Network Inversion for Uncertainty-Aware Out-of-Distribution Detection

Shaik Rehna Afroz (22B3932)
EE 691 R&D Project
IIT BOMBAY

May 7, 2025

Contents

1 Introduction

1.1 Problem Statement

- Classifiers are excellent at distinguishing between in-distribution classes but often struggle when presented with out-of-distribution (OOD) samples, misclassifying them with high confidence.
- OOD samples are data points that don't belong to the same distribution as the classifier's training data.
- A model trained on MNIST digits (0–9) may incorrectly classify inputs like clothing images (e.g., trousers from FashionMNIST) with high confidence. This misclassification is dangerous in real-world deployments (e.g., medical imaging, autonomous driving) where OOD inputs can lead to catastrophic outcomes.

  Example: A classifier trained on MNIST is shown a FashionMNIST image. It labels the image as a digit (e.g.,"3") with 95% confidence—despite the image being completely unrelated.
  We need to make the model which flags these kind of inputs as OOD and also if it misclassifies the OOD data it should do it with low confidence

1.2 Why Should This Problem Be Solved

- Solving OOD detection is critical for building safe, interpretable, and deployable AI systems.
- **Safety**: OOD inputs in medical, industrial, or autonomous systems can cause fatal decisions.
- **Generalization**: Robust systems should recognize when an input does not belong to the training distribution.
- **Trust**: Users need interpretable confidence estimates to trust predictions.
- **Performance**: OOD-aware models avoid confidently wrong outputs, leading to better downstream decision-making.

1.3 Why Is It a Challenging Problem

- **Data Distribution Gaps**: Real-world OOD samples can be very different from training data.
- **Confidence Miscalibration**: Softmax outputs are often overconfident, even for unfamiliar inputs.

2 Literature Review

2.1 Network Inversion

- In recent times, the applications of Neural Networks(NN) increased a lot. From image recognition, medical diagnosis to self-driving cars, NN are almost everywhere.
- Despite of their huge usage, they mostly remain like a blackbox.We generally know the output from a NN, but not the internal learnings happened in the NN.This hinders their interpretabiltiy and reliabilty

2.1.1 Paper : Network Inversion and its Applications

- The paper ”Network Inversion and its Applications” (https://arxiv.org/pdf/2411.17777) introduces Network Inversion (NI)- a technique to address this issue by determining the input from a known output and system mapping, providing insights into NN decision-making.
- **Methodology:**
- This method involves three key stages:
  First, the classifier is trained and frozen (weights locked).
  Next, an untrained generator (with conditioning mechanisms) learns to synthesize diverse input samples that would produce specific target outputs when fed through the frozen classifier.
  This conditioned generator effectively inverts the classifier's mapping - creating input distributions that reliably trigger desired output classifications while maintaining output diversity.
- Key characteristics:
  Classifier remains fixed during inversion
  Generator training is driven by classifier feedback
  Conditioning ensures label-consistency in generated samples
  Diversity losses prevent mode collapse
- This approach essentially ”reverse-engineers” the classifier's decision boundaries by exploring the input space that would lead to particular classifications.”

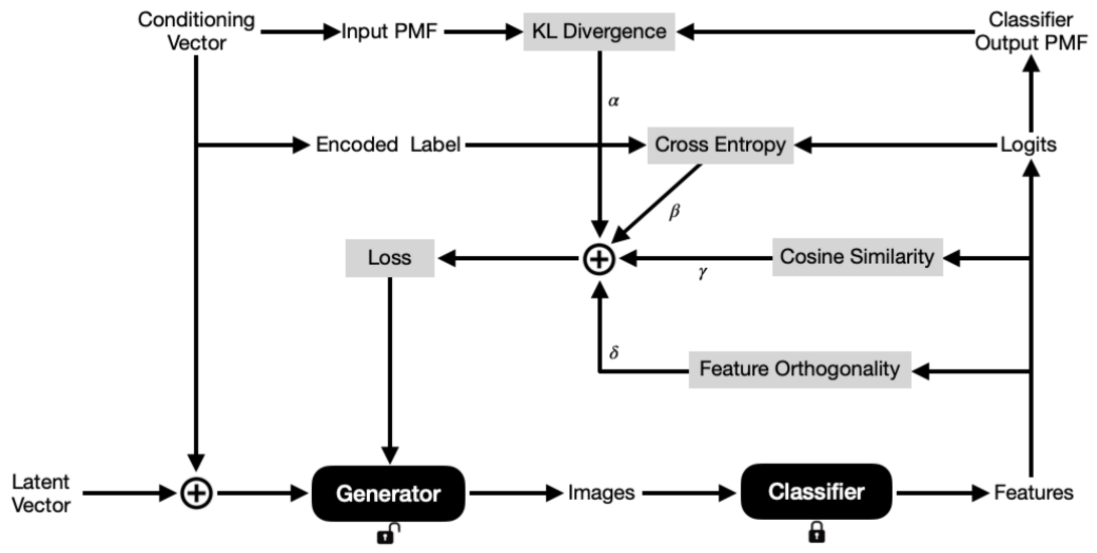


Figure 1. Proposed Approach to Network Inversion

- **Conditioning Mechanisms:**
- **Label conditioning:**
  In generative tasks, generators are conditioned on label embeddings (e.g., in cGANs), where the label encodes class information. However, network inversion requires more than simple embeddings because multiple inputs can lead to the same output. Thus, the generator needs richer guidance to capture input diversity, which is achieved using vectors and matrices
- **Vector conditioning:**
  For an N-class task, N-dimensional vectors are generated from a normal distribution, then softmaxed to represent the input distribution. The argmax index of the softmaxed vector serves as the conditioning label in the loss function, improving the generator's performance
- **Intermediate Matrix Conditioning:**
  Matrix conditioning uses an NXN matrix where the elements in a specific row and column (same index) are set to 1, encoding label information. This conditioning matrix is concatenated with the latent vector intermediately after up-sampling it to NXN spatial dimensions, while the generation upto this point remains unconditioned.
- **Vector-Matrix Conditioning:**
  In Intermediate Matrix Conditioning, the generation starts unconditioned. Both vector and matrix conditioning are combined: vectors are used to condition the generator up to NXN spatial dimensions, after which the conditioning matrix is concatenated for further generation. The argmax index of the vector, corresponding to the row or column set to high in the matrix, serves as the conditioning label.
- Network Inversion (NI) aims to generate images that, when passed through a classifier, produce the same label as the conditioned generator. Using a straightforward cross-entropy loss can cause mode collapse, reducing diversity. To address this, the generator is trained to learn the data distribution of different classes, with a combined loss function L_{inv}:

$$\mathcal{L}_{\text{Inv}} = \alpha \cdot \mathcal{L}_{\text{KL}} + \beta \cdot \mathcal{L}_{\text{CE}} + \gamma \cdot \mathcal{L}_{\text{Cosine}} + \delta \cdot \mathcal{L}_{\text{Ortho}}$$

where $\mathcal{L}_{\text{KL}}$ is the KL Divergence loss, $\mathcal{L}_{\text{CE}}$ is the Cross Entropy loss, $\mathcal{L}_{\text{Cosine}}$ is the Cosine Similarity loss, and $\mathcal{L}_{\text{Ortho}}$ is the Feature Orthogonality loss. The hyperparameters $\alpha, \beta, \gamma, \delta$ control the contribution of each individual loss term defined as:

$$\mathcal{L}_{\text{KL}} = D_{\text{KL}}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

$$\mathcal{L}_{\text{CE}} = -\sum_i y_i \log(\hat{y}_i)$$

$$\mathcal{L}_{\text{Cosine}} = \frac{1}{N(N-1)} \sum_{i \neq j} \cos(\theta_{ij})$$

$$\mathcal{L}_{\text{Ortho}} = \frac{1}{N^2} \sum_{i,j} (G_{ij} - \delta_{ij})^2$$

