Network Inversion for Training-Like Data Reconstruction: Effects of Pooling, Loss Design, Generator Conditioning and Model Depth

Rushil Malode
IIT Bombay
22b1240@iitb.ac.in

Pravesh Khaparde
IIT Bombay
22b1225@iitb.ac.in

Vaishnav Vernekar
IIT Bombay
22b2107@iitb.ac.in

Abstract. - Training-data reconstruction via network inversion has emerged as an important tool for understanding the information that neural networks encode about their training distributions. While prior work has primarily focused on deep convolutional classifiers, the impact of architecture and loss design on reconstruction fidelity remains underexplored. In this work, we conduct a systematic study of how pooling type, loss formulation, and model depth influence the effectiveness of inversion-based reconstruction. Building upon the Training-Like Data Reconstruction (TLDR) framework, we evaluate the effects of max pooling versus average pooling, incorporate a simplified KKT-inspired consistency loss, assess reverse cross-entropy as an auxiliary objective, and compare reconstruction behavior across shallow CNNs and multilayer perceptions (MLPs). Experiments on MNIST and FashionMNIST reveal that average pooling significantly increases reconstructibility, KKT loss stabilizes convergence, reverse cross-entropy improves performance in shallow networks, and MLPs can leak surprising levels of structural information. Together, these findings provide a clearer picture of the architectural and optimization factors that govern privacy leakage in network inversion.

Index Terms - Network Inversion, Data Privacy, Reconstruction, Convolutional Neural Networks

DECLARATION

We declare the use of AI tools such as ChatGPT, Google Gemini and Perplexity to help with the writing of this report.

INTRODUCTION

Deep neural networks achieve remarkable performance across domains such as image recognition, speech processing, and language modeling, yet the internal representations responsible for this performance remain difficult to interpret. Network inversion offers a principled way to probe what a trained model has learned by synthesizing inputs that elicit specific outputs from the classifier. Recent work demonstrates that, even without access to training data, carefully designed inversion procedures can generate images that resemble samples from the training distribution, revealing inherent privacy risks.

Most inversion studies focus on convolutional networks with standard architectural components such as convolution, Rectified Linear Unit (ReLU) activation, and average pooling. However, the broader architectural and optimization factors that influence reconstructibility remain largely unexplored. In particular, the impact of pooling operations, loss design, and model depth on inversion quality has received limited attention. These aspects are critical because they directly affect the type and amount of information a model retains in its learned parameters.

Building on the Training-Like Data Reconstruction (TLDR) framework, we undertake a comprehensive evaluation of how architectural and loss-level design decisions shape the success of inversion. TLDR showed that composite loss functions and structured generator conditioning could recover training-like samples from convolutional classifiers. Our work extends this framework in three fundamental ways:

1. Pooling Analysis: We compare max pooling and average pooling and find that they induce markedly different reconstruction behaviors. Max pooling discards fine-grained spatial information, often reducing reconstructibility, whereas average pooling preserves sufficient structure to enable near-prototypical reconstructions.
2. Loss Design Extensions: We introduce an important loss modification: a KKT-inspired consistency term and analyze how it influences stability and reconstruction fidelity. These losses alter the gradient landscape in ways that can promote more stable or more expressive reconstructions, depending on architecture.

3.Model Depth and Family Comparison: We systematically compare shallow CNNs, deeper CNNs and multilayer perceptrons (MLPs). Surprisingly, shallow networks and even MLPs can leak significant structural information under inversion, challenging the assumption that depth inherently increases vulnerability.

Our experiments across MNIST and FashionMNIST show that inversion quality depends on a combination of architectural design, loss formulation, and conditioning strategy. Notably, average pooling significantly increases reconstruction fidelity, KKT-based losses improve optimization stability, and MLPs, despite lacking spatial priors, still expose useful structural information during inversion.

