Numerical methods project

HUBER-ENERGY VARIATIONAL AUTOENCODER

Grégoire Mourre David Premachandra

Master 2 Applied and Theoretical Mathematics

2023-2024

Contents

Abstract

In this file, we test several architectures for some well known datatsets. Each time, we first present the parameters and the architecture of the encoder and decoder. Then, we plot the results.

1 First test with the MNIST dataset (convolutional encoder and decoder)

Encoder Architecture

```
mnist_encoder = tf.keras.Sequential([
                InputLayer(input_shape = mnist_parameters[’input_dim’]),
                Conv2D(32, (3, 3), activation=’relu’),
                Conv2D(64, (3, 3), activation=’relu’),
                Flatten(),
                Dense(128, activation=’relu’),
                Dense(mnist_parameters[’latent_dim’], activation=None)
])
```

Decoder Architecture

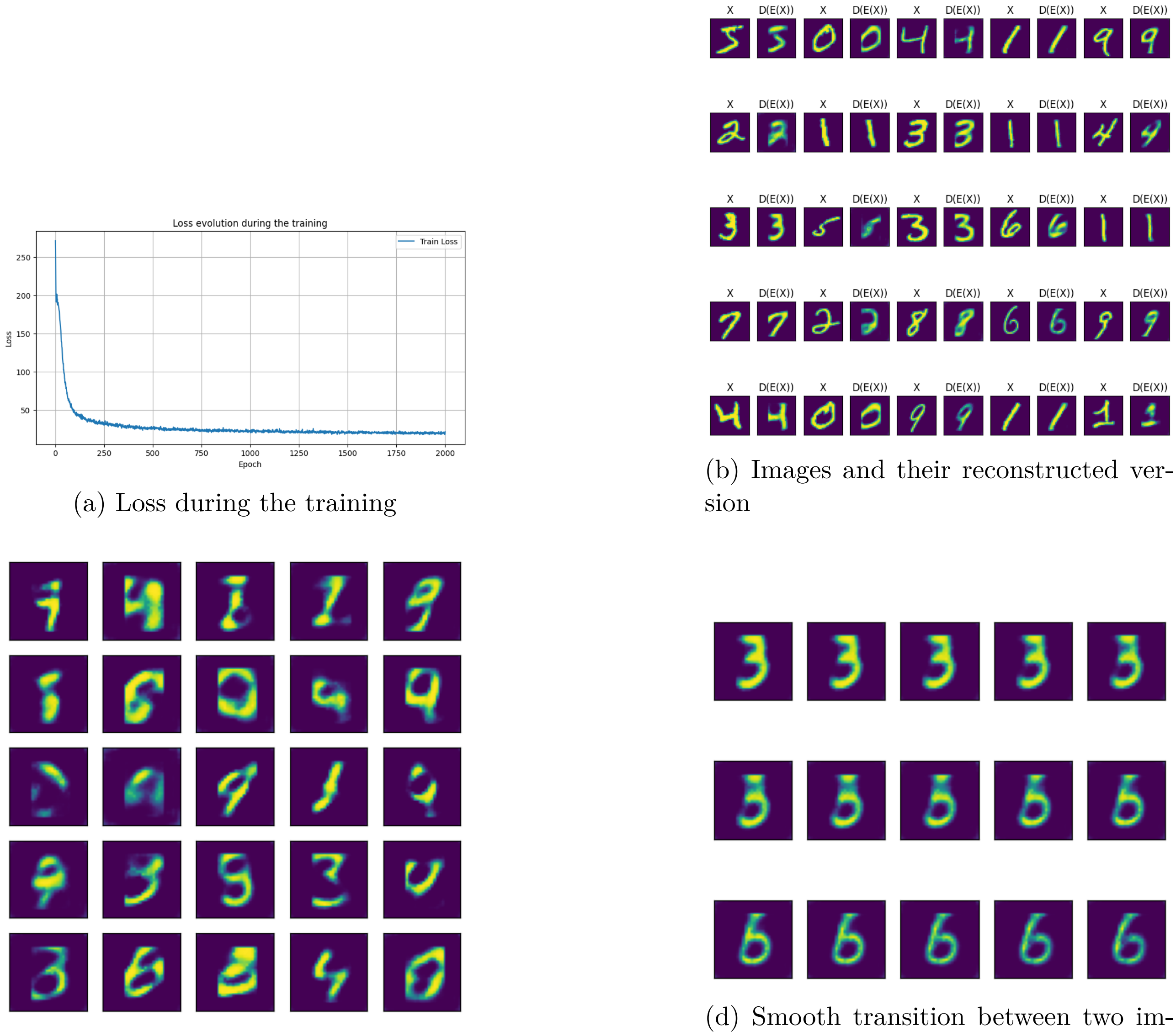
```
mnist_decoder = tf.keras.Sequential([
                InputLayer(input_shape = mnist_parameters[’latent_dim’]),
                Dense(256, activation=’relu’),
                Dense(1024, activation=’relu’),
                Dense(8*8*8, activation=’relu’),
                Reshape((8,8,8)),
                Conv2DTranspose(16, (8, 8), activation=’relu’),
                Conv2DTranspose(4, (8, 8), activation=’relu’),
                Conv2DTranspose(1, (7, 7), activation=’sigmoid’)
])
```

