XAI: Cluster Variational Inference Notes

Jack Li

October 2023

Abstract

...

1 Workflow

Variational Inference model explanation
The purpose of this report is to introduce the Variational Inference Cluster algorithm and experiment with it.

Situations:
1. ML model interpretability is attracting more attention in order to improve model performance and robustness in industrial applications.
2. Briefly outline the importance of machine learning insterpretability in the medical field, financial field, and autonomous driving. Explainable AI and adversarial attacks highlight the importance of understanding models.
3. Explainable AI (XAI) aims to understand the model's decision-making process.
4. Adversarial attacks exploit key features to influence the output.

Problem:
1. Applying the model in the wrong field.
2. The model outputs something incorrect, and people are not able to understand why.
3. Users blindly trust the model.
4. Do XAI and adversarial attack techniques share the same results?

Solution:
1. Examine and compare state-of-the-art techniques.
2. Use Variational Inference for model explanation.

Evaluation:
Show the comparison of results.

2 Introduction

Machine learning encompasses a variety of tasks, including classification, segmentation, and modern techniques such as generative models. One fundamental aspect of machine learning is classification, which involves categorizing data into labeled and unlabeled datasets. This process includes techniques from both supervised and unsupervised learning. A key feature

of unsupervised learning is clustering, an algorithm designed to group data into distinct clusters based on inherent similarities. Clustering is an unsupervised learning technique aimed at partitioning data into different clusters. One of the significant challenges in this field is the difficulty in understanding how models work analytically, which can lead to unpredictable results in critical applications such as medical diagnosis, financial fraud detection, and autonomous driving. Therefore, understanding model interpretability for key features is crucial. This is where the concept of explainable AI (XAI) becomes relevant. XAI is a subfield of artificial intelligence focused on making AI models more transparent and understandable to humans.

Another important area of research is adversarial attacks, which further underscores the importance of XAI and the need to comprehend model behavior. While XAI aims to elucidate the model's decision-making process, adversarial attacks seek to exploit the model's vulnerabilities.[1] [2] The robustness of models is crucial for their real-world applications, and understanding the dynamics between XAI and adversarial attacks is essential for improving model interpretability and reliability. Despite their differences, both fields highlight the necessity of understanding AI models.

In this paper, we explore state-of-the-art techniques in explainable AI (XAI) and adversarial attacks, focusing on how they interpret features in images differently. Through our exploration, we found that images in high-dimensional spaces significantly contribute to model complexity. This complexity aligns with the challenges posed by both XAI and adversarial attacks. Specifically, while XAI aims to elucidate the decision-making process of complex models, adversarial attacks exploit the vulnerabilities inherent in these high-dimensional representations. Understanding these dynamics is crucial for improving model robustness and interpretability. As we advocate for the idea of XAI, we introduce the Variational Inference Clustering algorithm and experiment with it. This algorithm is a popular XAI method used to interpret features in images.

3   Related Work

. . .
In this paper, we explore state-of-the-art XAI and adversarial attack techniques and how they interpret features in images differently. From the foundations of machine learning and deep learning, we know that improved predictive accuracy often comes with increased model complexity. This complexity can decrease the in-sample error (E_{in}) but may increase the out-of-sample error (E_{out}) [3, 4, 5]. To better understand models, it is essential to interfere with their decision-making processes.
The GradCAM[6], algorithm is a popular XAI algorithm.

4   Experiments

. . . VI Structures:

The variational inference algorithm is a popular XAI method used to interpret features in images. The clusters example[7]

Formulation:

$$z_j \sim \mathcal{N}(m^2, s^2) \text{ for } j = 1, ..., K$$

$$c_i \sim \mathcal{U}(K) \text{ for } i = 1, ..., N \tag{1}$$

$$x_i \sim \mathcal{N}(c_i^T \mu, 1) \text{ for} i = 1, ..., N$$

$$y \sim CNN$$

ELBO function:

$$ELBO = E_q[log\ p(x, \hat{z})] - E_q[log\ q(\hat{z})] \tag{2}$$
$$= E_q[log\ p(x, z, c, y) - log\ q(z, c, y)] \tag{3}$$

Encoder $q(.)$:

$$log\ q(z, c, y) = log\ [q(z|m, s^2)\ q(c|\phi)\ q(y|z, c)]$$

$$= log\ \left[\prod_j q(z_j|m_j, s_j^2)\ \prod_i q(c_i|\phi_i)\ q(y|z, c) \right] \tag{4}$$

$$= \sum_j log\ q(z_j|m_j, s_j^2) + \sum_j log\ \phi_j + log\ pred_y$$

$$\propto \sum_j \frac{-1}{2} \left(log\ s^2 + \frac{(z_j - m_j)^2}{s_j^2} \right) + \sum_j log\ \phi_j + log\ pred_y$$

Decoder $p(.)$:

$$p(c_i) = \frac{1}{K}$$

$$p(y) = N(y|0, 1)$$

$$log\ p(x_i \mid c_i, z_j, y) = log\ \prod_j p(x_i \mid z_j,\ y)^{c_{ij}} \prod_j\ p(c, z|y)\ p(y)$$

$$\propto \sum_j c_j log\ p(x_i \mid z_j)\ \sum_j c_j \log\left(-\frac{1}{2\pi}e^{-\frac{1}{2}\left(\frac{z_j-\mu_y}{1}\right)^2}\right)$$

$$\propto \sum_j c_j log\ p(x_i \mid z_j) - \frac{1}{2}\ \sum_j c_j \left(\frac{z_j-\mu_y}{1}\right)^2$$

$$\propto -\frac{1}{2}(\sum_j c_j \left(\frac{x_i-m_j}{1}\right)^2 + \ \sum_j c_j \left(\frac{z_j-\mu_y}{1}\right)^2)$$

$$ELBO = E_q[log\ p(x, z, c, y) - log\ q(z, c, y)]$$

$$= E_q[-\frac{1}{2}(\sum_j c_j\ (x_i - m_j)^2 + \ \sum_j c_j\ (z_j - \mu_y)^2) + \frac{1}{2}\left(\sum_j \log\ s^2\right) - \sum_j \log\ \phi_j - \log\ \text{pred}_y]$$