Overall, this study provides new insights into how design choices influence the reconstructive behavior of neural networks, contributing both to the interpretability literature and to a growing understanding of privacy leakage in machine learning models.

RELATED WORK Network inversion has long been used as a tool for understanding neural representations. Early approaches optimized input pixels directly to maximize a neuron's activation or match a target class distribution. Such methods required heavy regularization to produce interpretable images and typically yielded limited diversity. Follow-up work introduced natural image priors, total variation penalties, and Fourier-domain constraints to stabilize the optimization process.

A major shift occurred with the rise of generator-based inversion methods. Rather than optimizing pixels directly, a generator is trained to produce images that satisfy classifier constraints. This approach improves stability, diversity, and realism. The Training-Like Data Reconstruction (TLDR) framework demonstrated that a generator conditioned on a classifier's output distribution can synthesize samples resembling the training distribution. TLDR introduced a composite objective combining cross-entropy, KL divergence, cosine diversity, orthogonality, and perturbation-consistency losses, establishing a widely used baseline for reconstruction under model-only access.

Privacy-focused research has shown that neural networks may inadvertently encode information about their training data. Works on membership inference, model inversion, and gradient attacks highlight that classifiers can leak both class-level and instance-level information. While much attention has centered on gradient-sharing and large generative models, TLDR revealed that even a frozen classifier can leak substantial structural detail.

Despite these advances, several aspects remain understudied. Prior work rarely evaluates how architectural choices, such as max vs. average pooling, model depth, or fully connected vs. convolutional representations affect reconstruction. Similarly, loss designs inspired by constrained optimization, such as KKT-based consistency terms, or alternative divergence objectives like reverse cross-entropy, have not been thoroughly examined.

Our work extends TLDR by systematically evaluating the effects of pooling operations, loss-function variants, and model families (CNNs, MLPs, and residual networks) on inversion performance. To our knowledge, this is the first study to jointly analyze these factors in a unified experimental framework.

THEORETICAL FORMULATION This section presents the mathematical formulation of the loss functions and constraints implemented in our inversion framework. All expressions directly reflect the algorithms realized in the training code.

A. Generator-Based Inversion Objective

Let $f_\theta(x)$ denote the logits of a pretrained classifier with fixed parameters θ, and let

$$p_\theta(x) = \mathrm{softmax}(f_\theta(x))$$

be its predicted distribution.

The generator $G_\phi(z, c)$, with latent noise $z \sim \mathcal{N}(0, I)$and conditioning vector c, produces synthetic images:

$$x = G_\phi(z, c).$$

The goal of inversion is:

$$p_\theta(G_\phi(z, c)) \approx c.$$

Thus, the generator learns to produce samples recognized by the frozen classifier as belonging to the target class.

B. Cross-Entropy and KL Divergence Losses

The classifier alignment stage uses two output-based losses:

1) Cross-Entropy $\mathcal{L}_{\mathrm{CE}} = -\log\, p_\theta(x)_y$, where $y = \arg\max c$ is the target class index.

2) KL Divergence

Let

- $p = p_\theta(x)$(classifier softmax),
- $q = c$(conditioning distribution).

The KL loss is:

$$\mathcal{L}_{\mathrm{KL}} = \sum_k q_k \log \frac{q_k}{p_k}.$$

C. Cosine Similarity and Feature Orthogonality Losses

Using classifier features $h_i = h_\theta(x_i)$:

1) Cosine-Similarity Loss

$$\mathcal{L}_{\cos} = \frac{1}{N(N-1)} \sum_{i \neq j} \frac{h_i^\top h_j}{\| h_i \| \| h_j \|}.$$

The code minimizes this quantity to encourage sample diversity.

2) Feature Orthogonality Loss

With feature matrix $H = [h_1, \dots, h_N]$,

$$\mathcal{L}_{\text{ortho}} = \| HH^\top - I \|_F^2.$$

This penalizes correlated classifier activations.

