Semi-Supervised GAN for MNIST with 100 Labels: K+1 Discriminator and Feature-Matching for Low-Label Digit Classification

Manel LOUNISSI
Université Paris Cité
manel2.lounissi@gmail.com

Sandeep-Singh NIRMAL
Université Paris Cité
nirmalsinghsandeep@gmail.com

Brice SAILLARD
Université Paris Cité
brice.saillard.bs@gmail.com

Hamady GACKOU
Université Paris Cité
hamady.gackou@etu.u-paris.fr

Short Abstract

This report presents a faithful reproduction of the semi-supervised GAN framework introduced by Salimans et al. for low-label image classification. The original method is applied to the MNIST dataset in a setting with only 100 labeled examples (10 per class), while the remaining training images are treated as unlabeled. A simple supervised CNN trained on the same 100 labels is used as a baseline reference. The semi-supervised model follows the $K+1$ discriminator formulation and the feature-matching generator loss proposed in the original paper, without any methodological extension or modification. All architectural and optimization choices are derived from the literature and from the accompanying project codebase. The results are interpreted strictly as an empirical confirmation of the techniques introduced by Salimans et al. in this constrained MNIST scenario. [1][2]

1 Introduction

Deep learning models often rely on large labeled datasets, whereas in many real applications labeling is costly and only a small fraction of data can be annotated. Semi-supervised learning aims to mitigate this issue by combining a small labeled subset with a large unlabeled pool to improve the performance of a classifier. In the context of image recognition, this is particularly relevant for domains where expert annotation is expensive or slow.

Generative Adversarial Networks (GANs) provide a flexible framework to model complex data distributions by training a generator against a discriminator in a two-player game. Salimans et al. proposed several techniques to stabilize GAN training and demonstrated that the discriminator can be repurposed as a $K+1$-class classifier for semi-supervised learning. In this setting, the discriminator learns to separate real examples of K classes from generated (fake) samples, and the generator is trained using a feature-matching objective rather than directly trying to fool the discriminator.[2]

The present work is conducted in the context of a Master 2 project on deep learning and semi-supervised learning at Université Paris Cité, under the supervision of Blaise Hanczar (Université Paris-Saclay). The goal is to reproduce, as faithfully as possible, the semi-supervised GAN approach of Salimans et al. on MNIST with only 100 labeled images, using a PyTorch implementation organized as a GitHub project with modules for models, training, utilities, experiments, and report. No theoretical novelty is introduced; the study focuses on careful implementation, experimental protocol, and analysis of the observed behavior.[1]

2 Description of the Methods

2.1 Baseline: Supervised CNN with 100 Labels

The supervised baseline is implemented as a simple convolutional neural network trained exclusively on the 100 labeled MNIST samples. The architecture follows a standard design for digit recognition and is implemented in the `BaselineCNN` class inside the `models/` directory of the project. The overall structure of the baseline CNN is summarized in Figure 1.

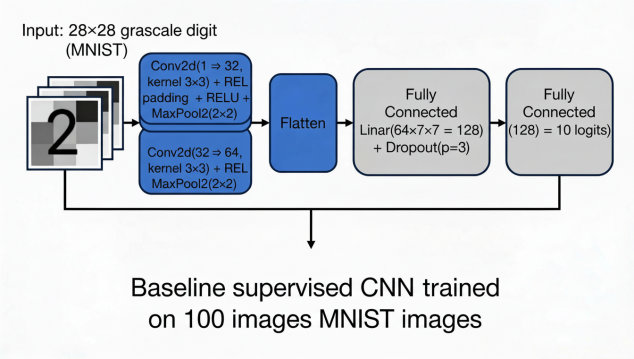

Figure 1: Architecture of the supervised CNN baseline used for training on 100 labeled MNIST samples.

Concretely, the baseline network consists of a feature extractor followed by a classifier. The feature extractor applies two convolutional layers with ReLU activations and 2×2 max-pooling:[1]

- first convolution: $1 \to 32$ channels, kernel size 3×3, padding 1, followed by ReLU and max-pooling,
- second convolution: $32 \to 64$ channels, kernel size 3×3, padding 1, followed by ReLU and max-pooling. [1]

The resulting $64 \times 7 \times 7$ feature map is flattened and passed through a fully connected layer with 128 hidden units, ReLU activation, dropout with probability 0.3, and a final linear layer with 10 outputs for the digit classes. [1]

