Model Heavy-Tailed Latent Space in Variational AutoEncoders

Canzoneri Daniele, Grillea Francesco

June 11, 2024

Abstract

In this research, we explore the use of heavy-tailed distributions to represent latent space in Variational AutoEncoders (VAEs). Specifically, we employed Normal, LogNormal, and Inverse Gaussian distributions as latent space representations. Our investigation includes a comparison of these distributions in terms of their Reconstruction Capability, Denoising Capability, and Generation Capability. The evaluation is conducted on the well-known MNIST dataset as well as a synthetic dataset generated using a LogNormal distribution. This approach aims to assess whether employing a heavy tailed latent space can more effectively model different input spaces.

github.com/francescogrillea/HeavyTailed-VAE

1 Introduction

Variational Autoencoders (VAEs) have become fundamental in generative modeling, allowing the learning of complex data distributions through latent space representations. The traditional approach involves using a Normal distribution to model the latent space, leveraging the reparameterization trick to facilitate backpropagation during training. However, real-world data often exhibit heavy-tailed distributions [1], which a Normal latent space may not be able to capture. To address this, we propose the use of heavy-tailed distributions [2, 3], specifically the LogNormal and Inverse Gaussian distributions, as alternatives to the Normal distribution for the latent space in VAEs. Heavy-tailed distributions can better model data with outliers or those exhibiting non-Gaussian behavior, potentially leading to improved generative performance and more robust learned representations [4].

We conduct our experiments using both the MNIST dataset and a synthetic dataset generated using a LogNormal distribution. Our primary objective is to determine if a LogNormal latent space can better model an input space that follows a LogNormal distribution. Then, we assess the performance of employing these distributions in VAEs across three key metrics: Reconstruction Capability, Denoising Capability, and Generation Capability. By providing this comparative analysis, we hope to provide a comprehensive understanding of how different latent space distributions affect the performance and capabilities of VAEs.

2 Methodology

2.1 Variational Auto-Encoder

Variational autoencoders (VAEs) are a type of generative model that combine neural networks and probabilistic graphical models to generate new data samples similar to a given dataset. They consist of an encoder that maps input data to a latent space, and a decoder that reconstructs the data from this latent representation. Unlike traditional autoencoders, VAEs incorporate a probabilistic element by learning the distribution parameters (mean and variance) of the latent space, which enables them to generate diverse and realistic outputs by sampling from this distribution. This approach allows VAEs to handle complex data distributions and has applications in image generation, anomaly detection, and unsupervised learning. Figure 1 illustrates the Variational AutoEncoder's architecture.

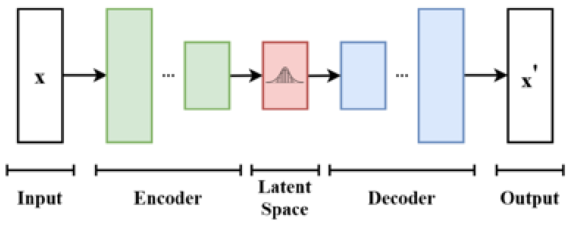


Fig. 1: Variational Auto-Encoder basic architecture

2.2 Latent Space Distribution

In the context of variational autoencoders (VAEs), drawing samples from a learned distribution presents a significant challenge due to the requirement for differentiability in the training process. Specifically, the backpropagation algorithm requires that the gradient flows from the output back to the input to update the model parameters effectively. However, the sampling operation is non-differentiable, so we can't simply drawn a sample from a distribution characterized by some learned parameters and learn those parameters at the same time. To address this, *reparameterization tricks* have employed in order to drawn differentiable samples according to some underlying distribution of the latent space. Of course different distributions need different reparametrization tricks. Figure 2 shows the reparametrization trick for a Normal latent space and how can gradient flows back to input layer. Firstly we describe how to handle a Normal distribution which will be the baseline for our experiments, then we dive deeper into two heavy-tailed latent distributions.

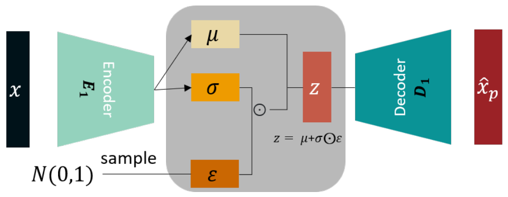


Fig. 2: Reparametrization Trick for Noraml latent sapce

Normal Distribution The classical latent distribution used in Variational Auto-Encoder is the Normal Distribution. Its probability distribution is

$$f(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

The intuition behind the reparametrization trick for Normal latent spaces is very simple: a random variable $\varepsilon \sim \mathcal{N}(0,1)$ can be transformed into $z \sim \mathcal{N}(\mu,\sigma)$ simply by

$$\varepsilon = \frac{z-\mu}{\sigma} \Rightarrow z = \varepsilon\sigma + \mu$$

Is easy to notice that now, since both sum and product are differentiable operations, the gradient can flow from z back to μ and σ and then back to input x, making the training process possible.

Algorithm 1 Sampling Algorithm for Normal distribution

```
1: Generate $\varepsilon \sim \mathcal{N}(0,1)$
2: Compute $z = \mu + \sigma * \varepsilon$
3: return $z$
```

LogNormal Distribution The first heavy-tailed distribution that we examine is the *LogNormal* distribution. Its probability distribution is defined for $x > 0$ as

$$f(x;\mu,\sigma) = \frac{1}{x\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\ln x-\mu)^2}{2\sigma^2}\right) \tag{1}$$

The LogNormal distribution is also defined as the probability distribution of a random variable whose logarithm follows a normal distribution:

$$Z \sim N(\mu,\sigma^2) \quad \Rightarrow \quad X = e^Z \sim LogNormal(\mu,\sigma^2)$$

This allows us to use the same reparametrization trick of the Normal distribution: to draw a sample from a LogNormal distribution, we just exponentiate a sample drawn from a Normal distribution. This process is differentiable since the sampling generation is differentiable, thanks to the reparametrization trick for Normal distribution, and the exponentiation operation is also differentiable.

Algorithm 2 Sampling Algorithm for LogNormal distribution

```
1: Generate $\varepsilon \sim \mathcal{N}(0,1)$
2: Compute $z = \mu + \sigma * \varepsilon$
3: return $exp(z)$
```

Inverse Gaussian Distribution The second heavy-tailed distribution examined is the *Inverse Gaussian* distribution. It is defined by two parameters

- **Mean** (μ): the mean of the distribution;
- **Shape** (λ): controls the dispersion of the distribution.

Its probability distribution is

$$f(x;\mu,\lambda) = \sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(-\frac{\lambda(x-\mu)^2}{2\mu^2 x}\right)$$

defined for $x > 0, \mu > 0, \lambda > 0$.

To sample from an Inverse Gaussian distribution we can exploit the following algorithm [5] that is also suitable for the back propagation in our NNs.

Algorithm 3 Sampling Algorithm for Inverse Gaussian distribution

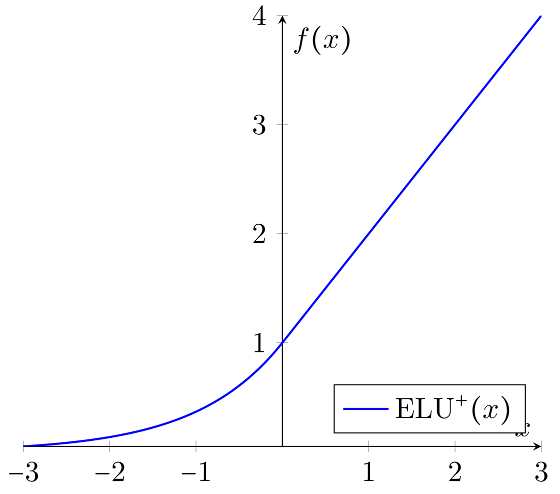
```
1: Generate $\varepsilon \sim \mathcal{N}(0,1)$
2: Set $y = \varepsilon^2$
3: Compute $x = \mu + \frac{\mu^2 y}{2\lambda} - \frac{\mu}{2\lambda}\sqrt{4\mu\lambda y + \mu^2 y^2}$
4: Generate $z \sim U(0,1)$
5: if $z \le \frac{\mu}{\mu+x}$ then
6:     return $x$
7: else
8:     return $\frac{\mu^2}{x}$
9: end if
```

The Inverse Gaussian distribution is defined only for positive μ and λ: the problem arise in the training of our neural network since the encoder outputs values also in the negative range. Notice also how in step 3 of algorithm 3 the argument of the square root is not guaranteed to be positive if μ and λ are not positive.

To solve this problem we interposed a custom activation function, called ELU^+ and defined as equation (2) between the encoder and the sampling layer.

$$ELU^+(x) = \begin{cases} e^x & \text{if } x < 0 \\ x+1 & \text{if } x \geq 0 \end{cases} = ELU(x) + 1 \tag{2}$$

With this modification we force the input of the sampling layer to be strictly positive.



Another option was to use the exponential function as the activation between the encoder and the sampling layer. In our experiment this produced, particularly in the first steps, very high distribution parameters, pushing the sample generated to `NaN`. We opted for the ELU^+ activation function since it is similar to the well studied $ReLU$ activation function which has good convergence properties.

Table 1 shows a simple comparison between the distribution employed in this study. For a better comprehension of the results presented in the next section, is important to notice that LogNormal distribution has an heavier tail than the InverseGaussian distribution.

Distribution	**Mean**	**Variance**	**Constraints**
Normal	μ	σ^2	$\sigma > 0$
LogNormal	$\exp\left(\mu + \frac{\sigma^2}{2}\right)$	$\left[\exp(\sigma^2) - 1\right] \exp\left(2\mu + \sigma^2\right)$	$\sigma > 0$
Inverse Gaussian	μ	$\frac{\mu^3}{\lambda}$	$\mu > 0, \lambda > 0$

Table 1: Distributions Comparison

2.3 Loss function

To train our VAEs we used the classical loss function composed of two main components: the reconstruction loss and the KL divergence loss.

- The reconstruction loss measures how well the VAE can reconstruct the input data from the latent representation. It is implemented as the mean squared error between the reconstructed data and the observed data.
- The KL divergence loss, on the other hand, regularizes the learned latent space by measuring how much the learned distribution of the latent variables given the input data deviates from the prior distribution (typically a distribution with "standard" parameters).

Mathematically, the total VAE loss function is the sum of the reconstruction loss and the KL divergence term: if x is the original data and $z = NN(x)$ is the reconstructed data we define

$$\mathcal{L} = \frac{1}{2}\Big(MSE(x,z) + \beta D_{KL}(P \parallel Q)\Big) \tag{3}$$

where P is the learned latent distribution and Q is a standard latent distribution (for example $Q \sim N(0,1)$).