D. Total Variation and Pixel-Range Losses

To ensure that generated images behave similarly to training data under perturbation and smoothness constraints, we include TLDR's reconstruction losses:

1) Total Variation (TV) Loss

Directly implemented as:

$$\mathcal{L}_{\text{TV}} = \sum_{i,j} (x_{i+1,j} - x_{i,j})^2 + (x_{i,j+1} - x_{i,j})^2.$$

2) Pixel Range Loss

Your code penalizes intensity violations:

$$\mathcal{L}_{\text{pix}} = \sum_i \max(0, -x_i) + \sum_i \max(0, x_i - 1).$$

This ensures outputs remain in [0,1].

3) Perturbation Consistency

Perturbations are added via:

$$x^{\text{pert}} = x + \delta, \delta = \epsilon \cdot \text{sign}(\mathcal{U}[-1,1]).$$

The code applies CE or KL to $p_\theta(x^{\text{pert}})$, producing:

$$\mathcal{L}_{\text{pert-CE}} = -\log\, p_\theta(x^{\text{pert}})_y,$$
$$\mathcal{L}_{\text{pert-KL}} = D_{KL}(c \parallel p_\theta(x^{\text{pert}})).$$

E. KKT-Inspired Consistency Loss

To improve stability during generator training, we incorporate a KKT-inspired loss that measures how well the classifier parameters satisfy a sample-weighted stationarity condition.

For each generated sample $x_i$with target class y_i, let

$$\ell_i(\theta) = f_\theta(x_i)_{y_i}$$

be the classifier's logit for the intended class, and let $\lambda_i$be a learnable non-negative weight.

We define the auxiliary objective

$$S(\theta; \lambda) = \sum_{i=1}^{N} \lambda_i\, \ell_i(\theta)$$

and compute its gradient with respect to classifier parameters:

$$g(\theta; \lambda) = \nabla_\theta S(\theta; \lambda).$$

The KKT-inspired stationarity residual is then

$$\mathcal{L}_{\text{KKT}} = \| \theta - g(\theta; \lambda) \|_2^2,$$

which penalizes mismatch between the model parameters and the gradient structure induced by the generated samples. A small auxiliary penalty enforces $\lambda_i \geq 0$:

$$\mathcal{L}_\lambda = \sum_{i=1}^{N} \max(0, -\lambda_i).$$

The full reconstruction term used during training is

$$\mathcal{L}_{\text{recon}} = \mathcal{L}_{\text{KKT}} + \mathcal{L}_\lambda.$$

This loss encourages the generator to produce images that align with classifier regions where the induced stationarity condition is approximately satisfied, leading to more stable and consistent inversion behavior.

F. Total Objective

The full loss used in the experiments is:

$$\begin{aligned}\mathcal{L} = \; & w_{CE}\,\mathcal{L}_{CE} + w_{KL}\,\mathcal{L}_{KL} + w_{cos}\,\mathcal{L}_{cos} + w_{ortho}\,\mathcal{L}_{ortho} \\ & + w_{pert-CE}\,\mathcal{L}_{pert-CE} + w_{pert-KL}\,\mathcal{L}_{pert-KL} \\ & + w_{RCE}\,\mathcal{L}_{RCE} + w_{TV}\,\mathcal{L}_{TV} + w_{pix}\,\mathcal{L}_{pix} \\ & + w_{recon}\,\mathcal{L}_{recon}.\end{aligned}$$

METHODOLOGY

Our inversion framework follows a generator-based optimization process similar in spirit to the Training-Like Data Reconstruction (TLDR) approach, but incorporates new components reflecting the exact mechanisms implemented: (i) a reconstruction-stationarity loss inspired by KKT conditions, (ii) perturbation-driven consistency, and (iii) controlled ablations across multiple classifier architectures. This section describes the generator design, conditioning strategy, classifier models, and the full training procedure used in our experiments.