- **Loss Functions:**
- **Cross Entropy (L_{CE}):**
  Cross Entropy is used to ensure the generator produces images with the correct labels. When labels are encoded into vectors, they are used in the loss function, allowing the generator to learn the conditioned image distribution. Training with vector conditioning requires the generator to learn the encoding mechanism for better exploration of the classifier's input space.
- **KL Divergence (L_{KL}):**
  KL Divergence minimizes the difference between the output distribution of generated images and the conditioning distribution, ensuring the generator learns the data distribution effectively.
- **Cosine Similarity (L_{Cosine}):**
  Cosine Similarity encourages diversity by reducing the angular distance between generated image features, ensuring both correct classification and diverse image generation for each label.
- **Feature Orthogonality (L_{Ortho}):**
  In Intermediate Matrix Conditioning, the generation starts unconditioned. Both vector and matrix conditioning are combined: vectors are used to condition the generator up to NXN spatial dimensions, after which the conditioning matrix is concatenated for further generation. The argmax index of the vector, corresponding to the row or column set to high in the matrix, serves as the conditioning label.
- **Applications**:
- **Interpretability** :
  The interpretability of inverted samples is analyzed by their features in the penultimate layer of the classifier. Inversion Accuracy refers to the percentage of images generated with desired labels same as the output labels from the classifier. The generator is trained to achieve an Inversion Accuracy $> 95\%$, meaning that most generated images are classified correctly.
- **Training-Like Data Reconstruction**
- **Out-of-Distribution detection**

2.1.2 Paper: Network Inversion for Training-Like Data Reconstruction

- Through Network Inversion we get a diverse set of images in the input space of the model for different classes. But the inverted samples are completely random
- The paper " Network Inversion for Training-Like Data Reconstruction " (https://arxiv.org/pdf/2410.16884) uses Network inversion for reconstructing training-like data entirely from the weights of the classifier without any insight of the training process.
- Reconstruction of training-like data was achieved by exploiting key properties of the training data in relation to the classifier including model confidence, robustness to perturbations gradient behavior along with some prior knowledge about the training data.

2.1.3 Paper: NETWORK INVERSION OF BINARISED NEURAL NETS

- The paper "NETWORK INVERSION OF BINARISED NEURAL NETS " (https://arxiv.org/pdf/2402.11995) introduces a novel approach to invert a BNN by encoding it into a Conjunctive Normal Form (CNF) propositional formula, which captures the network's structure.
- CNF formula: Encodes the computation of each neuron in the network and allows for both inference and inversion using satisfiability (SAT) solvers.
- By setting constraints on output variables, the formula can be used for inversion, generating inputs corresponding to a specific output.
- This method does not require hyperparameter tuning, unlike optimization-based approaches.
- The deterministic nature of CNF encodings ensures fine-grained control over the inverted samples, which helps mitigate issues like mode collapse commonly seen in generative models.

2.2 Uncertainty Estimation

2.2.1 Paper: Autoinverse: Uncertainty Aware Inversion of Neural Networks

- This paper(https://arxiv.org/abs/2208.13780v2) proposes Autoinverse - a highly automated approach for inverting neural network surrogates. This approach seeks inverse solutions in the vicinity of reliable data by taking into account the predictive uncertainty of the surrogate and minimizing it during inversion.
- Autoinverse improves neural inversion by considering predictive uncertainty of the surrogate model.It finds solutions near reliable training data, minimizing errors from poorly sampled or noisy data regions.Instead of forcing the surrogate to perfectly match the Native Forward Process(NFP), it adapts the inversion process to account for uncertainty.
- Autoinverse achieves this goal by, first by training a surrogate capable of predictive uncertainty. Second, relying on this trained surrogate and using a novel inversion cost function, Autoinverse finds accurate designs with minimal uncertainty. Autoinverse is applied on two inverse methods belonging to the two main neural inversion categories, i.e., optimization- and architecture-based

2.2.2 Paper: Uncertainty-Aware Out-of-Distribution Detection with Gaussian Processes

- This paper (https://arxiv.org/pdf/2412.20918) proposes proposes a Gaussian-process-based method for out-of-distribution (OOD) detection in deep neural networks (DNNs) that eliminate the need for OOD data during training.
- This approach includes multi-class Gaussian processes (GPs) to quantify uncertainty in unconstrained Softmax scores of a DNN and defines a score function based on the predictive distribution of the multi-class GP to distinguish in-distribution(InD) and OOD samples.
- This method is compatible with existing DNN architectures and requires only InD data for training, addressing a key limitation of prior OOD detection techniques.

3 Our Approach:

- To address the problem of OOD detection, Network inversion is used to generate OOD samples from the classifier's input space. These samples, which deviate from the trained distribution, are assigned to a "garbage" class
- This method uses an n+1 approach, where n is the number of classes present in the training data. Extra "garbage" class is added to the classifier's output for ood detection
- To make the classifier familiar with the OOD data, it is trained with an additional "garbage" class, initially populated by samples generated from random noise. This helps the classifier learn to recognize and isolate OOD samples

3.1 Training Process:

- After each training epoch, inverted samples for each class are added to the "garbage" class, further augmenting the dataset with OOD examples.
- This iterative retraining process helps the classifier differentiate between in-distribution and OOD data, improving its robustness.

3.2 Addressing class Imbalance:

- The inclusion of the "garbage" class creates a data imbalance, which is managed using a weighted cross-entropy loss function. This function dynamically adjusts weights to ensure effective learning despite the imbalance.

3.3 Classifier Architecture

The classifier is implemented as a Convolutional Neural Network (CNN) designed for 11-class classification (digits 0–9 and Garbage class). Below is a layer-wise breakdown:

- **Input:** Grayscale image of size $1 \times 28 \times 28$.
- **Convolutional Block 1:**
  - `Conv2d(1, 32, kernel_size=3, padding=1)`: Outputs 32 feature maps of size 28×28.
  - `BatchNorm2d(32)`: Normalizes activations for stable training.
  - `LeakyReLU`: Non-linear activation function.
  - `MaxPool2d(2, 2)`: Downsamples to 14×14.
- **Convolutional Block 2:**
  - `Conv2d(32, 64, kernel_size=3, padding=1)`: Outputs 64 feature maps.
  - `BatchNorm2d(64)`: Further stabilizes feature distribution.
  - `LeakyReLU`: Activation function.
  - `MaxPool2d(2, 2)`: Downsamples to 7×7.
- **Flatten Layer:**
  - Reshapes the output to a vector of size $64 \times 7 \times 7 = 3136$.