2.3.1 KL divergence

Now we study the KL divergence between two distributions. The KL divergence between two probability distribution P and Q is defined as

$$D_{KL}(P \parallel Q) = \int_X P(x)\ln\frac{P(x)}{Q(x)}dx$$

Normal distribution It is a well known result [6] that if P is the probability distribution of a $N(\mu_1,\sigma_1)$ random variable and Q is the probability distribution of a $N(\mu_2,\sigma_2)$ random variable, then

$$D_{KL}(P \parallel Q) = \frac{1}{2}\left[\frac{(\mu_1-\mu_2)^2}{\sigma_2^2} + \frac{\sigma_1^2}{\sigma_2^2} - \ln\frac{\sigma_1^2}{\sigma_2^2} - 1\right]$$

In our case the Q probability distribution will be the distribution of a standard Gaussian random variable $N(0,1)$ and so the expression becomes

$$D_{KL}(P \parallel Q) = \frac{1}{2}\left[\mu_1^2 + \sigma_1^2 - \ln\sigma_1^2 - 1\right] \tag{4}$$

LogNormal distribution Let's compute the KL divergence between two LogNormal distributions $LogNormal(\mu_1, \sigma_1)$ and $LogNormal(\mu_2, \sigma_2)$ with probability distribution defined as (1).

$$\begin{aligned}
D_{KL}(P \parallel Q) &= \int_0^\infty f(x; \mu_1, \sigma_1) \ln \frac{f(x; \mu_1, \sigma_1)}{f(x; \mu_2, \sigma_2)} = \\
&= \int_0^\infty f(x; \mu_1, \sigma_1) \left[- \ln x\sqrt{2\sigma_1^2 \pi} - \frac{(\ln x - \mu_1)^2}{2\sigma_1^2} + \ln x\sqrt{2\sigma_2^2 \pi} + \frac{(\ln x - \mu_2)^2}{2\sigma_2^2} \right] = \\
&= \int_0^\infty f(x; \mu_1, \sigma_1) \left[\ln \frac{\sigma_2}{\sigma_1} - \frac{(\ln x - \mu_1)^2}{2\sigma_1^2} + \frac{(\ln x - \mu_2)^2}{2\sigma_2^2} \right] = \\
&= \ln \frac{\sigma_2}{\sigma_1} - \mathbb{E}_P \left[\frac{(\ln X - \mu_1)^2}{2\sigma_1^2} \right] + \mathbb{E}_P \left[\frac{(\ln X - \mu_2)^2}{2\sigma_2^2} \right]
\end{aligned}$$

where X is a random variable with P as probability distribution. Since $X \sim LogNormal(\mu_1, \sigma_1)$ we have

$$\ln X \sim N(\mu_1, \sigma_1) \quad \Rightarrow \quad \mathbb{E}_P \left[\frac{(\ln X - \mu_1)^2}{2\sigma_1^2} \right] = \frac{Var(\ln X)}{2\sigma_1^2} = \frac{1}{2}$$

The last term is expanded as

$$\begin{aligned}
\mathbb{E}_P \left[\frac{(\ln X - \mu_2)^2}{2\sigma_2^2} \right] &= \frac{1}{2\sigma_2^2} \mathbb{E}_P \left[(\ln X - \mu_1 + \mu_1 - \mu_2)^2 \right] = \\
&= \frac{1}{2\sigma_2^2} \left(\mathbb{E}_P \left[(\ln X - \mu_1)^2 \right] + \mathbb{E}_P \left[2(\ln X - \mu_1)(\mu_1 - \mu_2) \right] + \mathbb{E}_P \left[(\mu_1 - \mu_2)^2 \right] \right) = \\
&= \frac{1}{2\sigma_2^2} \left[\sigma_1^2 + (\mu_1 - \mu_2)^2 \right]
\end{aligned}$$

Finally the KL divergence can be expressed as

$$\begin{aligned}
D_{KL}(P \parallel Q) &= \ln \frac{\sigma_2}{\sigma_1} - \frac{1}{2} + \frac{\sigma_1^2}{2\sigma_2^2} + \frac{(\mu_1 - \mu_2)^2}{2\sigma_2^2} = \\
&= \frac{1}{2} \left[\frac{\sigma_1^2}{\sigma_2^2} + \frac{(\mu_1 - \mu_2)^2}{\sigma_2^2} - \ln \frac{\sigma_1^2}{\sigma_2^2} - 1 \right]
\end{aligned}$$

which is exactly the same expression of the KL divergence between two Gaussian distributions.

Inverse Gaussian distribution It is known [7] that if P is the probability distribution of a $IG(\mu_1, \lambda_1)$ random variable and Q is the probability distribution of a $IG(\mu_2, \lambda_2)$ random variable, then

$$D_{KL}(P \parallel Q) = \frac{1}{2} \left[\ln \frac{\lambda_1}{\lambda_2} - 1 + \frac{\lambda_2 \mu_1}{\mu_2^2} - \frac{2\lambda_2}{\mu_2} + \frac{\lambda_2}{\mu_1} + \frac{\lambda_2}{\lambda_1} \right].$$

In our case the Q probability distribution will be the distribution of an Inverse Gaussian random variable $IG(1,1)$ and so the expression becomes

$$D_{KL}(P \parallel Q) = \frac{1}{2}\left[\ln \lambda_1 - 3 + \mu_1 + \frac{1}{\mu_1} + \frac{1}{\lambda_1}\right]. \tag{5}$$

2.4 Benchmark Dataset

During this study, we employed different datasets in order to evaluate if different input distribution can somehow affect the learned latent space distribution.

2.4.1 MNIST

The first benchmark dataset used in this study is the well-known MNIST dataset. Training set is made up by 60K instances of 28 × 28 grayscale images. However, as shown in figure 3 the input space distribution is very uneven.

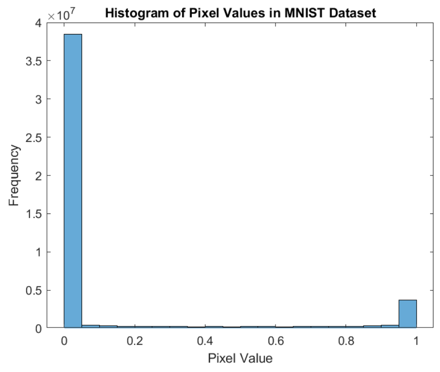


Fig. 3: MNIST intensity histogram with bin_size=20

Noised MNIST In order to make the input space evenly distributed and to make the model learn a more robust representation, we decided to augment MNIST images introducing random noise. Noise can follow Normal, LogNormal, Uniform or Inverse Gaussian distributions. Figure 4 shows the input space histogram after the noise injection.

Exponentiated MNIST Another augmentation technique employed during the experimental phase is the exponentiation of the MNIST dataset. The idea is to determine if a LogNormal latent space can be beneficial when the input space shares similar structural properties with the latent space, such as those introduced by exponentiation.

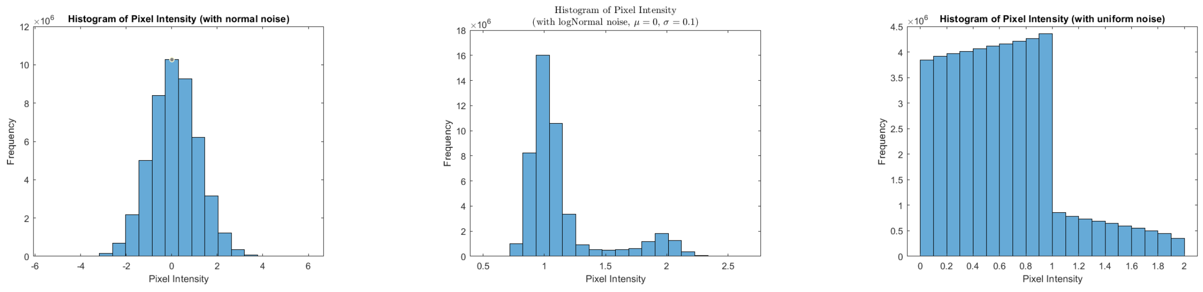


Fig. 4: Augmented MNIST intensity histogram with bin_size=20

2.4.2 Synthetic 2D Dataset

The other dataset used during the experimental phase is a synthetic dataset generated using a 2D LogNormal distribution. Points belonging to positive class (red) have been generated using $LogNormal(\mu = [0, 2], \sigma = [0.4, 0.4])$, while points belonging to negative class (blue) have been generated using $LogNormal(\mu = [2, 0], \sigma = [0.4, 0.4])$. Figure 5 shows the distribution of the generated datapoints.

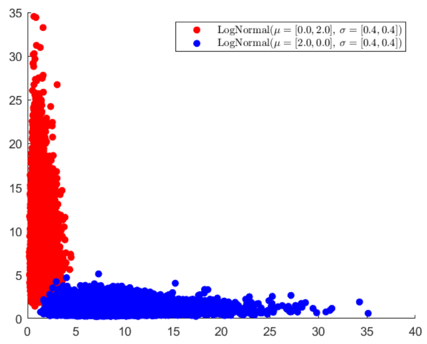


Fig. 5: Synthetic dataset following LogNormal distribution

3 Experimental Results

Before introducing the results obtained during the experimental phase, is important to assess the generated samples actually follow the reference distribution. The plots in figures 6 and 7 demonstrate how the histogram for a very high number of samples approximate respectively the LogNormal and Inverse Gaussian reference distributions. Note that the Normal distribution is not shown because is "intrinsically contained" in the LogNormal distribution.

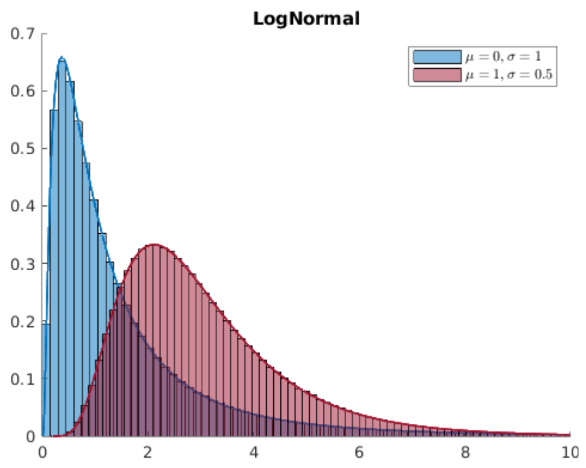


Fig. 6: LogNormal

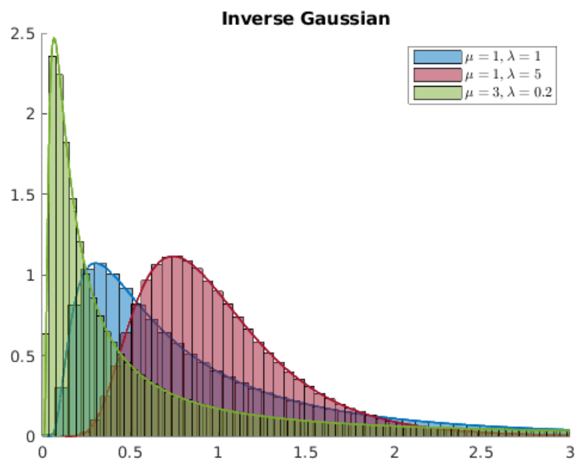


Fig. 7: Inverse Gaussian

3.1 Models Architecture

The model's architecture employed during the experimental phase is fixed, however, layer parameters have been chosen differently according to the dataset taken into account and so, its complexity. The main differences between the one used for synthetic dataset (Figure 8) and the one used for MNIST dataset (Figure 9) resides in the number of neurons per layer and in weights initialization, as explained in 3.2. For aesthetic sake, we represented all sampling layers in the same figure, but the encoder's output *follows only one flow* of the sampling layers. Moreover, even if activation functions are not reported, all of them are LeakyReLU (with $\alpha = 0.01$).

