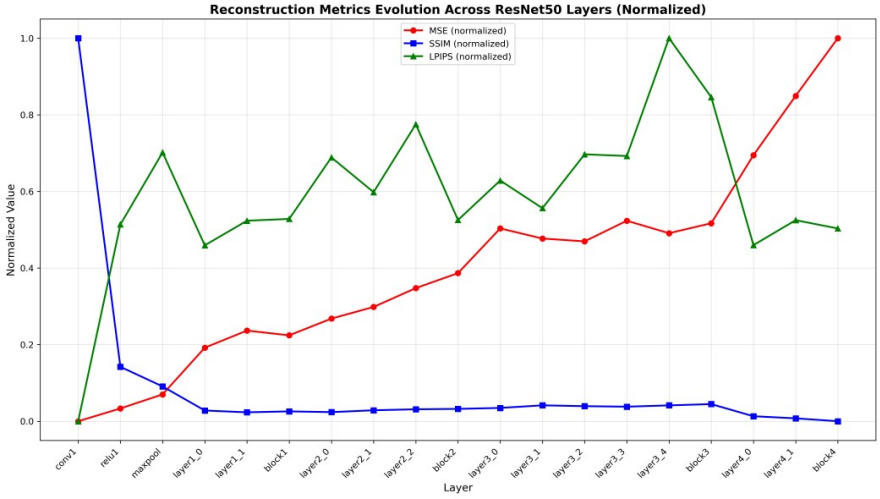


Figure 18: Normalized metrics evolution per layer attacked

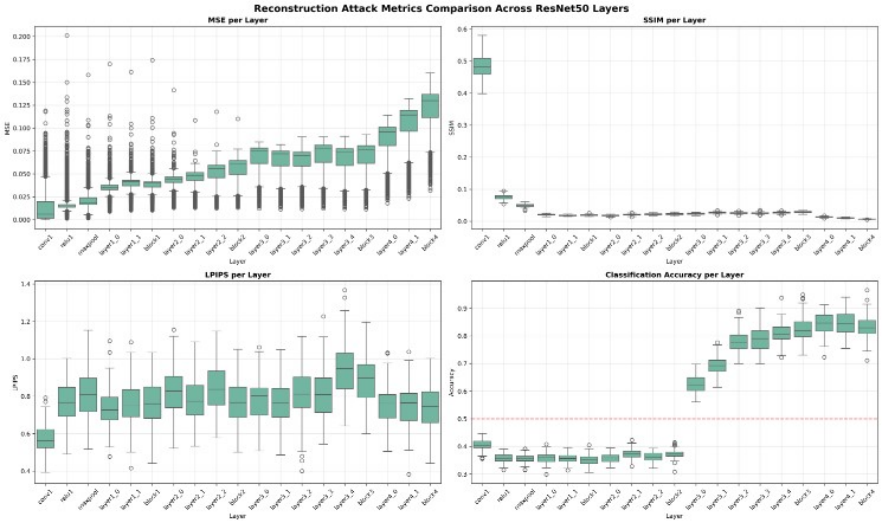


Figure 19: Reconstruction attack metrics comparison across layers

5.1 Mean Squared Error (MSE)

In order to quantify the reconstruction fidelity of our inversion attacks, we evaluated the **Mean Squared Error (MSE)** between the reconstructed images and the original training samples across different victim layers of the `ResNet-50` classifier. The values obtained from our experimental runs are consistent with the general trend highlighted in the literature, and particularly align with the observations made in the **Ginver** study [15], where the authors reported that reconstruction quality, as measured by MSE, systematically worsens as the split point of the model moves deeper into the network.

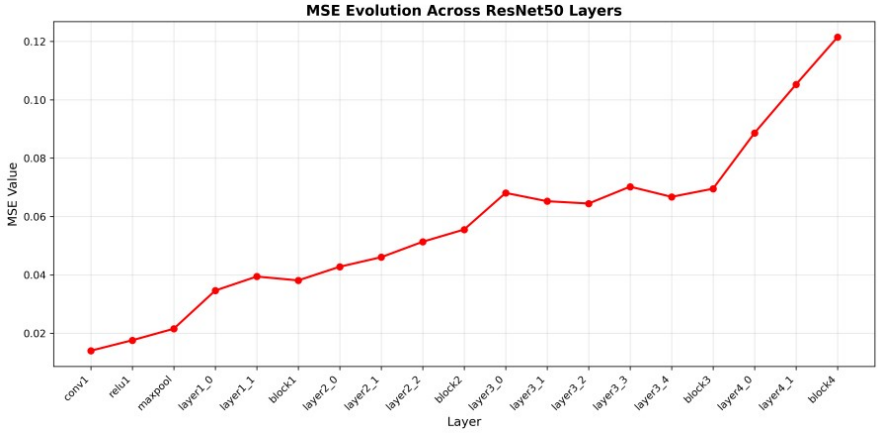


Figure 20: MSE evolution per layer attacked

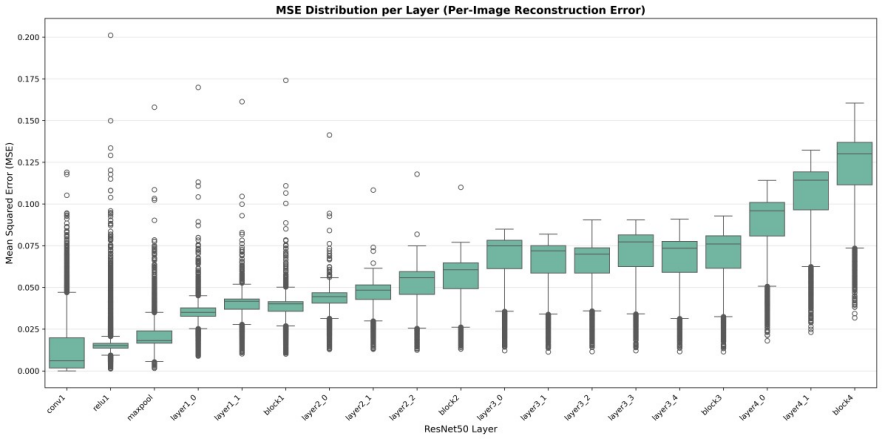


Figure 21: MSE distribution per layer

The results, searchable in table 6, show a **clear monotonic increase in MSE layer by layer**. For shallow layers, such as `conv1` and `block1`, the MSE values remain relatively low, reflecting that reconstructions from these early feature maps preserve pixel-level similarity with the original input. However, as we progress to deeper blocks such as `block2`, `block3`, and finally `lblock`, the MSE values become progressively higher. This trend quantitatively confirms that the deeper the layer chosen as the inversion target, the more information is lost in the intermediate representation and the more challenging the reconstruction task becomes.

Interestingly, this degradation is not linear: the largest jumps in MSE are observed when transitioning from early convolutional blocks to mid-level residual blocks. This aligns with the intuition behind **residual learning** [12], where deeper layers capture increasingly abstract semantic features rather than fine-grained pixel information. Thus, even though visually some reconstructions from deeper layers may still appear recognizable, the pixel-wise discrepancy measured by MSE becomes more pronounced.

In summary, our results corroborate the claim made by Yin et al. [15] that **MSE increases with depth**, highlighting that from a purely quantitative perspective, inversion from shallow layers yields more faithful reconstructions. This provides empirical support for the argument that **placing deeper layers inside a TEE** (Trusted Execution Environment) reduces measurable information leakage at the pixel level. However, as will be discussed later, human perceptual similarity does not always follow this strictly monotonic trend.

5.2 Structural Similarity Index Measure (SSIM)

The **Structural Similarity Index Measure (SSIM)** provides a complementary perspective to MSE by evaluating the preservation of luminance, contrast, and structural patterns between original and reconstructed images. Using the values searchable on table 6, we observe a **progressive decline in SSIM across deeper layers** of the `ResNet-50`, mirroring the trend documented in the **Ginver** work [15].

For early layers such as `conv1`, the SSIM values are relatively high, indicating that reconstructions from these shallow features still maintain strong perceptual similarity to the original inputs. However, as we move into deeper layers like `block1` and `block2`, the SSIM scores begin to decrease, reflecting that intermediate representations lose fine-grained structural information. This degradation becomes most pronounced in the deepest residual blocks, `block3` and `block4`, where SSIM values are significantly lower, confirming that reconstructions from these stages fail to preserve detailed structural correspondence.

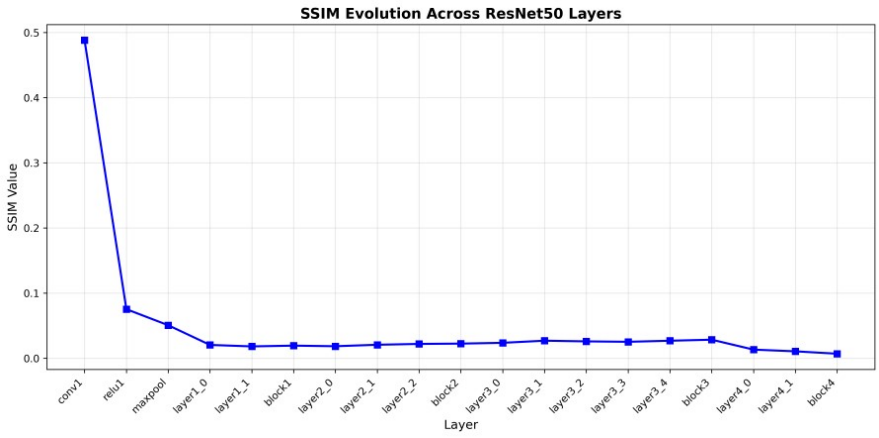


Figure 22: SSIM evolution per layer attacked

The decreasing SSIM trend reinforces the observation that **deeper layers cap-**

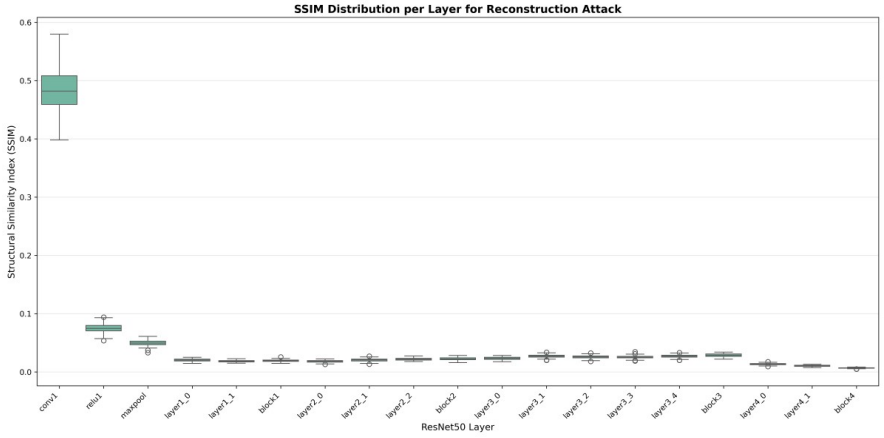


Figure 23: SSIM distribution per layer

ture more abstract and semantic features at the expense of pixel-level structural fidelity. While human observers may still perceive some reconstructions from mid-level blocks, such as `block2`, as recognizable, the SSIM metric quantitatively demonstrates the gradual erosion of structural similarity. This quantitative decline provides further evidence that deeper splits within a **Trusted Execution Environment (TEE)** reduce measurable leakage in terms of structural resemblance, consistent with the security arguments proposed in prior studies [12].

