# Generative Adversarial Networks

**Solomon Harvey  Alexandros Kouvatseas  Théo Courty**

## Abstract

Report of the project, where we explored generative modeling techniques using the MNIST dataset. We focused on improving the GAN's (Generative Adversarial Network) performance, by implementing alternative techniques like f-GAN and W-GAN. Our goal was to evaluate and compare the generative performance in terms of the metrics FID, Recall Precision, as well as visual quality of the samples.

## 1. Introduction

During this project, we tackled a basic image generation task using two distinct GAN variants: the f-GAN and the Wasserstein GAN (W-GAN). These models were evaluated and compared against the baseline Vanilla GAN. Significant modifications were necessary to accommodate the architectural differences between these advanced techniques. The experiments were conducted using the well-known MNIST dataset, which is commonly employed as a benchmark for image generation research.

### 1.1. Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs), introduced by Ian Goodfellow and colleagues in 2014 (**?** ), are a class of machine learning frameworks designed to generate new data samples that resemble a given training dataset. A GAN consists of two neural networks—the generator and the discriminator—that are trained simultaneously through an adversarial process. The generator creates synthetic data samples, while the discriminator evaluates them against real data, aiming to distinguish between authentic and generated samples. This adversarial training encourages the generator to produce increasingly realistic data over time.

The Vanilla GAN is the original and simplest form of GAN architecture. Its Generator and Discriminator are both implemented as multilayer perceptrons. The generator takes random noise as input and transforms it into data samples intended to mimic the real data distribution. The discriminator, on the other hand, receives both real and generated data samples and learns to classify them correctly. The training process involves a minimax game where the generator strives to deceive the discriminator, and the discriminator aims to accurately identify real versus fake data. This dynamic leads to the generator producing data that becomes increasingly indistinguishable from real data.

### 1.2. MNIST Dataset

The MNIST database is a widely-used benchmark in the field of Machine Learning and Computer vision. Due to its simplicity, its standardized format, and extensive use in literature, MNIST has been the go-to dataset to test different kind of models, varying from image classification to generation tasks. It offers a consistent baseline for comparing different techniques, as we do in this project too. It consists of 70,000 grayscale images of handwritten digits, from 0 to 9, where each image is a 28x28 pixel matrix.

## 2. f-GAN

The f-GAN framework is the first technique of a generative model that we used, with the main idea of leveraging the concept of f-divergences to measure the discrepancy between the real and generated data distributions. f-GANs offer a more generalized approach, allowing for a variety of divergence measures, that we mention later. By using a variational objective based on these f-divergences, the model gains flexibility in adapting the training objective to different problem settings.

### 2.1. Losses

The f-GAN architecture depends on the variational function, where an instance of it is calculated in equation 1. Through this function we have the function tailored specifically to a specific choice of f-divergence as seen in table 1.

$$T_\omega(x) = g_f(v_\omega(x)) \tag{1}$$

Here $v_\omega(x)$ is the linear output of the discriminator for a given sample $x$, and $g_f$ is the activation function used on the last layer of the discriminator and depends on hte f-divergence chosen (see table 1). The losses of the discriminator and the generator are calculated by equation 2 and 3 respectively. $\mathscr{L}_\mathcal{D}$ helps train the discriminator by maximizing the expected value of $T_\omega(x)$ when the input $x$ comes from the real data distribution $P$. Simultaneously, it minimizes the expected value of the conjugate $f_*$ of the f-divergence for samples drawn from the generated data distribution $Q$. This process encourages the discriminator to differentiate effectively between real and generated data. In the other hand, $\mathscr{L}_{\mathcal{G}}$ aims to produce samples that make the discriminator unable to distinguish them from the real data, by minimizing the expected value of $f_*(T_\omega(x))$ over samples from the generated distribution $Q$. From the objective function given in (**?** ), we can derive losses [1] for the discriminator ($\mathcal{D}$) and generator ($\mathcal{G}$) :

$$\mathscr{L}_{\mathcal{D}} = \mathbb{E}_{x\sim p}[T_\omega(x)] - \mathbb{E}_{\hat{x}\sim Q_\theta}[f_*(T_\omega(\hat{x}))] \tag{2}$$

$$\mathscr{L}_{\mathcal{G}} = -\mathbb{E}_{\hat{x}\sim Q_\theta}[f_*(T_\omega(\hat{x}))] \tag{3}$$

Where $x$ are real samples and $\hat{x}$ is the generated samples by the generator $\mathcal{G}$.