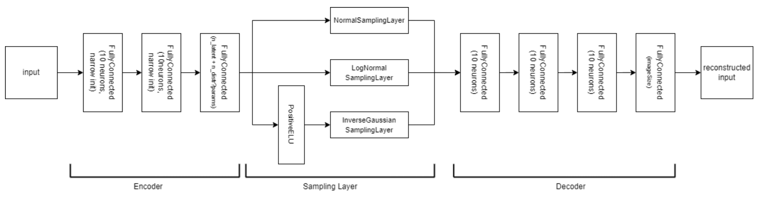

input
FullyConnected (10 neurons, narrow init)
FullyConnected (10neurons, narrow init)
FullyConnected (n_latent + n_distr_params)
NormalSamplingLayer
LogNormal SamplingLayer
PositiveELU
InverseGaussian SamplingLayer
FullyConnected (10 neurons)
FullyConnected (10 neurons)
FullyConnected (10 neurons)
FullyConnected (imageSize)
reconstructed input
Encoder
Sampling Layer
Decoder

Fig. 8: Model Architecture for Synthetic dataset

3.2 Weights Initialization

In all our experiments we used the default MATLAB weights initialization: Glorot [8]. However, we noticed that with this initialization some experiments with a LogNormal latent distribution suffered from instabilities in the training phase as we can see in figure 10a where the training could

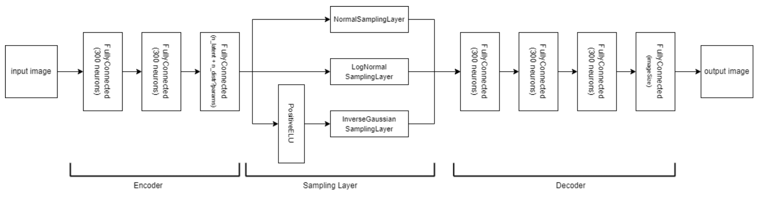
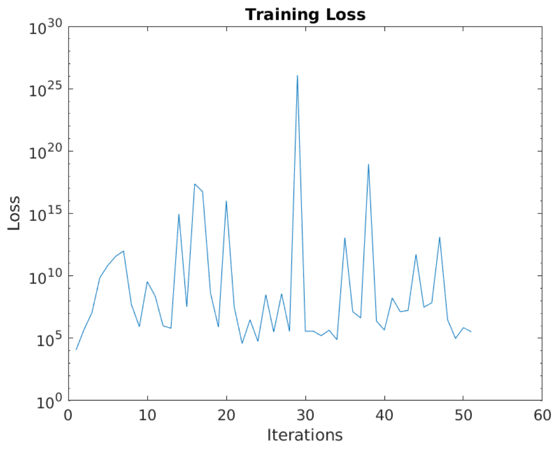
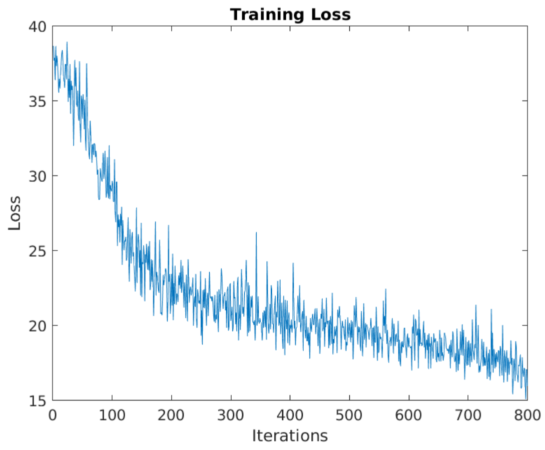

input image
FullyConnected (300 neurons)
FullyConnected (300 neurons)
FullyConnected (n_latent + n_distr*params)
NormalSamplingLayer
LogNormal SamplingLayer
PositiveELU
InverseGaussian SamplingLayer
FullyConnected (300 neurons)
FullyConnected (300 neurons)
FullyConnected (300 neurons)
FullyConnected (imageSize)
output image
Encoder
Sampling Layer
Decoder

Fig. 9: Model Architecture for MNIST dataset

not be completed due to the high values of the loss function. We think that the problem lies in the sampling operation from the latent space, since the heaviness of the LogNormal distribution's tail, determined by the variance $\mathcal{O}\big(exp(\sigma^2)\big)$, may lead to the generation of exploding features. This can be reduced with a different weights initialization, in particular by initializing the weights with smaller magnitudes. In this way, smaller weights lead to the propagation of smaller values to the sampling layer that generates smaller samples (at least in the first iterations). We see an improvement in the stability of the training as shown in figure 10b.



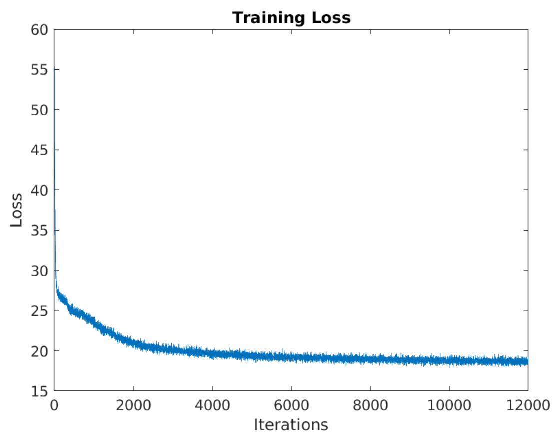
(a) Default weights initialization



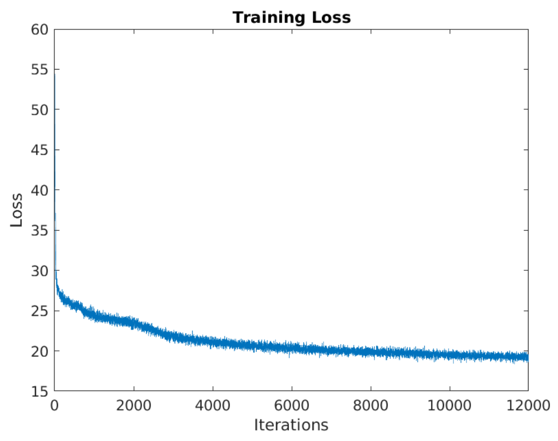
(b) $N(0, 0.01)$ weights initialization

Fig. 10: Comparison between different weights initialization on the synthetic dataset.

The default weights initialization, instead, was a good choice with a LogNormal latent space when our neural network was trained on the MNIST dataset. In this case the Glorot initialization was equivalent to a random initialization of the weights following a $N(0, 0.01)$ distribution and so the training did not suffer from instabilities as shown in figure 11. Moreover, we tried with even smaller weights but this did not led to any improvement.



(a) Default weights initialization



(b) $N(0, 0.001)$ weights initialization

Fig. 11: Comparison between different weights initialization on the MNIST dataset.

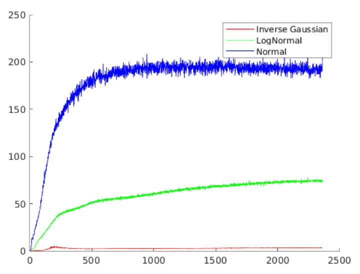
3.3 Parameters Selection

The loss function of a Variational Auto-Encoder is made up by the weighted sum of the Mean Squared Error and the Kullback-Leibler Divergency (as reported in equation 3). However, the choice of parameter β is crucial since it affects the total loss (and consequently the weight updating). So, a preliminary step before starting our analysis consists in finding a suitable β value. Table 2 shows a summary of the reconstruction error (MSE) obtained on the MNIST test set for different values of β.

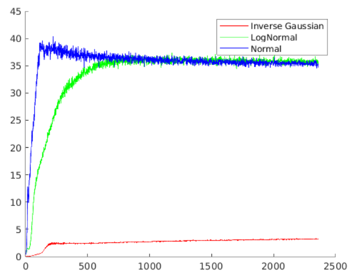
β	**Normal**	**LogNormal**	**Inverse Gaussian**
0.0	5.7492	5.9889	6.3569
0.01	5.6380	6.0242	6.3093
0.02	5.7418	6.0074	6.3297
0.03	5.8343	6.1501	6.4141
0.1	6.1855	6.5709	6.3293
0.2	6.7218	7.5128	6.3667
0.3	7.3572	8.3121	6.4305

Table 2: Reconstruction error on MNIST test set for different β values, using different latent space distributions.

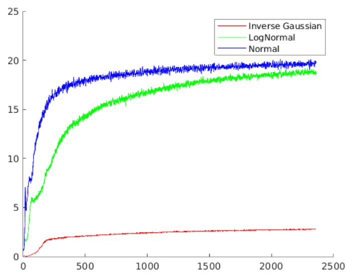
From table 2 we can observe an important behaviour of the latent spaces: using the Inverse Gaussian sampling layer, the reconstruction error for different values of β is much more stable than the one obtained using Normal or LogNormal sampling layers. Our intuition is that the learned Inverse Gaussian distribution is very close to the prior one, while learned parameters for Normal and LogNormal distributions are very different from the standard ones. This has been proved plotting the KL loss over time for all latent space distributions, figure 12. However, this preliminary analysis could be not sufficient to find the most appropriate β: in section 4.1 we discuss a the possibility of a more robust approach for setting that value.



(a) $\beta = 0$



(b) $\beta = 0.01$



(c) $\beta = 0.1$

Fig. 12: Comparison between different weights for the KL divergence. On the y-axis is reported the KL divergence between the learned latent distribution and the prior one.

Tables 3a and 3b summarize the training parameters employed respectively for synthetic and MNIST dataset. All the model performances we discuss now on, will be produced using that training parameters.

latent space	2
lr	0.0003
epochs	50
batch size	100
β	0.01

(a) Synthetic Dataset

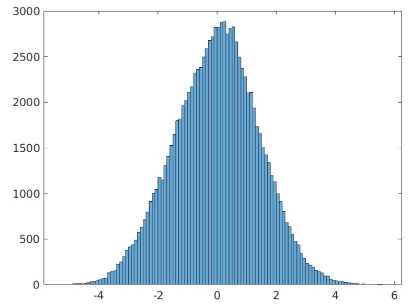
latent space	10
lr	0.0003
epochs	100
batch size	512
β	0.01

(b) MNIST Dataset

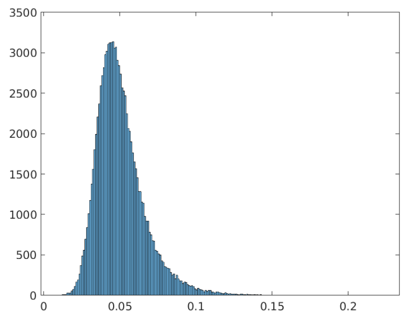
Table 3: Training Parameters

3.4 Latent space distribution

An interesting analysis performed concern studying the learned distribution of the latent space. Figures 13, 14 and 15 show the histogram of various parameters fed to the sampling layer by the encoder.

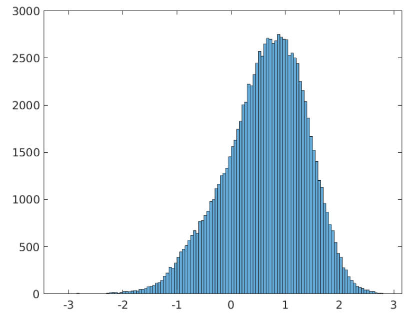


(a) Predicted μ

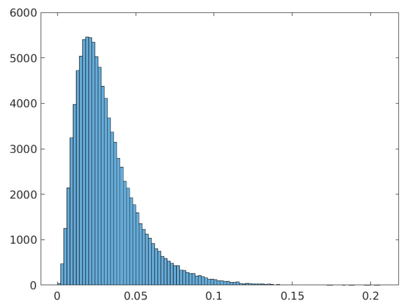


(b) Predicted σ

Fig. 13: Normal latent space distribution (experiments carried out on MNIST dataset).

