Variational Autoencoders as PIN Code Password Encryption

(Soon to be) Dr David Mills[*]
Department of AI
Utlandia
`davidmills@GG.ut`

Abstract

The four-digit pins used at every workplace are finally obsolete as part of the AI-fication. Instead, the new logins are latent space vectors, containing continuous values, resulting in far more security than mere four-digit pins! Each pin is encoded from a picture into the vector representations. The user now has to input the vector representations, which are then decoded into the image of the numbers, which again is classified by the super-accurate AI number classifier. What a world we live in. I will indeed get promoted after the boss learns about this.

1 Latent Space

A latent space is a space of multiple vector representations of original data, often of a smaller size or with fewer dimensions compared to the original. Think of it as creating a highly compressed digital ID card for every single piece of data (like an image of a number). This small, dense vector captures the essence of the original, making it much harder to guess or brute-force than a simple PIN.

2 Autoencoders

The simplest way of creating a latent representation is by using an Autoencoder. By passing the input data through an Encoder to a smaller size matrix (the latent space), then recreating the input data from the original through a Decoder. This simple technique enables a compressed representation and is often used for large image training, most famously in DALL-E. Essentially, an Autoencoder is a clever machine that learns to efficiently summarize an input and then perfectly reproduce it from that summary. If it can't reproduce it well, it knows the summary (the vector) is flawed.

[*]Goood Games.

Preprint. Under review.

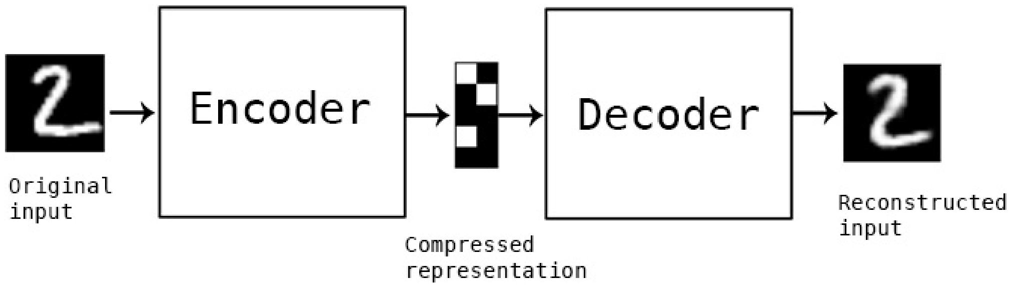

Figure 1: Autoencoder. *(Source: https://blog.keras.io/building-autoencoders-in-keras.html)*

Formally, the autoencoder is described as below, where X is the input data, E is the encoder, D is the decoder, z is the latent representation of X, and $\hat{X}$ is the reconstruction of X by the decoder.

$$E(X) = z, \quad D(z) = \hat{X} \tag{1}$$

The loss is then computed using the MSE (Mean Squared Error) loss function.

3 Variational Autoencoders

A Variational Autoencoder (VAE) takes the basic Autoencoder concept and adds a brilliant twist: probability. Instead of the encoder producing one single, fixed vector, the VAE's encoder generates two things for each input: a mean (μ) and a standard deviation (σ for variance).

- Standard Autoencoders: Create a very rigid, specific map. If someone tries to input a vector that's slightly off, the Autoencoder might not know what to do.
- Variational Autoencoders (VAEs): Create a smooth, continuous map, like a well-paved digital highway. Because the latent vectors are described by a probability distribution, the VAE ensures that similar inputs (e.g., a "3" written slightly differently) result in similar latent vectors that are close neighbors in the space.

This structural organization is key! It makes the latent space not just smaller, but meaningful and traversable.

3.1 Sampling and Training

To create this smooth map, the VAE does not simply output a coordinate. Instead, it defines a statistical distribution, a "cloud" of probabilities—where the data point is likely to exist. The encoder predicts the center of this cloud (the expected value or mean) and how spread out it is (the standard deviation):

$$\mathbb{E}[\mathbf{z}] = \boldsymbol{\mu} \quad \text{and} \quad \text{Std}(\mathbf{z}) = \boldsymbol{\sigma} \tag{2}$$