In summary, the SSIM results confirm that **structural fidelity decreases layer by layer**, fully consistent with the claim made by Yin et al. [15]. This supports the notion that adversaries face greater challenges in reconstructing structurally faithful images from deeper intermediate representations, even if the reconstructions remain semantically interpretable to humans.

5.3 Learned Perceptual Image Patch Similarity (LPIPS)

The **Learned Perceptual Image Patch Similarity (LPIPS)** metric was incorporated into our evaluation as a complement to **MSE** and **SSIM**, with the specific aim of capturing **human perceptual similarity** rather than pixel-level correspondence. This makes LPIPS particularly relevant for assessing the privacy risk of inversion attacks, since adversaries ultimately exploit human-recognizable features.

Unlike MSE and SSIM, which showed clear monotonic trends (increasing error and decreasing similarity, respectively, as the layer depth grows), the LPIPS results searchable in table 6, exhibit a much more **irregular pattern across layers**. Specifically, we find that the lowest LPIPS values (indicating the highest perceptual similarity) occur at both the very shallow layer (`conv1`) and again after the deeper residual block `block3`. Conversely, the highest LPIPS values, reflecting the strongest perceptual dissimilarity, appear at `block2`.

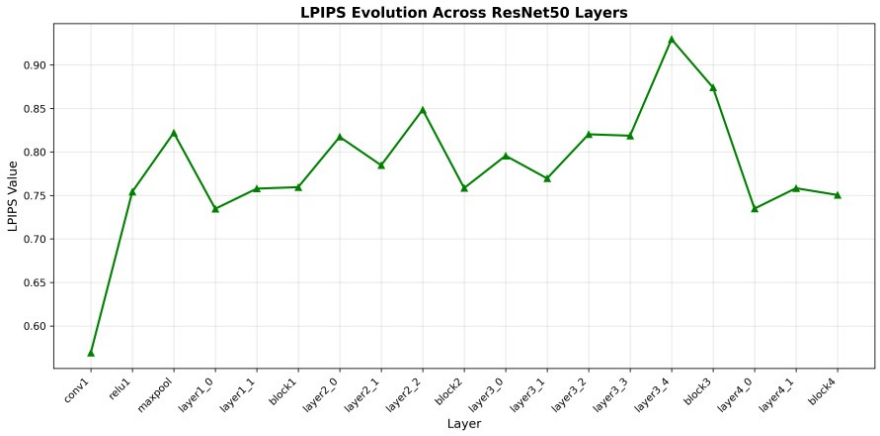


Figure 24: LPIPs evolution per layer attacked

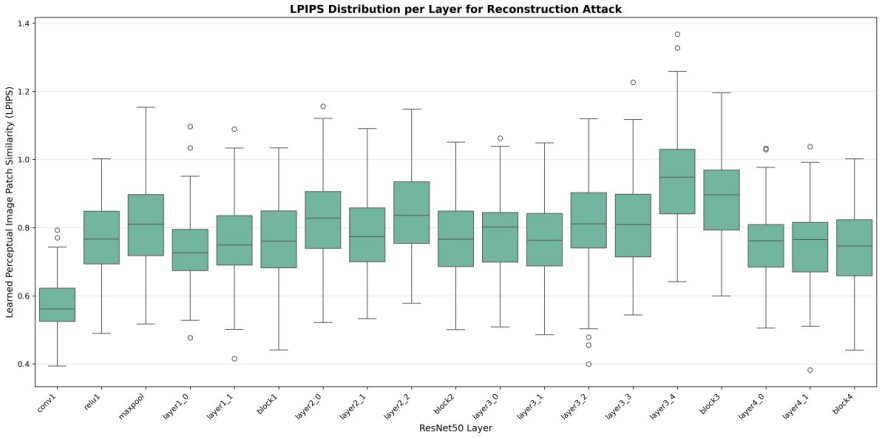


Figure 25: LPIPS distribution per layer

This non-monotonic trend suggests that reconstructions from certain mid-level features (particularly `block2`) are perceived as less similar to the original inputs than either earlier or later layers. This finding contrasts with the monotonic degradation captured by traditional metrics and highlights that **perceptual quality does not always decrease linearly with layer depth**. Instead, the perceptual recognizability of reconstructions depends heavily on the type of features encoded at different depths. For example, `block2` emphasizes complex mid-level patterns, which appear to be harder to invert into perceptually faithful reconstructions, while both early edges (from `conv1`) and higher semantic structures (from `block3` or beyond) yield reconstructions that align more closely with human visual perception.

In summary, the LPIPS metric provides crucial evidence that **human visual similarity is not captured by monotonic trends in pixel-based metrics**. Our results show that reconstructions from both very shallow and relatively deep layers can remain perceptually coherent, whereas mid-level features may paradoxically yield the least perceptually similar images. This reinforces the need to include perceptual metrics such as LPIPS when assessing the real privacy risks of inversion attacks, as relying solely on MSE or SSIM could lead to misleading conclusions about human-perceived similarity [8, 15].

5.4 Prediction accuracy

To complement the pixel-level and perceptual similarity measures, we also assessed the **prediction accuracy** of the reconstructed images when re-classified by the original `ResNet-50` on the `on-sample vs off-sample` task. This metric evaluates whether the reconstructed samples preserve the semantic information needed for the model to assign the correct label, thus providing additional insight into the practical privacy risks of the inversion attack [8, 15].

The results searchable in table 7 reveal a **non-monotonic trend across layers**. Specifically, in the first half of the `ResNet-50`, such as `conv1`, `block1`, and `block2`), prediction accuracy remains very low, around 40% on average. This indicates that reconstructions from these early feature maps, although perhaps pixel-wise closer to the original inputs, often fail to capture the semantic cues necessary for correct classification by the original network.

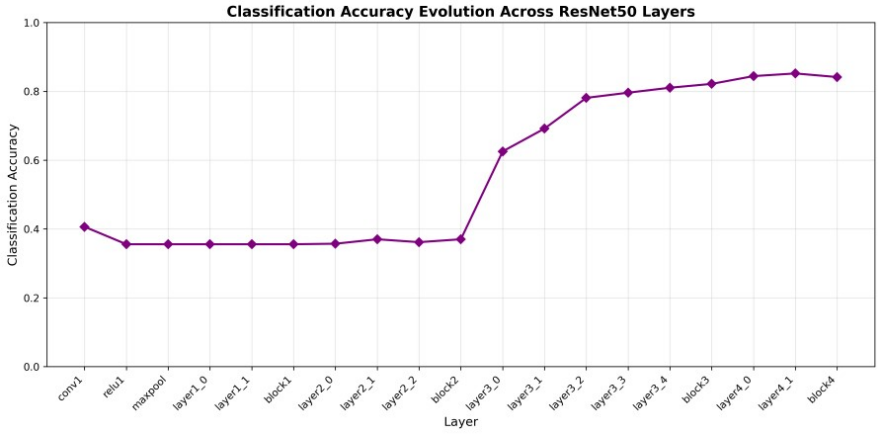


Figure 26: Prediction accuracy per layer attacked

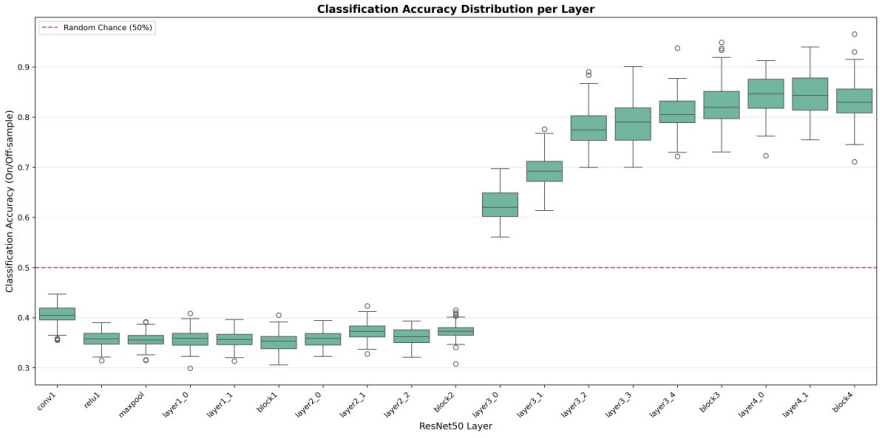


Figure 27: Classification accuracy distribution per layer

However, as we move into deeper layers such as `block3` and `block4`, the accuracy values increase substantially, reaching levels close to 80%. This improvement demonstrates that reconstructions from deeper layers, while structurally less faithful as mea-

sured by SSIM or pixel similarity, are more **semantically coherent**. In other words, the reconstructed images at these depths are more likely to be classified into the same category as their originals, which highlights their higher risk in terms of privacy leakage.

Overall, these findings emphasize that **prediction accuracy offers a complementary view** to MSE, SSIM, and LPIPS. Whereas the latter capture pixel-level and perceptual similarity, prediction accuracy directly reflects the model's ability to recover class-discriminative features from reconstructed samples. In our study, the sharp rise in accuracy in deeper blocks underscores the fact that reconstructions from these layers pose a greater **privacy threat**, since they preserve the semantic information needed for the target model to make consistent decisions.

5.5 Confusion Matrix

To further analyze the semantic coherence of the reconstructed images, we evaluated the **confusion matrices** obtained by re-classifying them through the original `ResNet-50` on the `on-sample vs off-sample` task. The values searchable in table 7 include `true positives (TP)`, `true negatives (TN)`, `false positives (FP)`, and `false negatives (FN)` for blocks `1-4`. These quantities summarize the classification outcomes: **TP** and **TN** correspond to correct predictions (on-sample identified as on-sample, off-sample identified as off-sample), whereas **FP** and **FN** capture misclassifications.

The results reveal that in the **first half of ResNet-50** (blocks `1` and `2`), the confusion matrices are dominated by misclassifications, with relatively low values of TP and TN and high counts of FP and FN. This indicates that reconstructions from shallow layers fail to retain the semantic information needed for reliable classification. For example, reconstructed on-sample images are often misclassified as off-sample (FN), and off-sample images are frequently mistaken for on-sample (FP).

In contrast, in the **second half of ResNet-50** (blocks `3` and `4`), the confusion matrices display a substantial increase in correct predictions. TP and TN values rise significantly, while FP and FN counts decrease. This shift demonstrates that reconstructions from deeper layers capture more discriminative features, enabling the classifier to correctly identify whether the input belongs to the on-sample or off-sample category. The higher TP and TN rates in deeper layers are consistent with the increased prediction accuracy reported in the previous subsection.