## 2.2. Optimization of f-GAN

As the model depends on the f-divergence chosen, so the training, especially its stability. We indeed found than using the Jensen-Shannon (JS) f-divergence results in stable convergence, whereas using Kullback-Leibler (KL) was always difficult to train due to exploding losses in the first epochs. To address this issue we add to think about the optimization process. The first step is to use Adam (**?** ) for both the discriminator and the generator with momemtum parameters $\beta_1 = 0.5$ and $\beta_2 = 0.999$, same as (**?** ). This optimizer is known for its stability towards convergence, and experimentally shows better results in our case than using SGD. By default, we used a learning rate of $\alpha = 2 \times 10^{-4}$, but we had sometimes to tweak it a little bit experimentally. To ensure gradients are not exploding during training we also implemented gradient clipping, a method that upper bound the gradient norm :

$$\nabla\mathscr{L} = c\frac{\nabla\mathscr{L}}{\|\nabla\mathscr{L}\|} \text{ if } \|\nabla\mathscr{L}\| > c \tag{4}$$

Where $c$ is a parameter fixed to $1$ arbitrarily. This method is also mentioned by (**?** ) to be useful without explicitly giving a value for $c$. We also found that using a weight decay regularization of $w = 10^{-3} - 10^{-5}$ to be useful on the discriminator only, as it is the most powerful model of the two.

## 2.3. Pre-training

Even with the previous conditions, the training (especially for KL) is not always stable. One way to address this is to use a sort of warm-up by using an already trained discriminator and generator. We choose to use the JS f-divergence trained with 6 epochs with previous parameters due to its stability in this setup ($w = 10^{-5}$). The idea behind this choice is to first optimize a generator capable of producing somewhat realistic samples within a small amount of epochs just to point out a good direction to the model (see Fig. 4 in appendix for an example of output). This can also help prevent mode collapse (state where the generator focuses on only producing one mode), as JS is kind of symmetric between precision and recall, and can cover a different variety of the modes. In practice, we compute this pre-training once, save the weights of both generator and discriminator, and initialize the model with them for the final training.

To ensure that the transition between JS and other f-divergence, in practice, we start with a learning rate of $\alpha_{\text{switch},1} = \alpha/100$, then update it to $\alpha_{\text{switch},2} = \alpha/10$ for the first epochs before increasing to the desired $\alpha$, which is aimed to be around to $\sim 10^{-4}$.

[1]As we are mostly interested by computing theirs gradients, we can split them from the objective function based on their dependence on $\theta$ and $\omega$

**Algorithm 1** f-GAN Model

```
Input: Generator model $\mathcal{G}$, Discriminator model $\mathcal{D}$, Optimizers $\mathcal{O}_G$ and $\mathcal{O}_D$, f-divergence $f$
Output: Trained f-GAN model
1. Initialization:
Set $\mathcal{G}, \mathcal{D}, \mathcal{O}_G, \mathcal{O}_D$, and activation function $g_f$

2. Variational Function:
$T_\omega(x) \leftarrow g_f(v_\omega(x))$

3. Discriminator Loss:
Sample noise $z \sim \mathcal{N}(0,1)$
Compute output: $T_{\text{fake}} \leftarrow T_\omega(\mathcal{G}(z))$ and $T_{\text{real}} \leftarrow T_\omega(x)$
Compute gradient loss $\nabla_\omega \mathscr{L}_{\mathcal{D}}$
Clip gradients: clip_grad($\mathcal{D}$, 1)[4]

4. Generator Loss:
Sample noise $z \sim \mathcal{N}(0,1)$
Compute output: $T_{\text{fake}} \leftarrow T_\omega(\mathcal{G}(z))$
Compute gradient loss: $\nabla_\theta \mathscr{L}_{\mathcal{G}}$
Clip gradients: clip_grad($\mathcal{G}$, 1)[4]

6. Update parameters :
Compute of step (descent) of $\mathcal{O}_{\mathcal{G}}(\nabla\mathscr{L}_{\mathcal{G}}, \theta)$
Compute of step (ascent) of $\mathcal{O}_{\mathcal{D}}(-\nabla\mathscr{L}_{\mathcal{D}}, \omega)$
```

