Inverting MNIST Neural Networks for Image Generation

Eric Guo, Yihao Wang
January 21, 2026
Dr. Yilmaz
Quarter 2 Project

***Abstract*—We trained three types of models, Base, Small, and NoActivations, all of which consist of only Linear and LeakyReLU connections to produce embeddings of MNIST images that appear similar to random samples. We inverted the models and attempted to generate new images by creating true random samples and passing them into the inverted model. We found that we were unable to generate new images, that we were able to reconstruct images from embeddings using the inverted model, and that the inverted models are highly sensitive to small perturbation.**

I. Introduction

In the past few years, image generative AI models have taken off in both popularity and performance. There are now commercially available applications able to generate entire videos from a prompt. However, these methods are very computationally expensive, and are not intuitively simple.

Instead, we want to use already well studied yet powerful models, specifically neural networks (NNs) and use them in a different way to generate images. We will train NNs to classify images from the MNIST dataset, then "invert" the models so that they take in labels as inputs and images as outputs, resulting in a generative AI model.

II. Related Work

Previous researchers have successfully created generative image models through other methods. Two methods are relevant to our invertable neural network: Autoencoders, Variational Autoencoders (VAEs), and Denoising Diffusion Models.

The purpose of traditional autoencoders, and thus also VAEs, is to create lower-dimensional representations, or "embeddings", of higher-dimensional data, such as images [1]. The embeddings can then be used to generate new images. For example, we can slightly perturb the embedding of an existing image, then "decode" the embedding back into the higher-dimensional image and get a new, generated image as a result.

The architecture of autoencoders is similar to symmetrical. An image enters the model through the "encoder" side, where the high dimensional input is operated on (e.g. through convolutions or neural networks) until it is shrunk down into a lower dimensional vector at the middle of the model. The lower dimensional vector is then further operated on by the "decoder" side, which increases the dimension of the vector until it is the same size as the input. The learning target of the model is to output an image that is the same as the input [1]. Since the encoding process intentionally destroys information by lowering dimension, if the model achieves the learning target, then it must be that less but enough information is being retained so that an image can be reconstructed. The middle vector after the encoder and before the decoder serves as the embedding for a particular image, and embeddings can be encoded (i.e. created) and decoded by splitting the trained model in half by its encoding and decoding portions. Autoencoders are essentially creating labels, or embeddings, of images without requiring the data be labeled initially.

Variational Autoencoders (VAEs) expand on this concept of traditional autoencoders. They were first proposed by Kingma and Welling [2] in 2013. In traditional autoencoders, only a single embedding exists for any given image. On the other hand, in VAEs, the embedding for any given image is not an embedding but a probabilistic distribution of probable embeddings. The encoder of VAEs outputs a series of means and variances, which represents a multivariate Gaussian distribution. An embedding is then found by sampling from the distribution represented by the means and variances [3].

This non-deterministic property of VAEs has several advantages. First, it helps to prevent overfitting. The decoder can no longer "memorize" the images and associated embeddings in the dataset because it will always see new, random embeddings. Second, it improves the stability of the generation process. In traditional autoencoders, small perturbations of the embedding may cause the embedding vector to fall outside of the "domain" of coherent images, and so decoding the resulting vector may not produce a recognizable image. With VAEs, the model is forced to learn a larger space of coherent images, as the embeddings will inherently vary [3].

Another relevant class of models are Diffusion Models. These models are intentionally designed for image generation, and diffusion underlies the state-of-the-art in image generation, such as Stable Diffusion. At a high level, Diffusion Models take a training image and slowly add noise, and then a model learns to identify and reverse the noise from the noisy image. After the model is trained, new images are generated by sampling completely random noise. The model then slowly removes noise, eventually resulting in a coherent image [4]. Our approach is similar to Diffusion Models in that our goal is to also generate images from sampled noise. However, the process from noise to image is different from diffusion.

III. Dataset and Features

We used the MNIST dataset from the torchvision Python library. Each instance is a 28x28 black and white image, and the labels were integers from 0-9 representing what digit the image represented. We turned the images into binary vectors of 10 dimensions, where the value corresponding to the correct digit was a 1 and the rest were 0s.