Overall, the evolution of the confusion matrices across layers **reinforces the findings of the prediction accuracy metric**. Reconstructions from shallow layers produce poor semantic fidelity and yield uninformative confusion matrices, whereas reconstructions from deeper layers exhibit much stronger alignment with the ground-truth categories. This result highlights that **deeper intermediate representations pose a greater privacy risk**, as they enable adversaries to reconstruct inputs that are not only perceptually convincing but also semantically consistent when re-evaluated by the original classifier [8, 15].

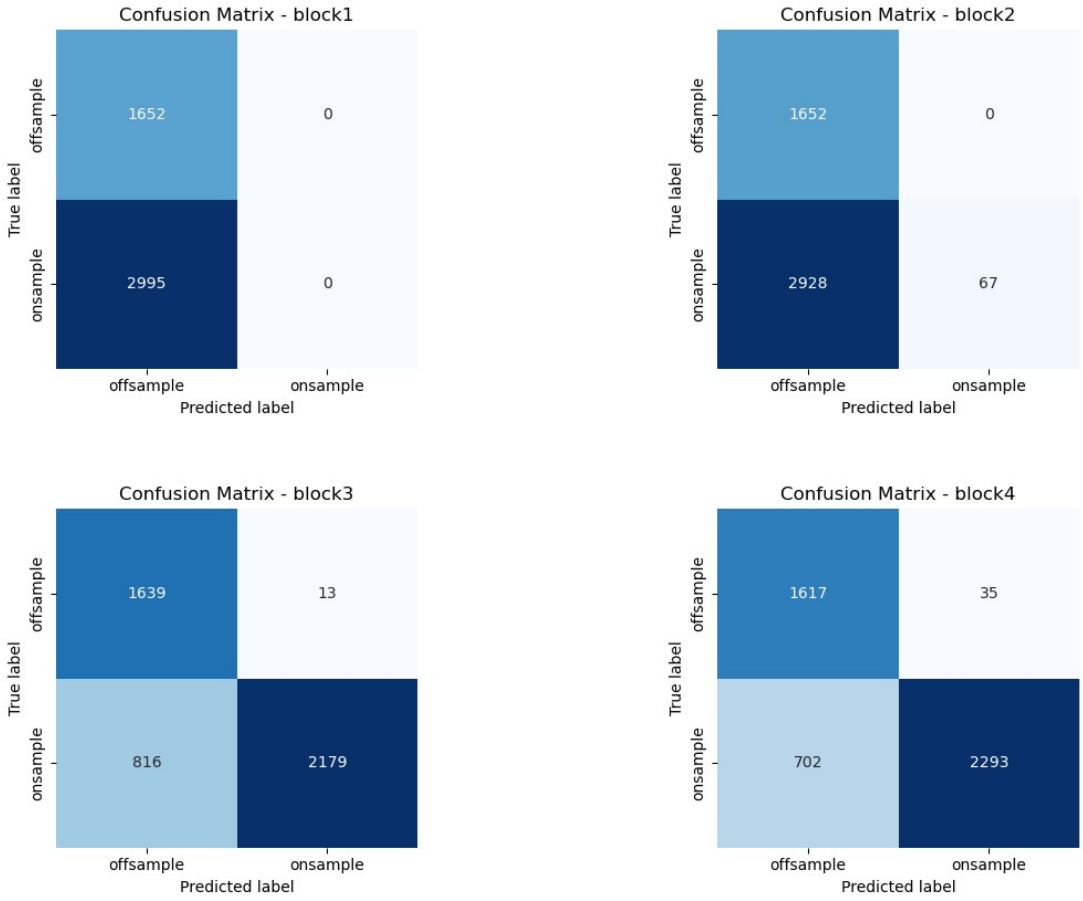


Figure 28: Confusion Matrices for Classification Accuracy across ResNet50 residual blocks

6 Discussion

6.1 White-box scenario

In the **white-box** setting, the adversary knows the full architecture and can obtain protected model's output of the target network (ResNet-50). We therefore construct a separate **layer-specific** mirrored decoder for each layer, using the known operations, such as convolutions, pooling, activations, of that layer to guide reconstruction. Intuitively, **having full knowledge should improve inversion quality**: for example, Liu *et al.* note that an attacker with white-box access generally achieves higher recovery performance than with only black-box access [8]. However, our experiments show that, in practice, **the ResNet-50 reconstructions were much poorer than expected**. In contrast to simple cases like MNIST, where even deep-layer inversions yield recognizable digits, we found that most inversions from ResNet-50 layers were blurry and unrecognizable. This suggests that complex architectures like ResNet-50 are intrinsically harder to invert faithfully. Indeed, recent inversion studies note that reconstruction quality *degrades* as model depth and complexity grow.

During model inversion training via grid search, we monitored the mean squared error (MSE) of the reconstructions. Recall the **MSE** loss for an image reconstruction is given by

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2,$$

where x_i are pixel intensities of the original image and $\hat{x}_i$ those of the reconstruction. A high MSE implies a poor reconstruction. In our white-box experiments, we observed very large MSE values for early layers, indicating huge pixel-wise errors. Concretely, the results were:

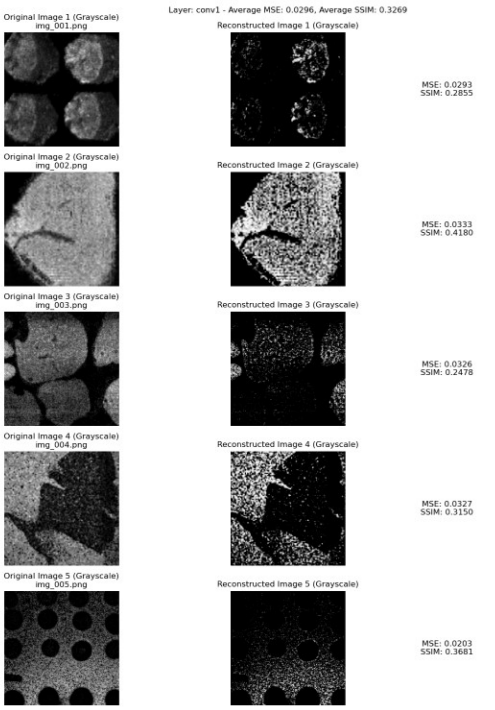
- **conv1** (first convolution) layer: The white-box reconstruction was noticeably better. Its MSE was significantly lower than for deeper layers, and the inverted image had much more similarity to the input (edges and basic shapes were visible). In fact, for `conv1` the white-box result exceeded the black-box baseline in quality.
- **MaxPool** layer output: The inversion loss remained extremely high (indicating large differences), and the reconstructed image bore no clear resemblance to the original. In other words, the decoder could not recover meaningful structure from the max-pooled features.
- Deeper layers (beyond **conv1**): Performance rapidly deteriorated again. MSE grew and output images became essentially noise. Because of these failures, we did not pursue further white-box inversions on later layers.

In summary, aside from the very first layer, the white-box decoders failed to reconstruct inputs effectively: high MSE early on meant no visual similarity to the originals.

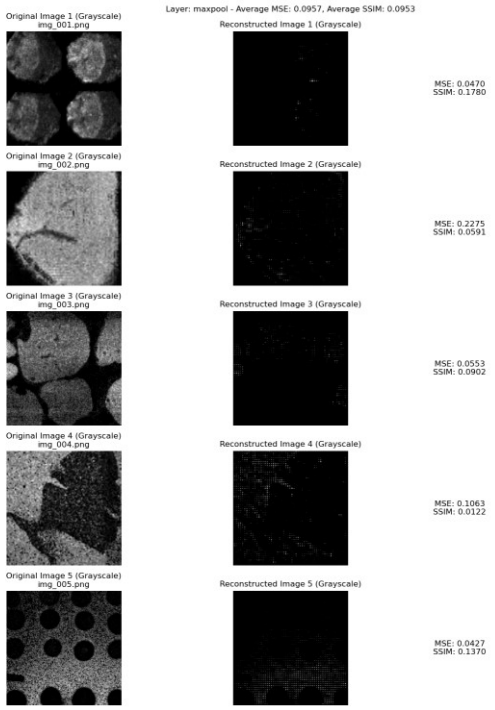
One likely reason for the white-box attack's failure on ResNet-50 is the presence of **non-injective** operations, such as `MaxPool`, in its architecture that discard information

and prevent accurate reconstruction from intermediate representations. This implies that constructing a fully inverted network architecture beyond such **non-invertible** layers is unfeasible. This explains why our white-box attack succeeded on the MNIST model, where all operations were invertible, but failed on ResNet-50; in the latter, only the earliest layer like `conv1` remains partially invertible, allowing shallow features to yield some recognizable reconstruction of the input. These observations align with Chen et al. [2], who note that non-injective network layers give rise to a space of *harmless perturbations*—distinct inputs that produce the same output—making exact inversion beyond those layers fundamentally impossible.

These **disappointing results for the white-box scenario** prompted us to abandon it for deeper layers. Instead, we shift to the black-box setting for those layers, following recent work of Yin *et al.* [15]. In other words, rather than building more mirrored decoders, we employ a generative inversion attack against the network outputs. This decision is also in line with approaches in privacy-preserving inference: exposing intermediate features (as would happen with white-box decoding) is known to be vulnerable to reconstruction [12]. Thus, moving to a black-box inversion strategy for deeper layers mitigates the clear-text exposure of early representations and follows the methodology of Ginver [15].



(a) Reconstruction attack in white-box scenario for layer conv1



(b) Reconstruction attack in white-box scenario for maxpool layer

6.2 Black-box scenario

Contrary to intuition, our experiments show that **black-box inversion attacks**, which assume no knowledge of the target layer's architecture, can actually produce more recognizable reconstructions than white-box attacks. This result is particularly

striking, as **black-box methods leverage significantly less information**. A similar phenomenon was noted by Arif *et al.*, who observed "surprising results in high-quality input data recovery" from intermediate ResNet outputs, explaining it by the fact that a residual block's output is essentially a noisy copy of its input. Likewise, **recent generative inversion approaches** [15], which are effectively black-box since they train solely on input–output pairs, have achieved semantic reconstruction quality nearly on par with white-box methods. In our case, **black-box reconstructions from ResNet-50's third block** have shown higher similarity in terms of human visual perception than the white-box reconstructions, even though the attacker had no architectural knowledge.

