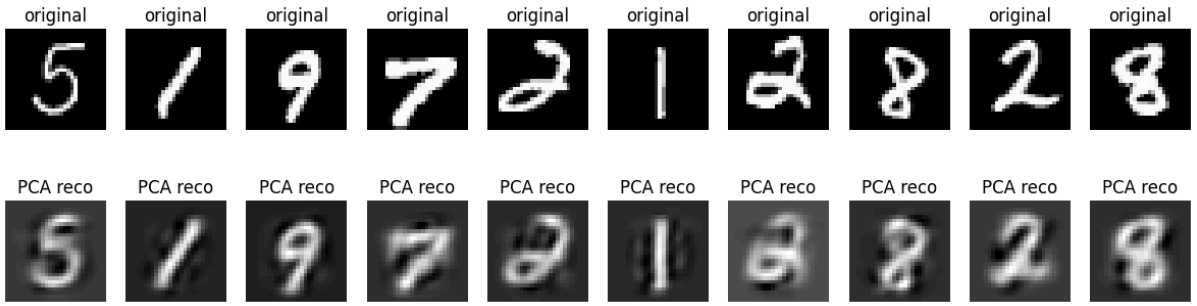
1. COMPARING PCA AND AUTOENCODERS

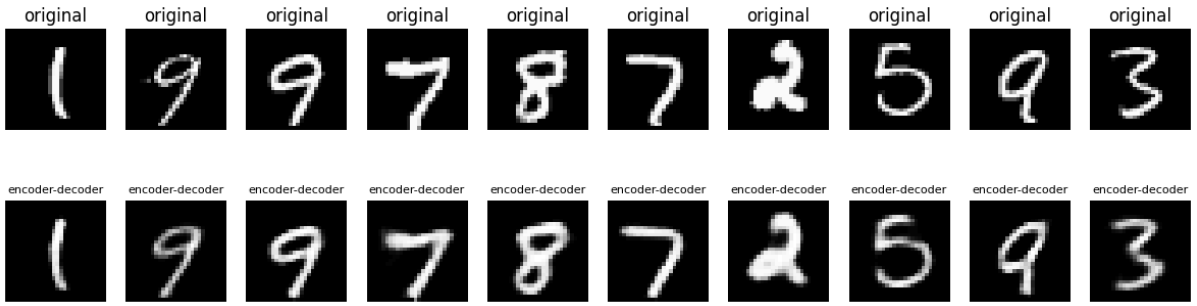
- **Linear vs. Non-linear:** This is the most important difference. PCA is restricted to linear transformations. An AE, with its activation functions (like ReLU), can learn complex, curved, non-linear manifolds. MNIST digits don't just vary along straight lines, so an AE is expected to create a more meaningful and efficient compression.

- **Expected Outcome:** As expected the **Autoencoder has a lower reconstruction error** (higher accuracy) than PCA. Its non-linear nature allows it to capture the underlying structure of the handwritten digits more effectively in 30 dimensions. The reconstructed digits from the AE should look clearer and more "digit-like" than those from PCA.

PCA OUTPUT



AUTO ENCODER OUTPUT

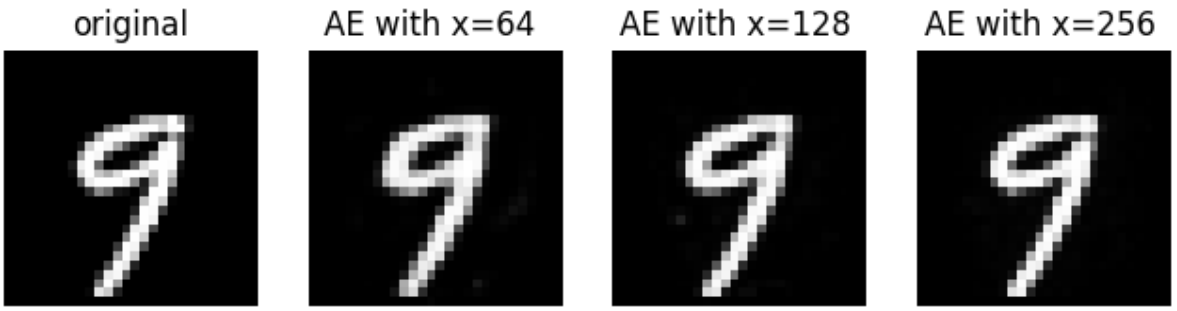



```
MSE of PCA : 0.018516837608128108
MSE of AE : 0.010623048257548362
```

Clearly AE outperforms

2. STANDARD AUTOENCODER

2.1

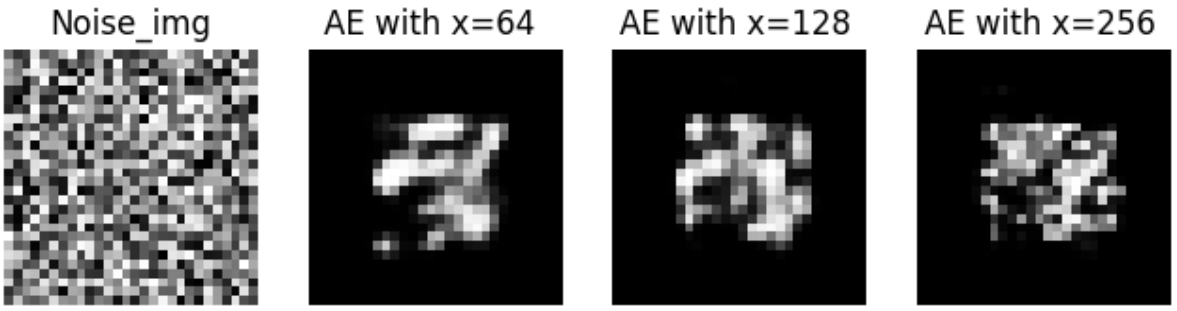


Quality of reconstruction is proportional to the number of neurons in hidden layer x.