- **Fully Connected Layer 1 (Penultimate):**
  - `Linear(3136, 128)`: Reduces dimensionality.
  - `LeakyReLU`: Activation function.
  - `Dropout(0.5)`: Randomly zeroes 50% of features during training to reduce overfitting.
- **Output Layer:**
  - `Linear(128, 11)`: Produces logits for 11 classes.
- **Outputs:**
  - `logits`: Final class scores.
  - `features`: Penultimate layer features for losses and visualization.

3.4 Generator Design

The generator synthesizes class-conditional grayscale images of shape $[B, 1, 28, 28]$, where B is the batch size. It takes as input a latent vector and a soft class-conditioning vector(Vector Conditioning), and follows a structured upsampling path:

- **Input:**
  - `z` $\in \mathbb{R}^{B\times d}$: Latent vector sampled from normal distribution $\mathcal{N}(0, I)$, where d is `latent_dim`.
  - `cond` $\in \mathbb{R}^{B\times C}$: Class-conditioning vector (softmaxed), where C is `num_classes`.
  - The two are concatenated to form a tensor of shape $[B, d + C]$.
- **Fully Connected Projection:**
  - `Linear(d+C, 128 × 7 × 7)` projects the input to shape $[B, 6272]$.
  - `BatchNorm1d` normalizes across the batch.
  - `ReLU` introduces non-linearity.
  - Output is reshaped to $[B, 128, 7, 7]$.
- **Upsampling via Transposed Convolutions:**
  - `ConvTranspose2d(128, 64, 4, 2, 1)` $\rightarrow [B, 64, 14, 14]$
  - `BatchNorm2d`, `ReLU`
  - `ConvTranspose2d(64, 32, 4, 2, 1)` $\rightarrow [B, 32, 28, 28]$
  - `BatchNorm2d`, `ReLU`
- **Final Convolution:**
  - `Conv2d(32, 1, kernel_size=3, padding=1)` $\rightarrow [B, 1, 28, 28]$
  - `Tanh()` activation constrains output pixels to range $[-1, 1]$
- **Output:**
  - Final generated image tensor of shape $[B, 1, 28, 28]$.

3.5 Loss Functions

The following loss functions that are mentioned earlier are used for the training of the generator

- Cross-Entropy Loss
- KL Divergence
- Cosine Similarity
- Feature Orthogonality Loss

3.6 Uncertainty Estimation Method

To quantify the model's uncertainty on a given input, we use a normalized deviation from the uniform distribution over class probabilities. The uncertainty estimate lies in the range $[0, 1]$, where values closer to 1 indicate higher uncertainty.

3.7 Procedure

Given the classifier logits $z \in \mathbb{R}^{B\times K}$ for a batch of B samples and K classes:

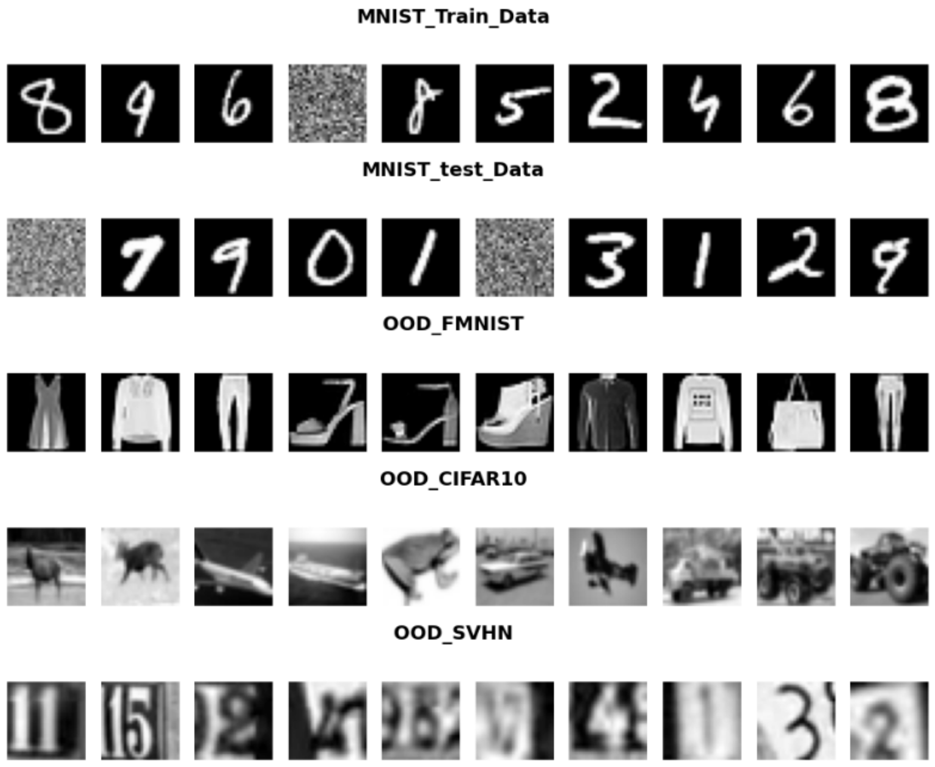
1. Compute softmax probabilities:
$$p = \text{Softmax}(z), \quad p \in \mathbb{R}^{B\times K}$$
2. Define a uniform distribution over K classes:
$$u = \left[\frac{1}{K}, \frac{1}{K}, \dots, \frac{1}{K}\right] \in \mathbb{R}^{K}$$
3. Compute squared deviation of the predicted probabilities from the uniform distribution:
$$a = \sum_{i=1}^{K} (p_i - u_i)^2$$
4. Compute squared deviation of the predicted one-hot class (argmax) from the uniform:
$$b = \sum_{i=1}^{K} (\hat{y}_i - u_i)^2, \quad \text{where } \hat{y} = \text{one-hot}(\arg\max(p))$$
5. Final uncertainty estimate:
$$\mathcal{U}(x) = 1 - \frac{a}{b}$$
This normalizes the deviation of the model's predicted distribution from the uniform baseline. A value of $\mathcal{U}(x)$ close to 0 implies high confidence, whereas values near 1 reflect a distribution close to uniform (i.e., maximum uncertainty).

3.8 Training Strategy and Epoch Flow

- The proposed pipeline performs iterative generator training using a frozen classifier, inversion accuracy estimation, uncertainty quantification, and classifier refinement. The process is designed to gradually improve the generator's ability to produce class-consistent samples while updating the classifier to better generalize when exposed to both inversion-generated synthetic samples and potential out-of-distribution (OOD) data
- MNIST dataset is used for training and testing was done on FMNIST, CIFAR10, SVHN

3.9 Data:

- The two parts of MNIST dataset : mnist-train(60000 images) and mnist-test(10000 images) are downloaded and appropriate image transformations are done(changing to tensors and normalizing to [-1,1])
- Since, mnist-train has 60000 images, 6000 guassian noise samples are added to it, ensuring balanced dataset This creates the training set on which the classifier will be trained for 1 epoch at the beginning
- Similarly, 1000 guassian noise samples are added to mnist-test dataset
- The train and test sets of the FMNIST are combined to make a single ood-test dataset
- Since, CIFAR10, SVHN are 32x32 RGB images, they are converted to 28x28 gray scale images
- Similar to the FMNIST, the train and test sets of the SVHN, CIFAR10 are also combined to create two more ood-test datasets