Model Parameters

```
mnist_parameters = {’input_dim’: (28,28,1),
                    ’1D data’: False,
                    ’latent_dim’: 12}

mnist_training_parameters = {’dataset’: train_images_mnist / 255.0,
                             ’learning_rate’: 0.0005,
                             ’epochs’: 2000,
                             ’batch_size’: 200}
```

Results



(a) Loss during the training

(b) Images and their reconstructed version

(c) Generated images

(d) Smooth transition between two images

Figure 1: Test of the architecture and the parameters presented above for the MNIST dataset

.

2 Second test with the MNIST dataset (dense encoder and decoder)

```
number_of_neurons = 128
```

Encoder Architecture

```
mnist_2_encoder = tf.keras.Sequential([
                InputLayer(input_shape = mnist_parameters['input_dim']),
                Flatten(),
                Dense(number_of_neurons, activation='relu'),
                Dense(number_of_neurons, activation='relu'),
                Dense(number_of_neurons, activation='relu'),
                Dense(mnist_parameters['latent_dim'], activation=None)
            ])
```

Decoder Architecture

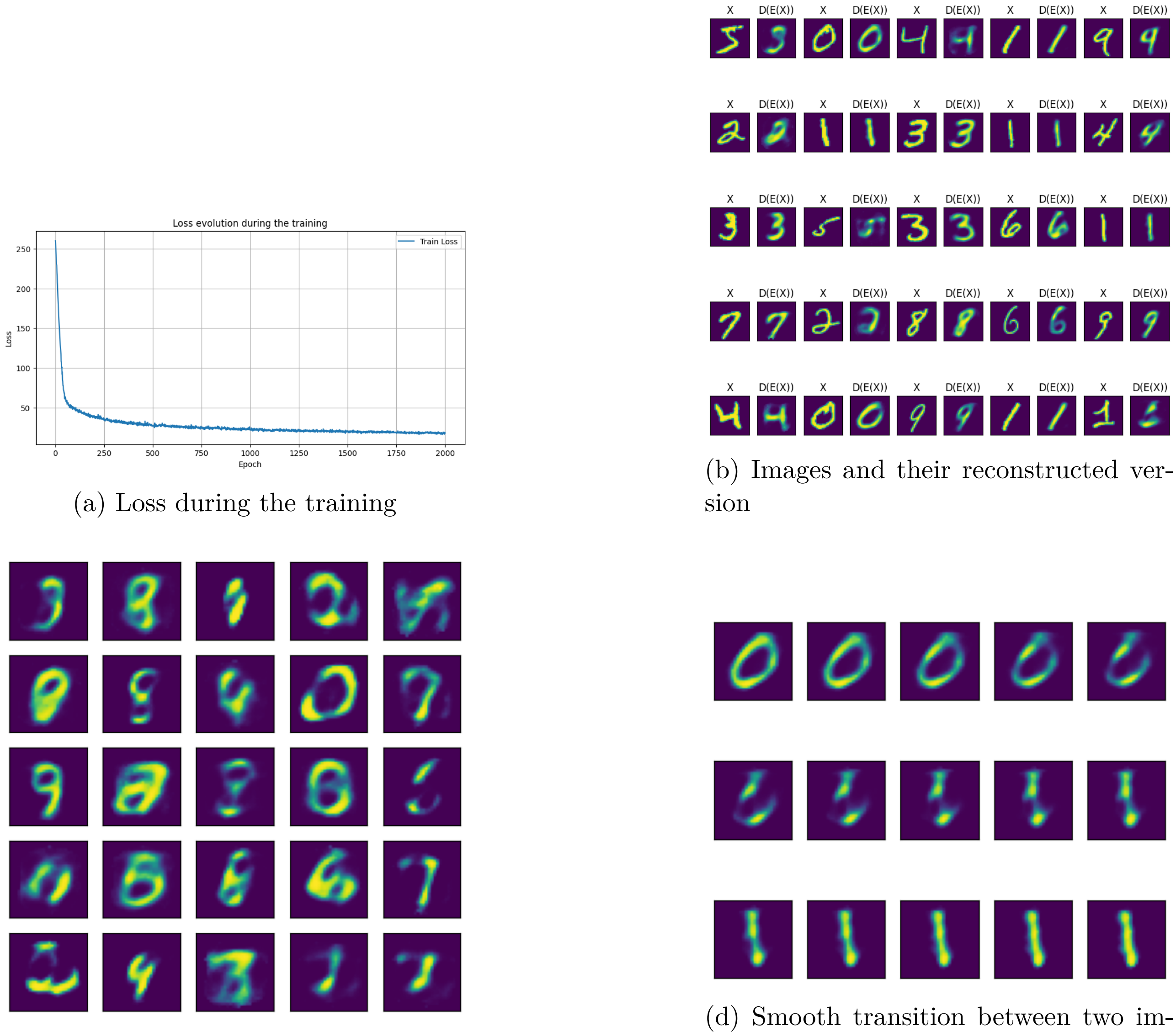
```
mnist_2_decoder = tf.keras.Sequential([
                InputLayer(input_shape = mnist_parameters['latent_dim']),
                Dense(number_of_neurons, activation='relu'),
                Dense(number_of_neurons, activation='relu'),
                Dense(28*28*1, activation='sigmoid'),
                Reshape((28,28,1))
            ])
```

Model Parameters

```
mnist_parameters = {'input_dim': (28,28,1),
                    '1D data': False,
                    'latent_dim': 12}

mnist_training_parameters = {'dataset': train_images_mnist / 255.0,
                             'learning_rate': 0.0005,
                             'epochs': 2000,
                             'batch_size': 200}
```

Results



(a) Loss during the training

(b) Images and their reconstructed version

(c) Generated images

(d) Smooth transition between two images

Figure 2: Test of the architecture and the parameters presented above for the MNIST dataset

.

3 Third test with the MNIST dataset (dense encoder and decoder with more neurons)

```
number_of_neurons = 256
```

Encoder Architecture

```
mnist_2_encoder = tf.keras.Sequential([
                InputLayer(input_shape = mnist_parameters['input_dim']),
                Flatten(),
                Dense(number_of_neurons, activation='relu'),
                Dense(number_of_neurons, activation='relu'),
                Dense(number_of_neurons, activation='relu'),
                Dense(mnist_parameters['latent_dim'], activation=None)
            ])
```