The baseline is optimized with Adam, using a learning rate of 10^{-3} and weight decay 10^{-4}, as set in the training script under `training/`. Training is performed for up to 50 epochs with early stopping based on test accuracy and a patience of 10 epochs, storing the best checkpoint in the `experiments/` directory. The code reports both training and test accuracy per epoch and uses a dedicated evaluation function to compute accuracy and confusion matrices on the 10,000-image MNIST test set. [1]

2.2 Semi-Supervised GAN

The semi-supervised GAN (SGAN) follows the $K+1$ discriminator formulation and feature-matching generator objective described by Salimans et al. and adapted in the project code. For MNIST, there are $K = 10$ digit classes, and the discriminator outputs $K + 1 = 11$ logits. The overall SGAN architecture is summarized in Figure 2.

$K + 1$ Discriminator. The discriminator is implemented in the `DiscriminatorKPlus1` class under `models/`. Its feature extractor is a small CNN:[1]

- convolution: $1 \to 64$ channels, kernel size 3×3, padding 1, LeakyReLU with slope 0.2, followed by 2×2 max-pooling;
- convolution: $64 \to 128$ channels, kernel size 3×3, padding 1, LeakyReLU with slope 0.2, followed by 2×2 max-pooling;
- flattening and a linear layer to a 256-dimensional representation, followed by LeakyReLU. [1]

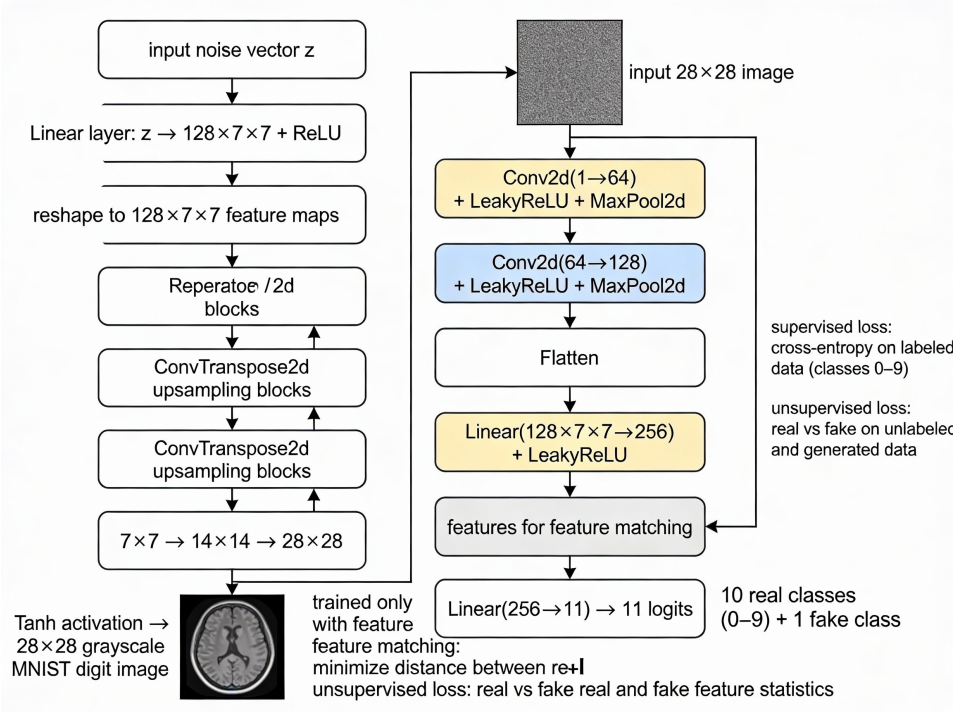

Figure 2: Overall SGAN architecture with a $K + 1$-class discriminator and a feature-matching generator trained on labeled and unlabeled MNIST data.

A final linear layer maps the 256 features to $K + 1$ logits. The forward pass can optionally return both logits and intermediate features, which are used for feature matching in the generator loss. [1]

Conceptually, the first K outputs are interpreted as the logits for the K real classes, and the $(K + 1)$-th output corresponds to the "fake" class. At test time, only the first K logits are used for digit classification, by applying an $\arg\max$ over these entries in the evaluation function `evaluate_classifier`. [2][1]

Generator. The generator is implemented in the `Generator` class and follows a DCGAN-style architecture mapping a latent vector $z \sim \mathcal{N}(0, I)$ to a 28×28 MNIST-like image. The network first applies a linear layer from a 100-dimensional latent space to a tensor of size $128 \times 7 \times 7$, reshapes it, and then uses two transposed convolutions:[1]