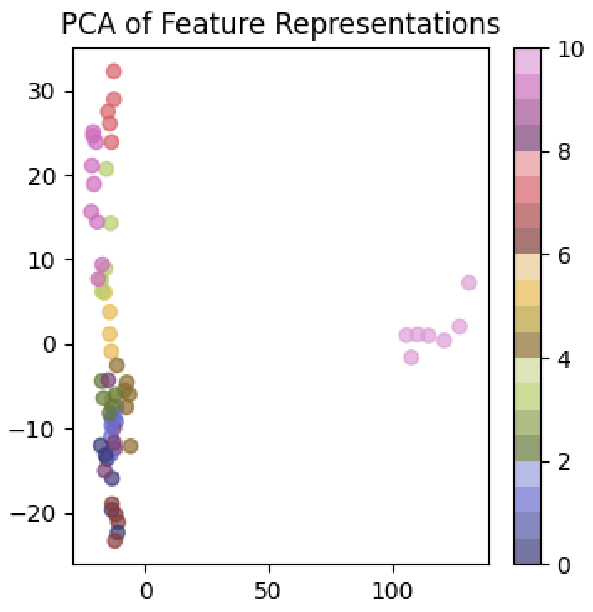


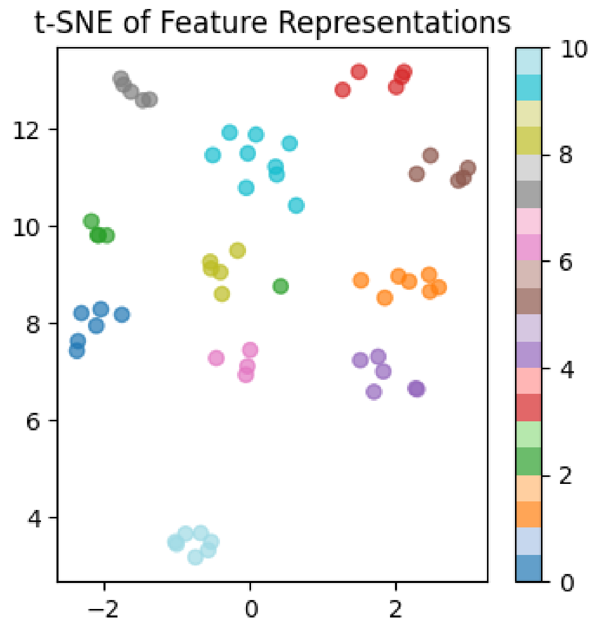
3.10 Hyperparameters and Settings

- Inversion Loss Weights: α, β, γ, δ
- Latent Dimension: 100
- Number of Classes: 11 (MNIST + Garbage class)
- Batch Size: 64
- Generator Steps per Epoch: 250
- Number of Epochs

3.11 Pre-training the Classifier for 1 epoch

- At the very beginning the classifier is pre-trained on the (mnist-train + guassian nosie) dataset for 1 epoch.
- Below figures show the PCA, t-SNE plot of this features generated for the 11 classses by this trained classifer
- The isolated dots in the PCA, t-SNE plots represent the garbage class





3.12 Overview of Iterative Process

1. **Initialize and Load Pretrained Classifier**:
   Load the pretrained classifier trained for 1 epoch on (MNIST train + Gaussian noise) dataset. Freeze its weights during generator updates.

2. **Generator Training (250 steps per epoch)**:
   - Sample latent vectors $z \sim \mathcal{N}(0, I)$ and random class-conditioning vectors $\mathbf{c}$ (softmax applied).
   - Generate synthetic images $\hat{x} = G(z, \mathbf{c})$.
   - Compute inversion loss $\mathcal{L}_{\text{inv}}$:

   $$\mathcal{L}_{\text{inv}} = \alpha \cdot \text{KL} + \beta \cdot \text{CE} + \gamma \cdot \text{CosSim} + \delta \cdot \text{Orthogonality}$$

   - Backpropagate and update generator.

3. **Uncertainty Estimation**:
   - Pass generated images through the classifier.
   - Estimate per-class uncertainty using the method described in section 3.6: Uncertainty Estimation Method .

4. **Save Generated Samples**:
   - Save generated images and label-wise sample grids.
   - Store conditioning and predicted labels for reproducibility and analysis.

5. **Classifier Retraining**:
   - Add generated samples to the training set.
   - Compute class-balanced loss weights.
   - Train classifier for one epoch with updated data.
   - Save model checkpoint for use in the next epoch.

3.13 Final Outputs

The process logs the following:

- Uncertainty estimates per class and epoch.
- Training loss and accuracy of the updated classifier.
- Saved checkpoints and generated images for qualitative inspection.
- This iterative scheme strengthens the generator's conditioning capability and encourages the classifier to adapt to hard or uncertain regions in the data space.

4 Experiments and Results

- Several experiments were conducted, during which the code was iteratively debugged and refined, hyperparameters were tuned. The experiments detailed below were performed using the finalized and corrected implementation.

4.1 Experiment -1

4.1.1 Hyperparameters and Settings

- Inversion Loss Weights: $\alpha = 500$, $\beta = 500$, $\gamma = 6000$, $\delta = 10$
- Latent Dimension: 100, Batch Size: 64
- Number of Classes: 11 (MNIST + Garbage class)
- Generator Steps per Epoch: 250
- Number of samples generated in each step = 1000
- Number of Epochs: 10

4.1.2 Test Accuarcy on OOD Datasets

Dataset	Before Inversion (%)	After Inversion (%)
Fashion-MNIST (FMNIST)	28.81	71.89
CIFAR-10	87.3	99.99
SVHN	87.71	100

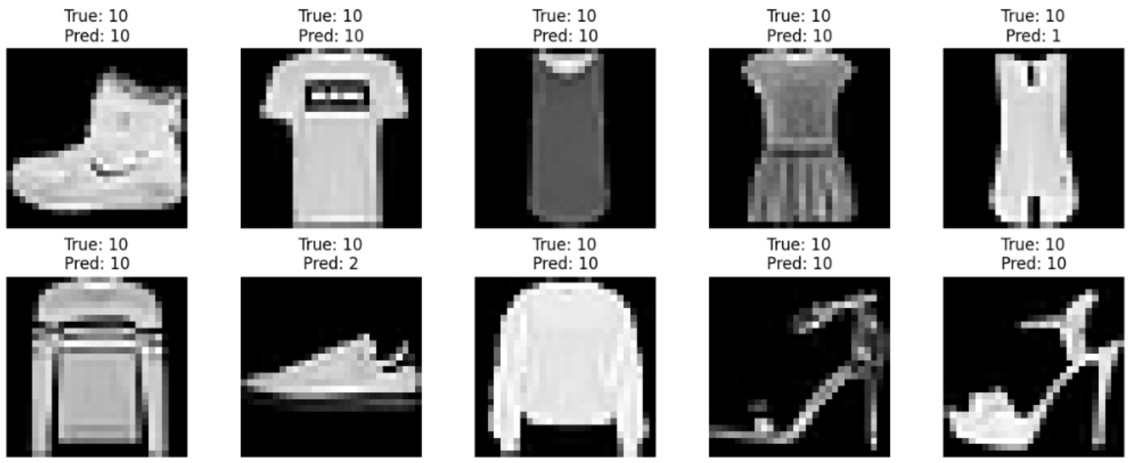
- The test accuracy of the classifier on the ood datatsets improved signinficantly after training using network inversion