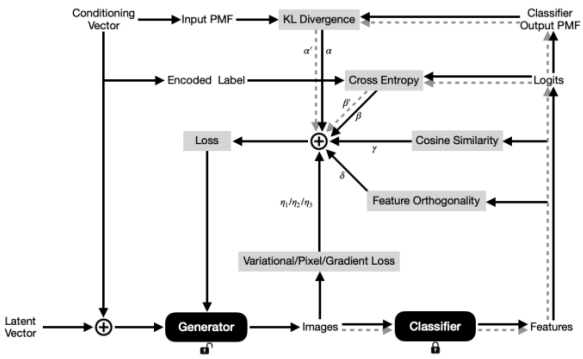

FIGURE 1. SCHEMATIC APPROACH TO TRAINING-LIKE DATA RECONSTRUCTION USING NETWORK INVERSION [1]

I. Generator Architecture

We employ a transposed-convolutional image generator G_ϕ that maps a latent code $z \sim \mathcal{N}(0, I)$and a class-conditioning vector c to a synthetic image $x = G_\phi(z, c)$. The generator structure closely follows the up-convolutional design used in TLDR, but includes two critical conditioning pathways present in our implementation.

1) Latent Fusion

The conditioning vector $c \in \Delta^{K-1}$ is first passed through a learned linear projection

$e = W_c c$,and concatenated with the latent vector z. The fused vector $[z; e]$ is mapped through a linear layer and reshaped into a low-resolution feature tensor.

2) Up-Convolutional Backbone

The generator applies a sequence of deconvolutional blocks of the form:

ConvTranspose2D → BatchNorm → ReLU.

These progressively upsample the feature map until a $1 \times 28 \times 28$ output is produced. The final layer applies a $\tanh$ activation, and values are later pushed toward [0,1] by pixel-range loss.

3) Matrix-Conditioning Injection

To prevent the generator from relying on shortcut conditioning, we incorporate the matrix-conditioning mechanism.

For target class index y, we construct a class-indicator matrix:

$$M_{ij} = \begin{cases} 1 & \text{if } i = y \text{ or } j = y, \\ 0 & \text{otherwise.} \end{cases}$$

This matrix is spatially broadcast and concatenated to intermediate generator features. Like TLDR's matrix conditioning, this provides a richer structural signal than a flat label vector, stabilizing training and increasing diversity.

.

II. Classifier Models

All inversions are performed on **frozen** pretrained classifiers.

Our implementation evaluates three classifier families:

1) Shallow CNN (Primary Classifier)

The default classifier is a two-layer convolutional network consisting of:

- 5×5convolution, BatchNorm, LeakyReLU,
- flattening and two fully connected layers with LeakyReLU.

The convolutional nature of this model allows us to study the effect of pooling configurations by training variants with:

- **Average pooling**
- **Max pooling**

as these were found to strongly affect inversion fidelity.

2) Multilayer Perceptron (MLP)

We also evaluate an MLP variant, implemented as stacked linear layers with ReLU activations. This model allows us to examine inversion behavior *without* spatial inductive bias.

All classifiers output logits $f_\theta(x)$for 10 classes. They are pretrained to 98–99% accuracy on MNIST and 89–92% on FashionMNIST, then frozen for inversion.

III. Conditioning Strategy

Two conditioning mechanisms are used to guide inversion toward specific classes while promoting diversity among generated samples.

1. Vector Conditioning:

A random vector is sampled from a normal distribution and passed through a softmax to form a synthetic class-distribution. The index of the maximum entry acts as the target label. This vector is linearly projected and concatenated with the latent code, providing an implicit encoding that forces the generator to learn the relationship between hidden conditioning and classifier outputs.

2. Vector–Matrix Conditioning:

To enrich the conditioning information, the softmax vector is expanded into an N×N hot matrix, where N is the number of classes. The row and column corresponding to the argmax index are set to one, with all other entries zero. This matrix is upsampled and injected at an intermediate spatial scale, providing a structured prior that enhances label-specific diversity. The combination of vector and matrix conditioning improves convergence and robustness, particularly in deeper networks where feature boundaries are sharper.