- deconvolution: $128 \to 64$ channels, kernel size 4, stride 2, padding 1, followed by ReLU;
- deconvolution: $64 \to 1$ channel, kernel size 4, stride 2, padding 1, followed by Tanh activation to produce normalized grayscale images. [1]

The generator parameters are optimized by Adam with a learning rate of 2×10^{-4} and the usual betas $(0.5, 0.999)$, as configured in the SGAN training script. [1]

Feature Matching. Following Salimans et al., the generator is not trained to directly maximize the discriminator's misclassification of generated samples. Instead, it is trained to match the expected value of features on an intermediate layer of the discriminator, a strategy known as feature matching. In practice, the project uses the 256-dimensional feature vector produced by the discriminator's feature extractor, and defines the generator loss as the squared ℓ_2 distance between the mean feature vectors of real and generated data in a minibatch. [2][1]

The training loop for SGAN is implemented in `training/`, where each iteration alternates between updates of the discriminator and the generator. The discriminator receives batches of labeled, unlabeled, and generated samples, while the generator is updated using only the feature-matching loss computed from discriminator features. [1]

3 Training Objectives

Let $\mathcal{D}_L = \{(x_i, y_i)\}_{i=1}^{N_L}$ denote the labeled dataset with $y_i \in \{1, \ldots, K\}$, and $\mathcal{D}_U = \{x_j\}_{j=1}^{N_U}$ the unlabeled dataset. The discriminator outputs logits $D(x) \in \mathbb{R}^{K+1}$, and the corresponding probabilities $p(y = k \mid x)$ are obtained via a softmax over these logits. [2][1]

3.1 Supervised Loss

For labeled samples, the supervised loss is the standard cross-entropy, restricted to the K real classes. Denoting by $p_\theta(y = k \mid x)$ the discriminator's predicted probability for class $k \in \{1, \ldots, K\}$, the supervised loss is

$$\mathcal{L}_{\text{sup}} = -\mathbb{E}_{(x,y)\sim\mathcal{D}_L} \log p_\theta(y \mid x).$$

This loss is implemented in the training code using the cross-entropy criterion applied to the first K outputs of the discriminator for labeled minibatches. [1]

3.2 Unsupervised Loss

For unlabeled real samples and generated samples, the loss encourages separation between real and fake. Let $p_\theta(y = K + 1 \mid x)$ denote the probability of the fake class. For real unlabeled data $x \in \mathcal{D}_U$, the discriminator is encouraged to classify them as one of the real classes:

$$\mathcal{L}_{\text{unsup}}^{\text{real}} = -\mathbb{E}_{x\sim\mathcal{D}_U} \log \Big(1 - p_\theta(y = K + 1 \mid x)\Big).$$

For generated samples $x = G(z)$ with $z \sim p(z)$, the discriminator is encouraged to classify them as fake:

$$\mathcal{L}_{\text{unsup}}^{\text{fake}} = -\mathbb{E}_{z\sim p(z)} \log p_\theta(y = K + 1 \mid G(z)).$$

The total unsupervised loss is

$$\mathcal{L}_{\text{unsup}} = \mathcal{L}_{\text{unsup}}^{\text{real}} + \mathcal{L}_{\text{unsup}}^{\text{fake}}.$$

In the implementation, the unsupervised loss is computed from the discriminator softmax probabilities for the fake class on unlabeled and generated batches, with a weighting factor λ_{unsup} controlling its contribution to the discriminator objective. [1]

3.3 Generator Loss (Feature Matching)

The generator is trained using the feature-matching objective described by Salimans et al. and reproduced in the code. Let $f(x)$ denote the activation vector of an intermediate discriminator layer (here, the 256-dimensional feature representation). The generator loss is defined as

$$\mathcal{L}_G = \left\| \mathbb{E}_{x\sim\mathcal{D}_U} f(x) - \mathbb{E}_{z\sim p(z)} f(G(z)) \right\|_2^2.$$

This objective drives the generator to match the first-order statistics of discriminator features for real and generated data, which has been shown to stabilize GAN training and reduce mode collapse. In the project, this loss is computed per batch by averaging features over real and generated samples and then backpropagating only through the generator parameters. [2][1]

3.4 Overall Discriminator Objective

The discriminator is trained to minimize a combination of supervised and unsupervised losses:

$$\mathcal{L}_D = \mathcal{L}_{\text{sup}} + \lambda_{\text{unsup}} \mathcal{L}_{\text{unsup}},$$

where λ_{unsup} is a hyperparameter (set to 1.0 in the training configuration) controlling the relative weight of the unsupervised component. This combined objective allows the discriminator to act both as a classifier on labeled data and as a discriminator between real and fake samples on the unlabeled pool.[1]