X	MSE error
64	0.0048
128	0.0019
256	0.0011

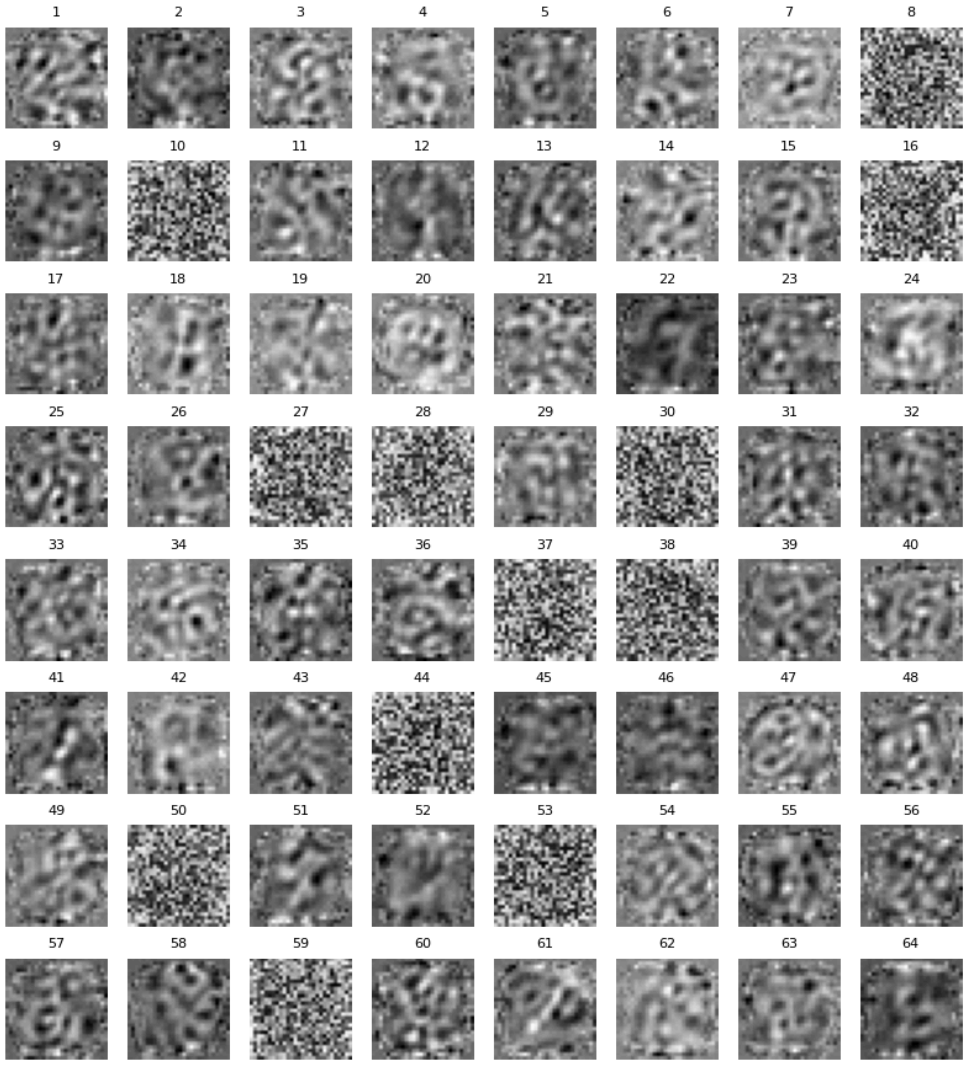
Clearly x=256 outperforms

2.2

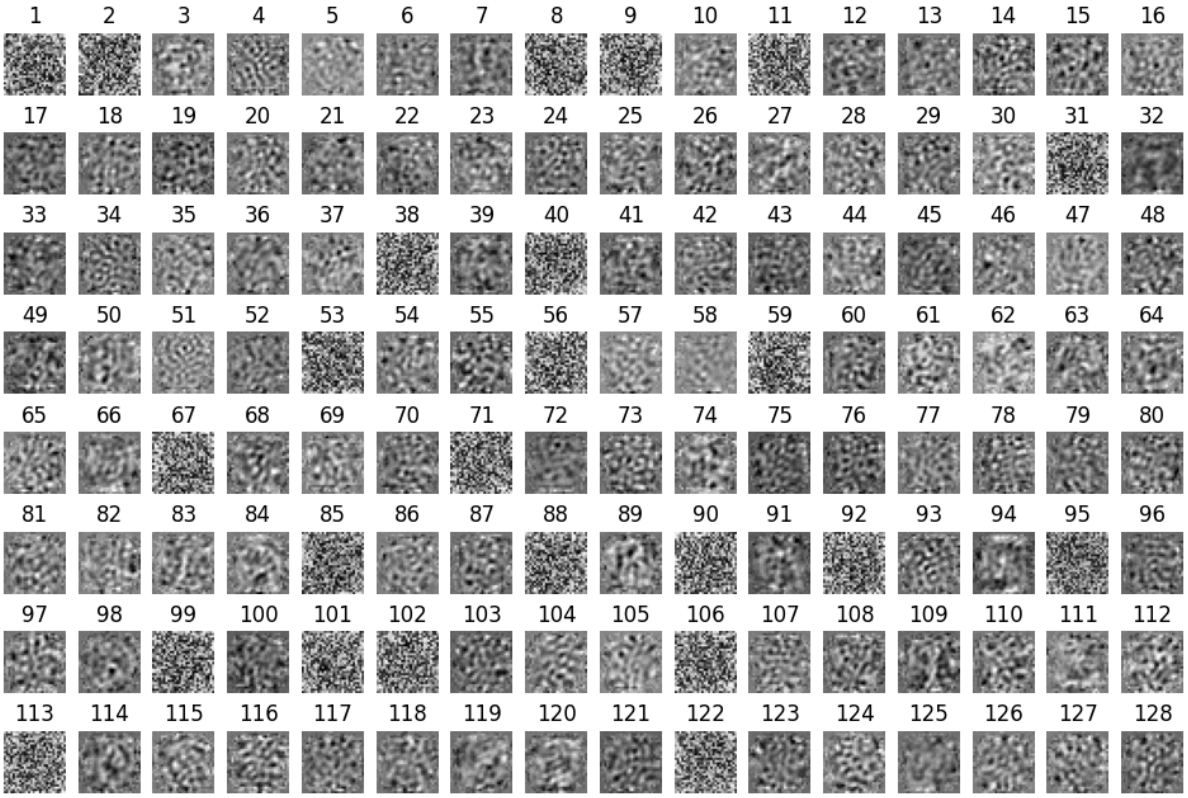


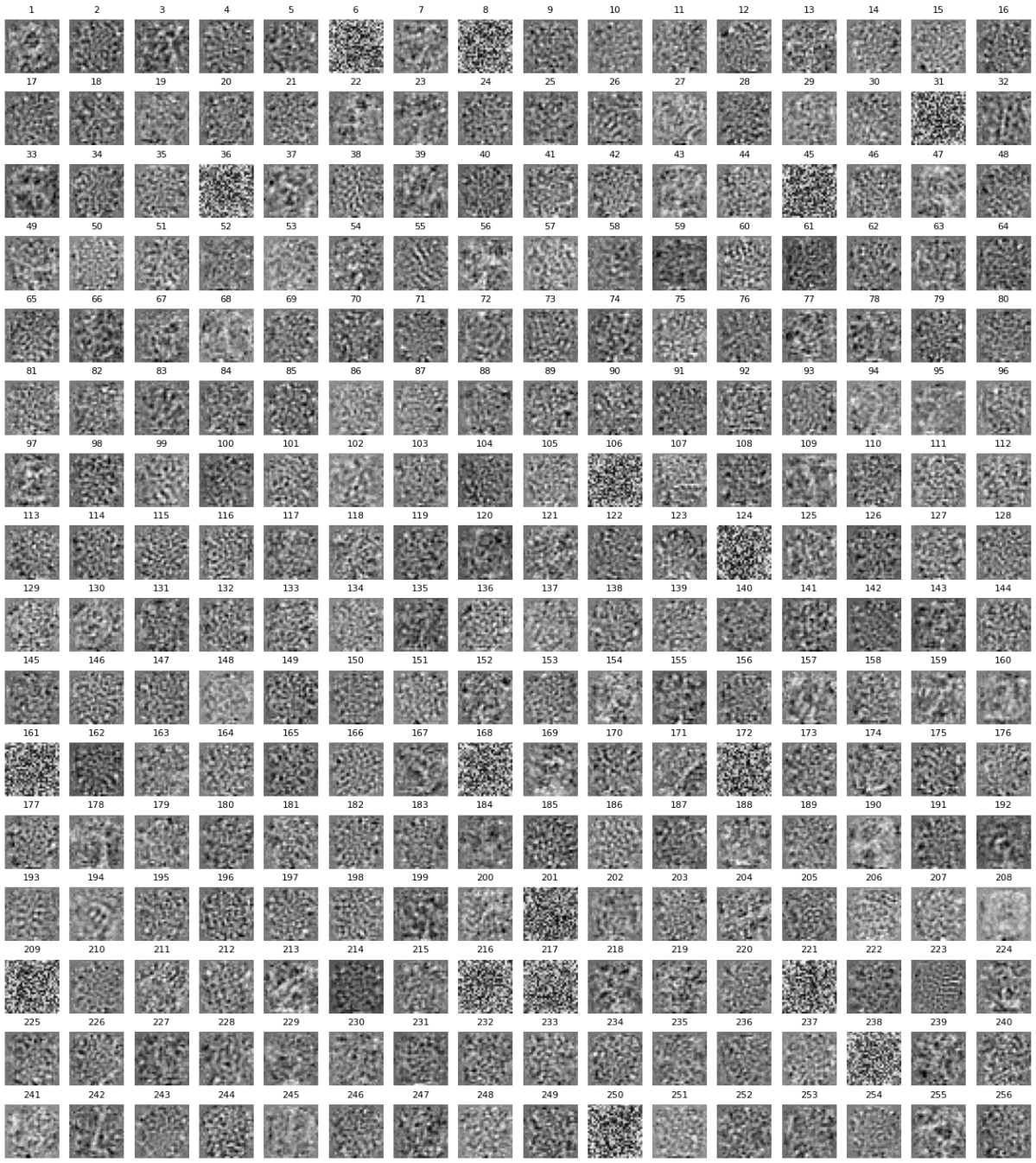
When a random noise image is passed as input to the trained autoencoders, the models do **not** reconstruct the random noise. Instead, they output a blurry, abstract image that vaguely resembles a composite or superposition of different digits. The output is clearly not random static; it has a structure that is characteristic of the MNIST dataset.

2.3


Learned Filters of the Encoder (x=64)
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64

Learned Filters of the Encoder (x=128)





Observation:

As seen in your visualizations for x=64, x=128, and x=256, the learned filters are not random noise. They exhibit clear structures.