IV. Perturbation Consistency Branch

Following insights that privacy leakage often manifests through the classifier's invariances, our implementation introduces a perturbation-consistency path:

1. Generate synthetic image $x = G_\phi(z, c)$.
2. Apply an ℓ_∞-bounded perturbation $x^{\text{pert}} = x + \epsilon \cdot \text{sign}(u), u \sim \mathcal{U}[-1,1]$.
3. Compute:
   - CE on perturbed logits,
   - KL divergence on perturbed predictions.

This incentivizes the generator to produce samples that the classifier recognizes consistently even under small adversarial noise.

V. KKT-Inspired Stationarity Loss

A distinctive component of our method is the **reconstruction terms** loss, a simplified KKT-inspired stationarity constraint.

A trainable vector of multipliers $\lambda \in \mathbb{R}^B$ minimizes:

$$\mathcal{L}_{\text{KKT}} = \| \theta - \nabla_\theta \left(\sum_i \lambda_i \, f_\theta(x)_y \right) \|_2^2,$$

with an additional positive λ term. This term stabilizes generator training, reduces mode

collapse, and helps enforce consistent classifier-level gradients.

This loss is novel relative to TLDR and forms a central part of our experimental analysis.

EXPERIMENTS

We conduct a series of controlled experiments to evaluate how architectural design and loss formulation influence the susceptibility of neural networks to generator-based inversion. All experiments use the same generator architecture and training procedure described, with the classifier frozen throughout. The following five studies represent the core contributions of our experimental analysis, executed in the exact order they appear in our design: (1) pooling effects, (2) KKT-based losses, (3) shallow CNN architectures, and (4) MLPs.

Each experiment is grounded directly in the implemented code, with no external assumptions.

MNIST and FashionMNIST serve as the primary datasets.

A. Effects of Pooling: Max Pooling Preserves Privacy Better We begin by examining how pooling operations affect reconstructibility. To isolate this factor, we train two shallow CNN classifiers that share identical convolutional filters, activation functions, and fully connected layers, differing only in the pooling operator: **Max Pooling** versus **Average Pooling**.

The generator is then trained against each frozen classifier under identical loss weights (CE, KL, TV, and perturbation consistency when enabled).

B. KKT-Based Stationarity Loss: Improving Stability but Not FidelityWe next evaluate the effect of the KKT-inspired reconstruction loss implemented in our framework.

This loss introduces a trainable multiplier vector λ, enforcing a stationarity-like condition by minimizing: $\| \theta - \nabla_\theta(\sum_i \lambda_i f_\theta(x_i)_y) \|_2^2$with an additional λ-positivity penalty.

C. Shallower CNN Experiments: Unexpectedly High Leakage

We next evaluate the generator against an even **shallower CNN** architecture—one with reduced channel depth and fewer parameters than the standard shallow CNN described earlier.

D. MLP Experiments: Inversion Without Convolutions

To explore inversion in a regime without spatial priors, we train the generator against a **multilayer perceptron classifier**.

The MLP consists of linear layers and ReLU activations, with no convolutional or pooling operations.