## 2.4. Results

One example of training is shown on Fig. 1, where we show both losses as well as the discriminator accuracy (refered as ”accuracy” alone later in the report). The latter is computed based on the value as $f'(1)$, if the output of the discriminator is greater (lower) than it, then it consider the sample as

*Table 1.* Different f-divergences used during the project. Table from (? ).

| NAME | ACTIVATION $g_f(v)$ | $f_*(t)$ | $f'(1)$ |
|---|---|---|---|
| GAN | $-\log(1+\exp(-v))$ | $-\log(1-\exp(t))$ | $-\log(2)$ |
| JENSEN-SHANNON (JS) | $\log(2)-\log(1+\exp(-v))$ | $-\log(2-\exp(t))$ | 0 |
| KULLBACK-LEIBLER (KL) | $v$ | $\exp(t-1)$ | 1 |
| REVERSE KL | $-\exp(-v)$ | $-1-\log(-t)$ | $-1$ |
| PEARSON $\chi^2$ | $v$ | $\frac{1}{4}t^2+t$ | 0 |
| SQUARED HELLINGER (SH) | $1-\exp(-v)$ | $\frac{t}{1-t}$ | 0 |

real (generated) sample. The trend of the accuracy kind of follows the losses and is a also a good metric of how training is going. Indeed, if the accuracy is near 1, the discriminator is distinguishing real from fake sample very well, whereas, if it goes near 0.5 it is just guessing. Ideally, we would like the accuracy slowly decreasing towards 0.5. We can see this kind of trend on Fig. 1, but in practice, because the discriminator is also training, going near that value can take a really high number of epoch without much increase in generated samples quality.

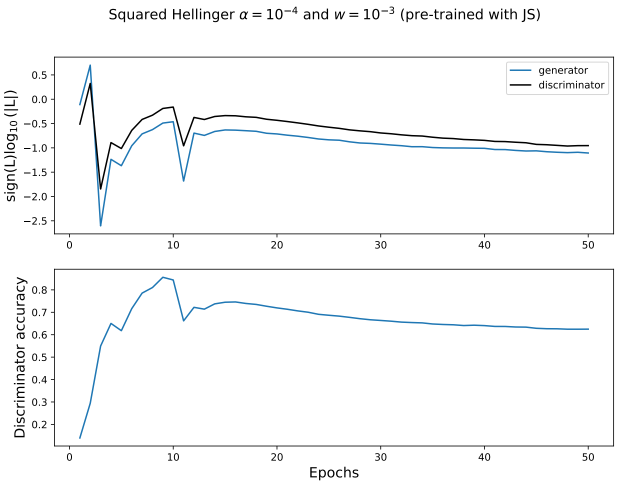

*Figure 1.* Training example using the Squared Hellinger (SH) f-divergence.

We present on Fig. 2 samples generated using the KL f-divergence, which was by eye inspection the better of all f-divergence. Other examples of samples with different f-divergence are given in Appendix A. Metrics computed on the testing platform are shown on the Table 2, and output examples of different f-divergences are given in appendix.