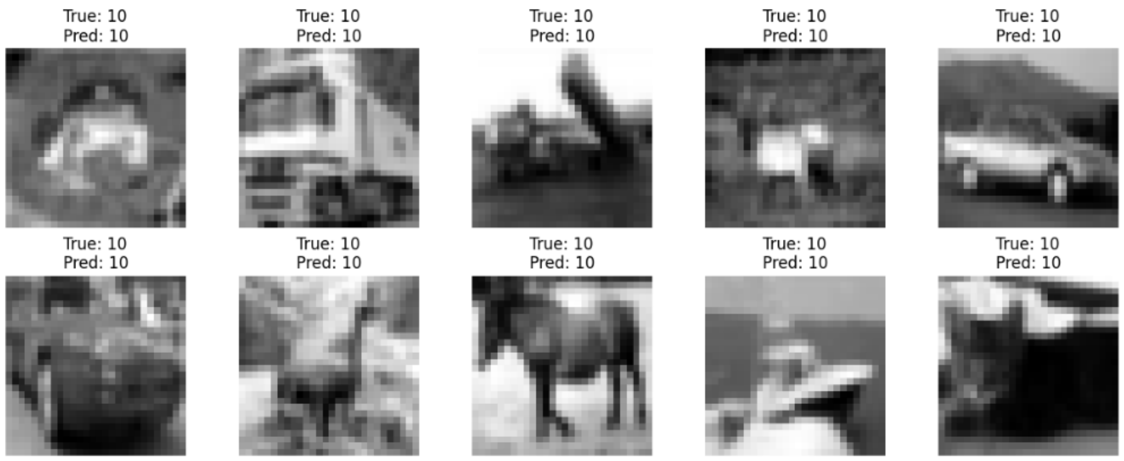
4.2 Average Uncertainty Estimates Across Epochs

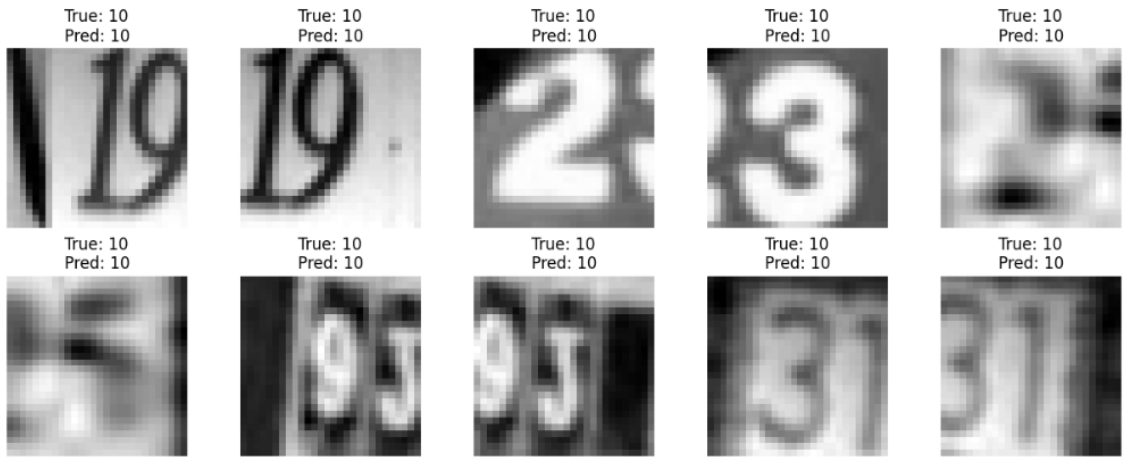
Epoch	Avg Uncertainty
1	0.9783
2	0.9778
3	0.9474
4	0.9719
5	0.9134
6	0.8922
7	0.9389
8	0.9459
9	0.9090
10	0.8736

- The Avg uncertainty is very high initially, indicating high uncertainty in the classifier's predictions. But as the num of epochs increased, the Avg uncertainty decreased as compared to the beginning.This indicates that classifier is becoming more confident over time on the generated samples, indicating improvement in generator conditioning and classifier robustness.

4.3 Predictions on FMNIST, CIFAR10, SVHN:







4.4 Experiment-2

4.4.1 Hyperparameters and Settings

- Inversion Loss Weights: $\alpha = 500$, $\beta = 500$, $\gamma = 6000$, $\delta = 10$
- Latent Dimension: 100, Batch Size: 64
- Number of Classes: 11 (MNIST + Garbage class)
- Generator Steps per Epoch: 250
- Number of samples generated in each step = 1000
- Number of Epochs: 20

4.4.2 Test Accuarcy on OOD Datasets

Dataset	Before Inversion (%)	After Inversion (%)
Fashion-MNIST (FMNIST)	28.81	87.9
CIFAR-10	87.3	99.998
SVHN	87.71	100

- The test accuracy on FMNIST has significantly improved (from 71.89% to 87.9%) when run for 20 epochs as compared to 10 epochs

4.5 Average Uncertainty Estimates Across Epochs

Epoch	Avg Uncertainty
1	0.980230
2	0.962530
3	0.652213
4	0.918750
5	0.983479
6	0.962047
7	0.954969
8	0.944361
9	0.940250
10	0.912971

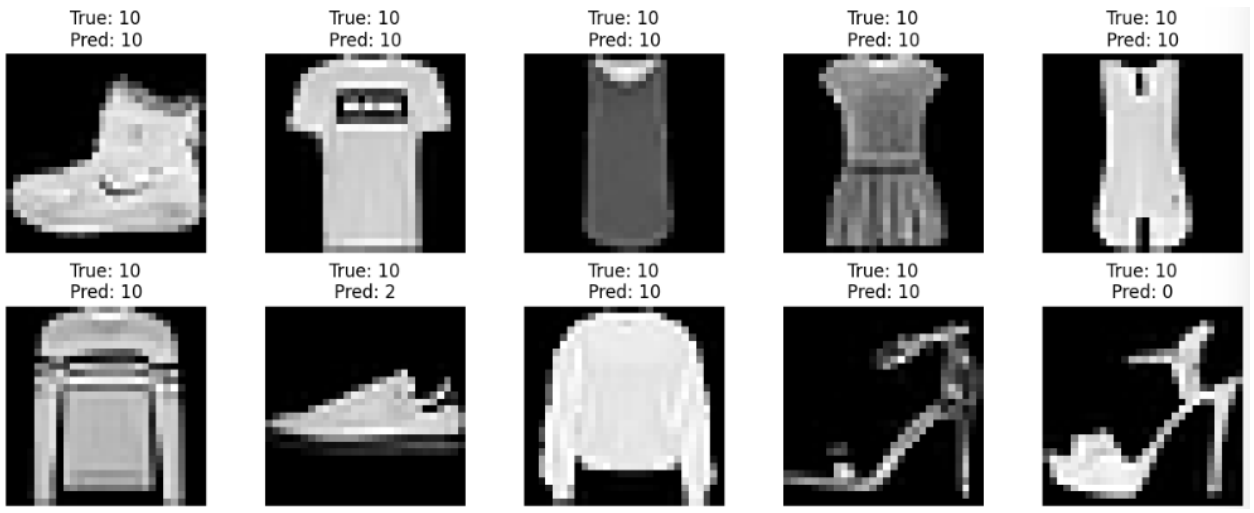
Epoch	Avg Uncertainty
11	0.916835
12	0.869079
13	0.880915
14	0.894120
15	0.936265
16	0.882743
17	0.894302
18	0.910122
19	0.874536
20	0.869838