IV. Methods

A. Basic Theory

A very basic neural network involves multiple layers that can be represented as the equation:

$$XW + B = Y$$

Where X is the input vector, W is the weights matrix, B is the bias vector, and Y is output vector. If W is a square matrix, then it follows that we can find some inverse W-1 where WW-1=I, the identity matrix. From this, we can obtain the input vector from the output vector:

$$
\begin{aligned}
XW + B &= Y \\
(XW + B) - B &= Y - B \\
((XW + B) - B)\, W^{-1} &= (Y - B)W^{-1} \\
XWW^{-1} &= (Y - B)W^{-1} \\
X &= (Y - B)W^{-1}
\end{aligned}
$$

If our input X is an image, and Y is some embedding representation of the image, then ideally by only training an "encoder" neural network that encodes X to Y we would also get the decoder that maps Y to X for "free", without any additional training. We can then apply the same processes used on autoencoders to generate new images. This is advantageous because it is very easy to create the encoder part, i.e. something that maps images X to embeddings (or labels) Y, but not easy to explicitly train something that maps Y to X.

This approach does constrain our neural network. First, the weights matrices must all be square matrices, otherwise the inverse cannot be calculated. Second, any activation functions used in the model must also be invertible, i.e. injective. So, for example, we cannot use ReLU as an activation function, but instead we can use LeakyReLU.

B. Implemented Approach

We trained three models: a Base model, a Small model, and a NoActivations model. The Base model takes in a 28x28 tensor of an image from the MNIST dataset, and applies a trainable linear transformation of the form $XW + B = Y$ followed by a LeakyReLU layer. This "block" of linear and then activation is then repeated three more times, for a total of four blocks. The Small model is similar to the Base model, in that it has only one block of linear followed by activation. The NoActivations model has four linear transformations in series. All models also begin with a flatten transform that turns the 28×28 tensor into a 1×784 vector and end with an unflatten transform that converts the vector back into a 28×28 tensor. As mentioned above, all our weights matrices must be square, so the size of the input and output vector for all linear transforms are the same (1×784) size.

All the models are intended to output a 28×28 tensor that should appear to be sampled from a target Gaussian distribution. The target Gaussian distribution depends on the label assigned to the input image: the mean of the Gaussian distribution is 2 times the label, and the variance is one. For example, if an image of 4 from the MNIST dataset is passed into any of the models, we expect the output to be similar to a 28×28 tensor of random numbers sampled from a Gaussian distribution of mean $2 \times 4 = 8$ and variance 1.

To achieve this target, our loss function consists of two parts. For each sample, we first compute the negative log likelihood (NLL) of the output being sampled from the target distribution. Then, we compute the variance of the sample and take the squared error of the variance from 1. The final loss of each sample is the sum of the NLL and variance error of the sample. Since we are training in batches, the loss of an entire batch is the mean of the final loss of each sample.

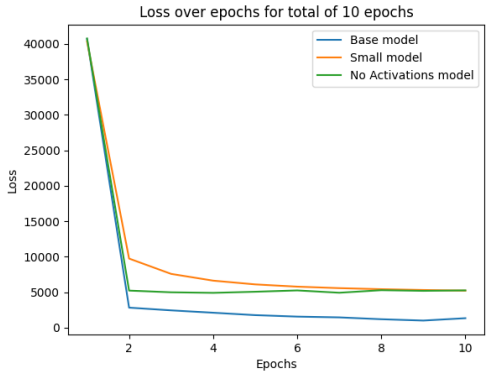

Fig. 1. Loss over epochs for 10-total-epochs run

For each output from the model N, its values $n \in N$, the total number of values $|N|$, and the label a for the input, the loss of N is

$$
\frac{|N| \ln(2\pi)}{2} + \frac{|N|}{2} \sum (n - 2a)^2 + (\text{Variance}\,(N) - 1)^2
$$

where the first two terms are the negative loss likelihood and the third term is the variance error.

The negative loss likelihood component is mathematically equivalent to the probability the model's output comes from the target distribution, so this component of the loss ensures the output is similar to a noise sample. The variance component is also include to ensure the outputs do not collapse to only repeating the mean of the target distribution, i.e. to make sure the outputs actually vary.

We trained the models in three runs: after 10 epochs, after 50 epochs, and after 100 epochs, 500 epochs. We used the Adam optimizer to train our models.

To generate new images, we invert the forward models. Then, we sample random noise from the target distribution and then pass the random noise into the inverted models.

V. Results

Figures 1, 2, 3, and 4 shows the loss of each model over epochs for the 10, 50, 100, and 500 epoch runs. In all four runs, the loss quickly decreases and stabilizes after less than 3 epochs. We found that continuing training beyond this point was detrimental to model performance.

