

Supplementary Material

for
A Non-Adversarial Approach to Idempotent Generative Modelling

1 NALIGN Torch Pseudocode

```
1 def train_batch(f, f_copy, model_optim, batch, loss_function, M_imle):
2     f.train()
3     #####
4     f_copy.load_state_dict(f.state_dict())
5     x = batch
6     loss = 0
7
8     ##### Reconstruction loss
9     x_rec = x + 0.01*torch.randn_like(x)
10    fx_rec = f(x_rec)
11    L_rec = loss_function(fx_rec, x_rec)
12    loss = loss + L_rec
13
14    ##### IMLE
15    x_imle = x.detach()
16    with torch.no_grad():
17        Nx, C, H, W = x_imle.shape
18        z_imle = torch.randn(M_imle, C, H, W)
19        fz_imle = f_copy(z_imle).detach()
20        # Compute the distance matrix between points1 and points2
21        dist_matrix = torch.cdist(x_imle.view(Nx, -1), fz_imle.view(M_imle, -1))
22        col_ind = torch.argmin(dist_matrix, axis=1)
23        fz_star = f(z_imle[col_ind])
24        L_imle = loss_function(x, fz_star)
25        loss = loss + L_imle
26
27    ##### Modified idempotency loss
28    z_naidem = torch.randn_like(x)
29    fz_naidem = f(z_naidem)
30    f_fz_naidem = f_copy(fz_naidem)
31    L_naidem_1 = loss_function(fz_naidem, f_fz_naidem)
32
33    q_naidem = fz_naidem.detach()
34    fq_naidem = f(q_naidem)
35    f_fq_naidem = f_copy(fq_naidem)
36    L_naidem_2 = loss_function(fq_naidem, f_fq_naidem)
37
38    L_naidem = 0.5 * (L_naidem_1 + L_naidem_2)
39    loss = loss + L_naidem
40
41    ##### Backpropagate
42    loss.backward()
43    model_optim.step()
44    model_optim.zero_grad()
```

2 Modified Idempotent Loss

The modified idempotent loss in NAIGNs ensures that the model behaves idempotently on data manifold points, source distribution points, and off-manifold points.

The original idempotent loss, introduced by (?), enforces idempotency on the source distribution by penalizing discrepancies between the model's output and its second application:

$$\mathcal{L}_{\text{idem}}(\theta) = \mathbb{E}_{\mathbf{z} \sim p_Z} [d(f_\theta(\mathbf{z}), f_{\theta^\perp}(f_\theta(\mathbf{z})))]$$

where \mathbf{z} is a sample from the source distribution p_Z .

We extend this loss by also applying it to points generated by the model during training, capturing off-manifold behavior. The modified loss is the average of the original idempotent loss and a similar term for model-generated points:

$$\mathcal{L}_{\text{idem}}(\theta) = \frac{1}{2} (\mathbb{E}_{\mathbf{z} \sim p_Z} [d(f_\theta(\mathbf{z}), f_{\theta^\perp}(f_\theta(\mathbf{z})))] + \mathbb{E}_{\mathbf{q} \sim p_\theta} [d(f_\theta(\mathbf{q}), f_{\theta^\perp}(f_\theta(\mathbf{q})))])$$

where \mathbf{q} represents samples generated by the model with inputs from the source distribution. This modification ensures the model is trained on both source and off-manifold points, encouraging it to project generated points back onto the manifold. By applying the loss to both types of points, the model becomes explicitly exposed to off-manifold behavior and learns how to act idempotently on such points as well.

3 Experimental Details: Degradation and Restoration

We conducted experiments on the MNIST dataset, resized to 32x32 to fit a smaller DCGAN model architecture. We applied four types of degradation: Gaussian blur, Gaussian noise, salt and pepper noise, and random deletion (setting random rows and columns to zero).

- **Gaussian blur:** We used the Torch transform’s Gaussian blur function. The amount of blur was controlled by the kernel size and a blur level parameter (sigma), where the kernel size was computed based on the blur level, and the sigma value matched the blur level.
- **Gaussian noise:** We used a sigma value of 1.0, implemented via the Torch transform’s Gaussian noise functionality.
- **Random row and column deletions:** Each row and column had a 20% chance of being deleted, with the pixel values set to -1.
- **Salt and pepper noise:** 20% of the pixels were deemed corrupted, and of those, 50% were set to salt (value 1) and 50% to pepper (value -1), as the MNIST dataset was normalized to the range of -1 to 1.