- **For x=64:** The filters are the most interpretable. Many of them look like primitive shapes: soft diagonal strokes, gentle curves, loops, and corners. Some are more like gradients that detect blurry spots.

- **For x=128 and x=256:** As the hidden dimension increases, the filters become slightly less distinct and more numerous. While some still represent clear strokes, others appear more complex or high-frequency, suggesting the network is learning more localized or redundant features.

3.SPARSE AUTOENCODER

Number of hidden neurons=1024

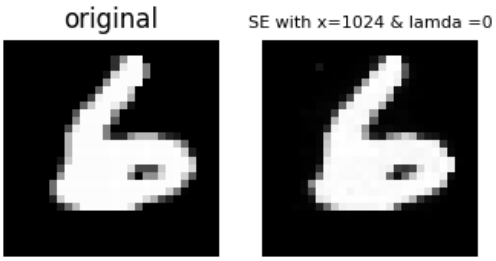
3.1

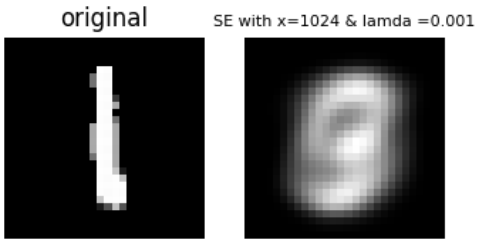
```
Model: Standard                  | Average Activation: 0.628777
Model: Sparse (lambda=1e-3)      | Average Activation: 0.000000
Model: Sparse (lambda=1e-4)      | Average Activation: 0.000000
Model: Sparse (lambda=1e-5)      | Average Activation: 0.010116
```