4.1 Full Connected Variational Inference
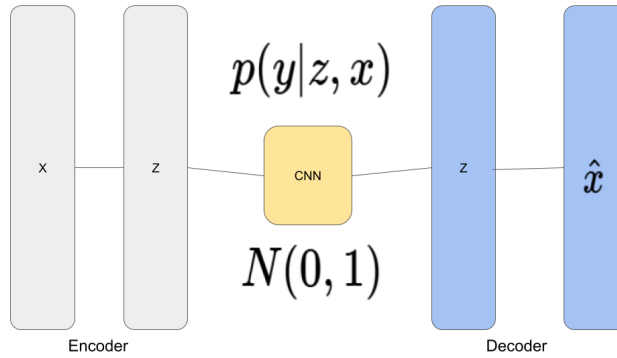
Figure 1: Using 784 $z = \{\mu, s\}$

Code 1: Full connected VI loss function

```
def loss_elbo(x, mu, log_var, phi, x_recon, model, predicted):
    # HACK: use the CNN model predition as the input
    t1 = 0.5 * (mu.view(1, -1) - mu_y.view(-1, 1)) ** 2
    t1 = phi * t1
    t1 = -t1.mean()

    # NOTE: Alternative implementation
```

```
8     t2 = 0.5 * (x - x_recon) ** 2
9     t2 = phi * t2
10    t2 = torch.mean(t2)
11
12    t3 = 0.5 * log_var.exp().mean()
13
14    t4 = -torch.log(phi + 1e-10).mean()
15
16    # HACK: use the CNN model predition as the input
17    model.eval()
18    input = x_recon.view(1, 1, 28, 28)
19    # Forward pass
20    outputs = model(input)
21    outputs = F.softmax(outputs, dim=1)
22    outputs = torch.clamp(outputs, 1e-10, 1 - 1e-10)
23    t5 = -torch.log(outputs[:, predicted])
24    return -(t1 + t2 + t3 + t4 + t5) + lamb
```
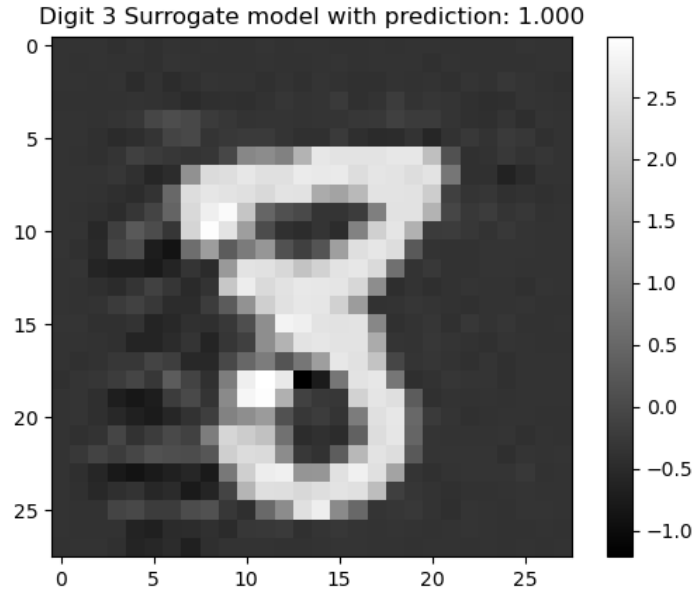

Figure 2: MNIST Prediction with 784 $z = \{\mu, s\}$

4.2   Conv2d Variational Inference
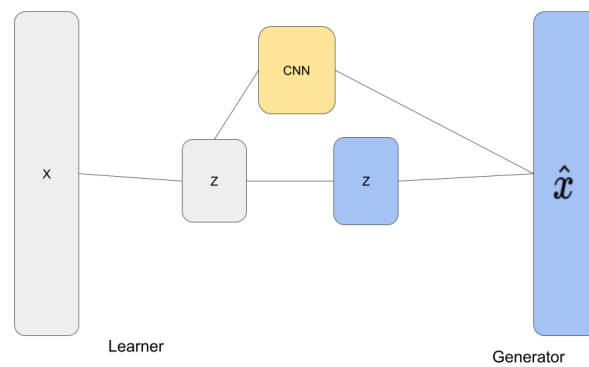
Figure 3: Using GAN and VAE method with conv2d
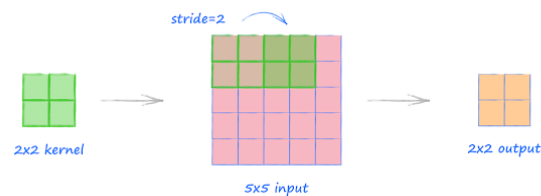
Figure 4: use conv2d kernel size 2x2 and stride 2

Code 2: GAN and VAE model

```python
class generator(nn.Module):
  def __init__(self, channels_img):
      super().__init__()
      self.channels_img = channels_img
      # self.features_d = features_d
      self.k = 2
      # decoder
      self.decoder = nn.Sequential(
          # NOTE: convTranspose2d output = (input -1)*s -2p + k + op
          # (14-1)*2 + 2 = img 28x28
          nn.ConvTranspose2d(
              channels_img, channels_img, kernel_size=2, stride=2,
                  padding=0
          ),
          nn.Tanh(),
      )

      self.p_layer = nn.Sequential(
          # NOTE: convTranspose2d output = (input -1)*s -2p + k + op
          # (14-1)*2 + 2 = img 28x28
          nn.ConvTranspose2d(
              channels_img, channels_img * self.k, kernel_size=2,
                  stride=2, padding=0
          ),
          nn.Tanh(),
```

```python
24            )
25
26        def forward(self, x, z):
27            p = self.p_layer(z)
28            p = F.softmax(p, dim=0)
29            # x_recon = self.decoder(z) * p[:, 0, :, :] + x * (p[:, 1, :,
                  :])
30            x_recon = self.decoder(z)
31            return x_recon, p
32
33
34  class learner(nn.Module):
35        def __init__(self, channels_img):
36            super().__init__()
37            self.channels_img = channels_img
38            # self.features_d = features_d
39            self.k = 2
40            # encoder
41            # latent mean and variance
42            self.mean_layer = nn.Sequential(
43                # NOTE: conv2d output = (input + 2p -k)/s +1
44                # (28-2)/2 +1 = img 14x14
45                nn.Conv2d(channels_img, channels_img, kernel_size=2,
                      stride=2),
46                nn.InstanceNorm2d(channels_img, affine=True),
47                nn.LeakyReLU(0.2),
48            )  # latent mean and variance
49            self.logvar_layer = nn.Sequential(
50                # NOTE: conv2d output = (input + 2p -k)/s +1
51                # (28-2)/2 +1 = img 14x14
52                nn.Conv2d(channels_img, channels_img, kernel_size=2,
                      stride=2),
53                nn.InstanceNorm2d(channels_img, affine=True),
54                nn.LeakyReLU(0.2),
55            )
56
57        def reparameterization(self, mean, var):
58            # mean = mean.view(mean.size[0], -1)
59            # var = var.view(var.size[0], -1)
60            epsilon = torch.randn_like(var).to(device)
61            z = mean + var * epsilon
62            return z
63
64        def encode(self, x):
65            mean, logvar = self.mean_layer(x), self.logvar_layer(x)
66            return mean, logvar
67
68        def forward(self, x):
69            mean, log_var = self.encode(x)
70            z = self.reparameterization(mean, log_var)
71            return z, mean, log_var
```