4 Reconstructed, Generated and Restored Sample Images

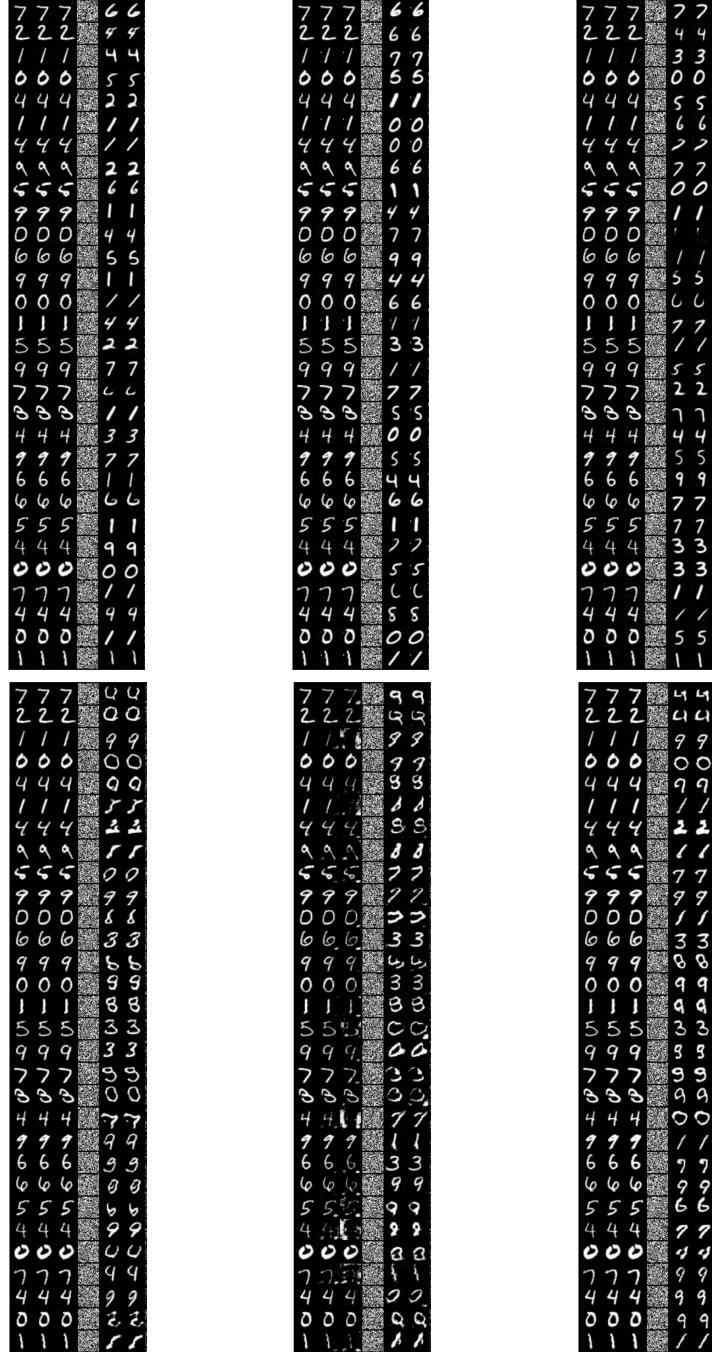