## 3. Conclusion

From Table 2, we can observe notable differences in performance among the models. First, we compare the KL f-GAN with the rKL variant. The KL f-GAN demonstrated higher efficiency and achieved better overall results compared to

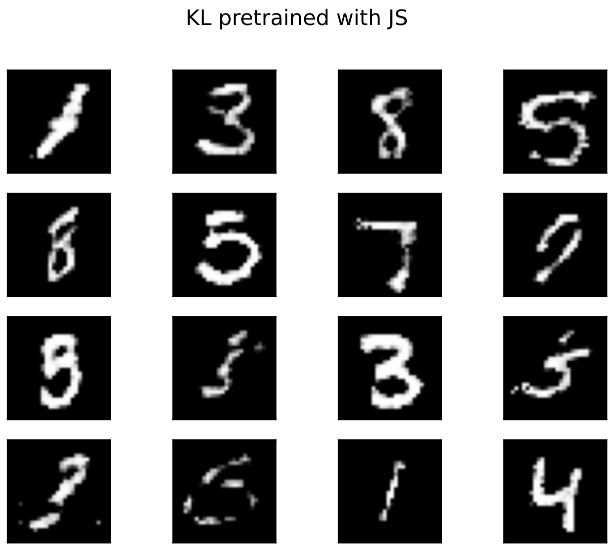

*Figure 2.* Output example based on the Kullback-Leibler (KL) f-divergence.

the rKL version.

| Model | VanillaGAN | KL f-GAN | rKL f-GAN |
|---|---|---|---|
| FID | 50.14 | 44.36 | 94.79 |
| Precision | 0.47 | 0.54 | 0.55 |
| Recall | 0.15 | 0.19 | 0.16 |
| Time | 97.38 | 103.84 | 82.06 |

*Table 2.* Results between the models tested on the platform. The Vanilla GAN is the one given on the platform.

## 4. Wasserstein GAN (WGAN)

### 4.1. Wasserstein Distance

Many f-divergences require the two distributions to overlap, causing vanishing gradients that halt the generator's progress. This kind of situation happens especially in high-dimensional manifolds, hence it is relatively common. A solution is to use a metric that measures distance in a more broad sense, not necessarily requiring distribution overlap. This is the case for the Wasserstein distance. By the Kantorovich-Rubinstein duality, if we denote by $\|\cdot\|_L$ the Lipschitz seminorm, the *Wasserstein distance* between two distributions $P$ and $Q$ is defined as

$$W(P,Q) = \sup_{\|f\|_L \leqslant 1} \Big\{ \mathbb{E}_{x\sim P}[f(x)] - \mathbb{E}_{y\sim Q}[f(y)] \Big\}.$$

Intuitively, $W(P,Q)$ measures the minimal effort required to move mass around to change $P$ into $Q$, thus avoiding the problem of non-overlapping distributions.

## 4.2. WGAN With Weight Clipping (WGAN-WC)

### 4.2.1. Theoretic Setup

Let us explain our setup, inspired by [11]. Denote by $P_r$ the real distribution of the data. Denote by $\theta$ the weights of the generator $g_\theta$ and by $P_\theta$ the probability distribution of $g_\theta$. We assume $P_\theta = g_\theta(Z)$ where $Z$ is a distribution on a low-dimensional latent space. Denote by $w$ the weights of the discriminator $f_w$, called the *critic* in the WGAN framework. If $x$ is a data sample, then $f_w(x)$ is the *score* given to $x$ by $f_w$. In the WGAN model, the score is not a probability distribution and can take any value in $\mathbb{R}$. A *Wasserstein GAN* (WGAN) is a GAN whose goal is to solve the min-max problem

$$\inf_\theta \sup_{-c\leqslant w\leqslant c} \mathbb{E}_{x\sim P_r}[f_w(x)] - \mathbb{E}_{z\sim Z}[f_w\circ g_\theta(z)]$$

where $c$ is clipping constant, simulating the Lipschtiz constraint. Hence the critic loss $L_{\text{crit}}$ and the generator loss $L_{\text{gen}}$ are equal to

$$\begin{cases} L_{\text{crit}}(w) &= \mathbb{E}_{z\sim Z}[f_w\circ g_\theta(z)] - \mathbb{E}_{x\sim P_r}[f_w(x)], \\ L_{\text{gen}}(\theta) &= -\mathbb{E}_{z\sim Z}[f_w\circ g_\theta(z)]. \end{cases}$$

### 4.2.2. Algorithm

Below is the algorithm for WGAN-WC. The full code may be found in the file `model_wgan_wc.py` on GitHub.