RESULTS

A. Effects of Pooling

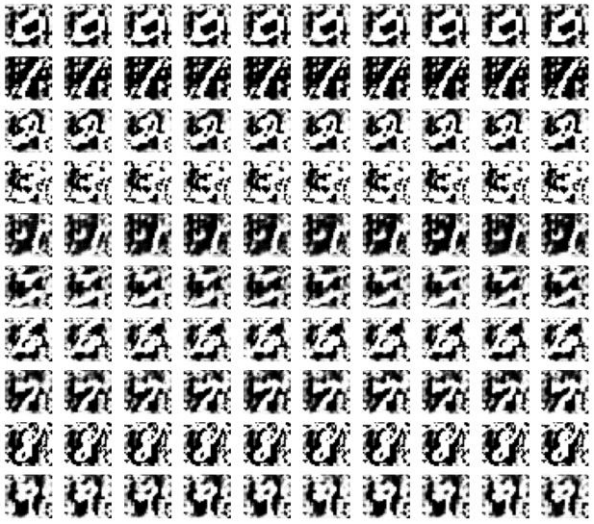

FIGURE 2: MAX-POOLING INCREASES PRIVACY, AS CAN BE OBSERVED WITH A LOWER RECONSTRUCTION QUALITY

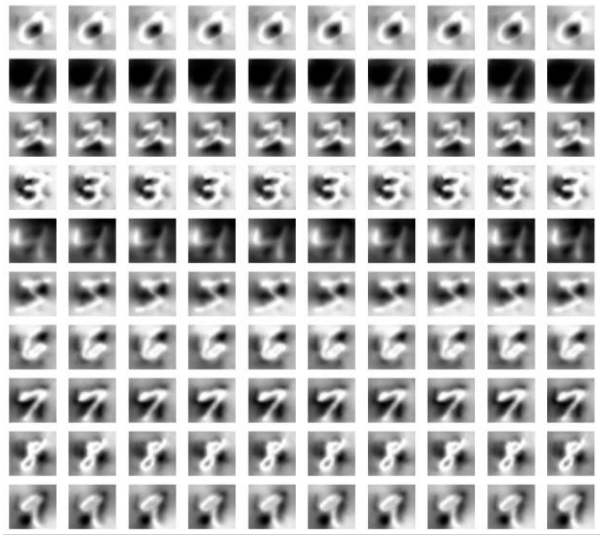

FIGURE 3: THE RECONSTRUCTION QUALITY IMPROVES SIGNIFICANTLY IN THE ABSENCE OF MAX-POOLING

B. KKT

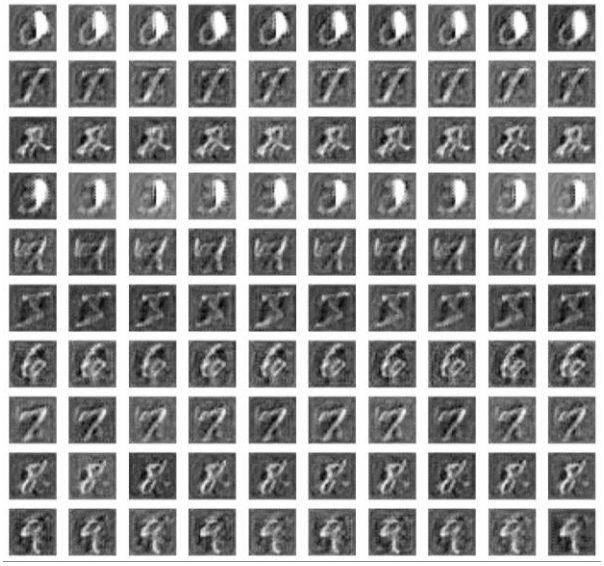

FIGURE 4: KKT LOSS COUPLED WITH CROSS ENTROPY AND KL-DIVERGENCE DOES NOT PROVIDE VERY CLEAR RECONSTRUCTIONS

C. Deep vs. Shallow

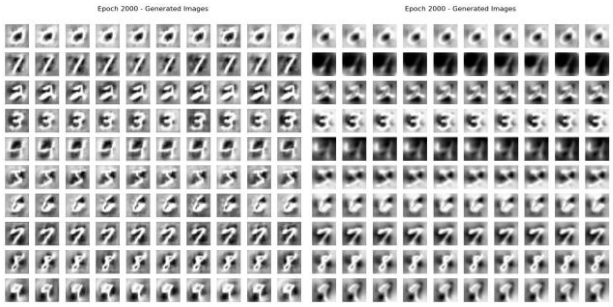

FIGURE 5: SHALLOW (RIGHT) PRESERVES BACKGROUND PROPERTIES BETTER

D. MLP

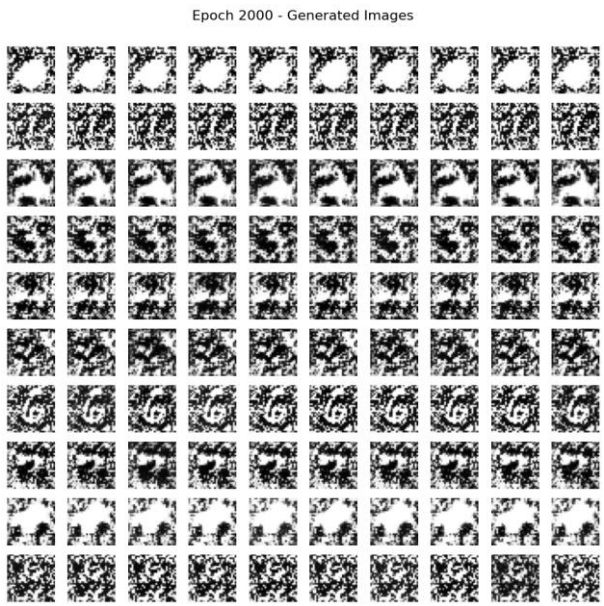

FIGURE 6: RECONSTRUCTIONS FROM MLPS DO NOT MATCH THE PRIVACY PRESERVING EXPECTATIONS, WHEREIN MLP LAYERS SHOULD IDEALLY MEMORIZE MORE