Code 3: GAN and VAE loss function

```python
epochs = 1000
leaner_epochs = 10
predicted = true_y
# predicted = 9
G = generator(1).to(device)
L = learner(1).to(device)


opt_G = torch.optim.Adam(G.parameters(), lr=0.005)
opt_L = torch.optim.Adam(L.parameters(), lr=0.005)


def loss_function(x, x_recon, mean, log_var, p):
    alpha = torch.sum(p[:, 0, :, :])
    # reproduction_loss = F.mse_loss(x_recon * p[:, 0, :, :], x * p[:,
        0, :, :])
    reproduction_loss = F.mse_loss(x_recon, x)
    KLD = -0.5 * torch.sum(1 + log_var - mean.pow(2) - log_var.exp())

    return reproduction_loss + KLD


# %%

for epoch in range(epochs + 1):
    for leaner_epoch in range(leaner_epochs + 1):
        opt_L.zero_grad()
        x = img.clone().to(device)
        x = x.view(1, 1, 28, 28)
        z, mean, log_var = L(x)
        x_recon, p = G(x, z)
        # Get the index of the max log-probability
        model.eval()
        critic_fake = F.softmax(model(x_recon), dim=1)[0][predicted]

        loss_L = -(torch.mean(critic_fake))
        # loss = loss_function(x, x_recon, mean, log_var) - torch.log(
            critic_real + 1e-5) * (
        #       -torch.log(critic_fake + 1e-5)
        # )

        loss_L.backward(retain_graph=True)
        opt_L.step()

    opt_G.zero_grad()
    x = img.clone().to(device)
    x = x.view(1, 1, 28, 28)
    z, mean, log_var = L(x)
    x_recon, p = G(x, z)

    model.eval()
    critic_fake = F.softmax(model(x_recon), dim=1)[0][predicted]
    t1 = -torch.sum(torch.log(critic_fake + 1e-5))
    t2 = loss_function(x, x_recon, mean, log_var, p)
```

```
52
53     # NOTE: original loss function
54     loss_G = t1 + t2
55     # NOTE: alternative loss function, from GAN
56     # loss_G = torch.mean(critic_fake)
57
58     loss_G.backward()
59     opt_G.step()
```

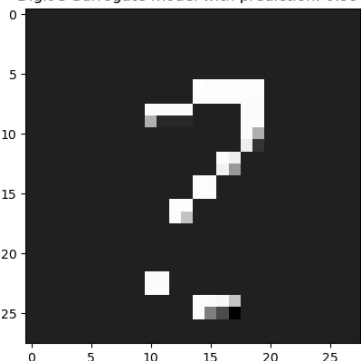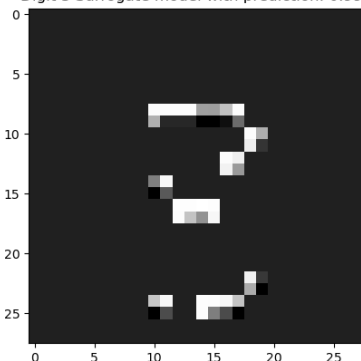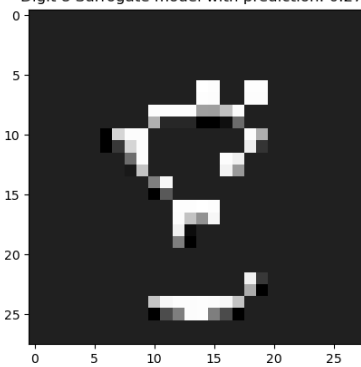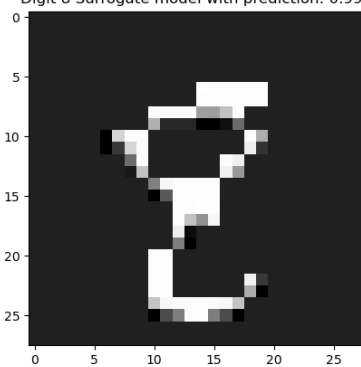Table 1: Experiments Comparison table with top n-th variance(v3)
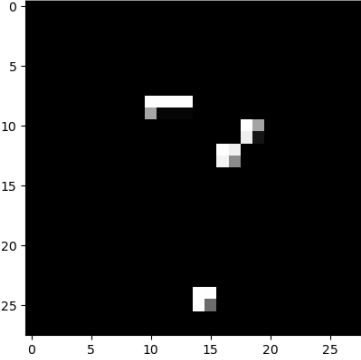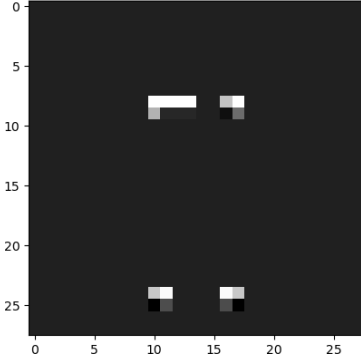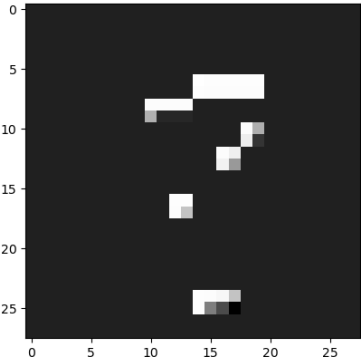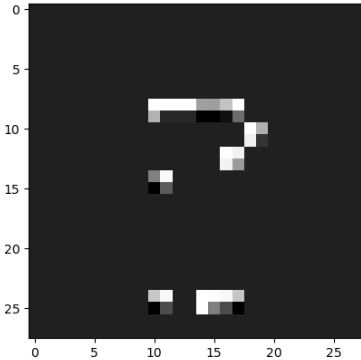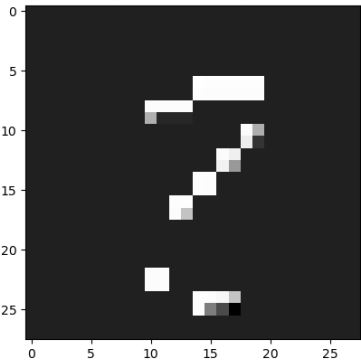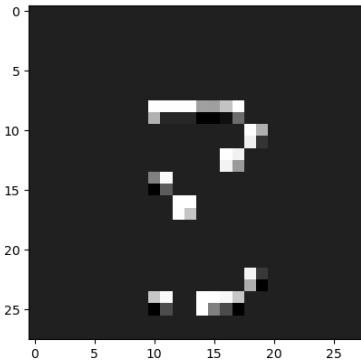
n = 5   Digit 3 Surrogate model with prediction: 0.666