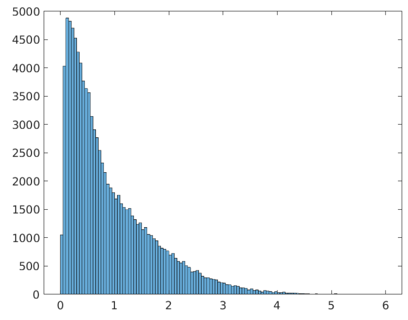


(a) Predicted μ

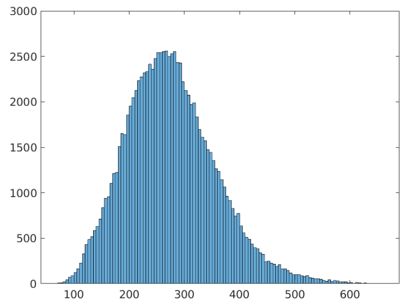


(b) Predicted σ

Fig. 14: LogNormal latent space distribution (experiments carried out on MNIST dataset).



(a) Predicted μ



(b) Predicted λ

Fig. 15: Inverse Gaussian latent space distribution (experiments carried out on MNIST dataset).

3.5 Reconstruction Capability

The first aspect we want to evaluate in Variational Auto-Encoder is the reconstruction capability. Table 4 shows a comparative analysis on the reconstruction error on test set between different learned latent spaces. The most significant aspect we can observe from the table is that, even if the synthetic dataset was generated using a LogNormal distribution, the Normal latent space is still slightly more accurate than the LogNormal latent space. Recalling that the LogNormal distribution is just an exponentiation of a Normal distribution, we think that using a Normal sampling layer, the Encoder component could have learned a representation of data points which is closer to their logarithm. Now, since data points seem to be generated from a Normal distribution, is easier for the VAE to learn a Normal latent space. Moreover, since the Normal distribution is numerically more stable than the LogNormal one, a Normal sampling layer is less prone to numerical instability. However, the reconstruction error obtained with a LogNormal sampling layer is very low, which means that the latent space have been learned successfully. On the other hand, using an Inverse Gaussian sampling layer, the reconstruction error increases a lot. This can be simply explained by the different statistical properties of Inverse Gaussian with respect to Normal or LogNormal, making the VAE very difficult to learn from this distribution.

About MNIST dataset, using a Normal sampling layer we achieve better performance thanks to its stability properties; however, LogNormal and Inverse Gaussian sampling layers are able to learn a good representation as well.

In order to exploit the LogNormal latent space capabilities, we ”forced” MNIST dataset to behave similar to the LogNormal distribution through element-wise exponentiation. From this augmentation we tried to prove that, in this case, the error obtained using a LogNormal sampling layer can be comparable to the one obtained using a Normal sampling layer on MNIST (or at least is better than the one obtained using a Normal sampling layer on the same dataset). Unfortunately, even in this case a LogNormal sampling layer performs worse than using a Normal sampling layer. The reason of this behaviour can be addressed, as for the synthetic dataset, to the stability of the Normal distribution.

Dataset	**Normal**	**LogNormal**	**Inverse Gaussian**
synthetic	0.0128	0.0327	0.1945
MNIST (fig. 16)	5.6380	6.0242	6.3093
MNIST-exp (fig. 17)	18.9708	21.029	22.1914

Table 4: Reconstruction error on test set

3.6 Denoising Capability

Another characteristic we want to evaluate in Variation Auto-Encoders is *denoising*. This section aims to provide an analysis on model's capability to remove random noise injected into MNIST images, employing two different augmentation techniques.

Noise Injection The first technique concern in injecting noise in an element-wise manner into the original image, i.e. $\tilde{x} = x + \varepsilon$, where x is the current image pixel, ε is a random noise drawn from Normal, LogNormal, Uniform or Inverse Gaussian distribution, and $\tilde{x}$ is the augmented version of the pixel. Table 5 summarizes the reconstruction error for different latent space distribution. Figures 18, 19, 20 and 21 help to visualize the denoising processes after injecting respectively Normal, LogNormal, Uniform and Inverse Gaussian noise into images.

As shown in table 5, a very interesting behaviour comes from employing an Inverse Gaussian sampling layer which demonstrate an astonishing capability of denoising in almost every type of noises. On the other hand, as expected, a Normal sampling layer shows better capabilities in denoising when injecting a normal noise.

Noise Type	**Normal**	**LogNormal**	**Inverse Gaussian**
Normal (fig. 18)	7.9923	8.1758	8.3661
LogNormal (fig. 19)	25.2696	29.9268	24.5316
Uniform (fig. 20)	16.5163	24.0331	8.7814
Inverse Gaussian (fig. 21)	26.3575	32.5612	20.6736

Table 5: Reconstruction error summary

Stacking Channels The second augmentation technique employed concerns in stacking m channels made by random (normal) noise to the original image. Doing so is possible to assess the denoising capability of the model, i.e. evaluating if the model is able to discard the information contained in the m channels, while generating a good enough reconstruction. When applying this augmentation technique, can be useful to take a closer look to the training approach, which is slightly different from the standard one. Even if the input image is a 3D image (width × height × channels), we're interested only in learning the representation embedded in the first channel. In order to do so, the decoder will return a 2D image (the first channel of the reconstruction) and, instead of computing the Mean Squared Error between two 3D images, it computes the Mean Squared Error between the first channel of the input image and the reconstructed image (which are both 2D images). Figures 22, 23, 24 and 25 help to visualize the denoising process, showing the first channel of the input image and its respective 2D reconstruction. Table 6 summarizes the reconstruction error for different latent space distribution.

Noise Type	Normal	LogNormal	Inverse Gaussian
Normal	6.1330	6.4419	6.6628
LogNormal	6.2493	6.5027	6.7255
Uniform	6.3211	6.5725	6.8283
Inverse Gaussian	6.4158	6.7951	6.7700

Table 6: Reconstruction error summary

3.7 Generation Capability

The last characteristic we want to evaluate in Variational Auto-Encoder is the generation capability of new images. So, once the latent space distribution is learned, we can sample from it to generate new instances and feeding these samples to the decoder network. The generation process concerns sampling from a distribution which parameters are given by the average of the learned distribution parameters on the test set, for each class. Figures 26 and 27 shows generated digits with Normal, LogNormal and Inverse Gaussian sampling layers.

4 Conclusions

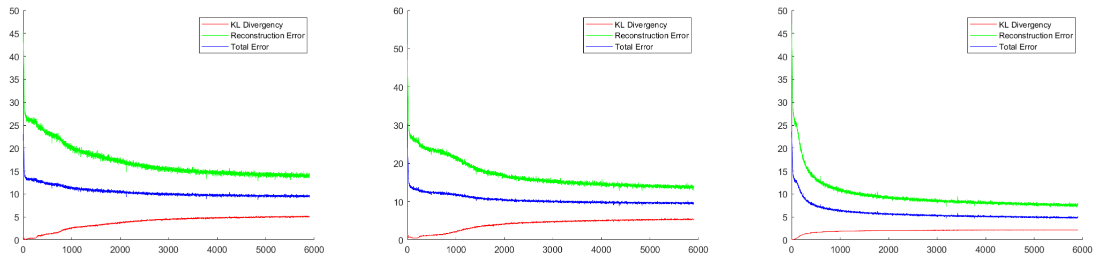
In this study, we explored the effectiveness of employing heavy-tailed distributions to represent the latent space in Variational Autoencoders. Our focus was on comparing Normal, LogNormal, and Inverse Gaussian distributions in terms of

- Reconstruction Capability: as reported in section 3.5, using a Normal latent space distribution we achieve the lowest reconstruction errors, even if the input space follows a LogNormal distribution (as in the case of the synthetic dataset).
- Denoising Capability: as reported in section 3.6, the Inverse Gaussian latent space demonstrate incredible performances in removing nose much better than its counterparts on LogNormal, Uniform and Inverse Gaussian noise type. On the other hand, if employing normal noise, with a Normal latent space distribution we achieved better performances.
- Generation Capability: generation's quality cannot be measured by rigorous metrics, so we evaluate it empirically through the generated images (fig. 26, 27, 28). In this case all different sampling layers show similar generation capabilities.

4.1 Future Works

This section explores potential directions and enhancements for advancing this research. It aims to identify limitations and propose innovative approaches to address them, encouraging further investigation and development.

Different KL Weight In section 3.3, we perform a preliminary analysis for choosing a suitable weight β for the KL loss in the total loss. So, our approach was based on comparing the reconstruction error on the test set and choosing the value of β which minimizes the error. However, a more accurate analysis shows us that the loss components affect the total loss in a different way we thought, as shown in figure 29. Therefore, to improve the quality of the overall model, a good idea could be to give a different weight to the KL loss element, letting it, for example, $\approx$ 3 times more important.



(a) Normal Latent Space (b) LogNormal Latent Space (c) Inv.Gaussian Latent Space

Fig. 29: How different loss components affects the total loss. In the plots, both components have the same weight.

Latent Representation Analysis Another interesting analysis that can be conduct concerns in analyzing the latent representations generated by the encoder on a LogNormal input space (like the synthetic dataset). Precisely, analyze the correlations (if any) between the μ and σ generated by a VAE with a Normal sampling layer and the ones generated by a VAE with a LogNormal sampling layer. If this correlation exists, it proves that using a Normal latent space is better than using a LogNormal one, even if the input space distribution follows a LogNormal distribution.

Ad-Hoc Loss Function One of the main goal of this work was to study the use of a heavy tail latent space in order to better catch the behaviour of outliers. A different approach to force the encoder to learn a more robust representation of the outliers would be to use a different loss function instead of the standard mean squared error employed in this work. Specific loss functions could be used, for example we think that a loss function that gives greater importance to outliers would help the encoder to learn better.

Forcing heavier tail latent distribution Another interesting approach to check whether an heavy tail latent space could be more suitable in case of heavy tailed distributed data would be to modify the KL divergence in the loss function in order to "force" the learned distribution's tail to be heavier. For example, an interesting experiment would be to generate a synthetic dataset as we have done in section 2.4.2 following an Inverse Gaussian with a small λ (e.g. $\lambda = 0.01$, the smaller the λ the heavier the tail) and computing the KL divergence (see equation (5)) from a prior Inverse Gaussian with the same λ used to generate the data.

References

[1] M. Dogariu, L.-D. Stefan, B. A. Boteanu, C. Lamba, B. Kim, and B. Ionescu, "Generation of realistic synthetic financial time-series," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 18, pp. 1 – 27, 2022.

[2] F. Liang, L. Hodgkinson, and M. W. Mahoney, "Fat-tailed variational inference with anisotropic tail adaptive flows," 2022.

[3] M. Cococcioni, F. Fiorini, and M. Pagano, "Modelling heavy tailed phenomena using a lognormal distribution having a numerically verifiable infinite variance," *Mathematics*, vol. 11, p. 1758, 04 2023.