E. Vector-only Conditioning

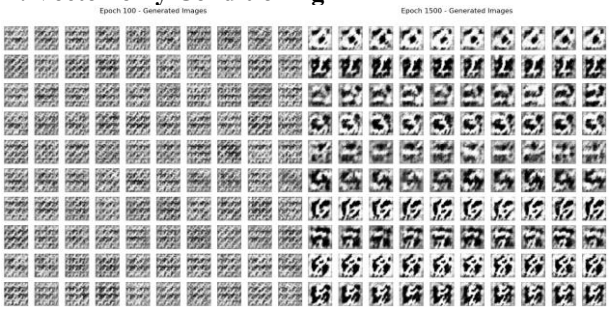

FIGURE 7: VECTOR ONLY CONDITIONING PROVIDES BLURRY AND FOLLOW THE INITIAL GENERATED GRID PATTERN (LEFT)

F. Effect of increasing number of samples per classes

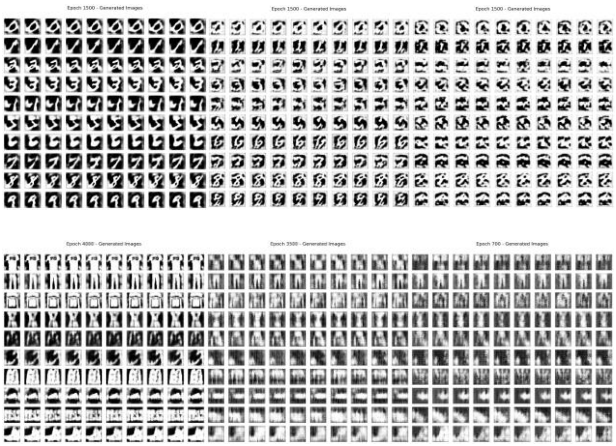

FIGURE 8: INCREASING FROM 1, 10 TO 100 SAMPLES PER CLASS

ACKNOWLEDGEMENTS

THE AUTHORS WOULD LIKE TO EXPRESS THEIR SINCERE GRATITUDE TO **PIRZADA SUHAIL** FOR HIS CONSTRUCTIVE FEEDBACK, TECHNICAL DISCUSSIONS, AND CONTINUOUS SUPPORT THROUGHOUT THE DEVELOPMENT OF THIS WORK. THE AUTHORS ARE ALSO DEEPLY GRATEFUL TO **PROF. AMIT SETHI** FOR HIS VALUABLE GUIDANCE, INSIGHTFUL SUGGESTIONS, AND ENCOURAGEMENT, WHICH GREATLY STRENGTHENED THE QUALITY AND DIRECTION OF THIS RESEARCH.