n = 5   Digit 3 Surrogate model with prediction: 0.646

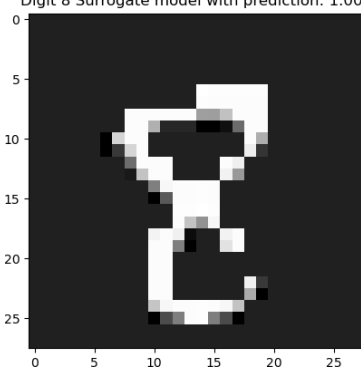n = 10   Digit 3 Surrogate model with prediction: 0.866

n = 10   Digit 3 Surrogate model with prediction: 0.981

n = 12   Digit 3 Surrogate model with prediction: 0.953

n = 12   Digit 3 Surrogate model with prediction: 0.997

n = 13   Digit 3 Surrogate model with prediction: 0.994

n = 13   Digit 3 Surrogate model with prediction: 0.995

Table 2: Experiments Comparison table with top n-th variance(v3)

n = 15
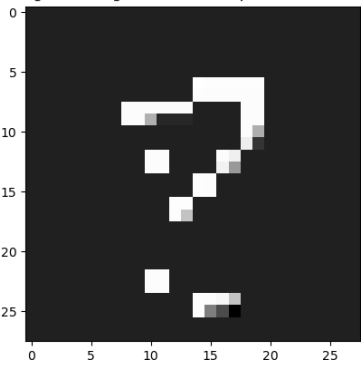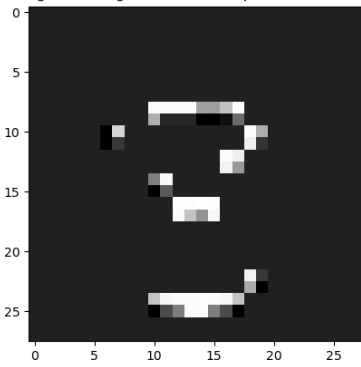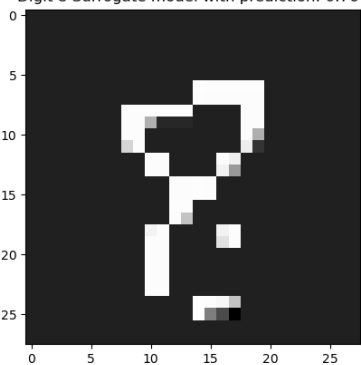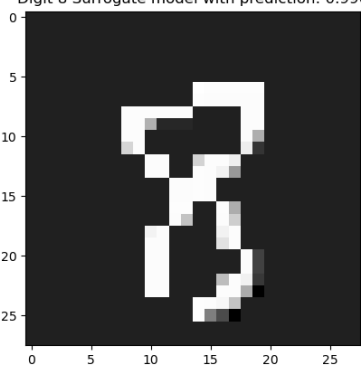
Digit 3 Surrogate model with prediction: 0.998

n = 15

Digit 3 Surrogate model with prediction: 0.996

n = 20

Digit 8 Surrogate model with prediction: 0.764

n = 20

Digit 8 Surrogate model with prediction: 0.278

n = 25

Digit 8 Surrogate model with prediction: 0.990

n = 25

Digit 8 Surrogate model with prediction: 0.998

n = 30

Digit 8 Surrogate model with prediction: 1.000

n = 30

Digit 8 Surrogate model with prediction: 1.000

Table 3: Train one class Experiments Comparison table with mask $C < 0.1$ (v4)
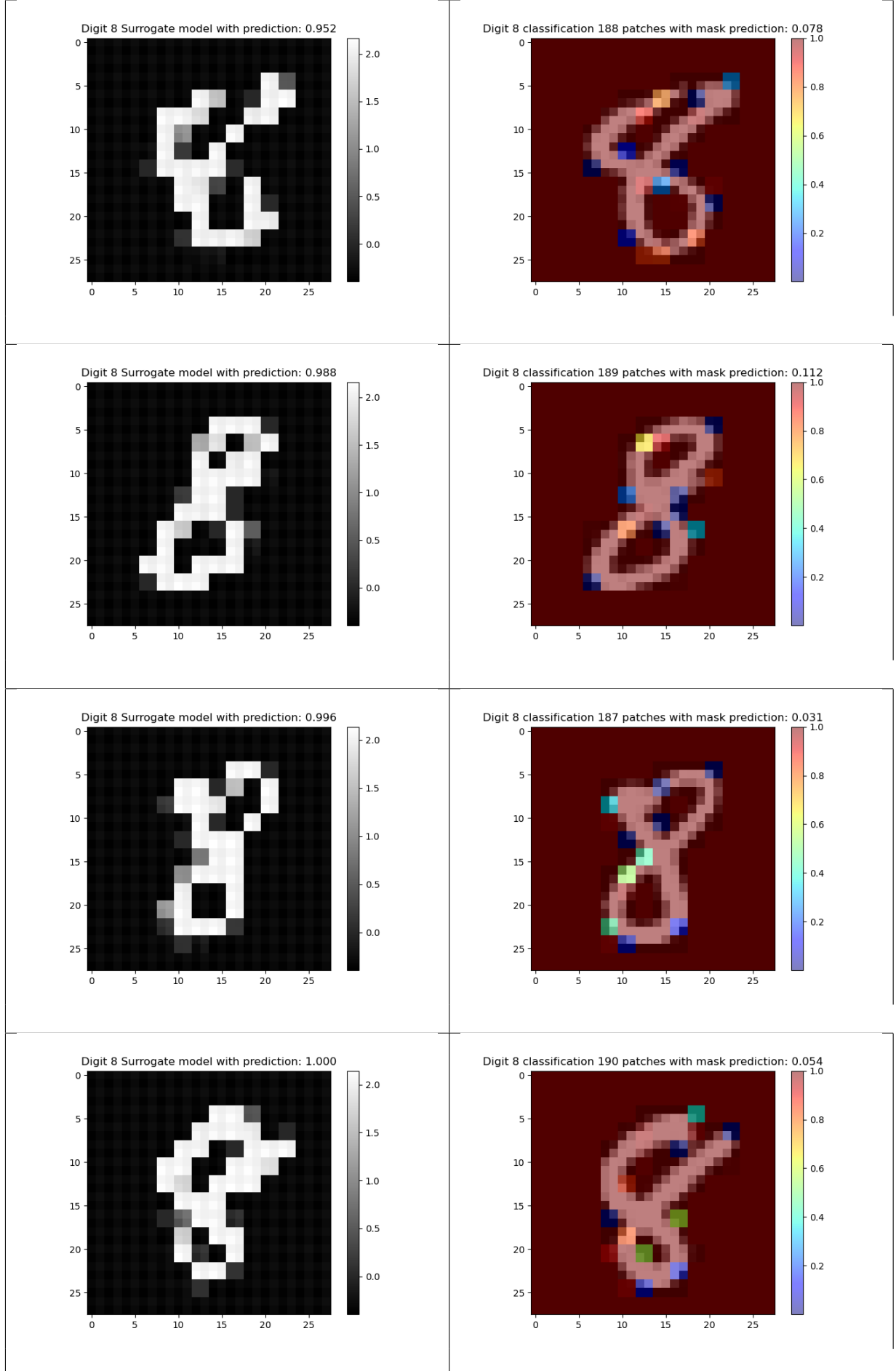
5   Experiment Results 2.0