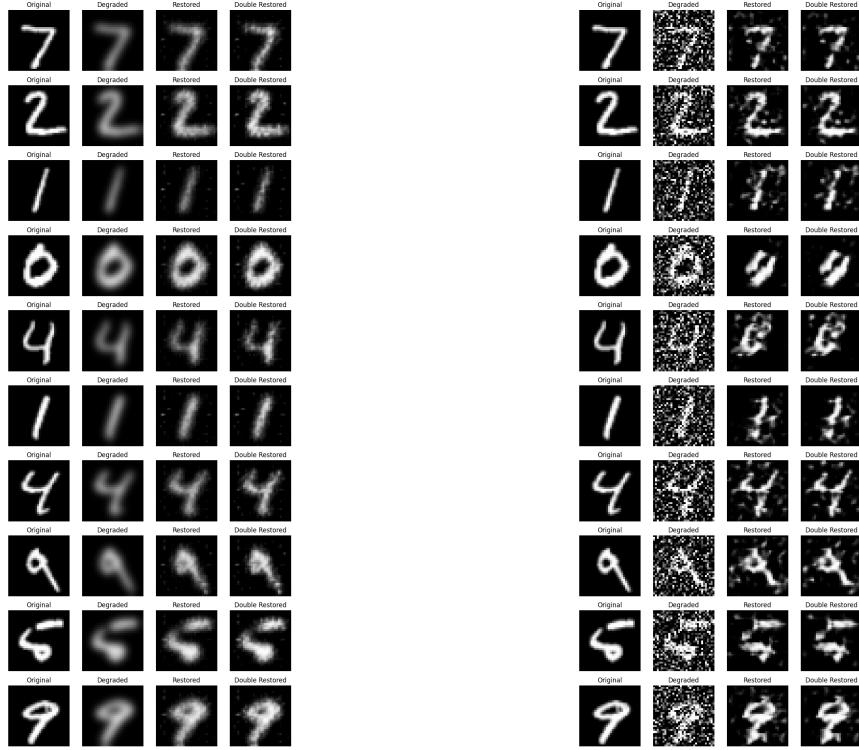
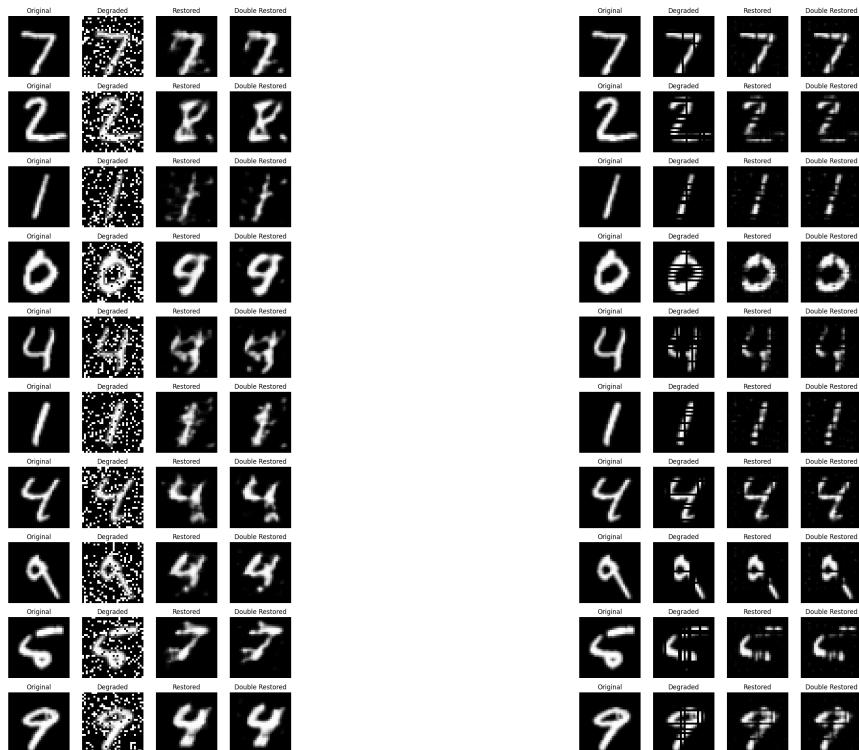