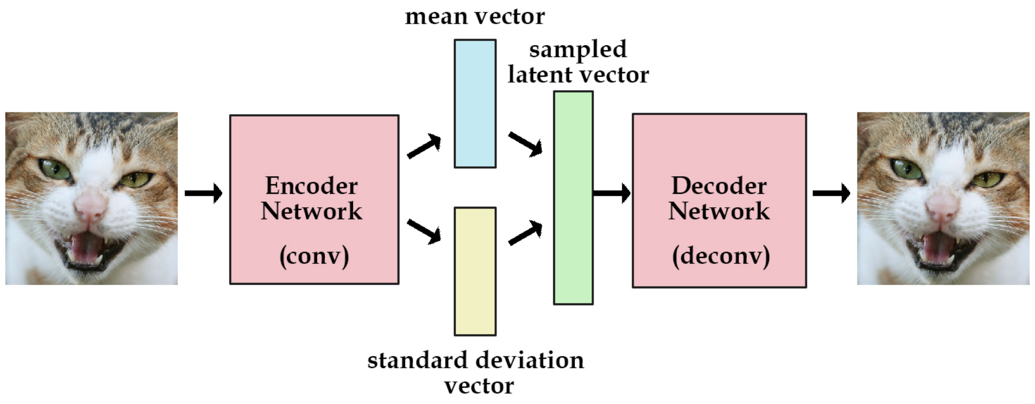

Figure 2: Variational Autoencoder architecture. The encoder maps input X to a distribution defined by μ and σ, from which z is sampled using the reparameterization trick. *(Source: `https://www.piyushmalhotra.in/Autoencoder-Implementations/VAE/`)*

The Reparameterization Trick
To train the network, we need to sample a point $\mathbf{z}$ from this cloud to feed into the decoder. However, purely random sampling breaks the chain of mathematics needed for backpropagation (training). The VAE solves this with the *Reparameterization Trick*.

Instead of asking the network to be random, we take the deterministic parameters ($\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$) and add the randomness externally using a noise variable, $\boldsymbol{\epsilon}$, drawn from a standard normal distribution. This allows us to calculate the latent vector $\mathbf{z}$ as:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \text{where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}) \tag{3}$$

This trick allows the model to learn the parameters while still maintaining the necessary stochastic (random) element.

3.2 The Loss Function

The VAE is trained using a specific objective function called the **Evidence Lower Bound (ELBO)**. Think of this as a tug-of-war between two opposing goals:

1. **Accuracy:** The model wants to reproduce the image perfectly.
2. **Regularization:** The model wants to keep the latent space neat and organized (preventing overfitting).

Mathematically, this is expressed as:

$$\mathcal{L} = \underbrace{\mathbb{E}_q[\log p(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Loss}} - \underbrace{D_{KL}(q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))}_{\text{KL Divergence}} \tag{4}$$

The **Reconstruction Loss** measures how well the output matches the input. The **KL Divergence** (Kullback-Leibler) measures the difference between our learned distribution and a standard Gaussian distribution. It penalizes the model if the latent clouds get too spread out or drift too far apart. For Gaussian distributions, this penalty can be calculated directly as:

$$D_{KL} = -0.5 \sum (1 + \log(\boldsymbol{\sigma}^2) - \boldsymbol{\mu}^2 - \boldsymbol{\sigma}^2) \tag{5}$$

By minimizing this combined loss, the VAE learns to generate crisp images while ensuring the latent space remains continuous and well-structured. Figure 3 tries to illustrate how the different numbers are placed in a latent space, and how the output image changes by altering the different latent dimensions.

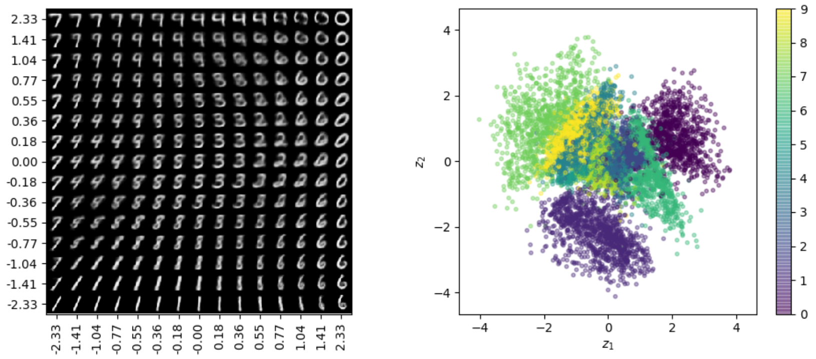

Figure 3: Latent Space illustration. *(Source: `https://tiao.io/post/tutorial-on-variational-autoencoders-with-a-concise-keras-implementation/`)*

3.3 Why VAEs are the Superior Choice

This probabilistic approach means the VAE is not just memorizing the PIN images; it's learning the underlying concept of the digits. This makes the system far more robust and secure:

1. Better Security: It's practically impossible for a hacker to guess a valid vector in this continuous, high-dimensional space. There are literally infinite possibilities, unlike the mere 10,000 options for a 4-digit PIN.
2. Flexibility: The VAE can generate a slightly different (but still valid) vector every time for the same PIN image, further confusing potential intruders.