Formulation (Equation ideal Similar to [7]) Structure (Concept similar to [8]):

$$z_j \sim \mathcal{N}(\mu, \sigma^2) \text{ for } j = 1, ..., K$$

$$c_i \sim \phi(K) \text{ for } i = 1, ..., N$$

$$x_i \sim \mathcal{N}(c_i^T z, 1) \text{ for } i = 1, ..., N \tag{5}$$

$$y \sim CNN$$
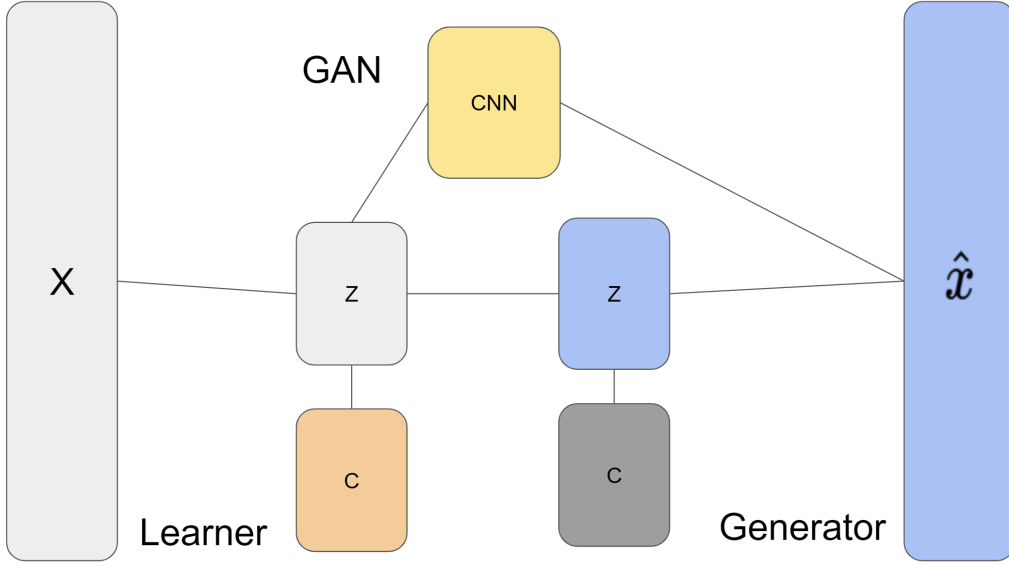
Figure 5: Using GAN and VAE method with conv2d

The goal is to minimize the loss function:

$$\underbrace{arg\ max}_{ELBO} \quad \underbrace{arg\ max}_{z} \quad \mathbb{E}_{z \sim p_z(z)}[L(z)] + ELBO$$

$$ELBO = E_q[log\ p(x, z, c, y) - log\ q(z, c, y)]$$

$$= E_q[-\frac{1}{2}(\sum_j c_j\,(x_i - m_j)^2 + \sum_j c_j\,(z_j - \mu_y)^2) + \frac{1}{2}\left(\sum_j \log\ s^2\right) - \sum_j \log\ \phi_j - \log\ \text{pred}_y]$$

In my experiments, $k = 10$, Some issues with the model:

1. Mode collapse: The model is not able to generate diverse samples.
Remedy: add regularization term to the model[9].

$$\lambda \int q(z)\|G(z) - G^o(z)\|^2 dz$$
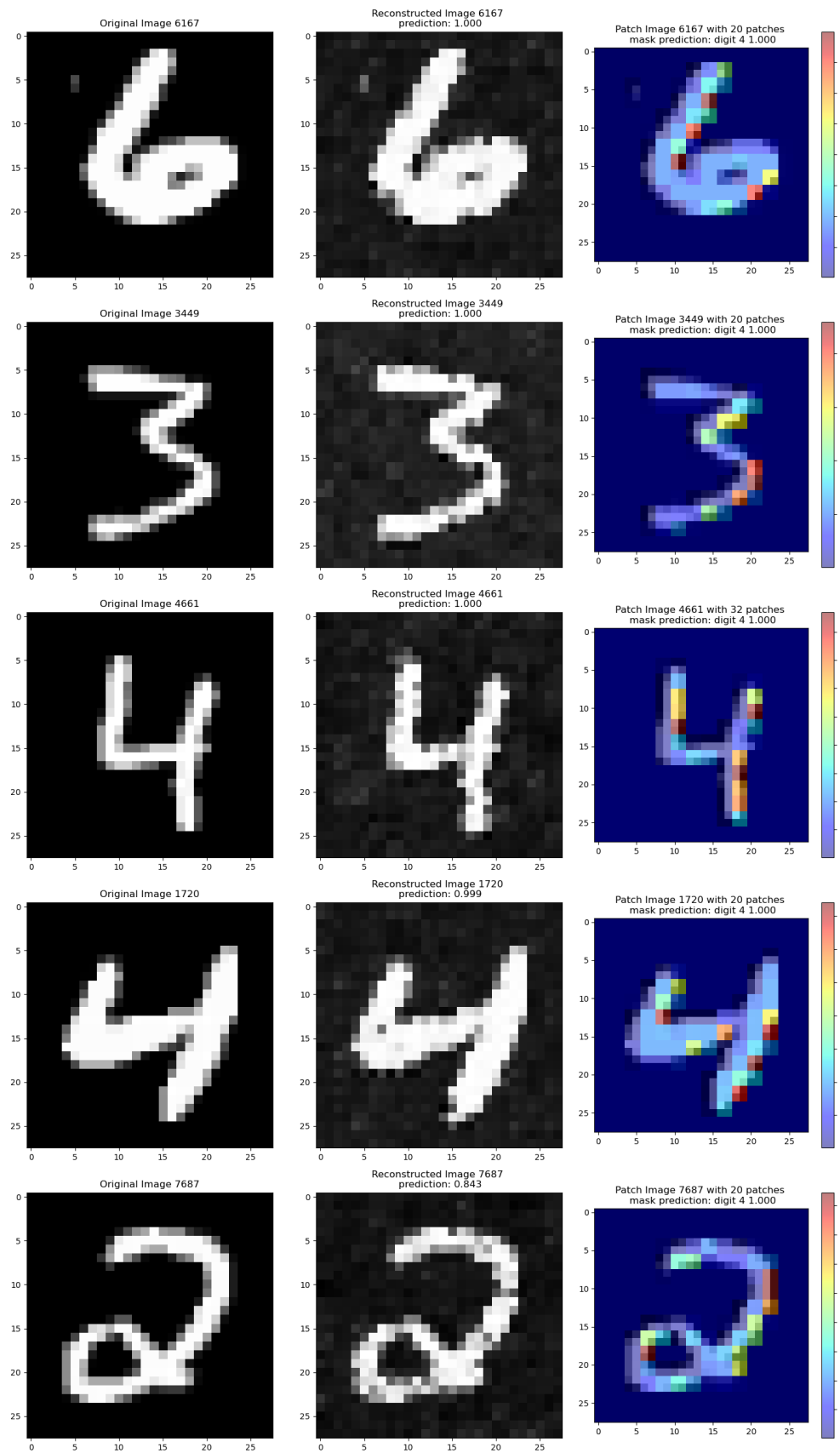
Where $G^o(z)$ is the previous data distribution.
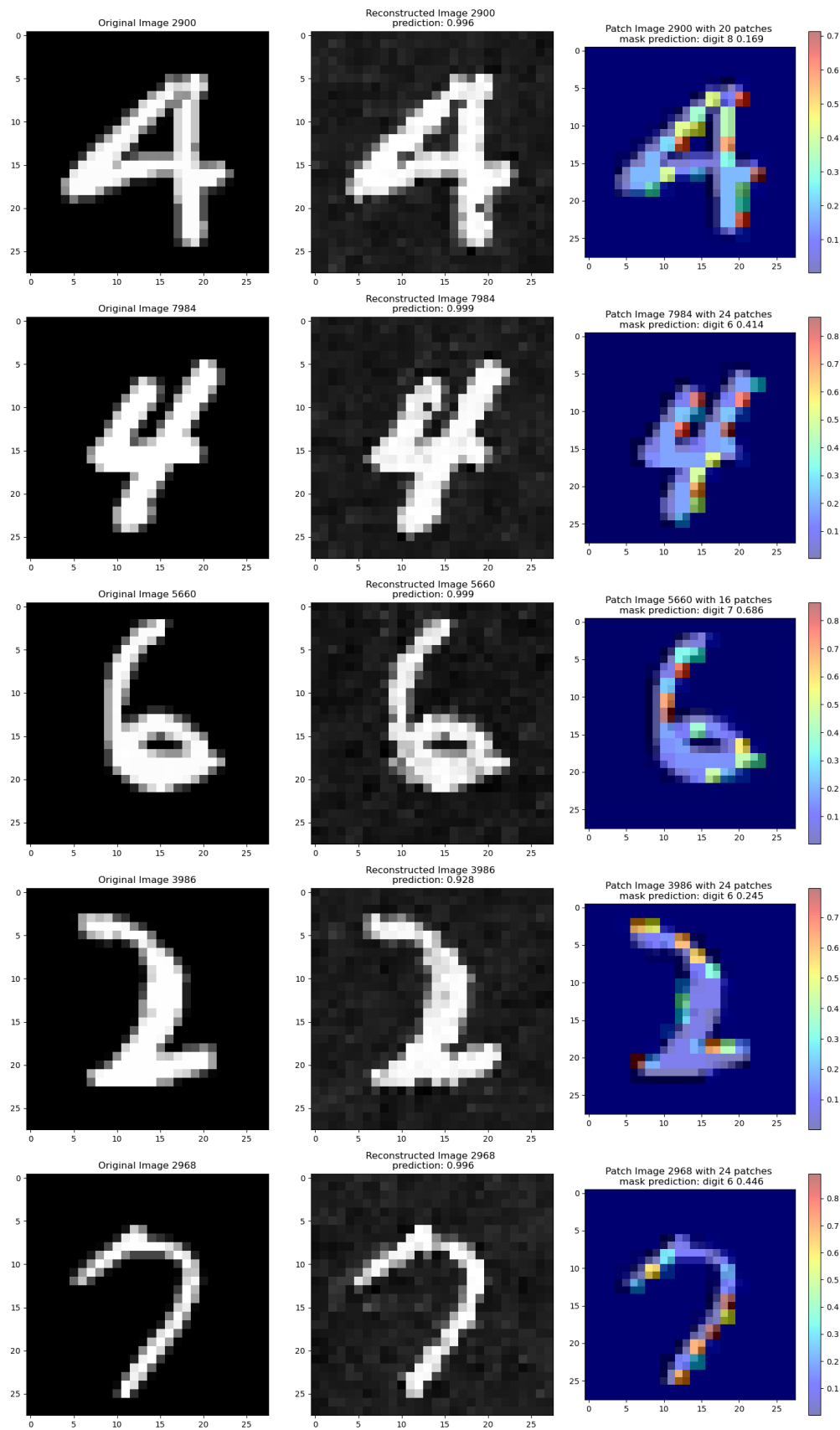
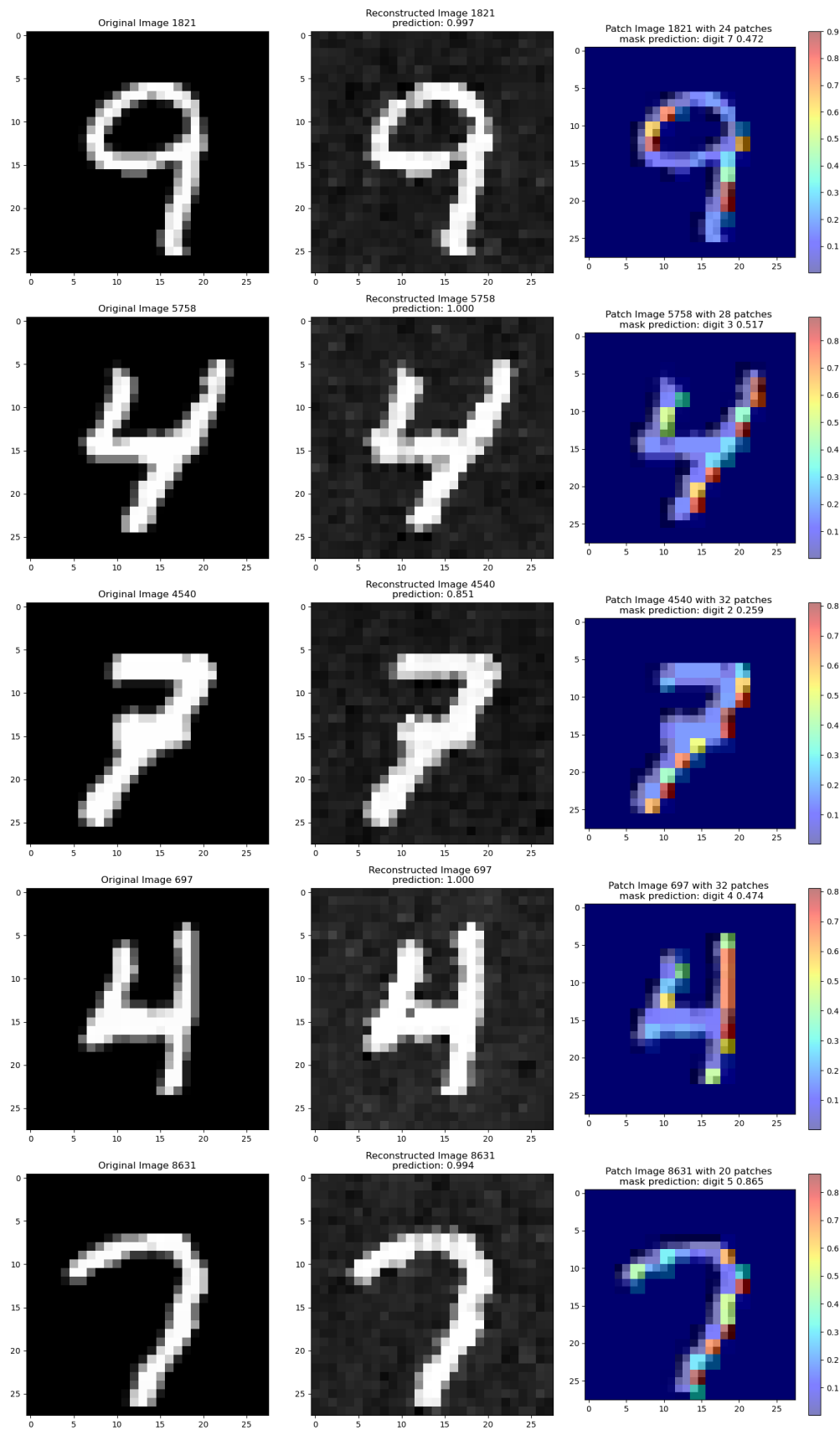Figure 6: Conv2d output: mode collapse