At 10 epochs, all three models (Base, Small, and NoActivations) were all able to convert an input image into some embedding matrix, and then decode the embedding back into the original image. Figure 5 shows the original image, the output of the forward model, and the reconstructed image after the inverted model in the first, second, and third columns respectively for all three models.

At 50 epochs, all three models were again able to reconstruct the image, and actually performs better than after 10

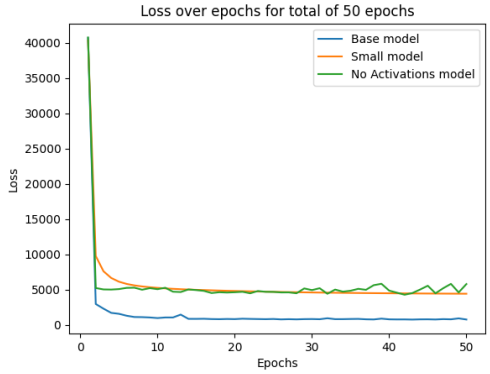

Fig. 2. Loss over epochs for 10-total-epochs run

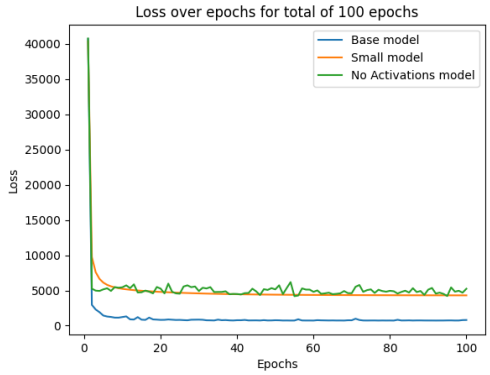

Fig. 3. Loss over epochs for 10-total-epochs run

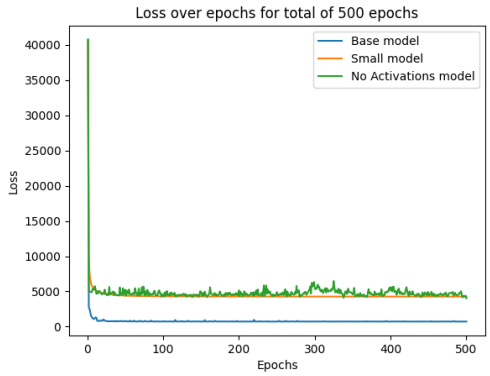

Fig. 4. Loss over epochs for 10-total-epochs run

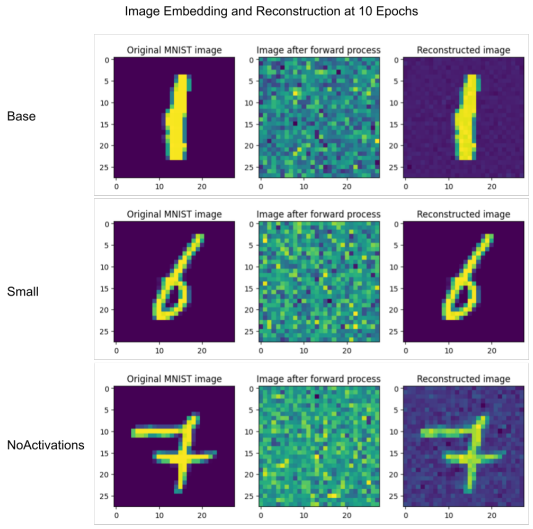

Fig. 5. Image reconstruction attempts after 10 epochs.

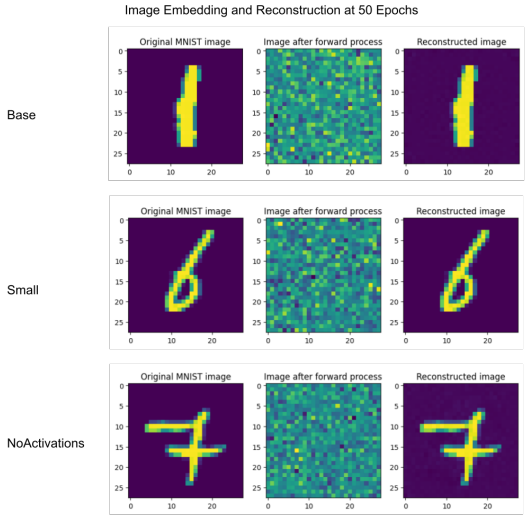

Fig. 6. Image reconstruction attempts after 50 epochs.

epochs. Notice that there is less noise in the reconstructed image as compared to the 10 epochs. Figure 6 shows the results.