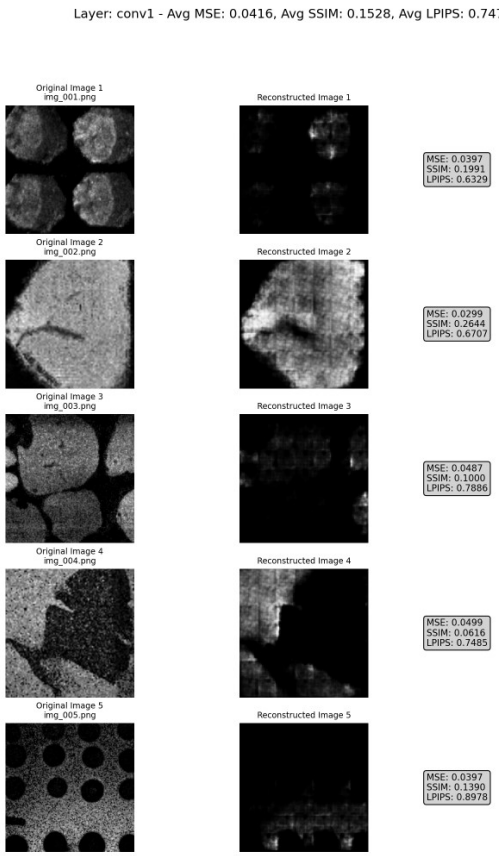
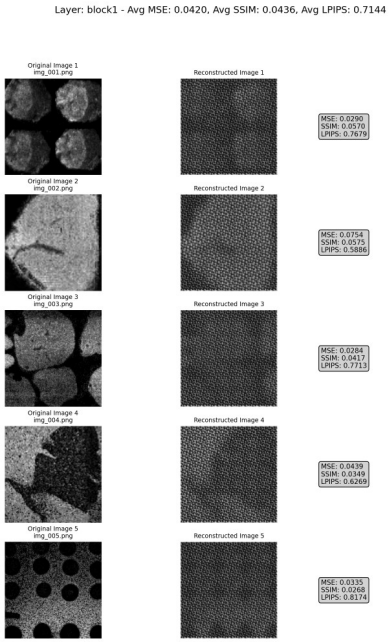
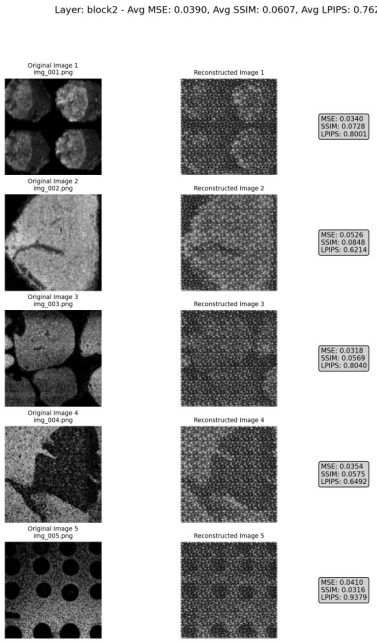


Figure 30: Reconstruction attack in black-box scenario for conv1 layer

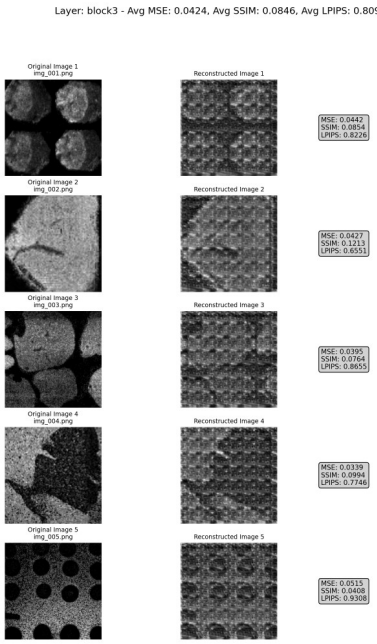
Notably, standard quantitative metrics fail to capture this effect. We find that metrics like **MSE**, **SSIM**, and **LPIPS** generally worsen for deeper layers, suggesting poorer reconstructions. Yet, **human observers** often found the `block-3` reconstructions more interpretable than those from `blocks 1 or 2`. This **misalignment** is well-documented: hand-crafted metrics frequently give weak or even contradictory signals relative to human judgment of image quality [15]. In fact, an image that is clearly recognizable to a person can receive higher SSIM or PSNR than a blurrier image, precisely because these metrics do not fully capture semantic content. Thus, relying solely on **pixel-level fidelity** to gauge privacy risk is misleading; **low metric scores do not guarantee that private information is hidden from human eyes** [8, 12].



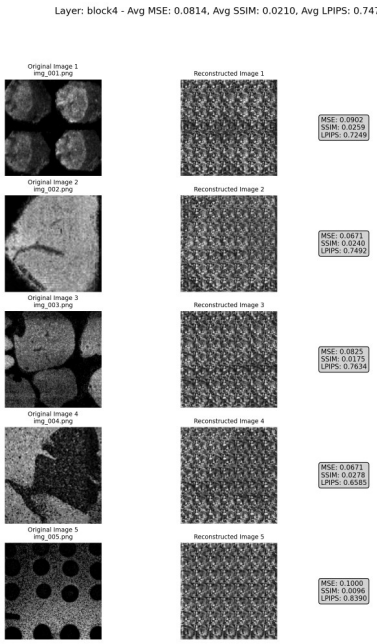
(a) Reconstruction attack in black-box scenario for block1



(b) Reconstruction attack in black-box scenario for block2



(c) Reconstruction attack in black-box scenario for block3



(d) Reconstruction attack in black-box scenario for block4

The practical implication is that simply shifting more layers into the **Trusted Execution Environment (TEE)** does not automatically improve privacy [12]. One might assume that protecting deeper layers, enclosing `block 3` within the enclave, would leave only higher-level features exposed and thus be safer. However, our findings suggest the opposite: the features output by `block 3` remain quite **human-interpretable**, so including `block 3` in the enclave does not necessarily reduce the information leaked by the remaining layers any more than protecting `blocks 1-2` would. In other words,

the **"depth" of the split does not guarantee security** if the exposed features themselves carry rich semantics. This echoes the intuition from **residual networks** that even high-depth outputs can reveal inputs almost entirely [8, 15].

Including more layers inside the TEE does not necessarily imply greater security is supported by the **prediction accuracy** obtained in the `on-sample vs. off-sample` classification task. Specifically, the accuracy for the first two convolutional blocks remains very low, around 40%, whereas from the third block onwards the accuracy rises to values close to 80%. This observation is further reinforced by the analysis of the **confusion matrices**, which detail how the attack models classify reconstructed samples and highlight the improved semantic coherence of deeper reconstructions. These findings may also be connected to the **greater visual similarity perceived by humans** in reconstructions generated from deeper layers, suggesting that semantic information becomes more recoverable as the network progresses.

Starting at ResNet-50's deeper layers (Block3 and beyond), the confusion matrices reveal a dramatic improvement in distinguishing **on-sample vs. off-sample inputs**. In these stages, the classifier achieves very high true positive and true negative rates for the on/off-sample classification task, reflecting its robust ability to correctly identify most genuine inputs and reject spurious ones. Notably, the model adopts a conservative stance: a non-trivial fraction of actual **on-sample images get flagged as off-sample (false negatives)**, while virtually **no off-sample images are misclassified as on-sample**. This asymmetry suggests the decision boundary is tuned to **avoid false positives** at the expense of some false negatives, ensuring that any image predicted as on-sample is highly likely to be a valid (in-distribution) sample. An attacker can directly exploit this behavior. By running reconstructed images through this classifier and discarding all reconstructions labeled as off-sample, the attacker would retain only those outputs confidently marked as on-sample; which, given the classifier's bias, are overwhelmingly realistic reconstructions. This means subsequent attack stages, such as training a secondary model or refining the reconstruction, can be performed exclusively on plausible, in-distribution inputs, free from the contamination of obvious off-sample outliers. In essence, the attacker leverages the on/off-sample classifier as a quality filter (analogous to off-sample detection methods like OffsampleAI[11]) to focus further attacks solely on realistic inputs, thereby maximizing the efficacy of the black-box attack pipeline.

This behavior aligns with what we know about **CNNs**: early convolutional layers extract very low-level cues, such as edges, colors, and simple textures, while deeper layers gradually assemble these into higher-level patterns and object parts [15]. In `ResNet-50`, for example, the `third block`'s filters respond to more complex shapes or object fragments than the first two blocks. Consequently, reconstructions from `block 3` tend to capture the overall layout and identity of the scene, making them *semantically coherent*, even if fine pixel detail is lost. Humans naturally latch on to these semantic signals, so deep-layer reconstructions can "make sense" visually despite poor **MSE** or **SSIM**. This explains why images inverted from `block 3` looked more recognizable to people than those from shallower blocks. It follows that deeper layers encode more **semantic information** and are intrinsically more interpretable, underscoring why

conventional metrics alone are insufficient for assessing privacy leakage [8, 12].

Our findings underscore the **power of a generic inversion model**. In the style of Yin *et al.*'s **Ginver attack** [15], we train one generator that can invert feature maps from **any layer**. As Ginver's authors note, once trained, "Ginver can infer the victim's unseen model inputs without remaking the inversion attack model." This **generative approach** means the adversary need not build a new network for each target layer or architecture, greatly improving **scalability**. At the same time, however, we must heed Ginver's own **limitations**. Despite claiming applicability to `ResNet-50`, Yin *et al.* only evaluated their method on toy CNNs: a `6 conv + 2 FC` network on CIFAR-10 and a `7 conv + 2 FC` network on FaceScrub, and presented only one white-box `ResNet-50` example. Even that `ResNet-50` result produced only "blurry" reconstructions. In short, their evidence for **real-world deep nets** is weak, which likely explains why our **black-box inversion on ResNet-50** yields different and often better reconstructions.

More importantly, the **threat is real**. We routinely obtain **moderate-to-high-fidelity reconstructions** even in the **black-box setting**, which represents a serious privacy breach. Liu *et al.* observe that **any leakage about membership** in a trained model can expose sensitive information [8]. For instance, knowing that a patient's record was used in a drug-dose model already reveals their health status. Our own experiments on **pharmaceutical data** confirm that protected attributes can be recovered from outputs, and they do so cheaply; only a few hundred euros of GPU time. Thus, **model inversion in black-box settings is not merely academic**: it directly endangers private training data [12, 15].

Given this, defenders can't ignore **inversion attacks**. Even if an attacker lacks detailed architecture knowledge, **model owners should proactively measure privacy leakage**. One practical **"blue team" strategy** is to run an inversion attack against our own model, layer by layer, and see which splits reveal the most. This empirical test identifies exactly which intermediate feature maps leak input information. In the **Trusted Execution Environment (TEE)** context, this knowledge guides how far into the network to keep layers encrypted. Indeed, recent **TEE-Partitioning work** prescribes splitting a DNN into a small **"privacy-sensitive" part**, kept inside the enclave, and a larger **"insensitive" part** on the GPU [12]. By analogy, we suggest using our **reconstruction attack** to find the optimal split: keep all layers that yield high-quality reconstructions and thus high privacy risk inside the TEE, and only offload the rest. Notably, in our tests we saw that attempting to invert deeper layers required much longer training, and as a consequence higher attack cost, and slower inference since deeper splits increase latency, meaning a clear **security–efficiency tradeoff**. Balancing this tradeoff is exactly the point of TEE partitioning. In fact, we advocate using such tools or our own **Ginver-based attack tests** [15] to empirically choose the safest partition. In summary, the existence of effective **black-box inversion**, even on sensitive pharma models [11], means we must defend. By measuring which layers leak the most and then shielding those layers inside the enclave, we can mitigate the threat while balancing performance [8].