**Algorithm 2** WGAN-WC

```
Require: The critic learning rate $\alpha = 0.00025$, the generator learning rate $\beta = 0.00005$, the clipping parameter $c = 0.01$, the batch size $m = 64$, the number of critic iterations per generator iteration $n_{\text{critic}} = 5$.
Require: Initial critic parameters $w_0$, initial generator parameters $\theta_0$.
 1: while $\theta$ has not converged do
 2:     for $1 \leqslant t \leqslant n_{\text{critic}}$ do
 3:         Sample a batch $\{x_i\}_{i=1}^m \sim \mathbb{P}_r$
 4:         Sample a batch $\{z_i\}_{i=1}^m \sim Z$
 5:         Compute $\delta_w \leftarrow \nabla_w\{\frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z_i)) - \frac{1}{m}\sum_{i=1}^m f_w(x_i)\}$
 6:         Update $w \leftarrow w - \alpha \cdot \text{RMSProp}(w, \delta_w)$
 7:         Clip $w \leftarrow \text{clip}(w, -c, c)$
 8:     end for
 9:     Sample a batch $\{z_i\}_{i=1}^m \sim Z$
10:     Compute $\delta_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z_i))$
11:     Update $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, \delta_\theta)$
12: end while
```

## 4.3. WGAN With Gradient Penalty (WGAN-GP)

### 4.3.1. Theoretic Setup

As presented in [12], instead of weight clipping, we may simulate the Lipschitz constraint by adding a gradient penalty to the critic loss. More specifically, the critic loss becomes

$$L_{\text{crit}}(w) = \underbrace{\mathbb{E}_{z\sim Z}[f_w\circ g_\theta(z)] - \mathbb{E}_{x\sim P_r}[f_w(x)]}_{\text{Original critic loss}} + \underbrace{\lambda \mathbb{E}_{\hat{x}\sim P_{\hat{x}}}\left[ (\|(\nabla_{\hat{x}} f_w(\hat{x})\| - 1)^2 \right]}_{\text{Gradient penalty}}$$

where

$$\begin{cases} \lambda > 0, \\ P_{\hat{x}} \sim \varepsilon P_r + (1-\varepsilon)P_\theta, \\ \varepsilon \sim \mathcal{U}(0,1). \end{cases}$$

### 4.3.2. Algorithm

Below is the algorithm for WGAN-GP. The full code may be found in the file `model_wgan_gp.py` on GitHub.

**Algorithm 3** WGAN-GP

```
Require: The critic learning rate $\alpha = 0.00025$, the generator learning rate $\beta = 0.00005$, the gradient penalty parameter $\lambda = 10$, the batch size $m = 64$, the number of critic iterations per generator iteration $n_{\text{critic}} = 5$.
Require: Initial critic parameters $w_0$, initial generator parameters $\theta_0$.
 1: while $\theta$ has not converged do
 2:     for $1 \leqslant t \leqslant n_{\text{critic}}$ do
 3:         Sample a batch $\{x_i\}_{i=1}^m \sim \mathbb{P}_r$
 4:         Sample a batch $\{z_i\}_{i=1}^m \sim Z$
 5:         Sample $\{\varepsilon_i\}_{i=1}^m \sim \mathcal{U}(0,1)$
 6:         Define $\{\hat{x}_i\}_{i=1}^m \leftarrow \{\varepsilon_i x_i + (1-\varepsilon_i) g_\theta(z_i)\}_{i=1}^m$
 7:         Compute $\delta_w \leftarrow \nabla_w \{\frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z_i)) - \frac{1}{m}\sum_{i=1}^m f_w(x_i) + \frac{\lambda}{m}\sum_{i=1}^m (\|\nabla_\theta f_w(\hat{x}_i)\| - 1)^2\}$
 8:         Update $w \leftarrow w - \alpha \cdot \text{RMSProp}(w, \delta_w)$
 9:         Clip $w \leftarrow \text{clip}(w, -c, c)$
10:     end for
11:     Sample a batch $\{z_i\}_{i=1}^m \sim Z$
12:     Compute $\delta_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z_i))$
13:     Update $\theta \leftarrow \theta - \beta \cdot \text{RMSProp}(\theta, \delta_\theta)$
14: end while
```