However, at 100 epochs the Base model fails to reconstruct the image, and at 500 epochs, both the Base and the NoActivations model fail to reconstruct the image. The Small model continues to be able to reconstruct a coherent image, but there are artifacts in the reconstructed image that are much more apparent than at 10 and 50 epochs. Figures 7 and 8 show the results at 100 and 500 epochs, respectively.

For all models at all epochs, we failed to generate coherent new numbers. Figure 9 shows a representative example of all three inverted models after 50 epochs of training. A random 28×28 tensor was sampled from a Gaussian distribution with mean 4 and variance 1 was passed into the inverted model, and the output was plotted. A different noise sample was used

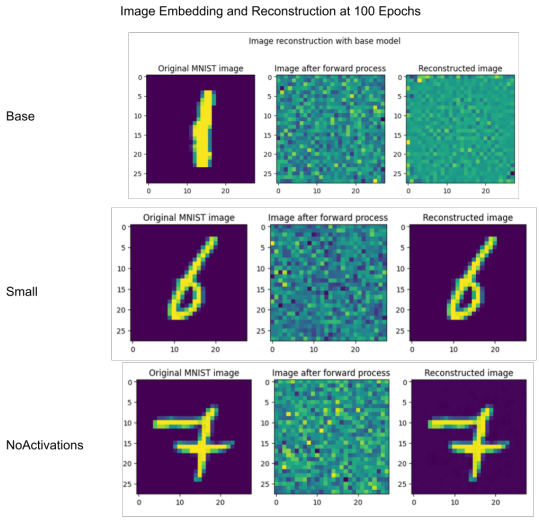

Fig. 7. Image reconstruction attempts after 100 epochs.

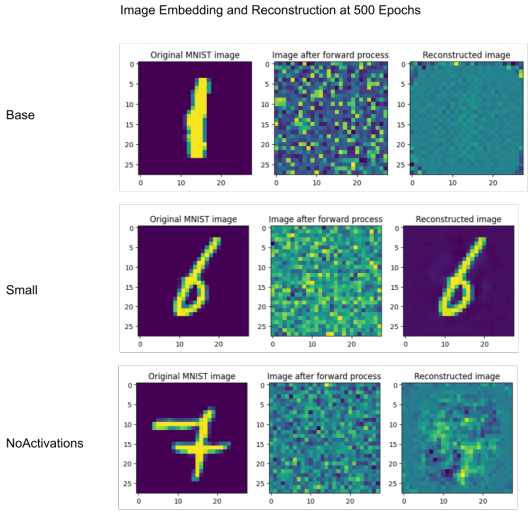

Fig. 8. Image reconstruction attempts after 500 epochs.

for each model. It is interesting to note that the inverted Small model appears to generate a much less noisy image from the noise, and the NoActivations model also generates an output with less uniform values (i.e. more peaks and troughs) than the Base model.

As also mentioned in the Methods section, the intention for the forward version of all the models is to output a tensor that appears similar to a random sample from a target Gaussian distribution. This implies that the distribution of this Y tensor should be similar to a Gaussian distribution. We tested this by inputting an MNIST image shown in Figure 10 into all the models and plotting the distribution of the outputs as compared to the Target Distribution of mean 14 and variance 1. Figures 11, 12, and 13 show the distribution density plots. In all models for all epochs, the output distributions did not match the target distribution. We also tested the distribution of the outputs on

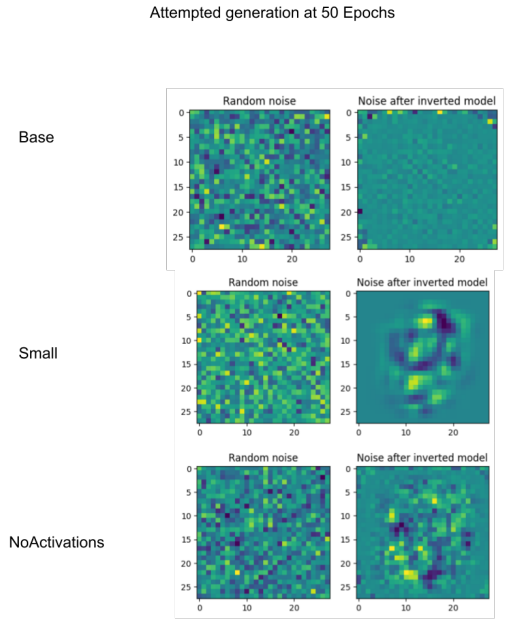

Fig. 9. Attempts to generate new number images after 50 epochs of training.

different input images, and found that the output distributions varied significantly.