6.3 Ethical and Social Implications

Our study underlines the **dual-use** nature of model inversion and reconstruction techniques. On one hand, developing these attacks serves a legitimate purpose: it helps security researchers and practitioners understand system vulnerabilities and fortify defenses. Indeed, academic analyses of inference attacks provide holistic risk assessments that guide safer model deployment [8]. Similar generative inversion attacks have been demonstrated in collaborative or split inference settings [15], highlighting that the threat we explore is not merely hypothetical. On the other hand, the same techniques could be weaponized by adversaries to compromise privacy. We therefore frame our work as a proactive security analysis aimed at revealing vulnerabilities so that they can be patched, rather than as tools for exploitation. All results are discussed in the spirit of transparency and **responsible disclosure**, ensuring that stakeholders are informed of risks without revealing unnecessary details to would-be attackers.

There are significant security risks for both individuals and organizations if such vulnerabilities are not addressed. A successful reconstruction of biomedical images could expose personal health data of patients or test subjects, breaching doctor-patient confidentiality and data protection agreements. In the pharmaceutical industry, reconstructed model inputs might reveal confidential drug discovery data or proprietary experimental results. We summarize a few key *potential consequences* of unchecked reconstruction attacks below:

- **Privacy violations:** Personal or sensitive information about individuals, such as patient medical conditions identifiable from recovered images, could be exposed, in contravention of privacy laws like GDPR (which can impose fines up to 4% of global annual turnover for serious infractions).
- **Intellectual property loss:** Proprietary data, such as pharmaceutical R&D findings, could be leaked, resulting in loss of competitive advantage and possible financial harm to organizations.
- **Legal and reputational damage:** Both healthcare providers and companies might face lawsuits, regulatory penalties, and reputational harm if they are found negligent in protecting data used in AI models.

Finally, it is noteworthy that the reconstruction attacks presented here achieved a high success rate despite being developed with limited resources within a student research setting. This underscores how even a relatively resource-constrained attacker can obtain alarming results. A more sophisticated adversary, equipped with greater expertise, computational power, and time could potentially achieve reconstructions of even higher fidelity, exacerbating the threat. This realization reinforces the urgency of addressing these vulnerabilities. In summary, our findings serve as a call to action for the field: to recognize the ethical and social implications of advanced ML models in sensitive domains and to proactively enhance their resilience against privacy-compromising attacks.

6.4 Future Work

Even with sensitive model layers protected inside a **TEE**, there remains room to enhance inversion attacks by exploiting any available **architectural knowledge** of the private model partition. In scenarios where an adversary knows the network architecture, but not the secret weights of the enclave-protected layers, they could design surrogate or generative models that mirror this structure to improve reconstruction fidelity. For instance, Yin *et al.* (2023) demonstrate in **Ginver** that an attacker can restore the original input from intermediate features in a split model by training a generative inversion network [15]. Extending this idea, a future attacker might leverage the known layer sequence (convolutions, activations, `maxpool` layers, etc.) to constrain their generative model, thereby better capturing how the protected layers transform inputs.

In addition, an important related direction is to address the challenges posed by **non-injective** operations in the model. Many architectures include layers like `maxpool` or dropout that irreversibly lose information, making exact inversion theoretically impossible. Future work could explore replacing or bypassing these non-injective layers in the attacker's reconstruction pipeline, for example, using learned upsampling, in order to achieve a closer approximation of the true inverse function. By ensuring that each forward operation has a corresponding invertible or at least approximable counterpart, one can push model inversion attacks closer to recovering inputs with high fidelity, even when faced with architectural components that normally obscure details.

Another vital avenue for research is investigating the fundamental trade-offs between **privacy** and **performance** in TEE-based model partitioning. Placing more layers inside a secure enclave provides stronger privacy by hiding a greater portion of the computation from the attacker, but this comes at the cost of increased computation time and communication overhead [12]. These **Trusted Execution Environments** introduce significant runtime overhead, so shifting the partition boundary deeper into the network to improve privacy can dramatically slow down inference. Conversely, shallow enclave partitions run faster but leave more informative intermediate features exposed to a potential adversary. Future research should build on these insights to determine optimal partition strategies: identifying which layer to split on to minimize reconstruction risk for a given overhead budget, or dynamically adjusting the partition based on the acceptable latency and threat level. Moreover, the attacker's own costs and limitations under TEE constraints merit careful study. If intermediate outputs can only be obtained via enclave calls with high latency and limited throughput, an adversary's ability to train or execute an inversion model at scale might be significantly hampered. An interesting research question is how this increased access latency impacts the feasibility of generative inversion attacks; for example, it may force attackers to use smaller query batches or limit the complexity of their generative networks to maintain reasonable attack timing. Incorporating such real-world constraints into holistic risk assessments will make evaluations more realistic.

7 Conclusion

In this thesis, we have conducted an extensive empirical study of black-box model inversion attacks on deep CNNs, such as ResNet-50, in a split-inference setting with Trusted Execution Environments (TEE) for privacy. Contrary to the claim by Yin et al. [[15]] that shifting the split point to deeper layers inherently improves privacy, our results show that deeper features can still betray sensitive content. In particular, inverted reconstructions from deeper layers, such as block 3 of ResNet-50, often appear more semantically coherent and recognizable to humans, even though objective measures (MSE, SSIM, LPIPS) report worse scores. This aligns with observations in other domains, like complex biological images [[11]]), that deep network representations preserve high-level structure even when pixel-level fidelity is low.

A key finding is that standard image-similarity metrics fail to capture the true privacy risk. High MSE or low SSIM can coexist with images that humans easily identify, meaning these metrics can substantially underestimate information leakage. Thus, protecting more layers in the TEE alone is not a panacea: even when three blocks are hidden, the intermediate outputs remain highly interpretable. Indeed, related work on layer partitioning [[12]] similarly reports that ResNet-50 features past block 3 can still yield recognizable content (SSIM still above trivial), indicating that privacy cannot be assured by naive metric thresholds. In other words, numeric thresholds on MSE/SSIM give a false sense of security when their values no longer correlate with human perception of the reconstructed images.

These insights motivate a more rigorous defense strategy. We advocate a "blue-team" inversion audit: model owners should actively test each layer's vulnerability by attempting reconstructions from its feature maps (as exemplified by comprehensive frameworks like ML-DOCTOR [[8]]). This layer-by-layer probing reveals exactly which intermediate outputs expose private details and thus helps set the enclave partition depth empirically. Our attack was demonstrated in a real-world use case, in collaboration with EMBL and the CloudLab team at URV, funded by the EU program, confirming that black-box inversion is practical and poses a serious privacy threat even with modest compute resources.

In conclusion, the choice of where to partition a model into a TEE should be guided by empirical evaluation, not by theoretical metric expectations. Defenses must account for human-perceived information leakage. In practice, this means validating any split against actual reconstructions: a design that appears secure under MSE/SSIM may still leak sensitive features. Our work reaffirms that robust privacy protection for split inference requires understanding what an adversary (or human reviewer) can see in the reconstructions, rather than relying solely on abstract similarity scores. Only with such a perception-driven evaluation can we ensure that TEE partitioning truly safeguards data confidentiality in real applications.

References

[1] Chang, Haonan ; Jia, Hengrui ; Yang, Yuxuan ; Wang, Bingyu ; Zhang, Tianwei: Ginver: Training-Free Collaborative Inversion Attacks Against Split Learning. In: *Proceedings of the ACM Web Conference 2023*, 2023, S. 1952–1962

[2] Chen, Lu ; Li, Shaofeng ; Huang, Benhao ; Yang, Fan ; Li, Zheng ; Li, Jie ; Luo, Yuan: Contrasting Adversarial Perturbations: The Space of Harmless Perturbations. In: *arXiv preprint arXiv:2402.02095* (2024). – URL https://arxiv.org/abs/2402.02095

[3] Chen, Si ; Kahla, Mostafa ; Jia, Ruoxi ; Qi, Guo-Jun: Knowledge-enriched distributional model inversion attacks. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, S. 9503–9513

[4] Fredrikson, Matt ; Jha, Somesh ; Ristenpart, Thomas: Model inversion attacks that exploit confidence information and basic countermeasures. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, S. 1322–1333

[5] Geiping, Jonas ; Bauermeister, Hartmut ; Drozdzal, Michal ; Uhrig, Jonas: Inverting gradients–how easy is it to break privacy in federated learning? In: *Advances in Neural Information Processing Systems*, URL https://arxiv.org/abs/2003.14053, 2020, S. 16937–16947

[6] He, Zecheng ; Zhang, Tianwei ; Lee, Ruby B.: Model inversion attacks against collaborative inference. In: *Annual Computer Security Applications Conference* ACM (Veranst.), 2019, S. 148–162

[7] Li, Ouxiang ; Wang, Xiao ; Huang, Jun ; Lee, Ruby B.: Model Inversion Attacks Through Target-Specific Conditional Diffusion Models. In: *arXiv preprint arXiv:2402.01193* (2024). – URL https://arxiv.org/abs/2402.01193

[8] Liu, Yang ; Ma, Xingjun ; Bailey, James ; Lu, Feng ; Zhang, Yonggang ; Qin, Zhan ; Ren, Kui: ML-Doctor: Holistic Risk Assessment of Inference Attacks Against Machine Learning Models. In: *USENIX Security Symposium* (2022), S. 4525–4542

[9] Nguyen, Ngoc-Bao ; Chandrasegaran, Keshigeyan ; Abdollahzadeh, Milad ; Cheung, Ngai-Man: Re-thinking model inversion attacks against deep neural networks. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, S. 14253–14262

[10] Nguyen, Ngoc-Bao ; Chandrasegaran, Keshigeyan ; Cheung, Ngai-Man: Label-only model inversion attacks via knowledge transfer. In: *Advances in Neural Information Processing Systems*, URL https://arxiv.org/abs/2305.17031, 2023

[11] Ovchinnikova, Katja ; Kovalev, Vitaly ; Stuart, Lachlan ; Alexandrov, Theodore: OffsampleAI: artificial intelligence approach to recognize off-sample mass spectrometry images. In: *BMC Bioinformatics* 21 (2020), Nr. 129, S. 1–11

[12] Rajasekar, Kishore ; Loh, Randolph ; Fok, Kar W. ; Thing, Vrizlynn L. L.: Privacy Preserving Layer Partitioning for Deep Neural Network Models. In: *arXiv preprint* arXiv:2404.07437 (2024)