4 Using Variational Autoencoders as Encryption

4.1 The Login Process: A Digital Handshake

We are leveraging the VAE's ability to create a consistent, structured latent space for our new, highly secure login system.

- Step 1: Enrollment: The user's PIN (as an image of the numbers) is passed through the VAE's Encoder. The resulting latent vector is what we store securely in the database as the user's "password." This vector is the secure, compressed ID.
- Step 2: Login: The user inputs their stored latent vector into the system. This vector is then passed through the VAE's Decoder.
- Step 3: Verification: The Decoder uses the latent vector to reconstruct (or decode) the original image of the numbers. This reconstructed image is then checked by a final, highly accurate AI Classifier. If the classifier says "Yes, that's a perfect '1234' image!", the login is approved.

4.2 Security Benefits: Why This is a Promotion-Worthy Idea

This system completely shifts the security paradigm from "What is your secret number?" to "Can you produce the correct digital blueprint of your secret number?"

- Hacking Prevention: If a hacker steals the stored latent vector, they still don't have the original PIN number! They only have a dense, unintelligible series of floating-point numbers that only the VAE's specific decoder can translate back into the original image.
- Perfect Secrecy: Since the login uses a high-dimensional vector instead of a simple number, there is no way to "peek" at the input and get a hint. This is a game-changer for workplace security, putting us at the forefront of AI-driven authentication.

This innovative use of VAEs moves our authentication beyond simple memorized secrets into the realm of complex, AI-generated digital signatures, ensuring a safer and more future-proof workplace.

5 MNIST

To train this monstrosity, the famous MNIST dataset will be used. Consisting of thousands of hand-drawn and labeled numbers, this is ideal for use with PIN codes.

Figure 4: Example of samples from the MNIST Dataset. *(Source: `https://datasets.activeloop.ai/docs/ml/datasets/mnist/`)*

6 Image Classifier

To ensure that the correct number is presented from the latent representation, an image classifier will be utilized. The classifier is trained on the MNIST dataset with an accuracy of 98% (WOW), and instead of having the normal labels of numbers from 0-9, I have included a 10th index. This index represents numbers that are not numbers, so that the solution can't be brute-forced (Mama Mia, he did it again :D). To obtain labeled data for the unknown category, the VAE was used to generate outlier numbers, which were then assigned to the unknown category. This makes for a more robust classifier. Figure 5 illustrates the classifier's predictions with its certainty above the numbers. Note that it will not vouch for a number, unless it is 98% certain that this is the number (Whaaaaa).

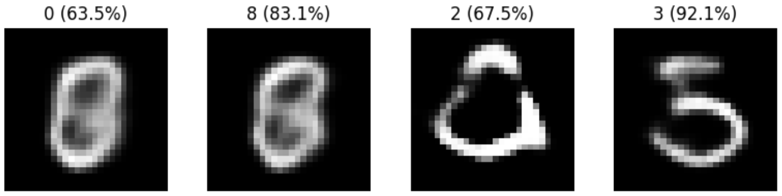

Figure 5: Classified with certainty

The network consists of several Convolutional Neural Network layers, with a kernel size of 3, with ReLU activation functions between each layer (ref my previous papers). A softmax function is applied to the output layer to ensure the output is a probability distribution. Figure 6 is an example illustration of how the classifier network looks. The loss function used during training is Cross-Entropy Loss.

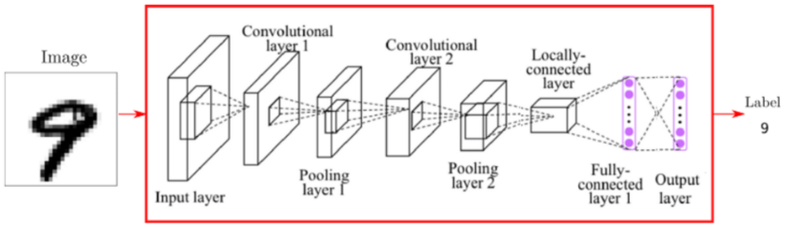

Figure 6: Classifier network example. *(Source:* `https://www.researchgate.net/figure/CNN-for-MNIST-data-classification-47_fig2_350908560`*)*

7 Model Results

The following figures illustrate the excellent performance of the decoder, which is trained to act as a number decoder. The latent spaces are slightly altered along different axes, and as a result, the number changes from one category to another.