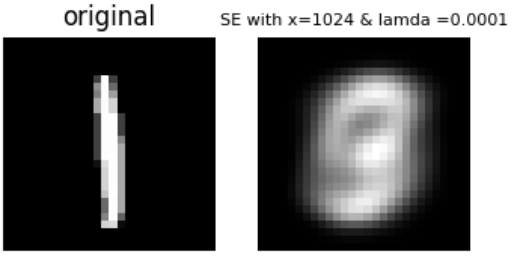
```
Observation:
The results from the experiment clearly and dramatically demonstrate
the effect of the L1 sparsity penalty. The standard over-complete
autoencoder (lambda = 0) has a high average hidden layer activation of
0.628777. This indicates that, on average, many of its hidden neurons
are active for any given input.

In stark contrast, even a very small sparsity parameter (lambda = 1e-5)
causes the average activation to plummet by over 98% to just 0.010116.
For stronger penalties (lambda = 1e-4 and 1e-3), the average activation
becomes effectively zero, meaning almost all hidden neurons are silent
for any given input.

Explanation (Why this happens):
This difference is the direct result of the modified loss function.

In the Standard AE, the only goal is to minimize reconstruction error.
Since the model is over-complete (1024 neurons > 784 pixels), it can
easily learn a "lazy" solution by having many neurons work together to
simply copy the input, leading to high overall activation.

In the Sparse AE, the optimizer has a second, competing goal: to
minimize the L1 norm of the hidden activations. The lambda parameter
controls the importance of this second goal. A higher lambda forces the
model to find a solution that uses the fewest possible active neurons
to reconstruct the image. This is why the average activation drops so
significantly—the model is being heavily penalized for every neuron it
"turns on."
```

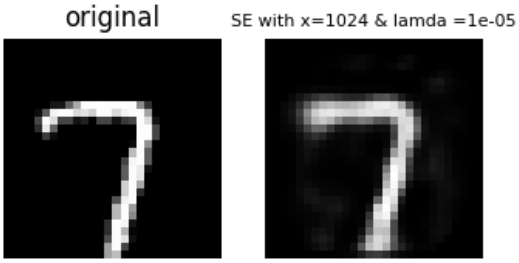
3.2

Lambda (L1 regularizer)	Reconstruction error
0	0.00085
1e-3	0.0672
1e-4	0.0670
1e-5	0.011









Observation:
Based on the reconstruction quality, we can infer the structure of the learned filters.

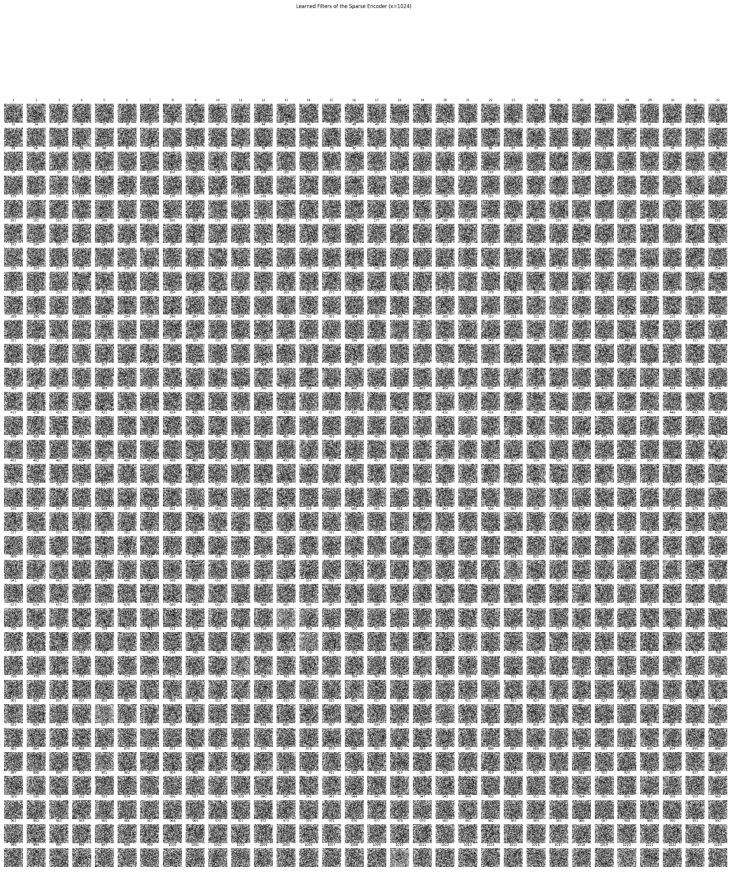
The Standard AE (lambda = 0) produces a near-perfect reconstruction. This suggests its filters are likely less distinct and potentially redundant. They work together to essentially memorize and reproduce the input.

The Sparse AE (lambda > 0) shows a clear trade-off. As lambda increases, the reconstruction becomes blurrier and more "prototypical." The reconstruction of the '1' morphs into a generic, blurry shape because the model is so heavily constrained that it can only activate a few of its most general-purpose feature detectors.

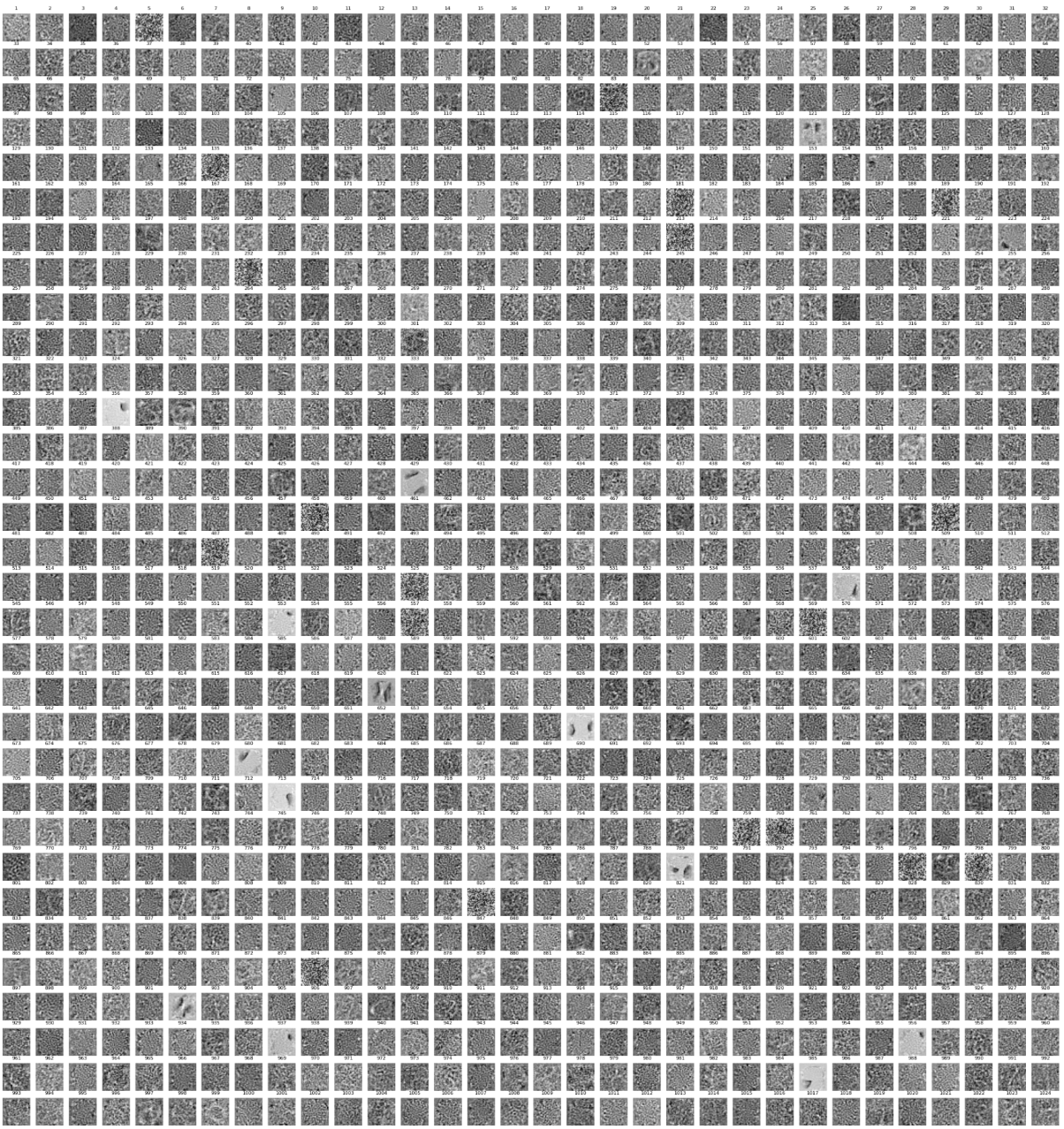
Explanation (Why the filters change):
This difference in reconstruction quality is a direct consequence of the kind of filters the models learn.