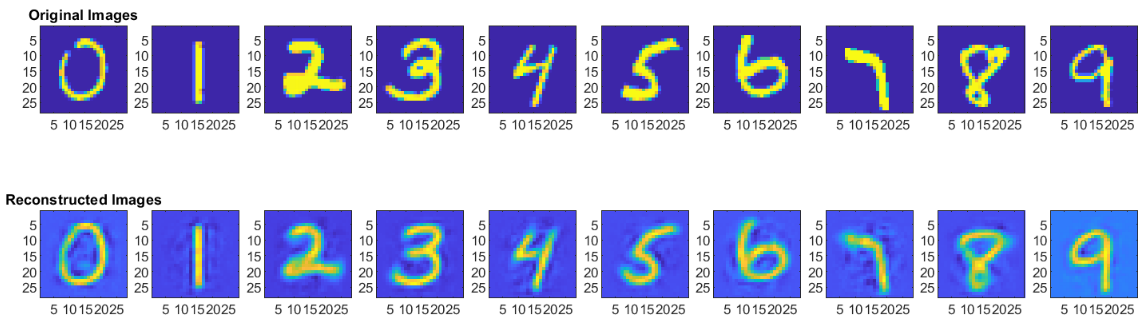
[4] K. R. Chen, D. Svoboda, and K. P. Nelson, "Use of student's t-distribution for the latent layer in a coupled variational autoencoder," 2020.

[5] J. R. Michael, W. R. Schucany, and R. W. Haas, "Generating random variates using transformations with multiple roots," *The American Statistician*, vol. 30, no. 2, pp. 88–90, 1976.

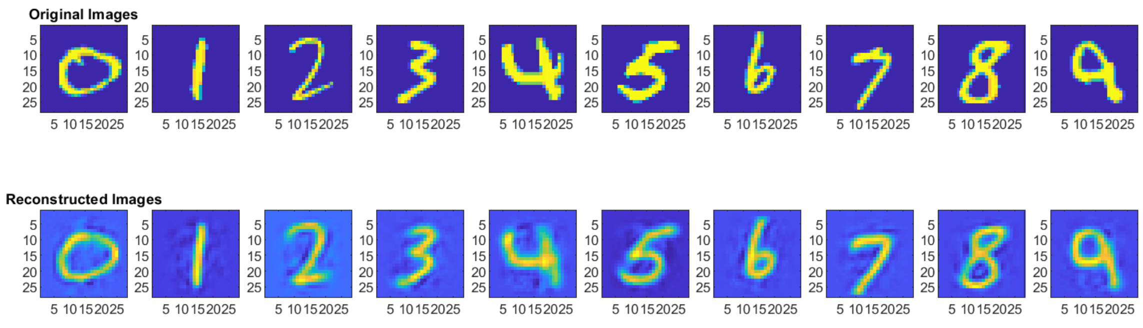
[6] Y. Zhang, J. Pan, L. K. Li, W. Liu, Z. Chen, X. Liu, and J. Wang, "On the properties of kullback-leibler divergence between multivariate gaussian distributions," in *Advances in Neural Information Processing Systems* (A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, eds.), vol. 36, pp. 58152–58165, Curran Associates, Inc., 2023.

[7] O. Bahadir and U. Cako, "On information geometry of the inverse gaussian distribution," 09 2021.

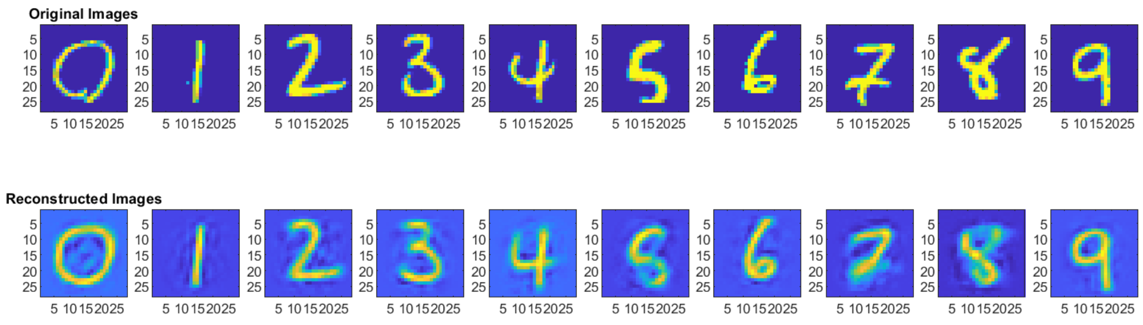
[8] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2010.



(a) Reconstruction using Normal latent space

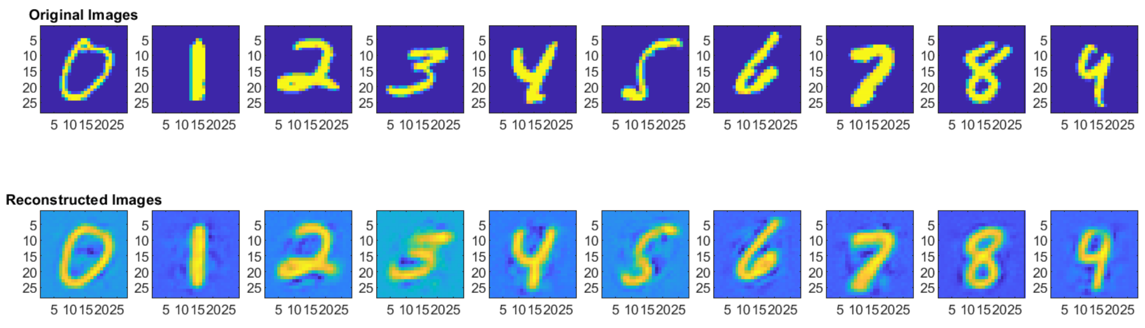


(b) Reconstruction using LogNormal latent space

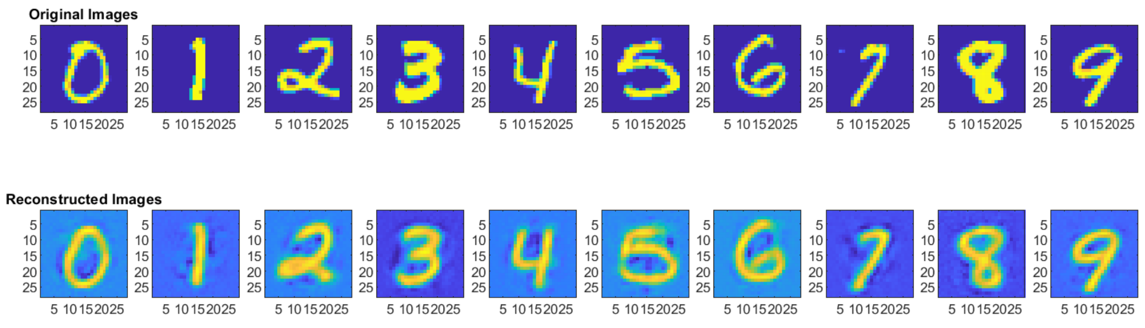


(c) Reconstruction using Inverse Gaussian latent space

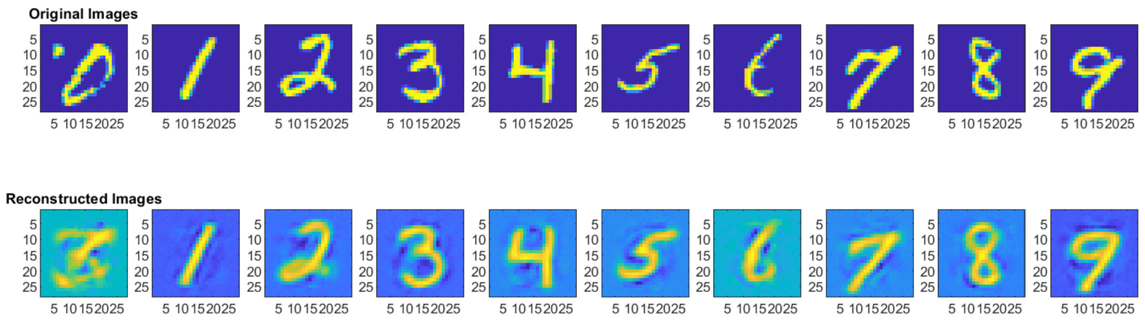
Fig. 16: Original and Reconstructed images using different latent space distributions on MNIST dataset.



(a) Reconstruction using Normal latent space

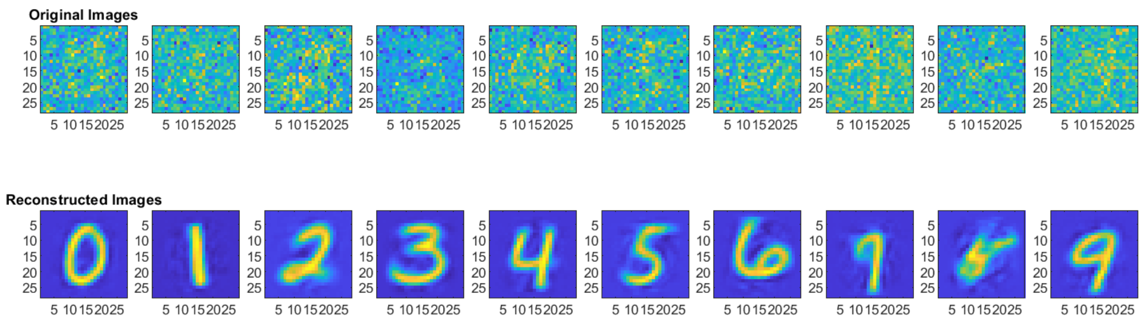


(b) Reconstruction using LogNormal latent space

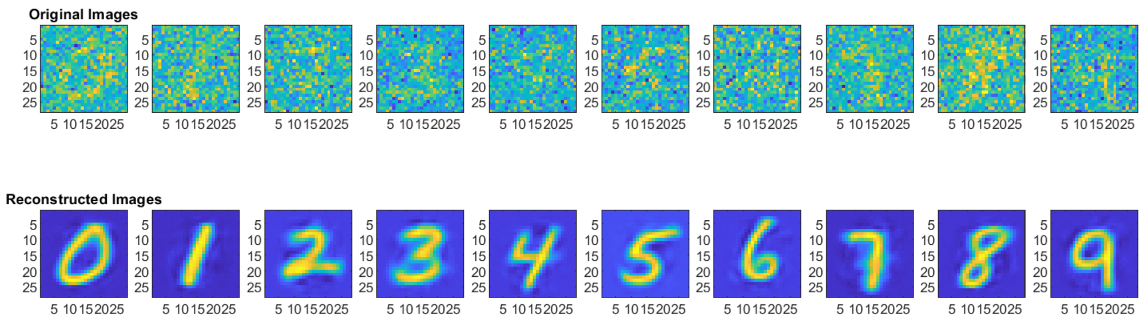


(c) Reconstruction using Inverse Gaussian latent space

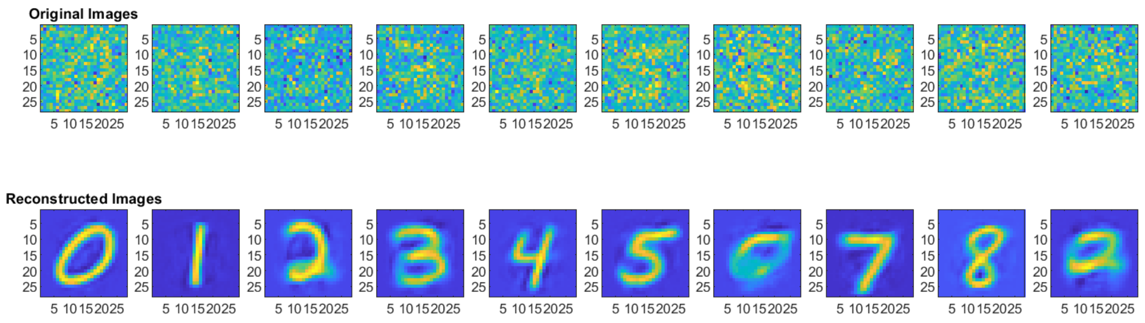
Fig. 17: Original and Reconstructed images using different latent space distributions on MNIST exponentiated.