7.1 VAE

The following figures illustrate how the decoder decodes different latent representations when altering a single dimension of the latent vector in the range [-3, 3].

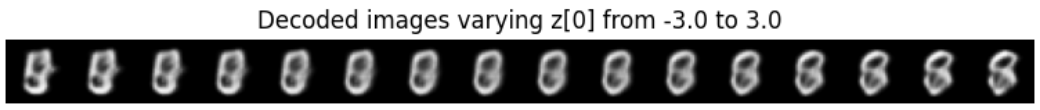

Figure 7: Dim0

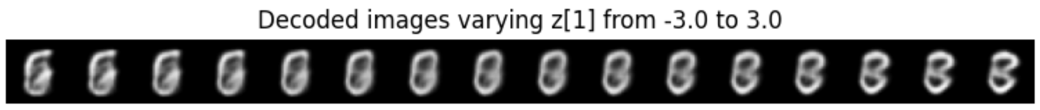

Figure 8: Dim1

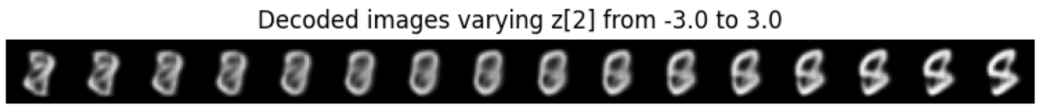

Figure 9: Dim2

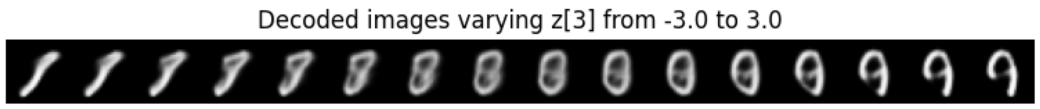

Figure 10: Dim3

Visualization of the latent space manipulation across four dimensions.

The following images are samples from the latent grid after altering a single dimension in Figure 11, the latent space in two dimensions (using PCA) is shown in Figure 12, and lastly reconstructions of the MNIST dataset in Figure 13, where the upper half are original samples, and the lower half are the reconstructions.

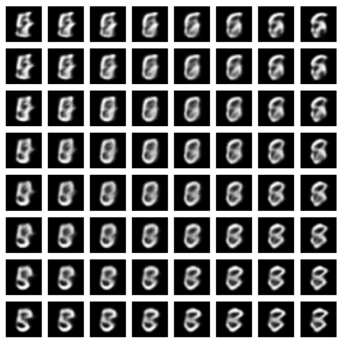

Figure 11: Latent Grid

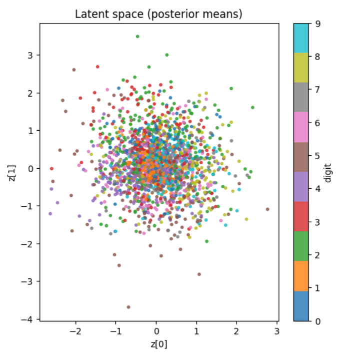

Figure 12: Latent space in 2D showing different numbers with labels.

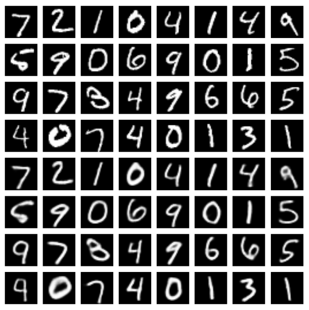

Figure 13: Reconstructions using the VAE. The numbers above the middle are real, while the ones below are reconstructed.

8 Endpoints

Different endpoints available in the login page:

8.0.1 GET /

Returns rendered HTML using session-stored small result and latent vectors.

8.0.2 POST /send_random

No request body. Generates four random latent vectors, runs inference, stores a small result in the session, and returns "OK".

8.0.3 POST /decode_predict

Accepts JSON with latent vectors. Examples:

List of vectors:

```
{ "z": [
    [0.12, -0.55, ..., 0.09],
    [0.03,  0.11, ..., -0.21]
  ]
}
```

Validation errors (HTTP 400) return JSON like:

```
{ "error": "Missing 'z' in request body" }
{ "error": "'z' must be a list or list of lists" }
{ "error": "Vector at index 0 has length 18; expected 20" }
```

9 Conclusion

As proven by this paper, this new pin code idea is genius. The classifier and decoder work immaculately together, especially when ensuring that the classifier is robust and knows when the generated numbers are not numbers...

To my haters who say that this design results in the possibility that one code can be input using different vectors, please... You are aware that a GUID can generate two identicals. If you knew my salary, you would stay silent... Imagine hahahaha.