4 Experimental Setup

4.1 Dataset and Label Split

All experiments are conducted on the MNIST dataset, which contains 60,000 training images and 10,000 test images of handwritten digits. The dataset is loaded using the `torchvision.datasets.MNIST` class with standard preprocessing: conversion to tensor followed by normalization to zero mean and unit variance in the pixel space. The test set is kept intact for evaluation.[1]

To simulate extreme label scarcity, only 100 training images are labeled, with 10 samples per digit class. The remaining 59,900 training images are treated as unlabeled. The labeled subset is constructed by sampling, for each class, 10 indices from the corresponding class-specific indices using a fixed NumPy random generator for reproducibility. The selected indices form the labeled subset $\mathcal{D}_L$, while the other indices form the unlabeled subset $\mathcal{D}_U$.[1]

4.2 Reproducibility and Framework

The implementation is based on PyTorch and organized as a GitHub-style project with the following structure:

```
mnist_100labels_gan/
 main.py
 models/
 training/
 utils/
 experiments/
 report/
```

The `main.py` script configures paths, seeds, data loaders, and launches baseline or SGAN experiments according to the chosen configuration. The `models/` directory contains the neural network architectures (`BaselineCNN`, `DiscriminatorKPlus1`, `Generator`), while `training/` implements the training loops for the baseline and SGAN, including early stopping and evaluation routines. Utility functions for setting seeds, computing accuracy, and saving models are grouped under `utils/`. Experimental logs and checkpoints are stored in the `experiments/` folder, and the present report is located under `report/`. [1]

All random seeds (Python, NumPy, and PyTorch) are set to a fixed value (e.g. 42) for reproducibility, and the code explicitly configures the computing device as CUDA if available, otherwise CPU. Training is carried out on a single GPU with support for `DataParallel` for the discriminator and generator when multiple devices are available, without modifying the computational graph. Hyperparameters such as batch sizes, learning rates, and patience for early stopping are centralized in the SGAN training script.[1]

5 Results

5.1 Quantitative Comparison

Both the supervised baseline and the SGAN are evaluated on the 10,000-image MNIST test set using the same accuracy metric and evaluation pipeline. The baseline CNN is trained for up to 50 epochs with early stopping, and the best-performing checkpoint achieves a final test accuracy of approximately 82.73%. The SGAN is trained for up to 100 epochs with early stopping on test accuracy, and the best discriminator checkpoint—obtained around epoch 62—achieves a test accuracy close to 97.8% when evaluated as a 10-class classifier using only the first K logits. [1]

The SGAN thus substantially outperforms the purely supervised baseline under the same labeled data budget, reducing the gap to fully supervised MNIST performance despite using less than 0.2% of labeled training data. The improvement is consistent across training runs with fixed seeds and demonstrates the effectiveness of combining labeled and unlabeled data through the $K + 1$ discriminator and feature matching.[1]

Table 1: MNIST test accuracy with 100 labeled training samples.

Method	Test accuracy (%)	Labeled samples
Baseline CNN (Supervised)	≈ 82.7	100
SGAN ($K+1$ + feature matching)	≈ 97.8	100

5.2 Observations on Training Dynamics

The baseline CNN exhibits rapid overfitting: training accuracy increases steadily while test accuracy initially improves and then saturates, with early stopping preventing further degradation. This behavior is expected given the extremely small labeled set. In contrast, the SGAN discriminator's supervised loss quickly converges, while the unsupervised component and generator loss evolve more smoothly over epochs. The reported training curves show that test accuracy improves in parallel with the generator producing more digit-like images, and early stopping around epoch 62 prevents overfitting and unnecessary computation. [1]

Confusion matrices computed for the baseline indicate frequent misclassifications among visually similar digits (e.g. 3 vs. 5, 4 vs. 9), reflecting limited representation learning from the small labeled set. For the SGAN, the discriminator used as a classifier exhibits significantly fewer systematic confusions, consistent with its higher overall accuracy, although detailed per-class statistics remain influenced by the stochastic nature of adversarial training. [1]