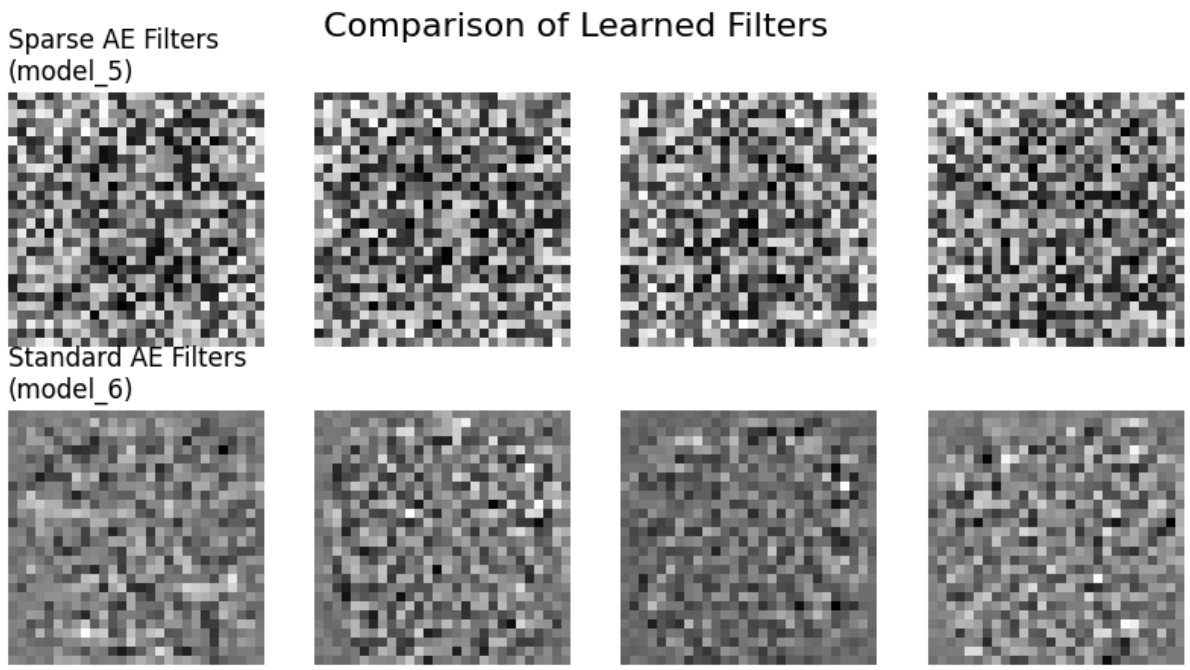
Standard AE Filters: Without a sparsity penalty, the over-complete network has no incentive to be efficient. Many filters can learn similar, overlapping features. The resulting filters are often less interpretable, noisy, and not clearly defined.

Sparse AE Filters: The sparsity penalty forces each neuron to become a highly specialized, efficient feature detector. Because activating a neuron is "expensive," the model ensures that each one learns to detect a unique and meaningful pattern (a specific edge, curve, or stroke). The resulting filters will be much more interpretable, distinct, and localized. They will look like the clear, fundamental building blocks of digits.



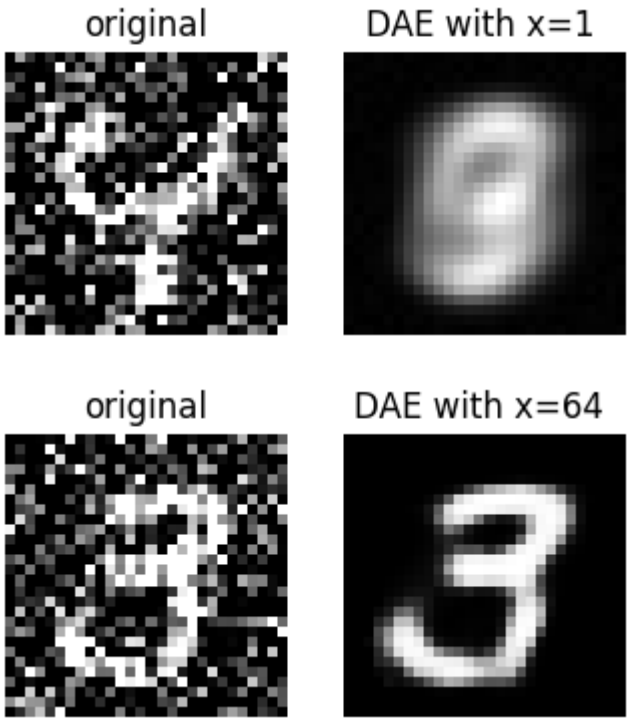
Learned Filters of the Auto Encoder (x=1024)