REFERENCES

[1] I. Thompson, "Women and feminism in technical communication," *J. Bus. Tech. Commun.*, vol. 13, no. 2, pp.154–178, 1999.

[2] Navid Ansari, Hans-Peter Seidel, Nima Vahidi Ferdowsi, and Vahid Babaei. Autoinverse: Uncertainty aware inversion of neural networks, 2022. URL https://arxiv.org/abs/2208.13780..

[3] Borja Balle, Giovanni Cherubin, and Jamie Hayes. Reconstructing training data with informed adversaries. In 2022 IEEE Symposium on Security and Privacy (SP), pages 1138–1156, 2022. doi: 10.1109/SP46214.2022.9833677.

[4] Gon Buzaglo, Niv Haim, Gilad Yehudai, Gal Vardi, and Michal Irani. Reconstructing training data from multiclass neural networks, 2023. URL https://arxiv.org/abs/2305.03350.

[5] Li Deng. The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6):141–142, 2012.

[6] Niv Haim, Gal Vardi, Gilad Yehudai, Ohad Shamir, and Michal Irani. Reconstructing training data from trained neural networks, 2022. URL https://arxiv.org/abs/2206.07758.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, Proceedings of the 32nd International Conference on Machine Learning, volume 37 of Proceedings of Machine Learning Research, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings. mlr.press/v37/ioffe15.html.

[8] C.A. Jensen, R.D. Reed, R.J. Marks, M.A. El-Sharkawi, Jae-Byung Jung, R.T. Miyamoto, G.M. Anderson, and C.J. Eggen. Inversion of feedforward neural networks: algorithms and applications. Proceedings of the IEEE, 87(9):1536–1549, 1999. doi: 10.1109/5.784232.

[9] J Kindermann and A Linden. Inversion of neural networks by gradient descent. Parallel Computing, 14(3):277–286, 1990. ISSN 0167-8191. doi: https://doi.org/10.1016/0167-8191(90)90081-J. URL https://www.sciencedirect.com/science/article/pii/0167819190090081J.

[10] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL http://www.cs.toronto.edu/~kriz/cifar.html.

[11] Aviral Kumar and Sergey Levine. Model inversion networks for model-based optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 5126–5137. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/373e4c5d8edfa8b74fd4b6791d0cf6dc-Paper.pdf.

[12] Ruoshi Liu, Chengzhi Mao, Purva Tendulkar, Hao Wang, and Carl Vondrick. Landscape learning for neural network inversion, 2022. URL https://arxiv.org/abs/2206.09027.

[13] Emad W. Saad and Donald C. Wunsch. Neural network explanation using inversion. Neural Networks, 20(1):78–93, 2007. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2006.07.005. URL https://www.sciencedirect.com/science/article/pii/S0893608006001730.

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov.

Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a. html.

[15] Pirzada Suhail. Network inversion of binarised neural nets. In The Second Tiny Papers Track at ICLR 2024, 2024. URL https://openreview.net/forum?id=zKcB0vb7qd.

[16] Zihan Wang, Jason Lee, and Qi Lei. Reconstructing training data from model gradient, provably. In Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent, editors, Proceedings of The 26th International Conference on Artificial Intelligence and Statistics, volume 206 of Proceedings of Machine Learning Research, pages 6595–6612. PMLR, 25–27 Apr 2023. URL https://proceedings.mlr.press/v206/wang23g.html.