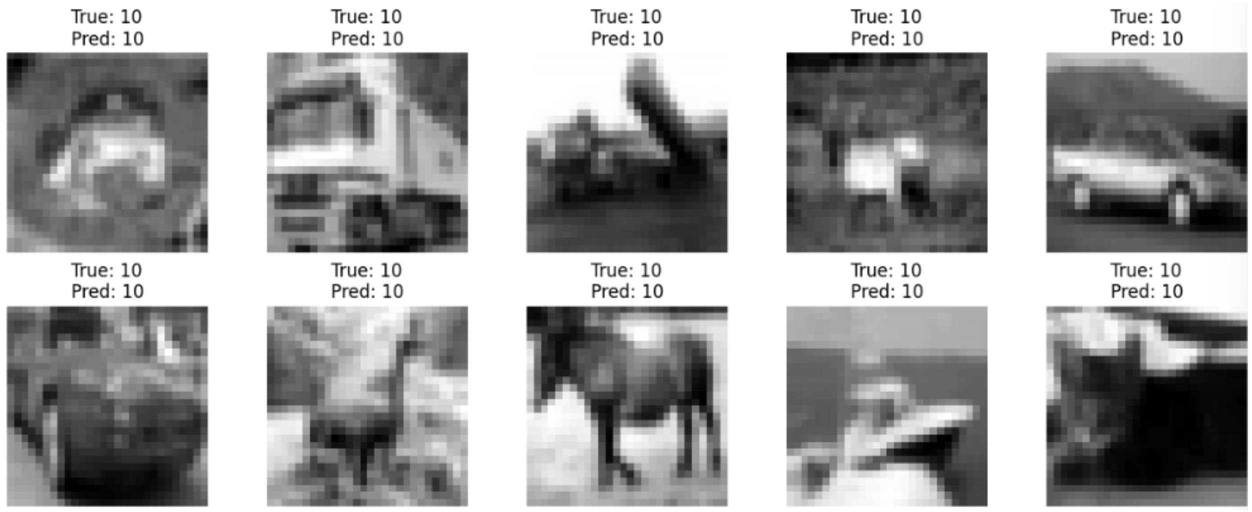
- The Avg uncertainty is very high initially, indicating high uncertainty in the classifier's predictions. The sharp drop at Epoch 3 is likely an anomaly — it may indicate temporary overfitting, mode collapse, or a change in generator behavior. From Epoch 7 onward, uncertainty becomes more stable and generally decreases, which reflects:

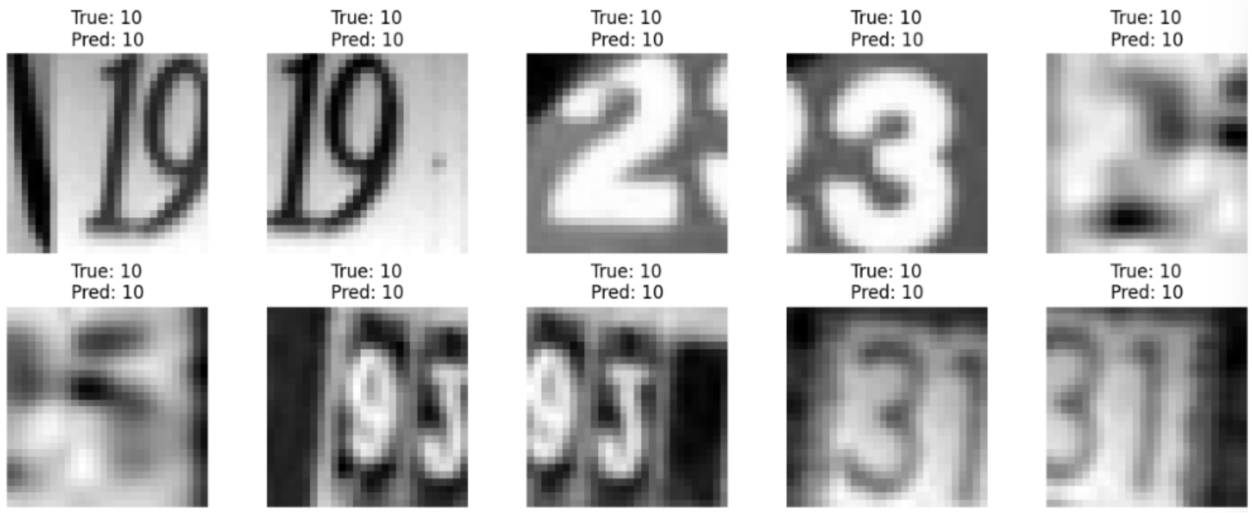
  Better generator conditioning.

  Classifier growing more confident with synthetic(generated) data.
- The gradual decline by Epoch 20 suggests that the model is learning useful structure from inversion-refined samples.

4.6 Predictions on FMNIST, CIFAR10, SVHN:







4.7 Experiment-3

4.7.1 Hyperparameters and Settings

- Inversion Loss Weights: $\alpha = 500$, $\beta = 500$, $\gamma = 6000$, $\delta = 10$
- Latent Dimension: 100, Batch Size: 64
- Number of Classes: 11 (MNIST + Garbage class)
- Generator Steps per Epoch: 250
- Number of samples generated in each step = 1000
- Number of Epochs: 50

4.7.2 Test Accuarcy on OOD Datasets

Dataset	**Before Inversion (%)**	**After Inversion (%)**
Fashion-MNIST (FMNIST)	30.97	73.96
CIFAR-10	98.8	100
SVHN	99.73	100
MNIST	98.66	98.44

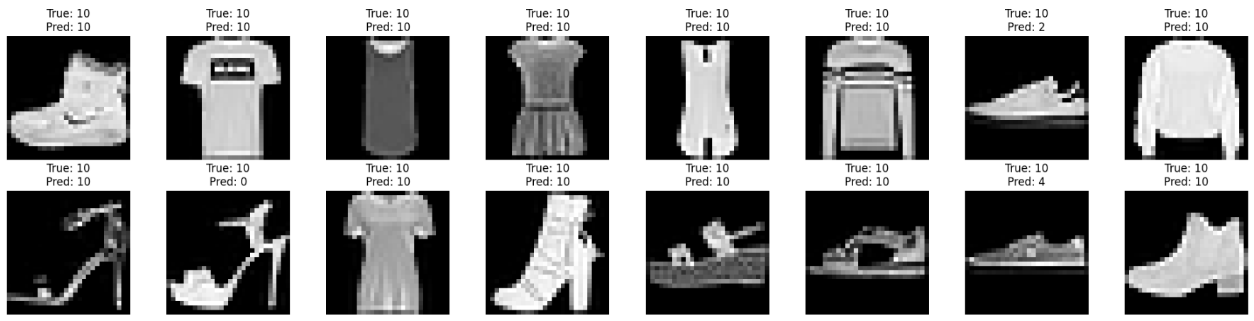
- The test accuracy on FMNIST has significantly improved after training using network inversion
- After training, the test accuracy of the classifier is also checked on the mnist-test dataset.
- This is to make sure if the original task of classifying in-distribution data is still correctly being done by the classifier or not. From the results we can infer that, training didnot effect the classifier's abitlity to classify in-distribution data correctly. It remained almost same and the OOD performance significantly improved