4 Denoising Autoencoders

4.1

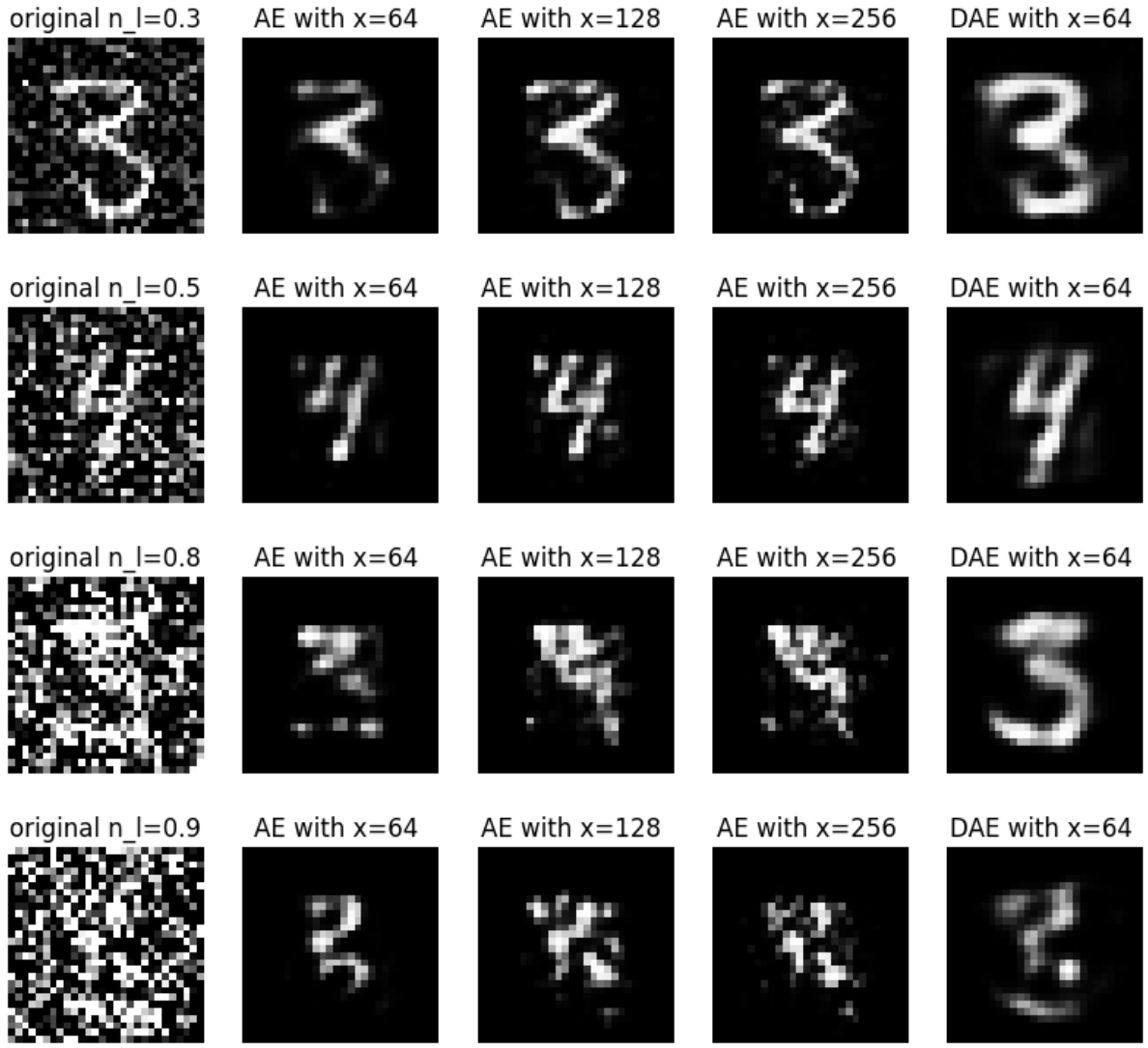


When a noisy image is passed to a **Standard Autoencoder**, it **fails to denoise** the image. As your results show, the output is a fragmented mess because the model, having only ever seen clean data, treats the noise as part of the signal and tries to reconstruct it

4.2

The **Denoising Autoencoder (DAE)**, in contrast, **successfully removes the noise** and reconstructs a recognizable, albeit blurry, digit. It works because it was explicitly trained to