(a) Denoising using Normal latent space

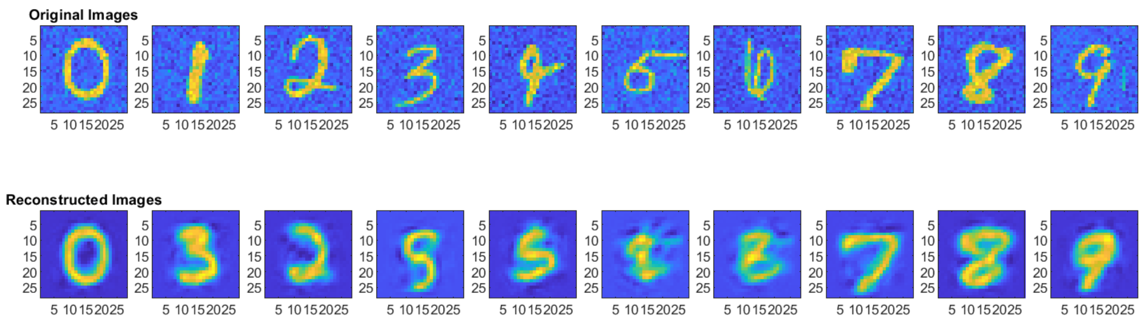


(b) Denoising using LogNormal latent space

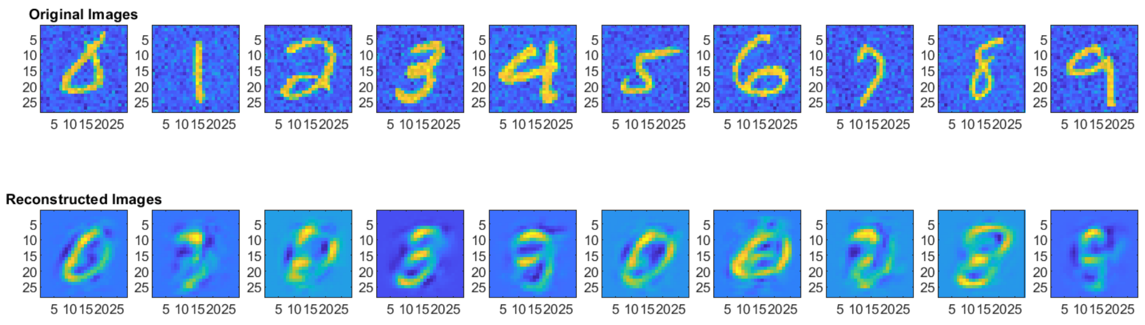


(c) Denoising using InverseGaussian latent space

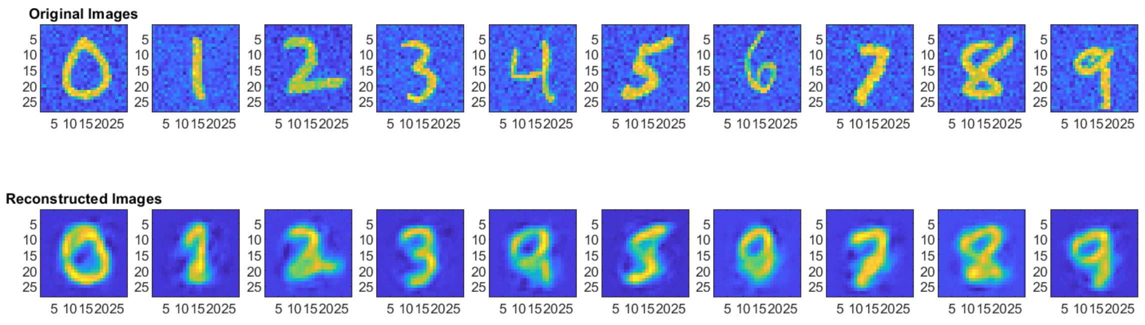
Fig. 18: Comparison between using different latent space distributions for removing noise generated using Normal distribution.



(a) Denoising using Normal latent space

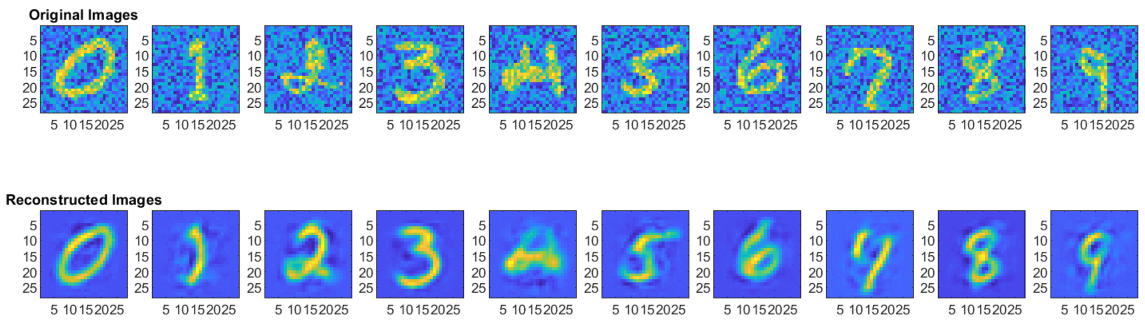


(b) Denoising using LogNormal latent space

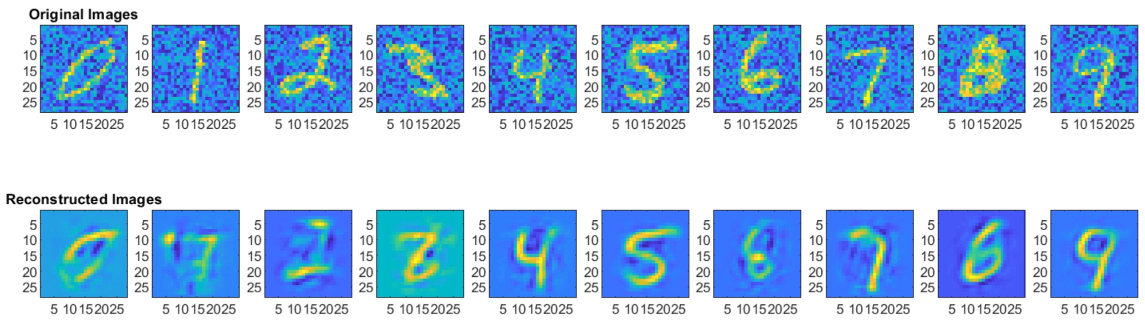


(c) Denoising using InverseGaussian latent space

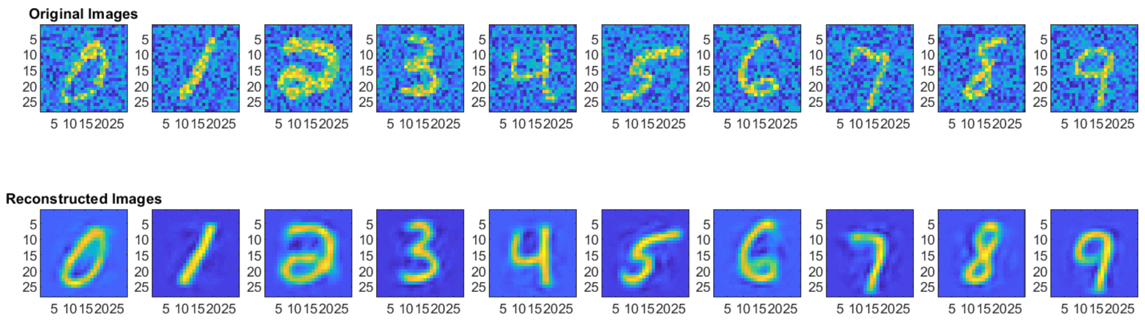
Fig. 19: Comparison between using different latent space distributions for removing noise generated using LogNormal distribution.



(a) Denoising using Normal latent space

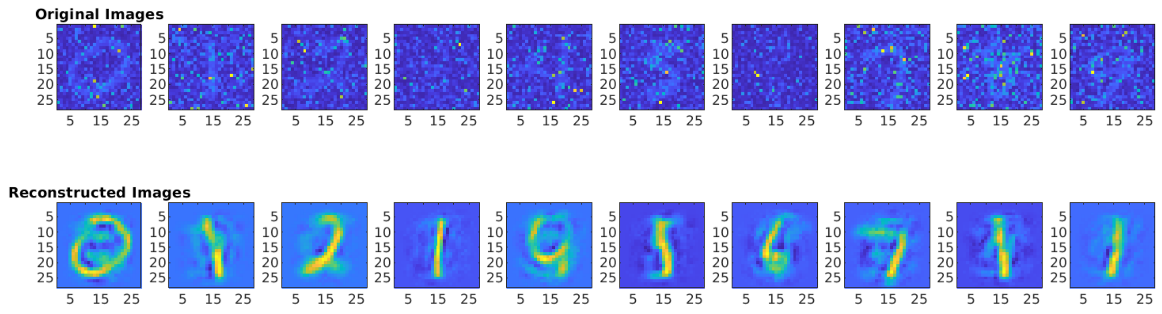


(b) Denoising using LogNormal latent space

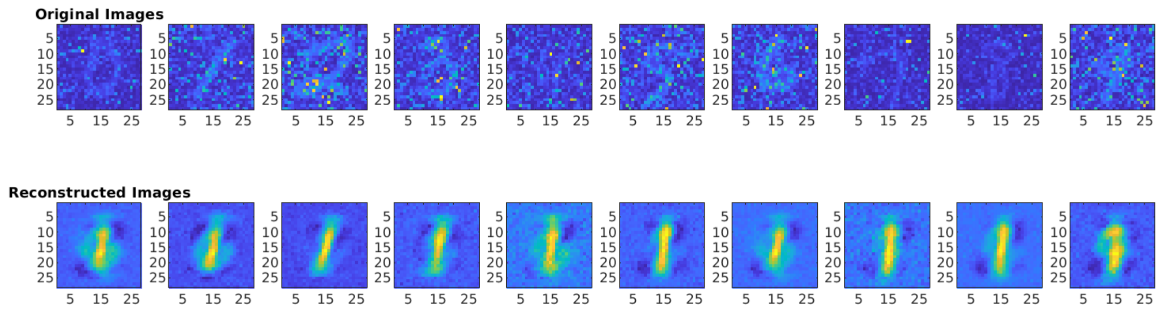


(c) Denoising using InverseGaussian latent space

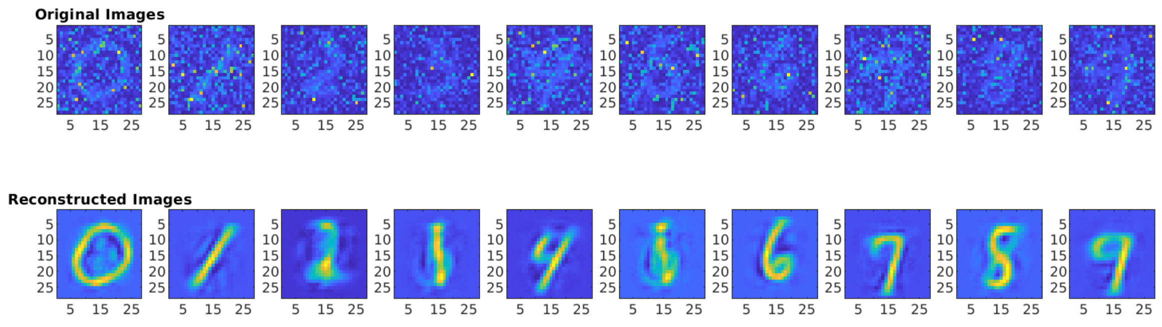
Fig. 20: Comparison between using different latent space distributions for removing noise generated using Uniform distribution.