### 4.4. Empirical Results

Below you shall find the results for WGAN-WC and WGAN-GP trained for 150 epochs. As you see, WGAN-GP significantly improves over WGAN-WC. This is explained by the fact that a gradient penalty is a far less restrictive condition on the critic than weight clipping. The only drawback is that WGAN-GP needs to be trained for longer, because a gradient penalty does not simulate the Lipschitz constraint as directly as weight clipping.

| Model | FID | Precision | Recall |
|---|---|---|---|
| WGAN-WC | 94.79 | 0.63 | 0.16 |
| WGAN-GP | 36.17 | 0.45 | 0.26 |

In Fig. 3 are presented some WGAN-GP generator outputs.

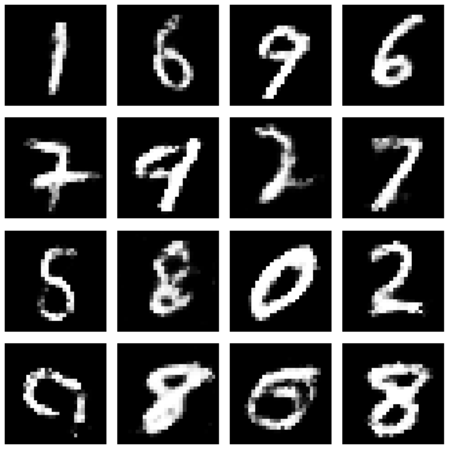

*Figure 3.* Some WGAN-GP generator outputs.

## References

[1] P. Langley (2000), *Crafting Papers on Machine Learning*, Proceedings of the 17th International Conference on Machine Learning (ICML 2000), Morgan Kaufmann.

[2] T. M. Mitchell (1980), *The Need for Biases in Learning Generalizations*, Computer Science Department, Rutgers University.

[3] M. J. Kearns (1989), *Computational Complexity of Machine Learning*, PhD Thesis, Department of Computer Science, Harvard University.

[4] R. S. Michalski, J. G. Carbonell, T. M. Mitchell (1983), *Machine Learning: An Artificial Intelligence Approach, Vol. I*, Tioga.

[5] R. O. Duda, P. E. Hart, D. G. Stork (2000), *Pattern Classification*, 2nd ed., John Wiley and Sons.

[6] N. N. Author (2021), *Suppressed for Anonymity*.

[7] A. Newell, P. S. Rosenbloom (1981), *Mechanisms of Skill Acquisition and the Law of Practice*, Cognitive Skills and Their Acquisition, Lawrence Erlbaum Associates, Inc.

[8] S. Nowozin, B. Cseke, R. Tomioka (2016), *f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization*, Advances in Neural Information Processing Systems, Curran Associates, Inc.

[9] I. Goodfellow et al. (2014), *Generative Adversarial Nets*, Advances in Neural Information Processing Systems, Curran Associates, Inc.

[10] D. P. Kingma, J. Ba (2014), *Adam: A Method for Stochastic Optimization*, arXiv preprint arXiv:1412.6980.

[11] M. Arjovsky, S. Chintala, L. Bottou (2017), *Wasserstein GAN*, Courant Institute of Mathematical Sciences, Facebook AI Research.