[13] Wang, Kuan-Chieh ; Oreshkin, Boris ; Zhang, Neil: Variational model inversion attacks. In: *Advances in Neural Information Processing Systems*, URL https://arxiv.org/abs/2104.10459, 2021

[14] Yang, Ziqi ; Chang, Ee-Chien ; Liang, Zhenkai: Adversarial neural network inversion via auxiliary knowledge alignment. In: *arXiv preprint arXiv:1902.04290* (2019)

[15] Yin, Yupeng ; Zhang, Tianwei ; Wang, Dongxiao ; Liu, Zihao: Ginver: Generative model inversion attacks against collaborative inference. In: *Proceedings of the ACM Web Conference 2023*, 2023, S. 1952–1962

[16] Zhang, Yuheng ; Sheng, Yanzhi ; Chang, Xiaojun ; Qi, Guo-Jun: The secret revealer: Generative model-inversion attacks against deep neural networks. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, S. 253–261

[17] Zhao, Xuejun ; Yu, Han ; Zhang, Yang L.: Exploiting explanations for model inversion attacks. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, S. 934–943

[18] Zhu, Ligeng ; Liu, Zhijian ; Han, Song: Deep leakage from gradients. In: *Advances in Neural Information Processing Systems*, URL https://arxiv.org/abs/1906.08935, 2019, S. 14747–14756

Appendix A: Code appendix

```python
################################################################################
#                                                                              #
#        ResNet Inversion Model Arquitecture proposed by Ginver Paper          #
#                                                                              #
################################################################################

class ResnetInversion_Generic(nn.Module):
    # nc  ngf  nz
    def __init__(self, nc, ngf, nz):
        super(ResnetInversion_Generic, self).__init__()

        self.nc = nc
        self.ngf = ngf
        self.nz = nz

        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(nz, 8*ngf, 4, 2, 1),
            nn.BatchNorm2d(8*ngf),
            nn.Tanh(),

            nn.ConvTranspose2d(8 * ngf, 4 * ngf, 4, 2, 1), #28*28
            nn.BatchNorm2d(4 * ngf),
            nn.Tanh(),

            nn.ConvTranspose2d(4 * ngf, 2 * ngf, 4, 2, 1), #56*56
            nn.BatchNorm2d(2 * ngf),
            nn.Tanh(),

            nn.ConvTranspose2d(2 * ngf, ngf, 4, 2, 1), #112*112
            nn.BatchNorm2d(ngf),
            nn.Tanh(),

            nn.ConvTranspose2d(ngf, nc, 4, 2, 1), #224*224
            nn.Sigmoid()
        )

    def forward(self, x):
        # Get original batch size
        batch_size = x.size(0)

        # Check feature map size to adapt reshaping
        if x.dim() == 2:  # Already flattened
            x = x.view(batch_size, self.nz, 7, 7)
        elif x.dim() == 4:  # Has spatial dimensions
            # Get current spatial dimensions
            _, _, h, w = x.size()

            # If spatial dimensions need adjustment, do it properly
            if h != 7 or w != 7:
                # Resize using adaptive pooling to preserve batch size
                x = F.adaptive_avg_pool2d(x, (7, 7))

        # Ensure batch size is preserved
        x = x.view(batch_size, self.nz, 7, 7)

        return self.decoder(x)
```

Code 2: Definition of Resnet Model Arquitecture proposed by Ginver Paper

Appendix B: Results appendix

Layer	MSE	SSIM	LPIPS
conv1	0.01401019562035799	0.4879419440142668	0.5691883022841802
relu1	0.01759685017168522	0.0752994833199078	0.7542718040101729
maxpool	0.02153596840798855	0.05067673142083931	0.8220393613881902
layer1_0	0.03462613746523857	0.02044212924462065	0.7346459498886706
layer1_1	0.03942584991455078	0.018192398862000445	0.7578323048136427
block1	0.03809519112110138	0.01936285924659451	0.7595072542310197
layer2_0	0.042782243341207504	0.01845180845265923	0.8173357073866169
layer2_1	0.04605131596326828	0.02071475715860098	0.7846812196308396
layer2_2	0.05133433640003204	0.021988532149271333	0.8485415860776468
block2	0.055535830557346344	0.022478984221435957	0.7584564460332044
layer3_0	0.0680731013417244	0.023693435219119766	0.795642141824336
layer3_1	0.06523124128580093	0.026938971472903687	0.7696019457069961
layer3_2	0.06443958729505539	0.025905228359577762	0.8202767115730046
layer3_3	0.07021969556808472	0.0252526314380501	0.8186722074577006
layer3_4	0.06672782450914383	0.02688576098572723	0.9295666606189719
block3	0.0695166364312172	0.028477220022302078	0.874025463942484
layer4_0	0.08859771490097046	0.013259337432296195	0.7348496364423581
layer4_1	0.10521765798330307	0.010628038711185001	0.7583262746709122
block4	0.12141445279121399	0.006961201757144857	0.7505212535723733

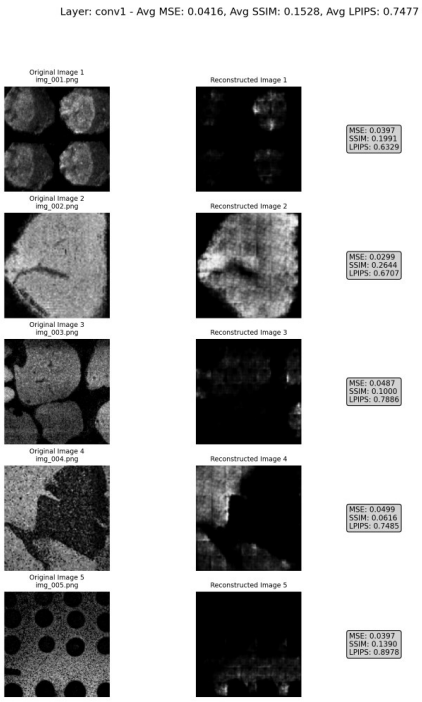
Table 6: Similirity metrics across attacked layers of ResNet-50 on blackbox scenario.

Layer	Classification accuracy	TP	TN	FP	FN
conv1	0.40628362384333977	260	1628	24	2735
relu1	0.3554981708629223	0	1652	0	2995
maxpool	0.3554981708629223	0	1652	0	2995
layer1_0	0.3554981708629223	0	1652	0	2995
layer1_1	0.3554981708629223	0	1652	0	2995
block1	0.3554981708629223	0	1652	0	2995
layer2_0	0.3570045190445449	7	1652	0	2988
layer2_1	0.3699160748870239	67	1652	0	2928
layer2_2	0.3615235635894125	28	1652	0	2967
block2	0.3699160748870239	67	1652	0	2928
layer3_0	0.6253496879707338	1268	1638	14	1727
layer3_1	0.6916290079621261	1570	1644	8	1425
layer3_2	0.780933935872606	1992	1637	15	1003
layer3_3	0.7959974176888315	2061	1638	14	934
layer3_4	0.8104153217129331	2131	1635	17	864
block3	0.8216053367764149	2179	1639	13	816
layer4_0	0.8442005595007531	2291	1632	20	704
layer4_1	0.8519474930062406	2345	1614	38	650
block4	0.8414030557348827	2293	1617	35	702

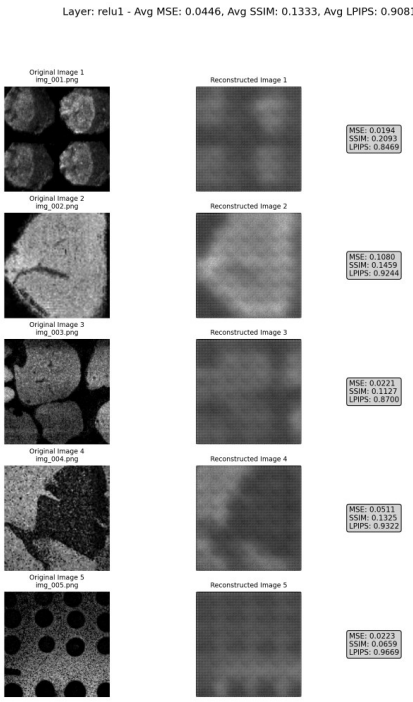
Table 7: Classification accuracy metrics across attacked layers of ResNet-50 on blackbox scenario.

Appendix C: Graphical results

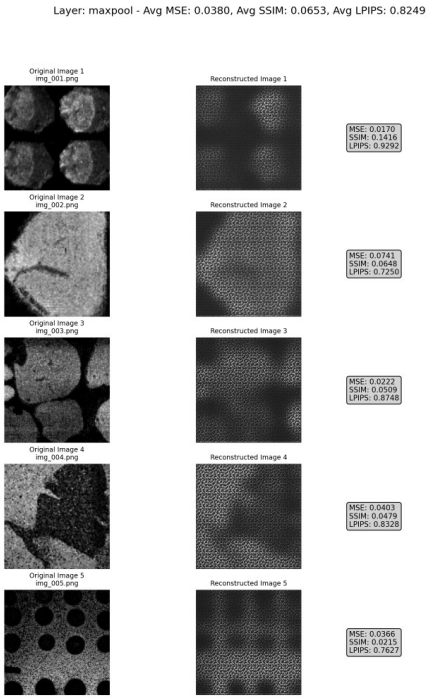
C.1 Reconstruction attacks



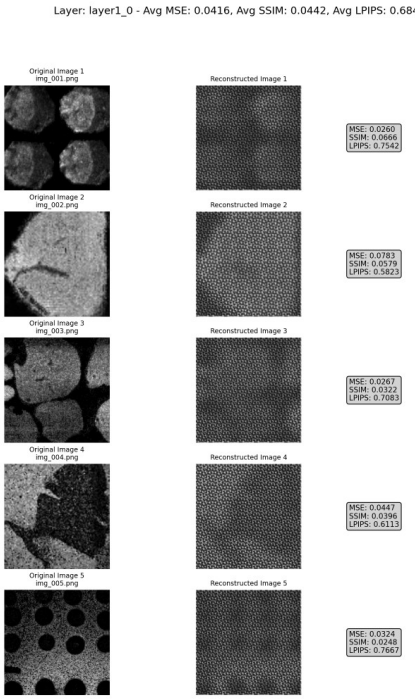
Results visualization of reconstruction attack for layer conv1 in black-box scenario



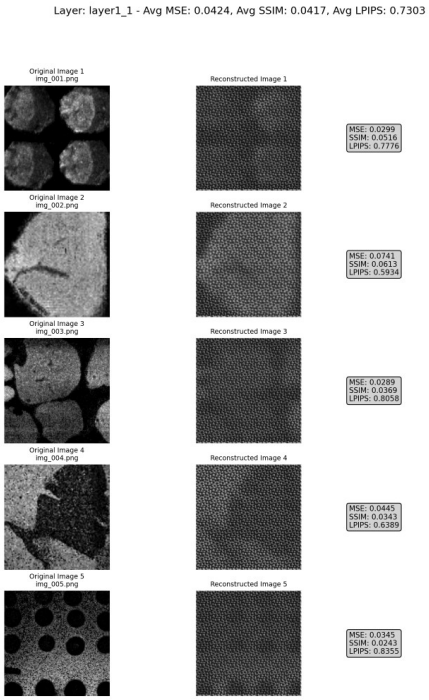
Results visualization of reconstruction attack for layer relu1 in black-box scenario