(a) Denoising using Normal latent space

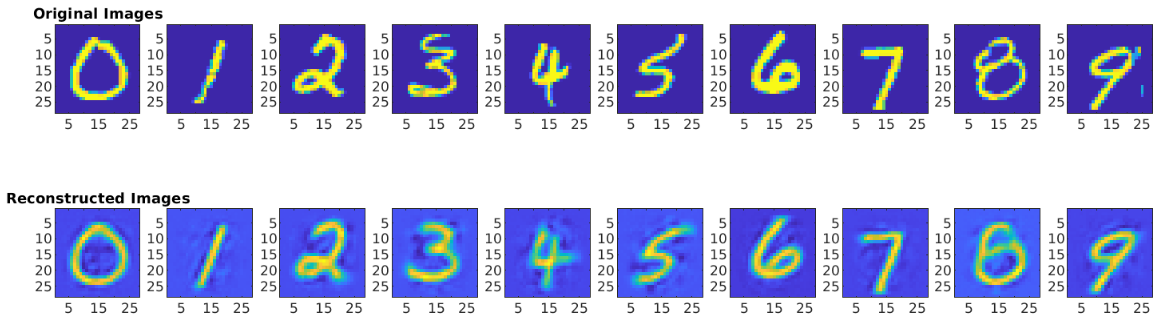


(b) Denoising using LogNormal latent space

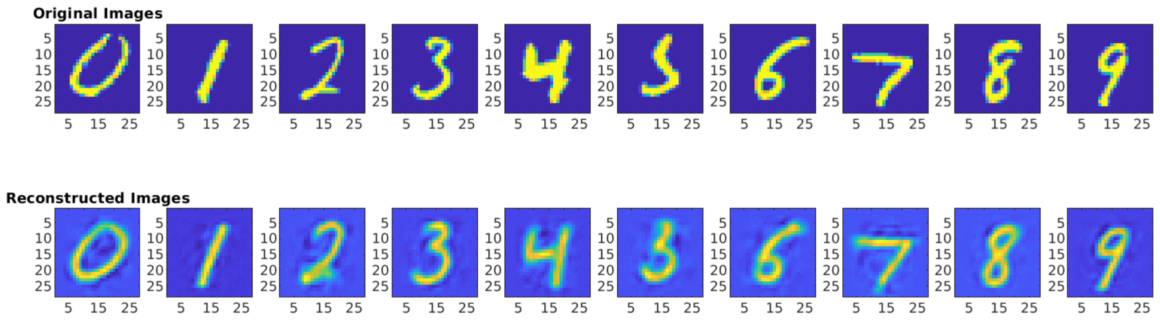


(c) Denoising using InverseGaussian latent space

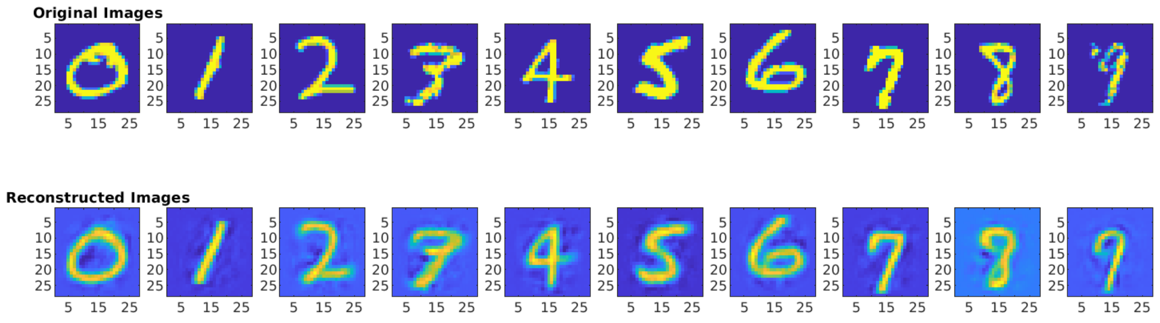
Fig. 21: Comparison between using different latent space distributions for removing noise generated using Inverse Gaussian distribution.



(a) Denoising using Normal latent space

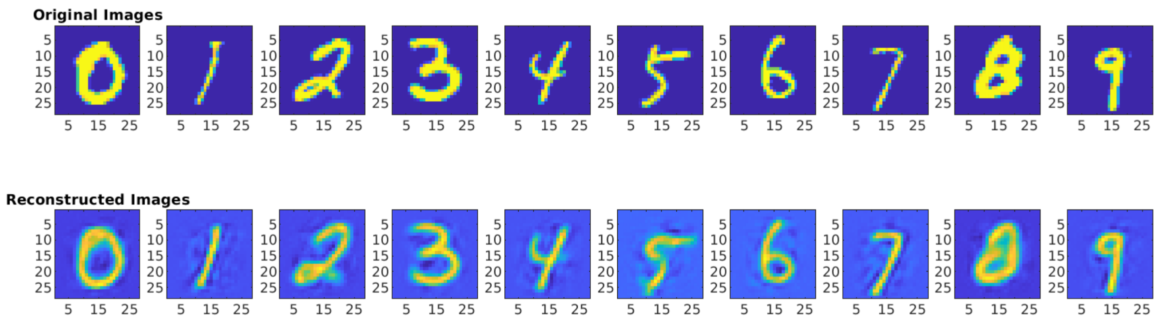


(b) Denoising using LogNormal latent space

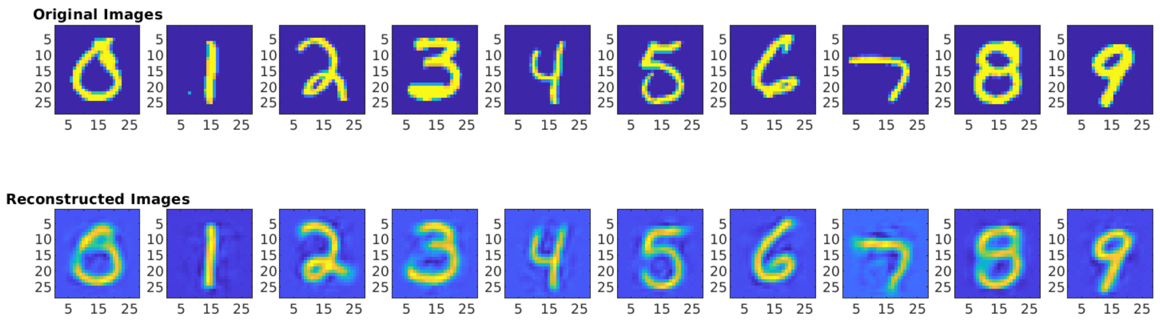


(c) Denoising using InverseGaussian latent space

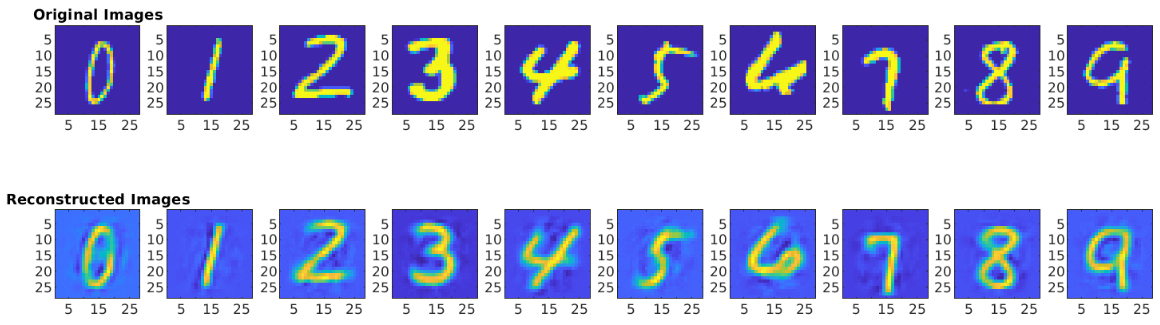
Fig. 22: Comparison between using different latent space distributions for removing noise generated using Normal distribution as additional channels for MNIST dataset (only the first channel is shown).



(a) Denoising using Normal latent space

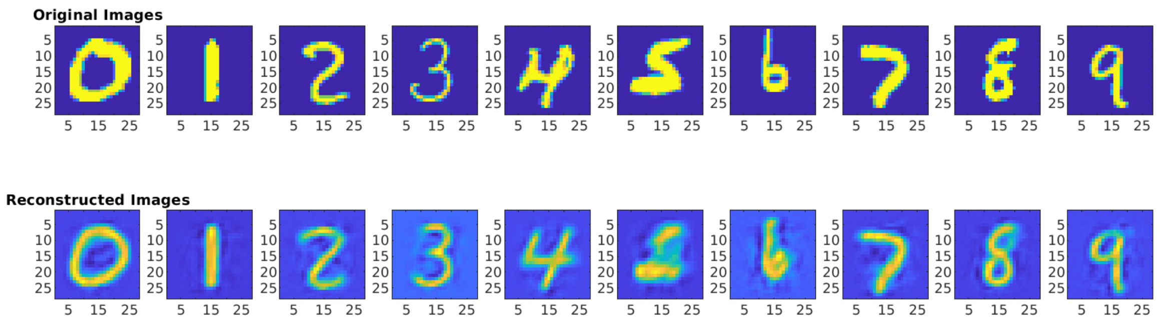


(b) Denoising using LogNormal latent space

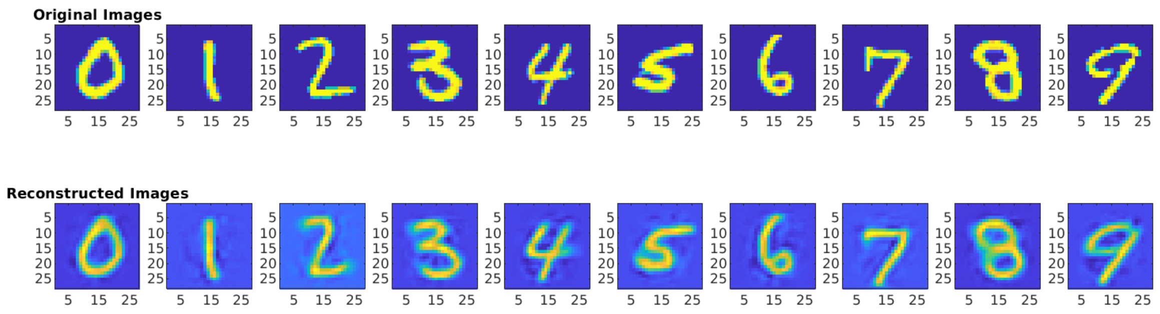


(c) Denoising using InverseGaussian latent space

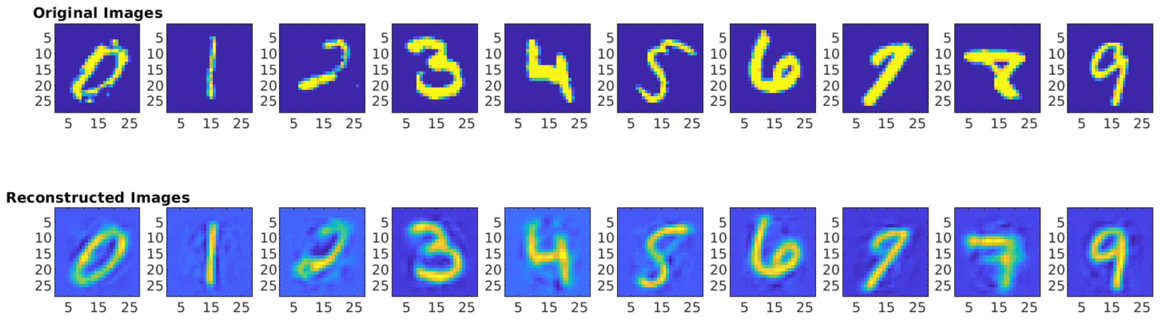
Fig. 23: Comparison between using different latent space distributions for removing noise generated using LogNormal distribution as additional channels for MNIST dataset (only the first channel is shown).