VI. Discussion

From the results, we hypothesize various causes to the problems observed and suggest solutions to mitigate these issues.

First, we observed that in all models, the loss of the epochs plateaued after only 2 epochs. Any training beyond 2 epochs did not significantly improve the models, and, moreover, the model began to decrease in performance and suffer from mode collapse. We hypothesize the forward model learned to output the same tensor regardless of the MNIST input image, and that this output satisfied the loss function better than the intended output of a random-like sample. We hypothesize this "mode-collapse" problem is also responsible for why at higher epochs, the complex models are unable to even reconstruct images, let alone generate new ones; because the forward process only outputs a single tensor/mode, the inverse process would likewise also only result in a single tensor.

Second, we observed that the inverse models were highly sensitive to noise. If the image after the forward process was only changed by an imperceptible amount (less than 0.001) and passed back through the inverse model, the inverse model would not be able to reconstruct the image. This made generating new images even more difficult, as the inverse model would be too sensitive to the noise used for image generation and would not converge to coherent images.

We hypothesize that both the depth of the models and the LeakyReLU activation functions are responsible for this sensitivity. From Figure 9, we note that the most complex Base model had the least coherent generated image, the

NoActivations model had more distinct features, and the Small model had the most coherent features. The depth of the NoActivations and Base models likely compounds errors in the embedding, random-like inverse-input tensor, and the high slope of InverseLeakyReLU adds to this sensitivity. We note that since the Small model has less depth and less InverseLeakyReLU layers, it has a more coherent generated output, which matches our hypothesis.

Finally, we note that the distribution of the outputs of all forward models do not match the target distribution. We hypothesize two causes. One, the models are not sufficiently complex enough to learn a mapping from an image to a Gaussian distribution. In analogous Diffusion Models, which also generates images from noise, the noise is slowly added or removed over many steps, in contrast to our method, which noises or de-noises the image in one step. Two, the loss function we used is not a good measure of how well the output matches the target distribution. Instead, another loss function, such as KL-Divergence, can be used.

VII. Conclusion

Our experiments have shown that it is infeasible to produce image generators by inverting basic neural networks. The constraints of the neural network and the inherent sensitivity make inverted neural networks unattractive as image generators. Future research in image generation should focus on architectures that slowly produce a coherent image from noise and that are inherently probabilistic, and should use a more sufficient loss metric, such as KL-Divergence.

VIII. Contributions

Yihao Wang wrote the code that trained the forward models and produced the slideshow. Eric Guo wrote the code to invert the forward models and ran experiments and figures.

References

[1] ErSE 222 Machine Learning in Geoscience Course, "Dimensionality reduction — autoencoders," https://dig-kaust.github.io/MLgeoscience/lectures/13$_aimred/autoencoders$, 2025, ac $2026-01-20$.

[2] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," arXiv preprint arXiv:1312.6114, 2013.

[3] ErSE 222 Machine Learning in Geoscience Course, "Generative modelling and variational autoencoders," https://dig-kaust.github.io/MLgeoscience/lectures/14$_vae/$, 2025, $accessed$: $2026-01-20$.

[4] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," arXiv preprint arXiv:2006.11239, 2020, accessed: 2026-01-20. [Online]. Available: https://arxiv.org/abs/2006.11239