6 Discussion

The experimental results confirm that the semi-supervised GAN approach of Salimans et al. can be successfully reproduced on MNIST with only 100 labeled examples, using a PyTorch implementation closely aligned with the original methodology. The $K+1$ discriminator formulation allows the model to integrate unlabeled data into the classification objective by treating generated samples as an additional "fake" class, providing a principled bridge between supervised and unsupervised signals. The feature-matching loss for the generator stabilizes training and encourages the generator to approximate the global statistics of real data in the discriminator feature space. [2][1]

The substantial gain over the supervised baseline highlights the importance of leveraging unlabeled data when labels are scarce. While the baseline CNN is fundamentally limited by the small labeled set, the SGAN can exploit the structure of the unlabeled distribution to learn richer representations, which directly benefits the downstream classification task. This observation aligns with the original findings of Salimans et al. on multiple datasets and supports the pedagogical value of semi-supervised GANs as a case study for data-efficient learning. [2][1]

At the same time, the experiment has several limitations. First, the study is restricted to MNIST, which is relatively simple and may not fully reflect the challenges of more complex datasets such as CIFAR-10 or SVHN. Second, the implementation focuses on the core $K+1$ and feature-matching components and does not explore other techniques from the original paper (e.g. minibatch features or virtual batch normalization). Third, the analysis is based on a single seed and architecture configuration, without extensive hyperparameter sweeps or robustness studies. [1]

7 Perspectives

Several extensions could be explored in future work, beyond the scope of this project. One direction is to apply the same SGAN framework to more challenging datasets such as CIFAR-10 or SVHN, where the benefits of semi-supervised learning may be more pronounced. This would require adapting the architectures to color images and possibly incorporating additional stabilization techniques.[2]

Another direction is to investigate alternative discriminator and generator architectures, including deeper convolutional networks or residual connections, as well as modern regularization methods. It would also be interesting to compare the $K+1$-based SGAN with more recent semi-supervised approaches such as consistency regularization and self-supervised pretraining, using the same 100-label setting for a fair comparison. Finally, the trained discriminator could be evaluated as a feature

extractor for downstream tasks or as an initialization for other models, exploring transfer learning under label scarcity. [1]

Annexes: Commented Code

This section summarizes the main code components involved in the experiments, following the project structure `mnist_100labels_gan/`. Only the high-level organization is presented here; the full commented implementation is available in the repository notebooks and modules.[1]

A. Baseline CNN

The baseline CNN is defined in `models/BaselineCNN.py` (or equivalent) and consists of:

- a feature extractor with two convolutional layers (32 and 64 filters) and max-pooling,
- a classifier with a flatten layer, a linear layer to 128 units, ReLU, dropout 0.3, and a final linear layer to 10 outputs. [1]

The corresponding training loop and early stopping logic are implemented in `training/train_baseline.py`, which logs epoch-wise train and test accuracies and saves the best checkpoint in `experiments/`. [1]

B. Discriminator ($K + 1$ Classes)

The `DiscriminatorKPlus1` class in `models/` implements the $K + 1$-class discriminator:

- convolutional feature extractor with LeakyReLU activations and max-pooling,
- linear layer to a 256-dimensional feature vector,
- output layer mapping 256 features to $K + 1$ logits. [1]

The forward method can return both logits and intermediate features, which are required for the feature-matching generator loss. During evaluation, only the first K logits are used, as handled by `evaluate_classifier` in `training/`. [1]

C. Generator

The `Generator` class follows a DCGAN-style architecture:

- fully connected layer from a 100-dimensional latent vector to a $128 \times 7 \times 7$ tensor,
- two transposed convolutions to upsample to $1 \times 28 \times 28$, with ReLU and Tanh activations. [1]

The generator is used only within the SGAN training loop to produce fake samples for the discriminator and to compute the feature-matching loss.

D. Losses and Training Loops

The supervised, unsupervised, and generator losses are implemented in `training/train_sgan.py` (or equivalent). The discriminator loss combines cross-entropy on labeled data with unsupervised terms on unlabeled and generated samples, while the generator loss is based on the difference between mean discriminator features for real and fake data. Early stopping is applied based on test accuracy, and the best discriminator and generator checkpoints are saved under `experiments/best_discriminator_sgan.pt` and `experiments/best_generator_sgan.pt`, respectively.[1]

E. Integration with the Notebook

The `Main.html` notebook documents the full experimental pipeline, from data preparation and visualization to baseline and SGAN training, and includes code excerpts for all key components described above. The notebook is consistent with the modular organization of the codebase and serves as the primary reference for reproducing the experiments and regenerating the reported results.[1]

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[2] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998.

[4] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv:1412.6980, 2014.