Figure 7: Conv2d output 1
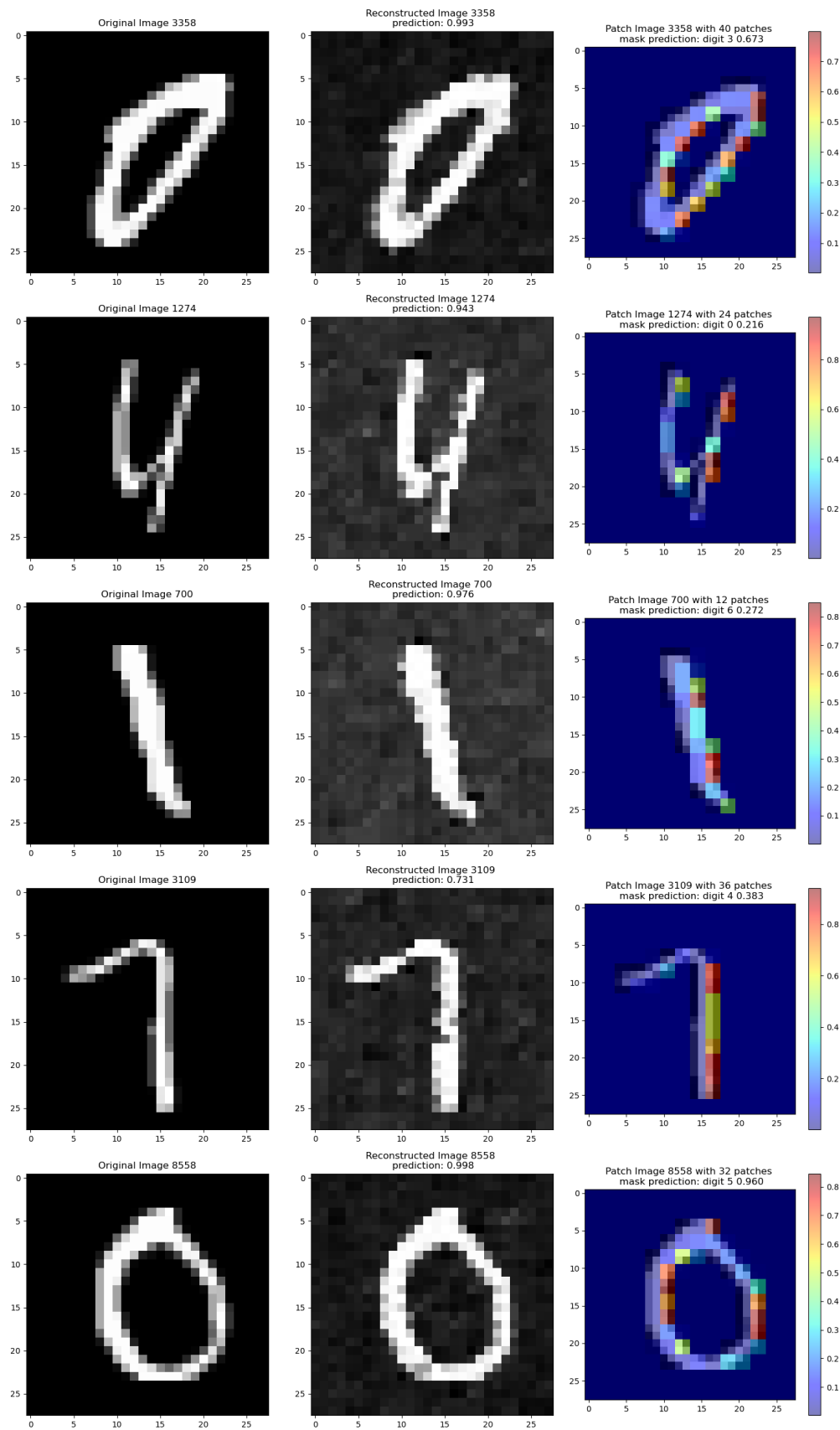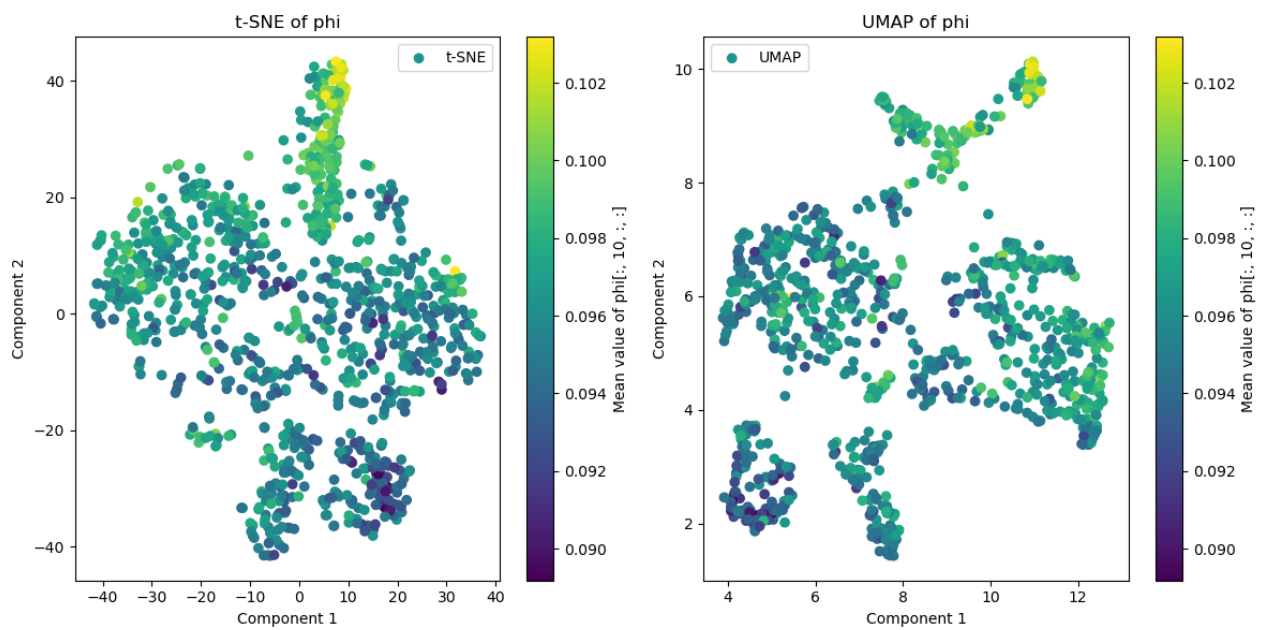
Figure 8: Conv2d output 2