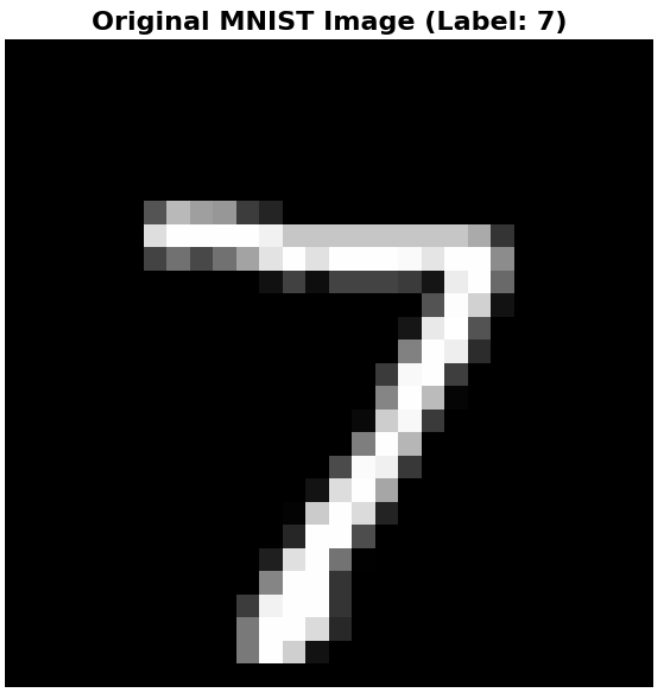

Fig. 10. Input image used for distribution experiments

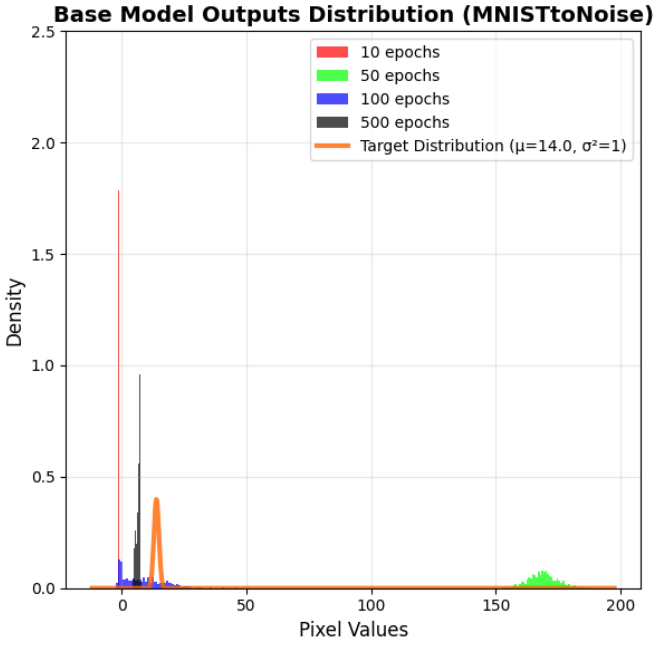

Fig. 11. Distributions of outputs from Base models after various epochs.

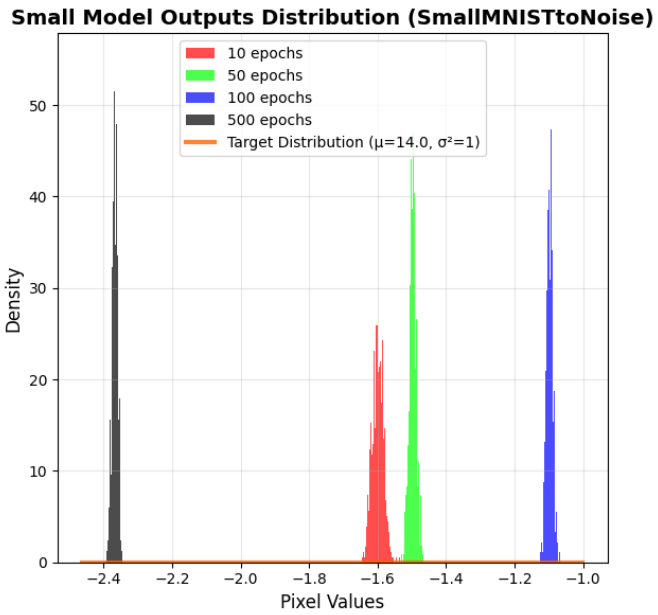

Fig. 12. Distributions of outputs from Small models after various epochs.

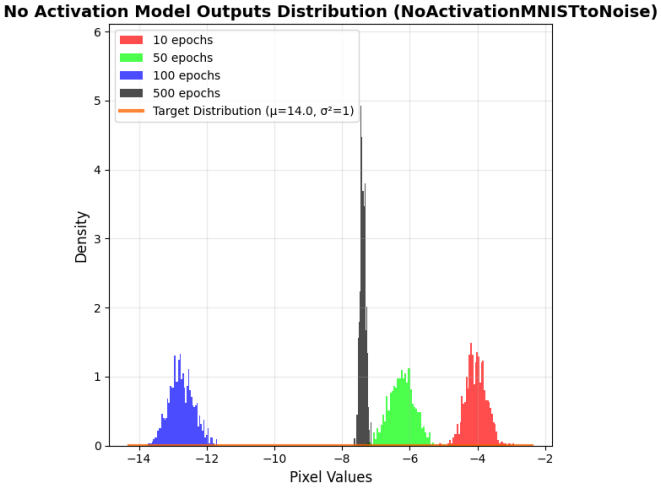

Fig. 13. Distributions of outputs from NoActivations models after various epochs.