[12] I. Gulrajani et al. (2017), *Improved Training of Wasserstein GANs*, Montreal Institute for Learning Algorithms, Courant Institute of Mathematical Sciences, CIFAR Fellow.

## A. Output for Other f-divergences

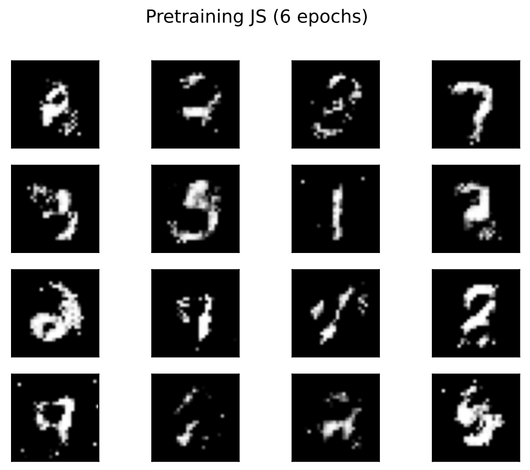

*Figure 4.* Samples generated by the JS model used for pre-training used afterward when choosing other f-divergences. Theses samples serve to pinpoint a reasonable direction to the future model, but we can see that the JS model haven't really converged yet.

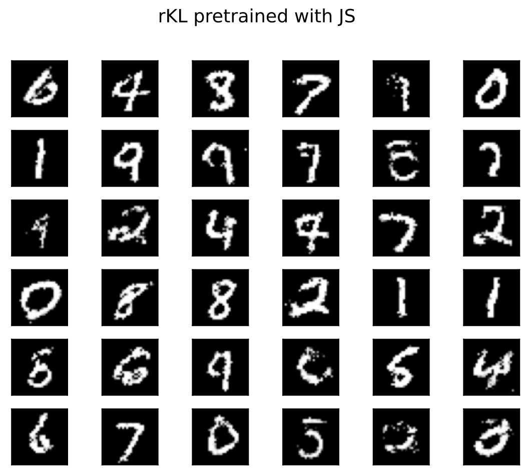

*Figure 5.* Example of output with the reverse Kullback-Leibler f-divergence.

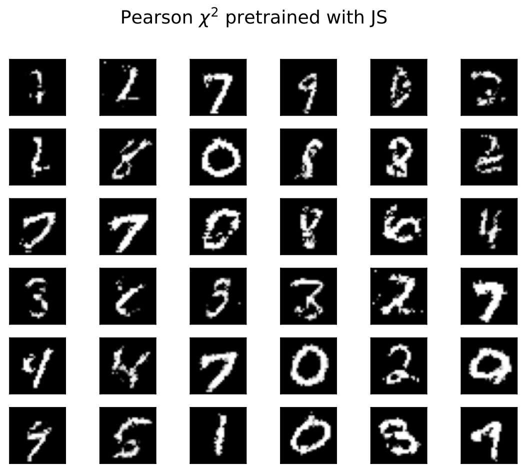

*Figure 6.* Example of output with the Pearson $\chi^2$ f-divergence.

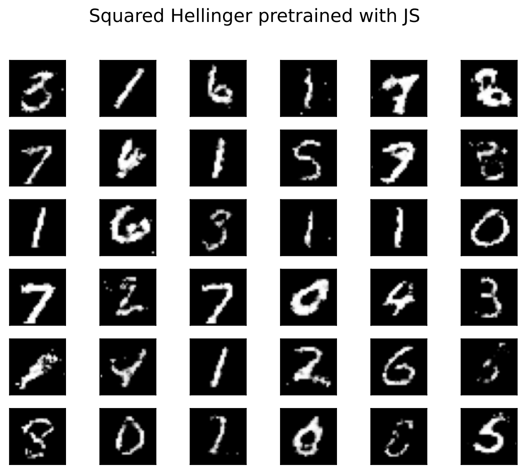

*Figure 7.* Example of output with the Squared Hellinger f-divergence.