Figure 9: Conv2d output 3

Figure 10: T-SNE vs UMAP

6 Future Work
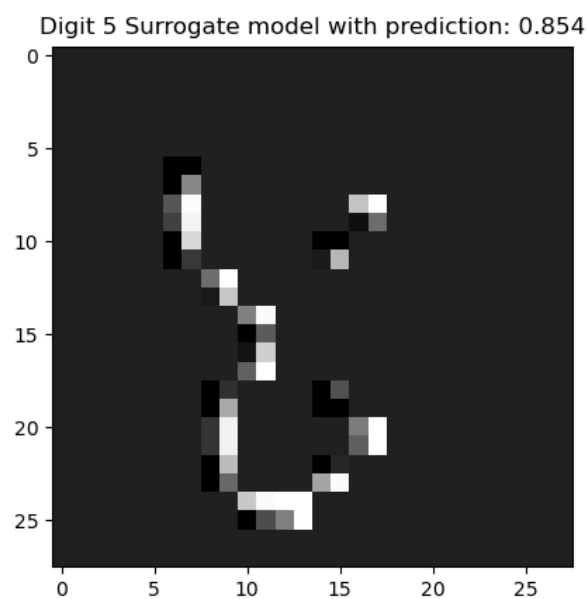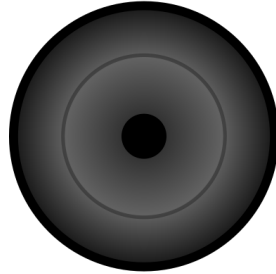

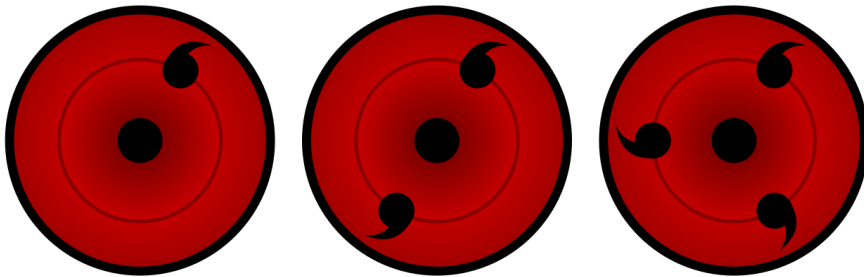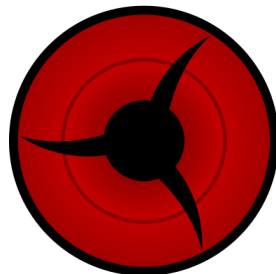Figure 11: Using conv2d and GAN

Sharingan Evolution:

Figure 12: Sharingan

7   Conclusion

. . .

123456

References

[1] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, "Synthesizing Robust Adversarial Examples," June 2018.

[2] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," Mar. 2017.

[3] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. New York, NY, USA: Cambridge University Press, 2014.

[4] L. Devroye, L. Györfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*. New York: Springer, 1996.

[5] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics, New York, NY: Springer, second edition, corrected at 12th printing 2017 ed., 2017.

[6] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization," *International Journal of Computer Vision*, vol. 128, pp. 336–359, Feb. 2020.

[7] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational Inference: A Review for Statisticians," *Journal of the American Statistical Association*, vol. 112, pp. 859–877, Apr. 2017.

[8] L. Mescheder, S. Nowozin, and A. Geiger, "Adversarial Variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks,"

[9] J. Su, "Variational Inference: A Unified Framework of Generative Models and Some Revelations,"