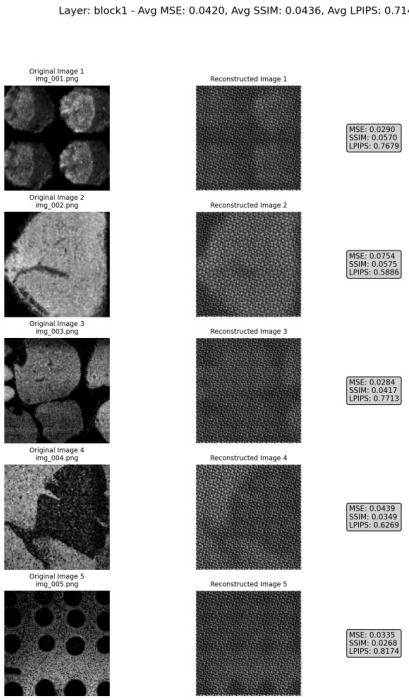
Results visualization of reconstruction attack for layer maxpool in black-box scenario



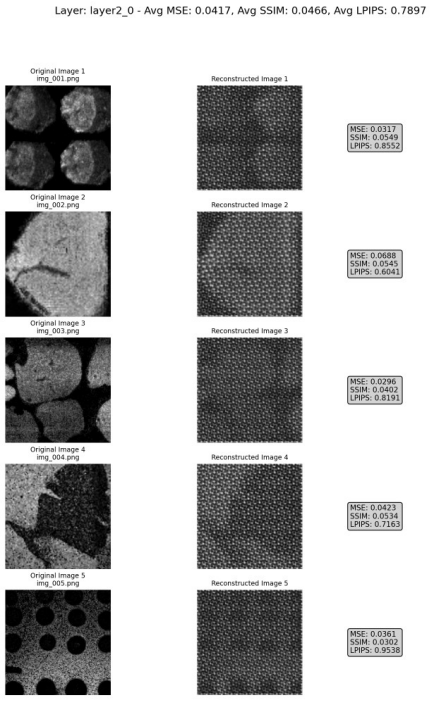
Results visualization of reconstruction attack for layer1_0 in black-box scenario



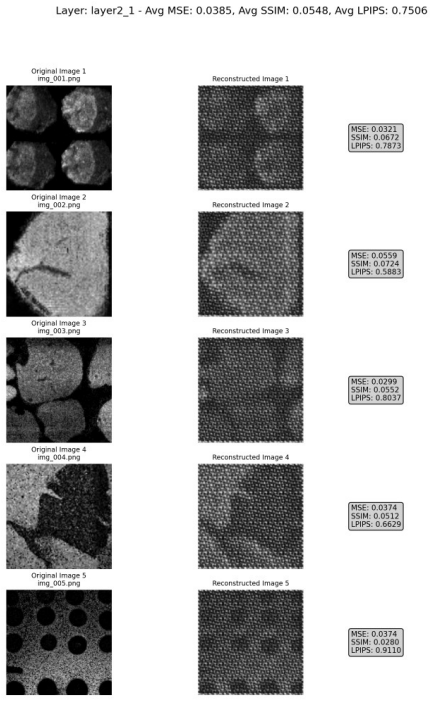
Results visualization of reconstruction attack for layer1_1 in black-box scenario



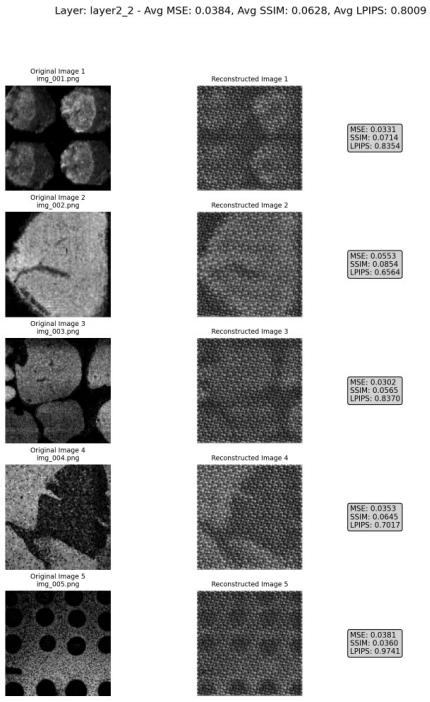
Results visualization of reconstruction attack for block1 in black-box scenario



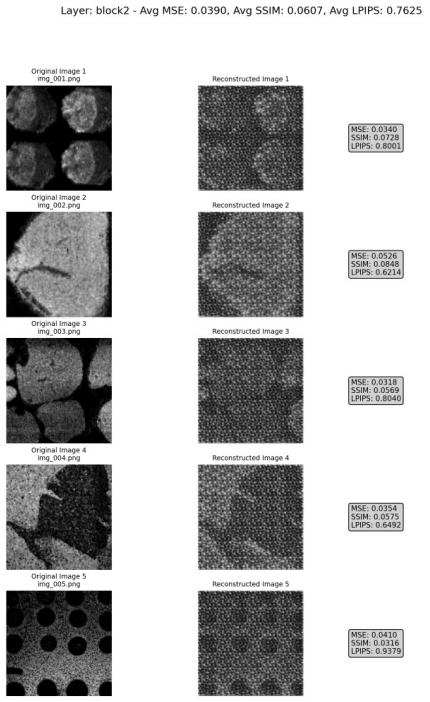
Results visualization of reconstruction attack for layer2_0 in black-box scenario



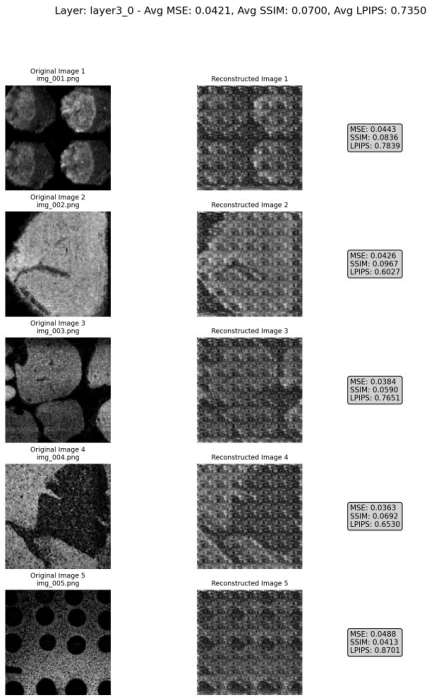
Results visualization of reconstruction attack for layer2_1 in black-box scenario



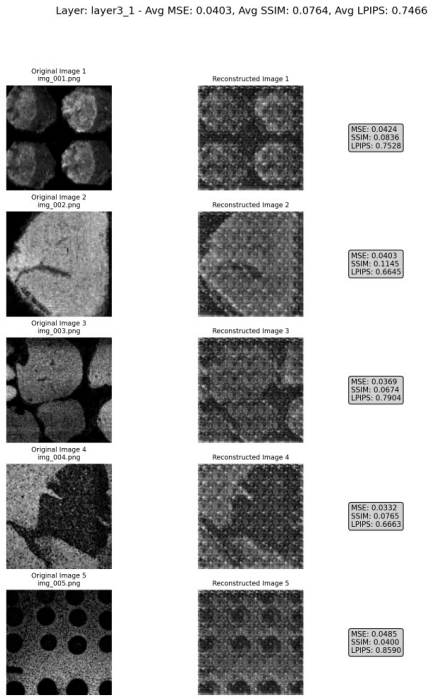
Results visualization of reconstruction attack for layer2_2 in black-box scenario



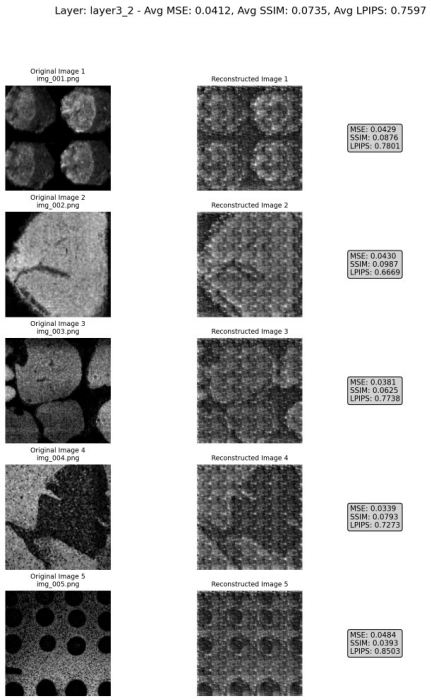
Results visualization of reconstruction attack for block2 in black-box scenario



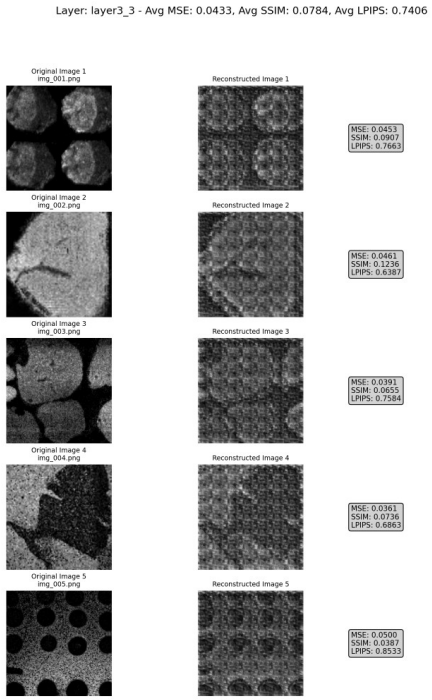
Results visualization of reconstruction attack for layer3_0 in black-box scenario



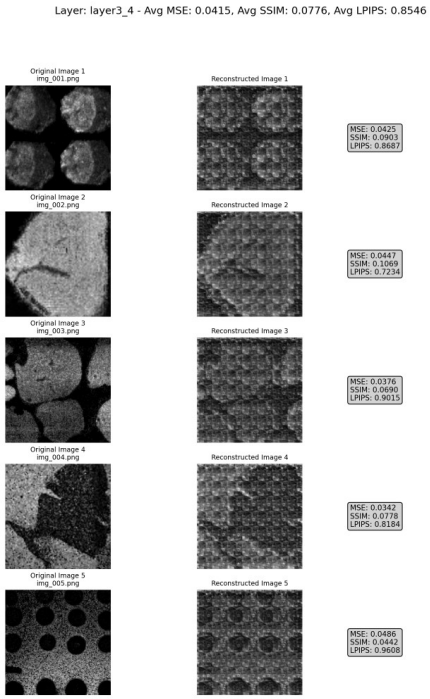
Results visualization of reconstruction attack for layer3_1 in black-box scenario