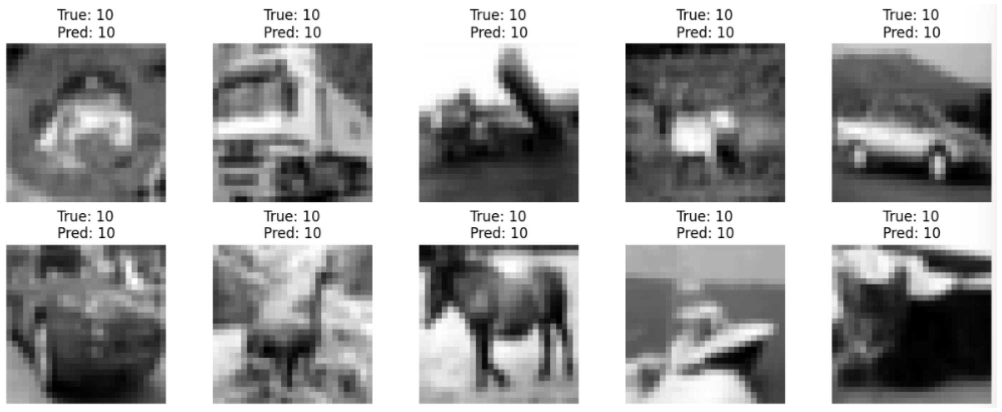
4.8 Average Uncertainty Estimates Across Epochs

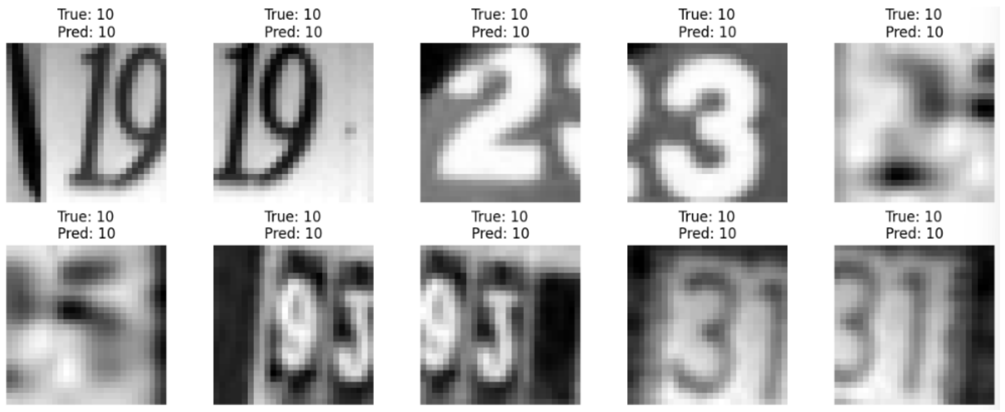
Epoch	Avg UE	Epoch	Avg UE	Epoch	Avg UE	Epoch	Avg UE	Epoch	Avg UE
1	0.855	11	0.000042	21	0.800	31	0.822	41	0.879
2	0.935	12	0.000283	22	0.792	32	0.847	42	0.808
3	0.964	13	0.958	23	0.815	33	0.883	43	0.802
4	0.690	14	0.960	24	0.867	34	0.859	44	0.703
5	0.130	15	0.949	25	0.824	35	0.878	45	0.828
6	0.002	16	0.962	26	0.851	36	0.786	46	0.829
7	0.005	17	0.946	27	0.839	37	0.865	47	0.856
8	0.685	18	0.927	28	0.870	38	0.877	48	0.829
9	0.081	19	0.890	29	0.832	39	0.837	49	0.606
10	0.002	20	0.844	30	0.836	40	0.793	50	0.676

- Epochs 1–3 show high UE (0.855 to 0.964), indicating classifier uncertainty.
- From around Epoch 13 onward, the model maintains high but balanced certainty, reflecting improved generator-conditioning and classifier synergy.
- At Epochs (31–50) UE is stable and moderate (0.79–0.88) for most epochs.
- The system evolves from high uncertainty (early exploration) → overconfident phase → stabilized adaptation.

4.9 Predictions on FMNIST, CIFAR10, SVHN:







4.10 Experiment-4

4.10.1 Hyperparameters and Settings

- α (Weight for the KL Divergence loss)changed form 500 to 1000 to improve the generated samples. β(weight for Cross-entropy loss) changed from 500 to 2000 to improve the generated samples. $\gamma = 6000$ (weight for cosine similarity loss) is not changed δ (weight for Feature orthogonality loss) changed from 10 to 100 to improve diversity in the generated samples
- Inversion Loss Weights: $\alpha = 1000$, $\beta = 2000$, $\gamma = 6000$, $\delta = 100$
- Latent Dimension: 100, Batch Size: 64
- Number of Classes: 11 (MNIST + Garbage class)
- Generator Steps per Epoch: 250
- Number of samples generated in each step = 1000
- Number of Epochs: 30

4.10.2 Test Accuarcy on OOD Datasets

Dataset	Before Inversion (%)	After Inversion (%)
Fashion-MNIST (FMNIST)	25.48	80.54
CIFAR-10	95.72	100
SVHN	96.9	100

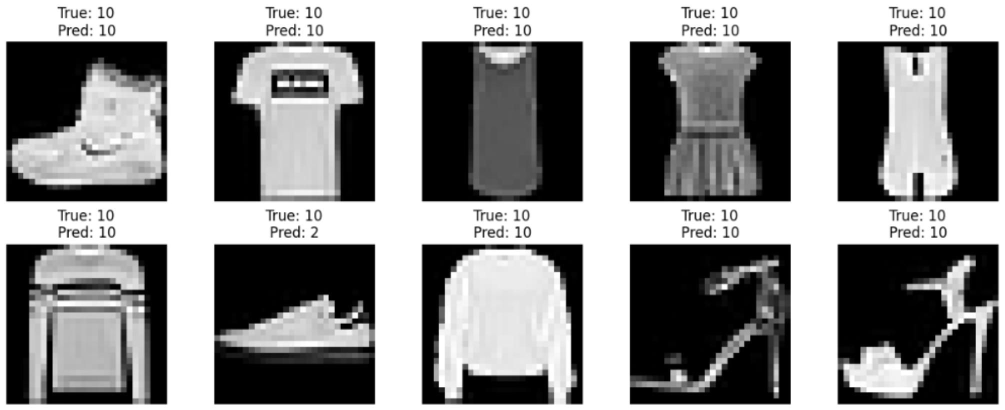
- The test accuracy on FMNIST has significantly improved after training using network inversion
- In this experiment the hyperparameters and no of epochs are changed as compared to the previous experiments
- The test accuracy has improved as compared to experiment-1 and 3 but less than that of experiment-2