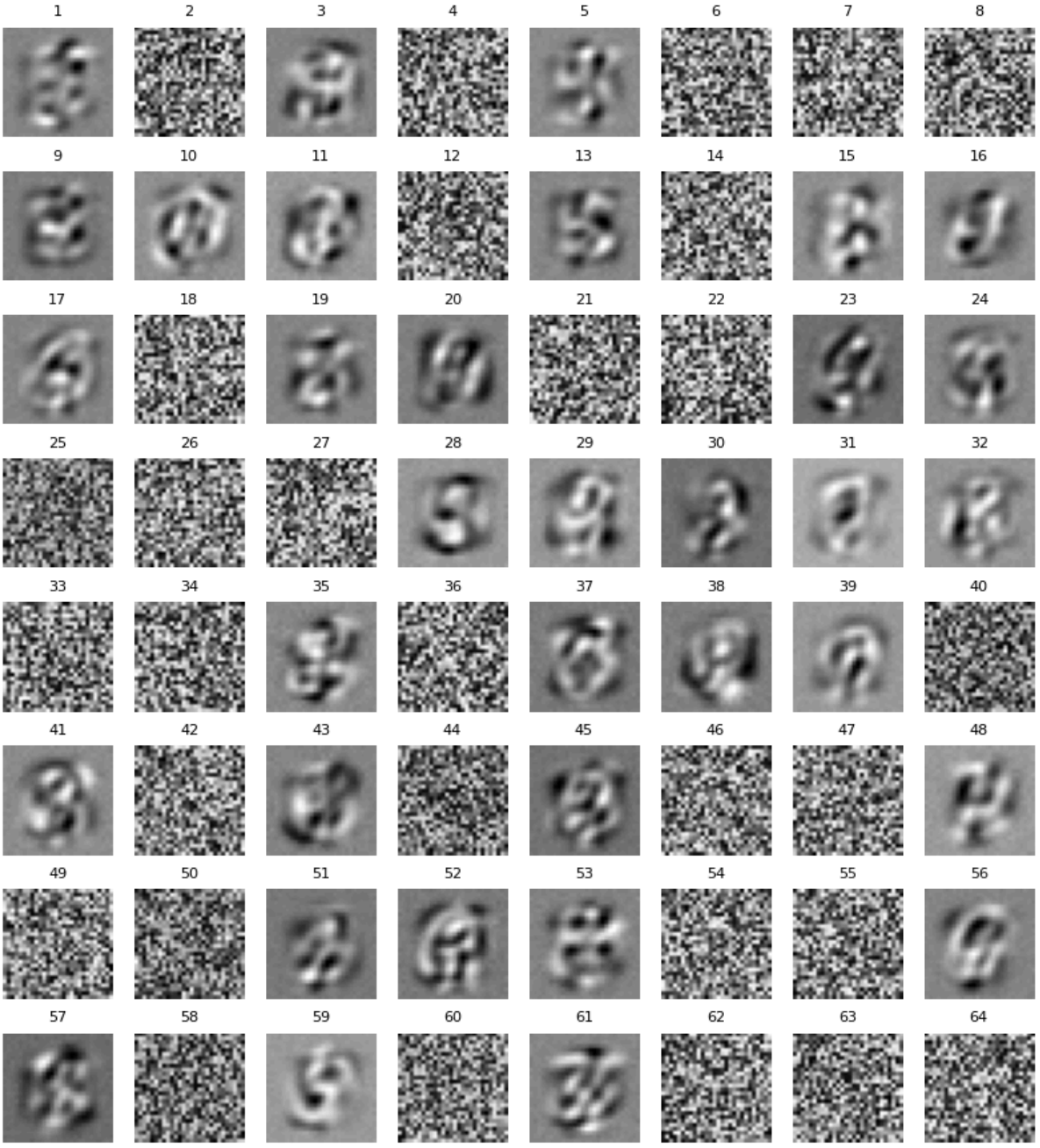
map corrupted inputs to their clean originals. As the noise level increases, the reconstruction becomes blurrier because the model has to make a "best guess" to fill in more missing information.



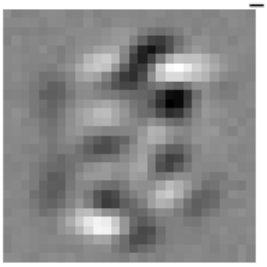
4.3

The DAE's filters are fundamentally different from a Standard AE's. Instead of learning small, local features like strokes and curves, the DAE has learned **holistic, template-like filters** that resemble entire digits (e.g., '0', '9', '3'). This happens because recognizing a whole digit shape is a much more robust strategy for denoising than relying on small features that might be completely wiped out by noise.

Learned Filters of the DAE for noise_level=0.5 (x=64)



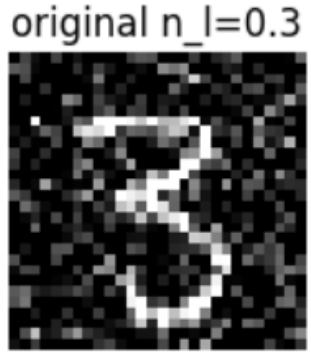
Learned Filters of the DAE for noise_level=0.5 (x=1)



5 MANIFOLD LEARNING

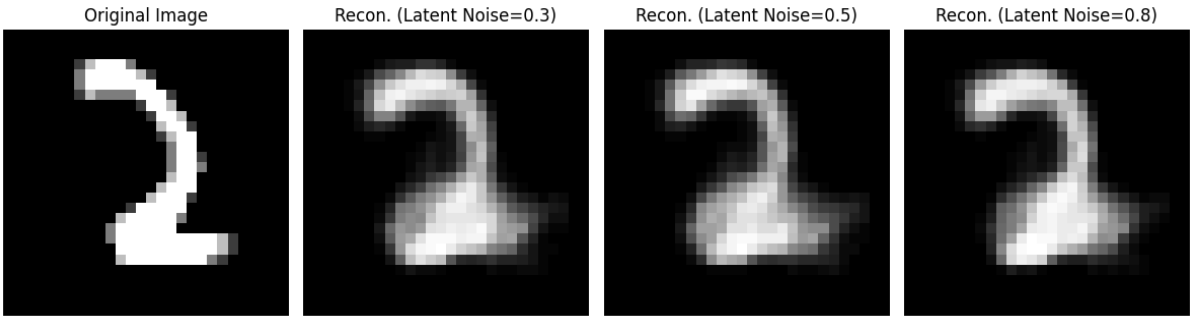
5.1

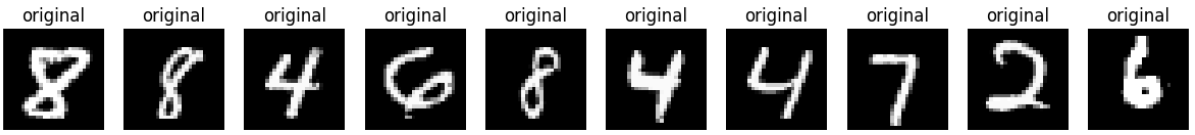
Adding random noise to a 784D image vector fails to produce a valid digit because you are moving a point **off the data manifold**. The space of all possible images is vast, but valid digits lie on a very small surface within it. A random move will almost certainly land in the empty, meaningless space between valid data points.