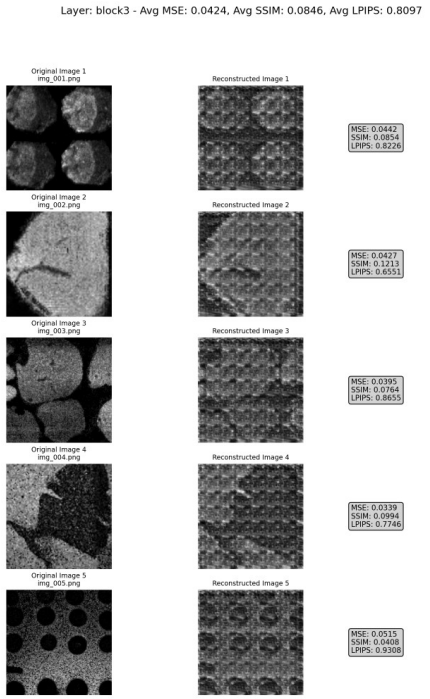
Results visualization of reconstruction attack for layer3_2 in black-box scenario



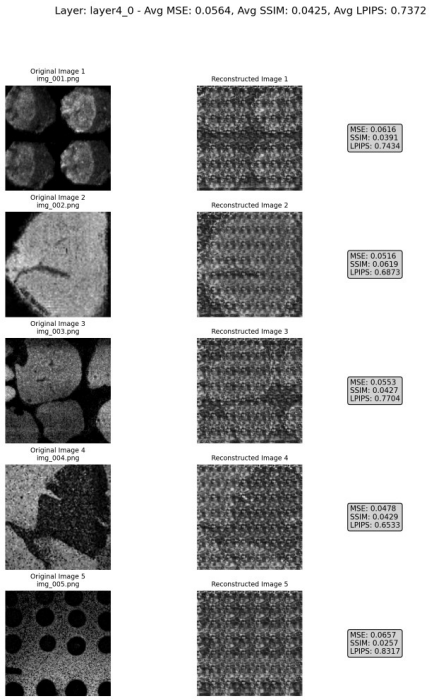
Results visualization of reconstruction attack for layer3_3 in black-box scenario



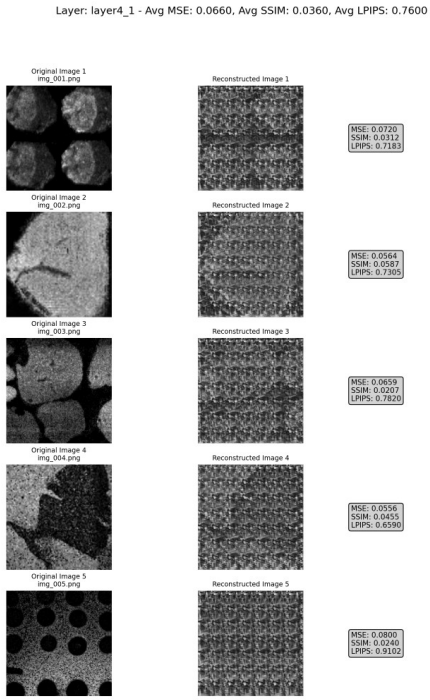
Results visualization of reconstruction attack for layer3_4 in black-box scenario



Results visualization of reconstruction attack for block3 in black-box scenario

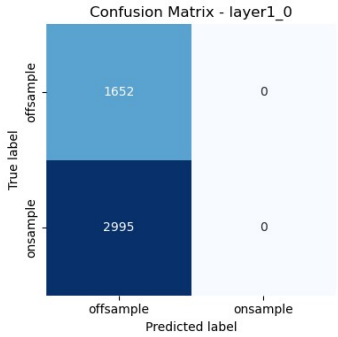


Results visualization of reconstruction attack for layer4_0 in black-box scenario

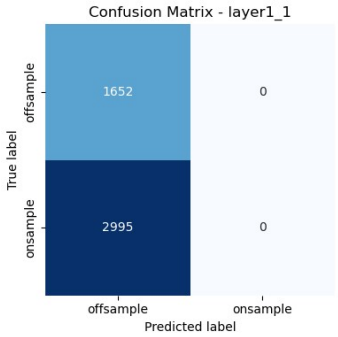


Results visualization of reconstruction attack for layer4_1 in black-box scenario

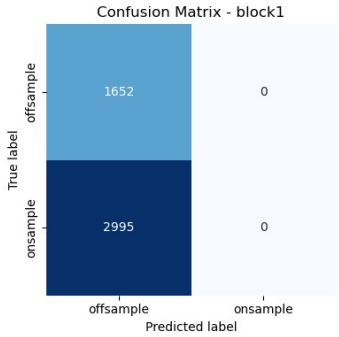
C.2 Confusion matrix for each layer on White-box scenario



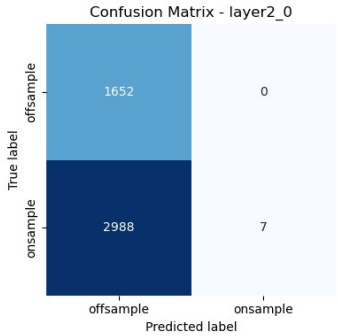
Confusion matrix of the attack model for layer1_0



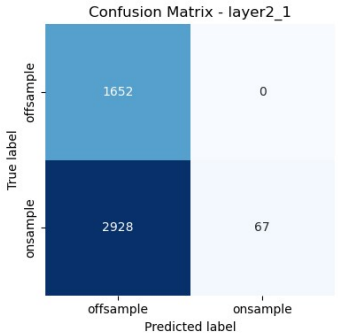
Confusion matrix of the attack model for the layer1_1



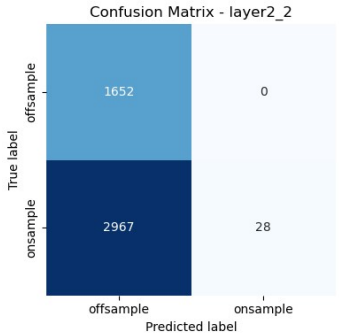
Confusion matrix of the attack model for block1



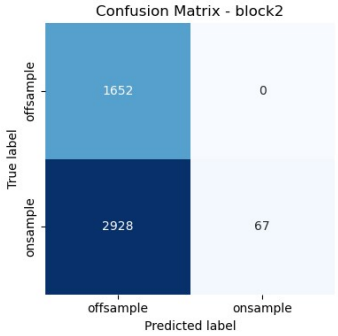
Confusion matrix of the attack model for layer2_0



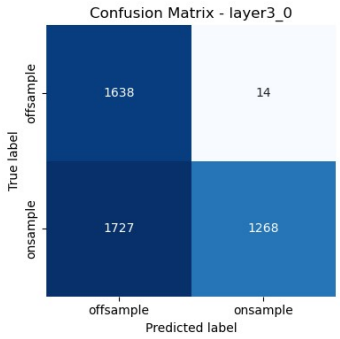
Confusion matrix of the attack model for layer2_1



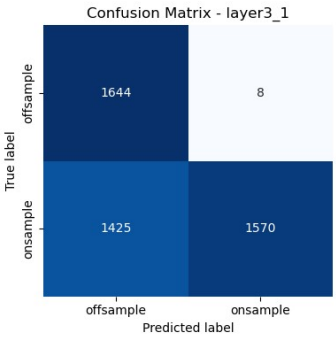
Confusion matrix of the attack model for layer2_2



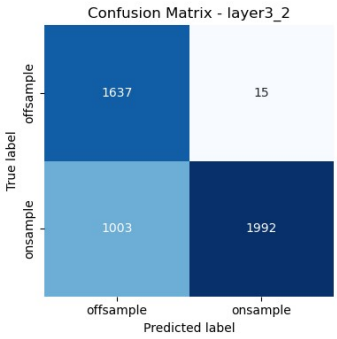
Confusion matrix of the attack model for block2



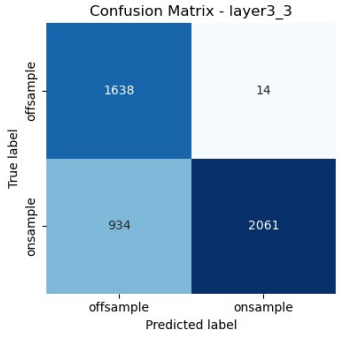
Confusion matrix of the attack model for layer3_0



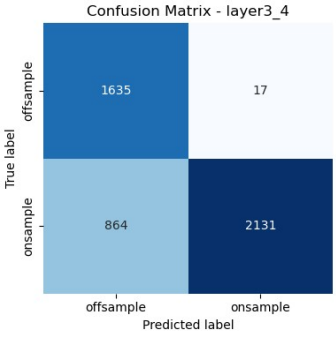
Confusion matrix of the attack model for layer3_1



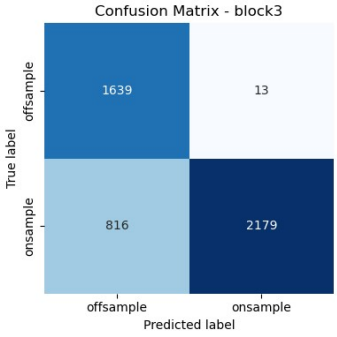
Confusion matrix of the attack model for layer3_2



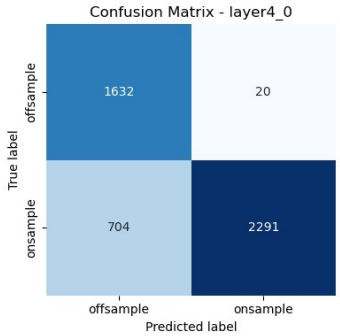
Confusion matrix of the attack model for layer3_3



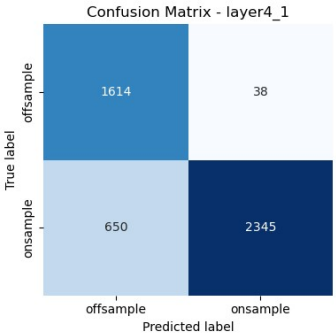
Confusion matrix of the attack model for layer3_4



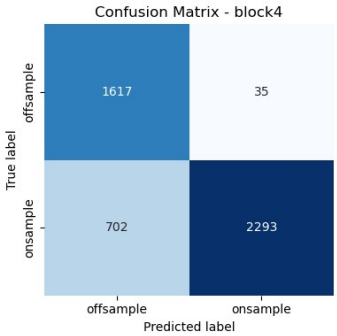
Confusion matrix of the attack model for block3



Confusion matrix of the attack model for layer4_0



Confusion matrix of the attack model for layer4_1



Confusion matrix of the attack model for block4