(a) Denoising using Normal latent space

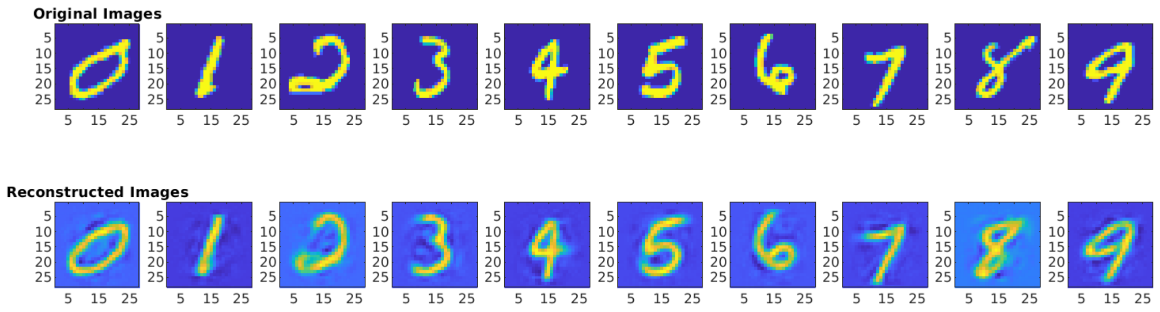


(b) Denoising using LogNormal latent space

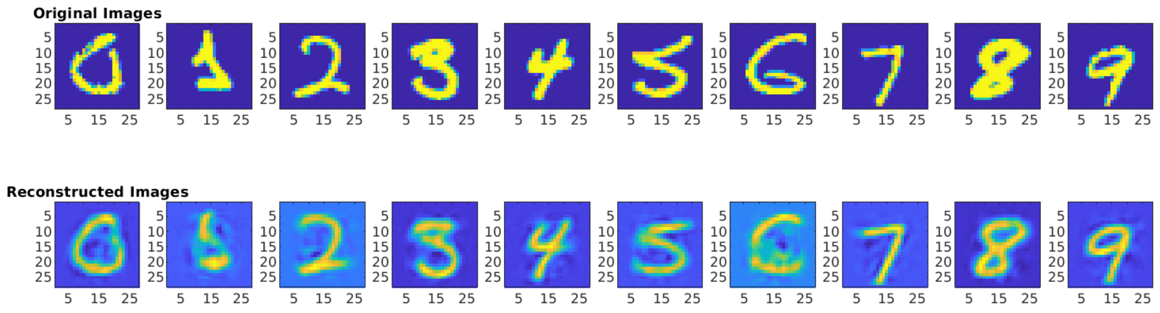


(c) Denoising using InverseGaussian latent space

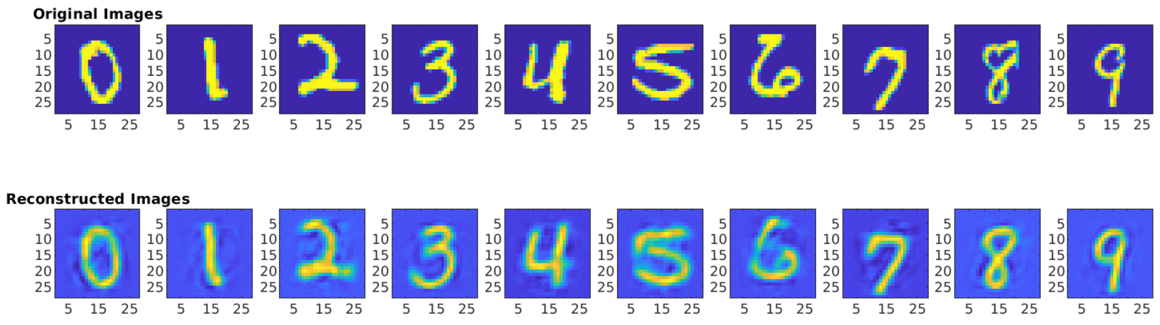
Fig. 24: Comparison between using different latent space distributions for removing noise generated using Uniform distribution as additional channels for MNIST dataset (only the first channel is shown).



(a) Denoising using Normal latent space

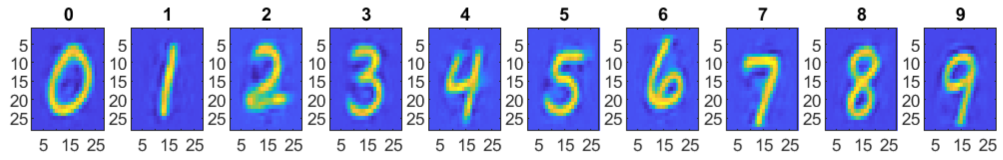


(b) Denoising using LogNormal latent space

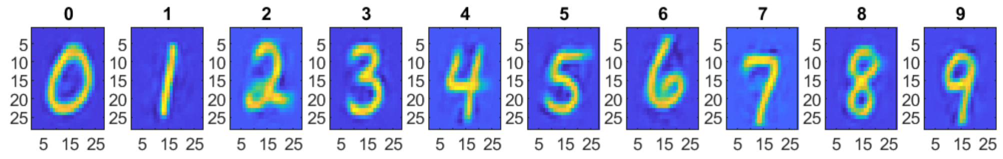


(c) Denoising using InverseGaussian latent space

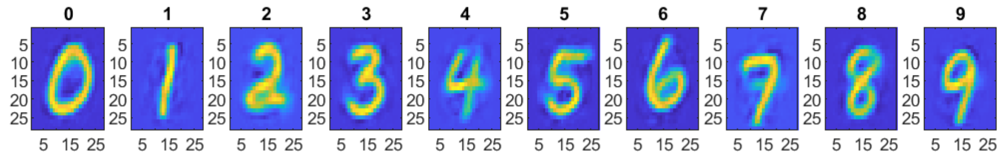
Fig. 25: Comparison between using different latent space distributions for removing noise generated using Inverse Gaussian distribution as additional channels for MNIST dataset (only the first channel is shown).



(a) Generation using Normal latent space

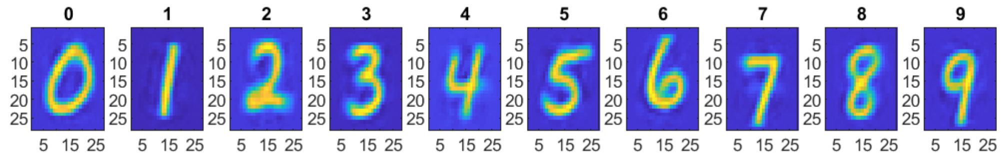


(b) Generation using LogNormal latent space

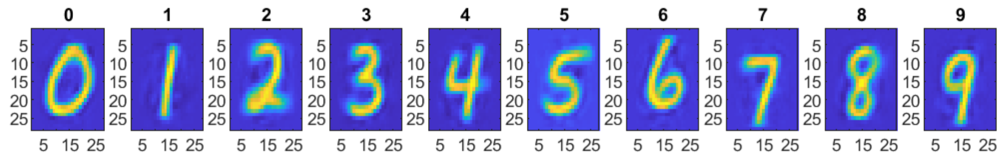


(c) Generation using InverseGaussian latent space

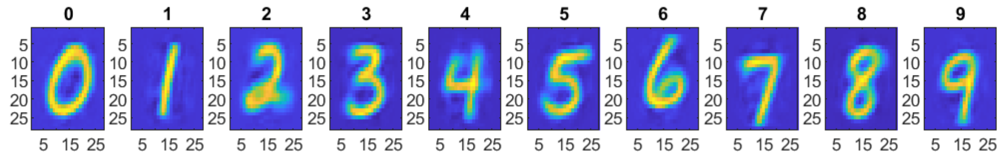
Fig. 26: Generated samples using different latent spaces.



(a) Generation using Normal latent space

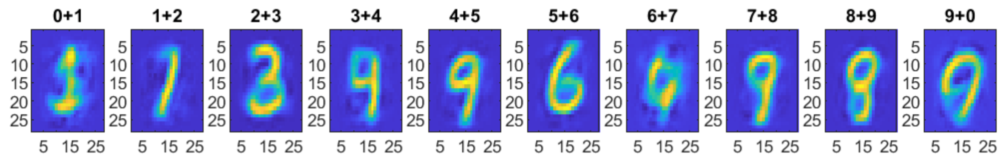


(b) Generation using LogNormal latent space

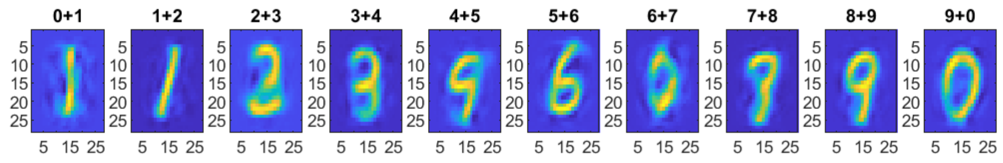


(c) Generation using InverseGaussian latent space

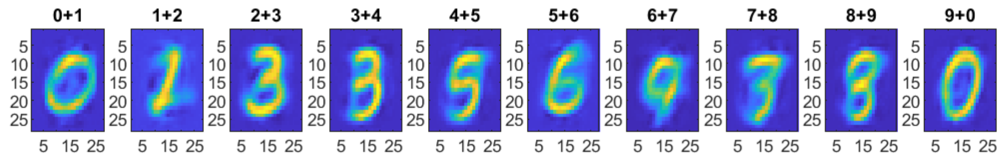
Fig. 27: Generated samples using different latent spaces using a model learned with noised dataset (normal noise).



(a) Generation using Normal latent space



(b) Generation using LogNormal latent space



(c) Generation using InverseGaussian latent space

Fig. 28: Generation of new samples by combining two different embeddings and using a model learned with noised dataset (normal noise).