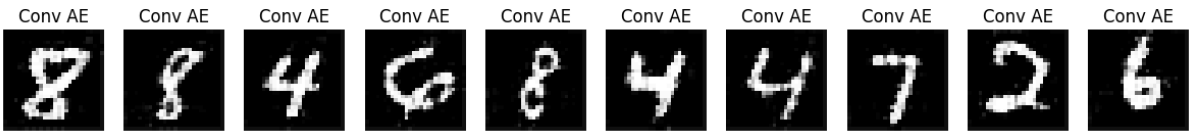


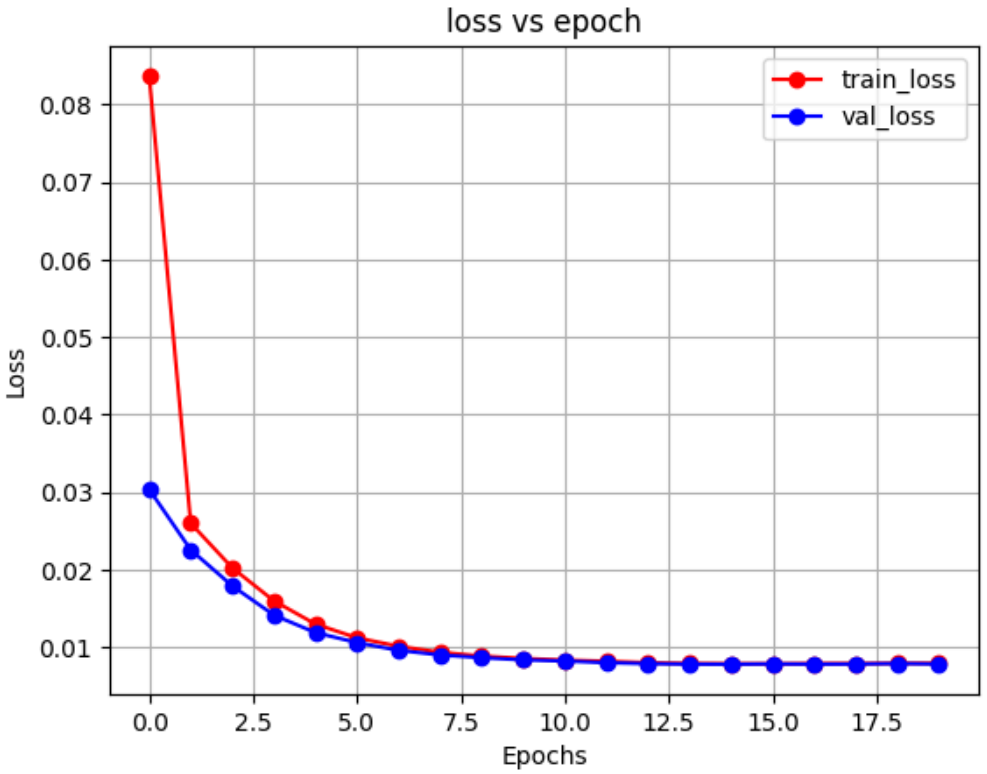
5.2 & 5.3

Your results show that adding noise to the 8D **latent vector** creates another plausible digit, not noise. This is because the autoencoder has learned a "map" of the digit manifold. The decoder only knows how to project coordinates from this map back onto the manifold, so even a noisy coordinate is mapped to the nearest valid digit shape, causing a smooth and meaningful change in appearance.

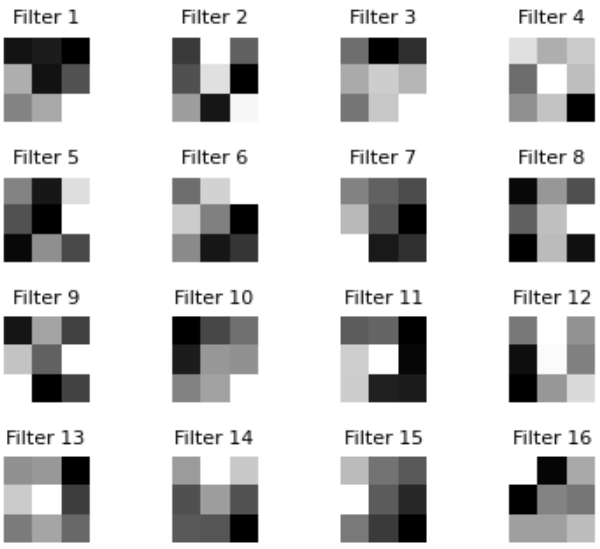


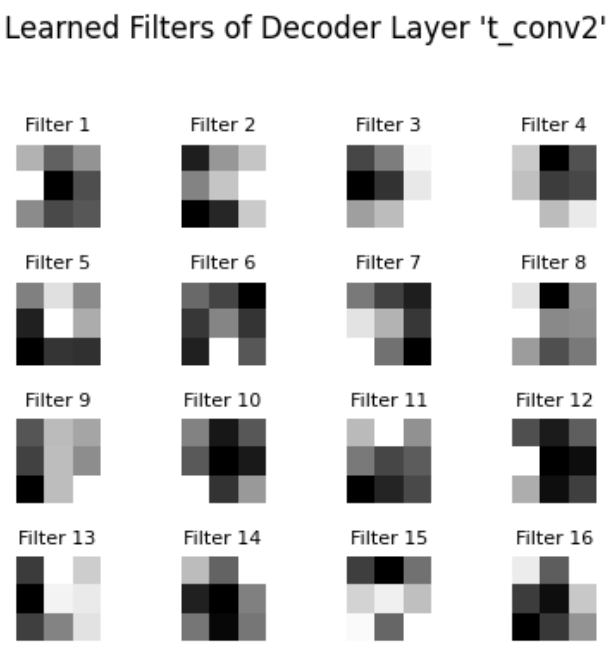


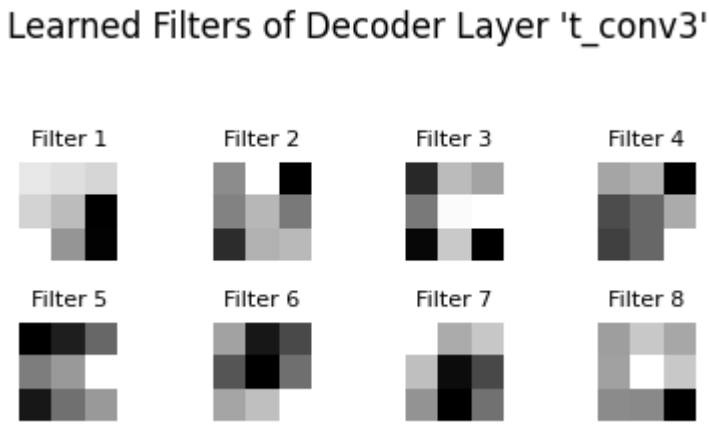




Learned Filters of Decoder Layer 't_conv1'







The model demonstrates excellent **convergence**, as shown by the training and validation loss curves which flatten out after approximately 10 epochs. The final **reconstruction error** is very low, evident from the sharp and accurate reconstructions of the test images.

The visualized decoder weights are not random noise; instead, they have learned **interpretable, primitive patterns** like edges, corners, and gradients. The network uses these simple, learned "building blocks" to effectively draw and reconstruct the more complex digit shapes.