Figure 1: Reconstruction, generation over 3 seeds (one seed per sub-figure) for NAIGN (top) and IGN (bottom). In each sub-figure, the first column depicts images from the test set. Columns 2-3 are the first and second applications of the model on the real images. Columns 3-5 depict noise sampled from the source distribution and the first and second applications of the model, respectively.

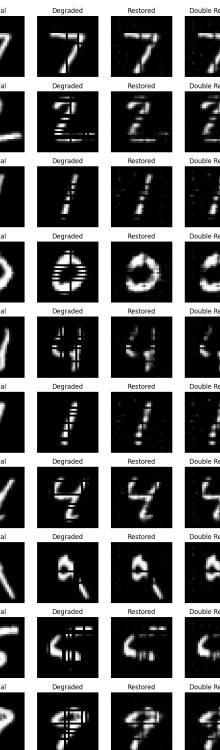


(a) Blur



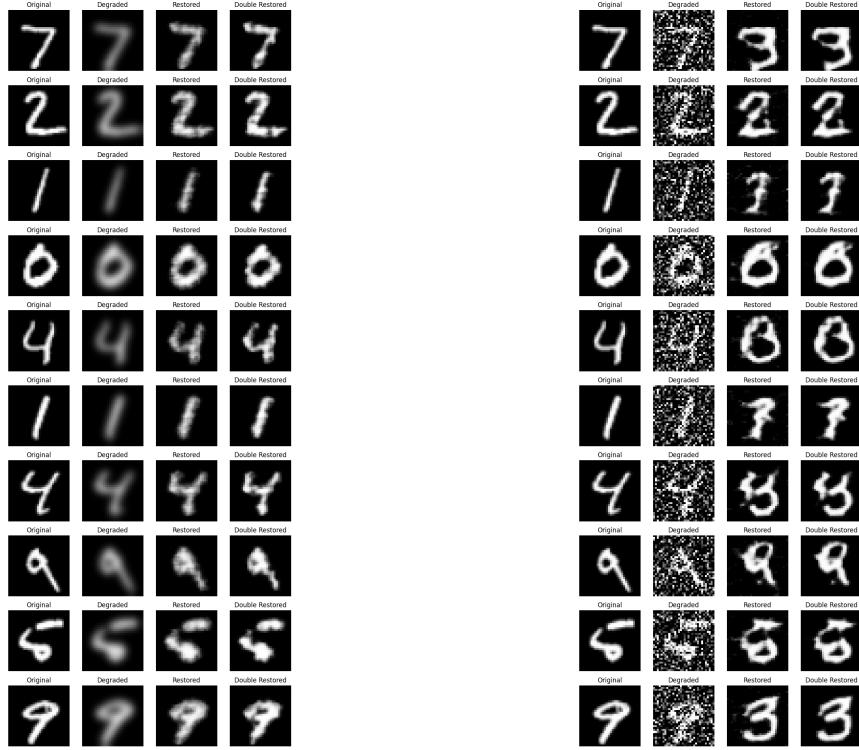
(c) Salt&Pepper

(b) Gaussian Noise

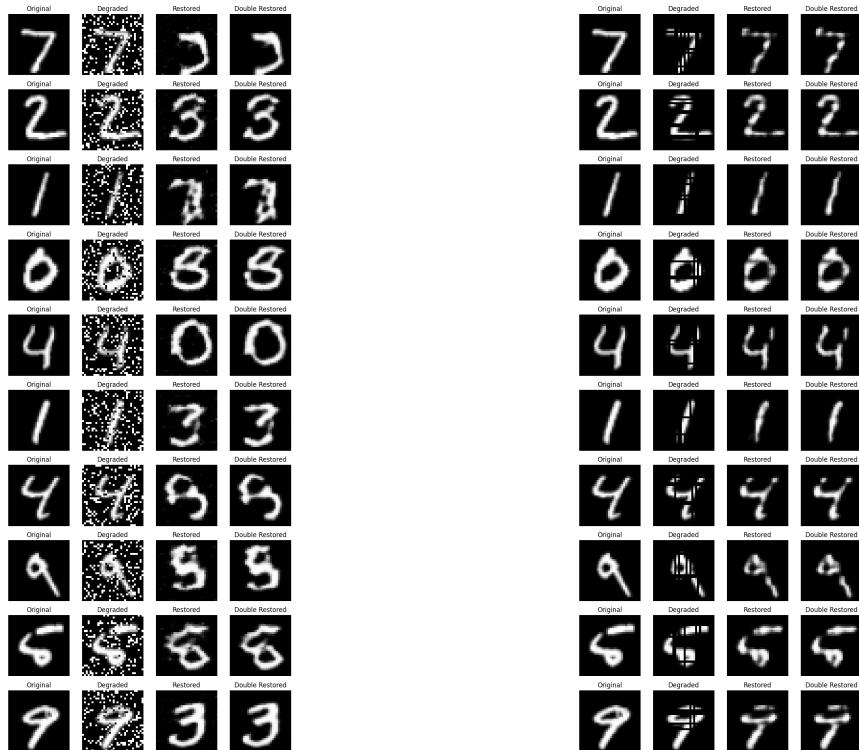


(d) LinesRows

Figure 2: Restoration for NAIGN.

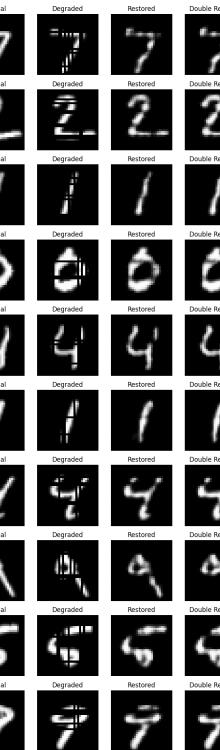


(a) Blur



(c) Salt&Pepper

(b) Gaussian Noise



(d) LinesRows

Figure 3: Restoration for IGN.