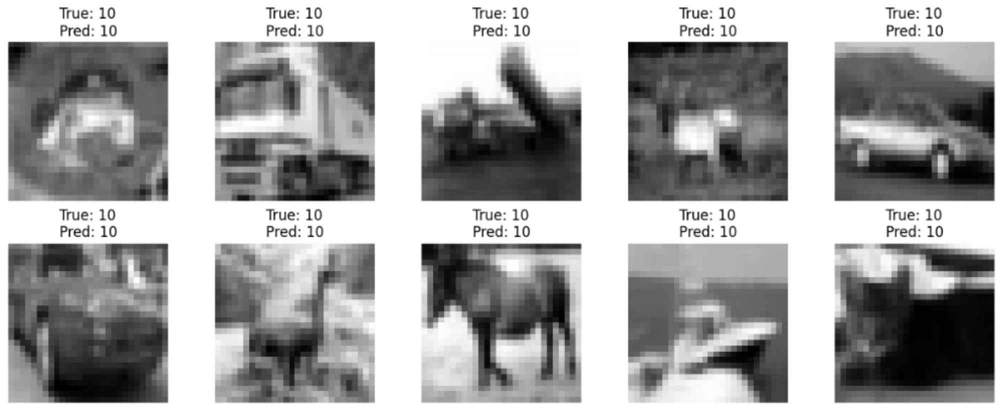
4.11 Average Uncertainty Estimates Across Epochs

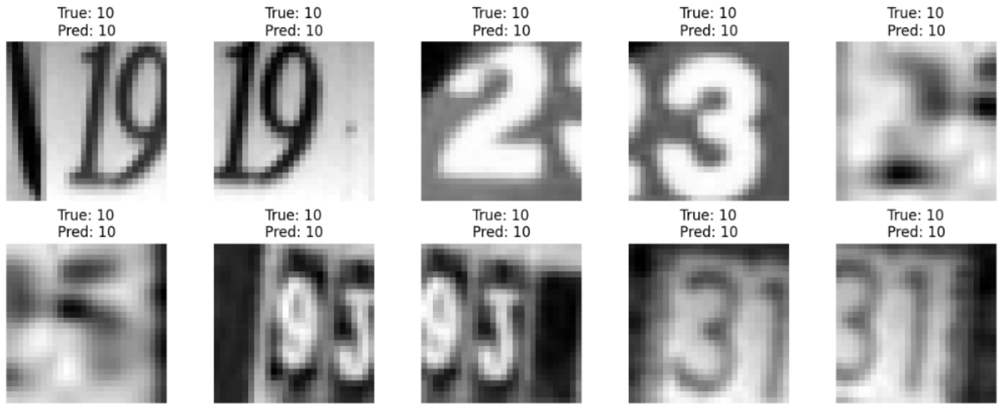
Epoch	Avg UE	Epoch	Avg UE	Epoch	Avg UE
1	0.9797	11	0.8933	21	0.8897
2	0.4496	12	0.9046	22	0.8886
3	0.9593	13	0.9034	23	0.8872
4	0.9410	14	0.8863	24	0.8503
5	0.9677	15	0.9111	25	0.8862
6	0.9521	16	0.8856	26	0.8690
7	0.9242	17	0.8941	27	0.9208
8	0.9330	18	0.8893	28	0.8304
9	0.8938	19	0.9023	29	0.8712
10	0.8741	20	0.8657	30	0.8955

- Epochs 1–3 show high UE (0.855 to 0.964), indicating classifier uncertainty.
- From around Epoch 13 onward, the model maintains high but balanced certainty, reflecting improved generator-conditioning and classifier synergy.
- At Epochs (31–50) UE is stable and moderate (0.79–0.88) for most epochs.
- The system evolves from high uncertainty (early exploration) → overconfident phase → stabilized adaptation.

4.12 Predictions on FMNIST, CIFAR10, SVHN:







5 Analysis and summary

Experiment	Epochs	FMNIST (Post-Inversion)	α, β	γ, δ
Exp-1	10	71.89%	500, 500	6000, 10
Exp-2	20	**87.9%**	500, 500	6000, 10
Exp-3	50	73.96%	500, 500	6000, 10
Exp-4	30	80.54%	1000, 2000	6000, 100

5.1 Impact of Network Inversion on OOD Generalization

- Across all experiments, network inversion consistently improved test accuracy on OOD datasets (Fashion-MNIST, CIFAR-10, SVHN). For instance:
- FMNIST accuracy improved from 71 to 88% (post-inversion)
- CIFAR-10 and SVHN showed near-perfect post-inversion accuracy, indicating strong generalization.
- Notably, in-distribution performance (MNIST) remained unaffected, affirming that inversion-based OOD refinement does not compromise original classification ability.

5.2 Generator Conditioning and Classifier Confidence

- The uncertainty estimates across epochs reveal how both generator and classifier evolve over time:
- **Early Epochs (1–3)**:
  High uncertainty (e.g., UE 0.85–0.98) was observed, signaling initial classifier confusion due to noisy or poorly conditioned samples from the generator.
  In some cases (e.g., Exp-2 Epoch 3), sharp dips in uncertainty likely indicate mode collapse or unstable generator output.
- **Mid Epochs (10–20)**:
  Uncertainty becomes more stable and gradually declines, suggesting:

  Improved generator conditioning, producing more class-consistent samples.

  Classifier adaptation, with increasing confidence on synthetic inputs.

  Experiments 2 and 4 notably show smoother transitions in this phase, correlating with better FMNIST performance.
- **Late Epochs (20–50 in Exp-3, up to 30 in Exp-4):**:
  The system enters a stable phase: UE stabilizes between 0.79–0.88, indicating a healthy balance between exploration (novel inputs) and exploitation (confidence).

  Some fluctuations (e.g., spikes in Exp-4 Epoch 27) reflect natural generator variations but don't degrade performance.

5.3 Influence of Epoch Count and Hyperparameters

- Longer training (Exp-2 with 20 epochs) led to the best FMNIST accuracy (71.89% for 10 epochs to 87.9% for 20 epochs), showing that the model benefits from extended generator refinement.
- But further increasing the number of epochs did not show improvement in the performance(87.9% for 20 epochs to 73.96% for 50 epochs)
- Experiment-4, with adjusted inversion loss weights ($\alpha = 1000$, $\beta = 2000$, $\gamma = 6000$, $\delta = 100$), outperforms Exp-1 and Exp-3 despite running only 30 epochs—highlighting the importance of tuning loss terms for better generator-classifier synergy.

6 Conclusion:

- This work demonstrates the effectiveness of network inversion as a strategy to enhance OOD detection by refining classifier robustness through generative feedback. Across multiple experiments with varying hyperparameters and training epochs, the approach consistently improved test accuracy on diverse OOD datasets such as FMNIST, CIFAR-10, and SVHN.
- Notably, uncertainty estimates revealed a transition from high initial classifier ambiguity to more confident and stable predictions, indicating improved generalization and better alignment between the generator and classifier. Furthermore, the ability to preserve in-distribution accuracy while boosting OOD performance confirms the reliability and adaptability of the method. This highlights the potential of inversion-based training to serve as a general-purpose refinement step in building more robust deep learning systems