Decoder Architecture

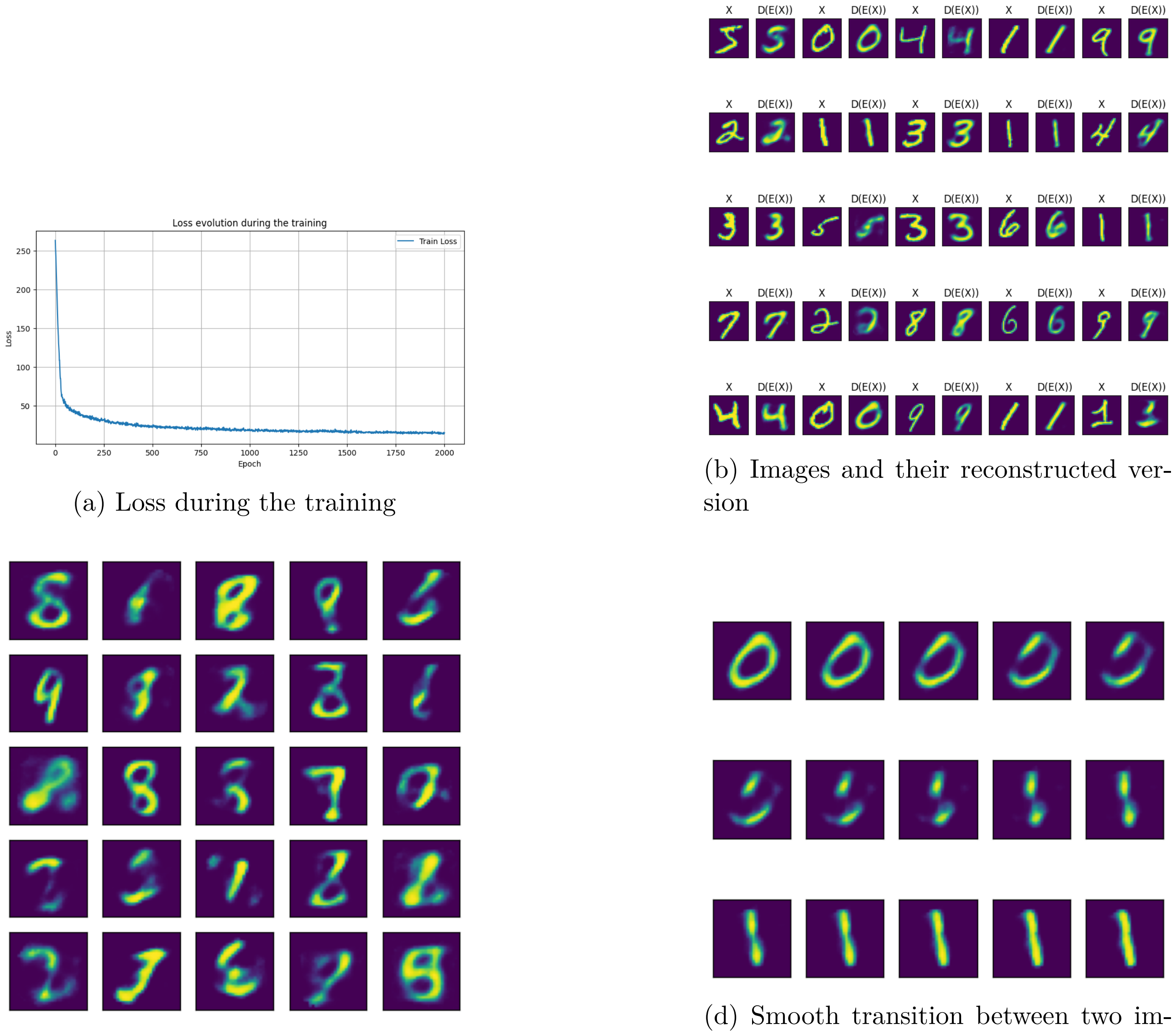
```
mnist_2_decoder = tf.keras.Sequential([
                InputLayer(input_shape = mnist_parameters['latent_dim']),
                Dense(number_of_neurons, activation='relu'),
                Dense(number_of_neurons, activation='relu'),
                Dense(28*28*1, activation='sigmoid'),
                Reshape((28,28,1))
            ])
```

Model Parameters

```
mnist_parameters = {'input_dim': (28,28,1),
                    '1D data': False,
                    'latent_dim': 12}

mnist_training_parameters = {'dataset': train_images_mnist / 255.0,
                             'learning_rate': 0.0005,
                             'epochs': 2000,
                             'batch_size': 200}
```

Results



(a) Loss during the training

(b) Images and their reconstructed version

(c) Generated images

(d) Smooth transition between two images

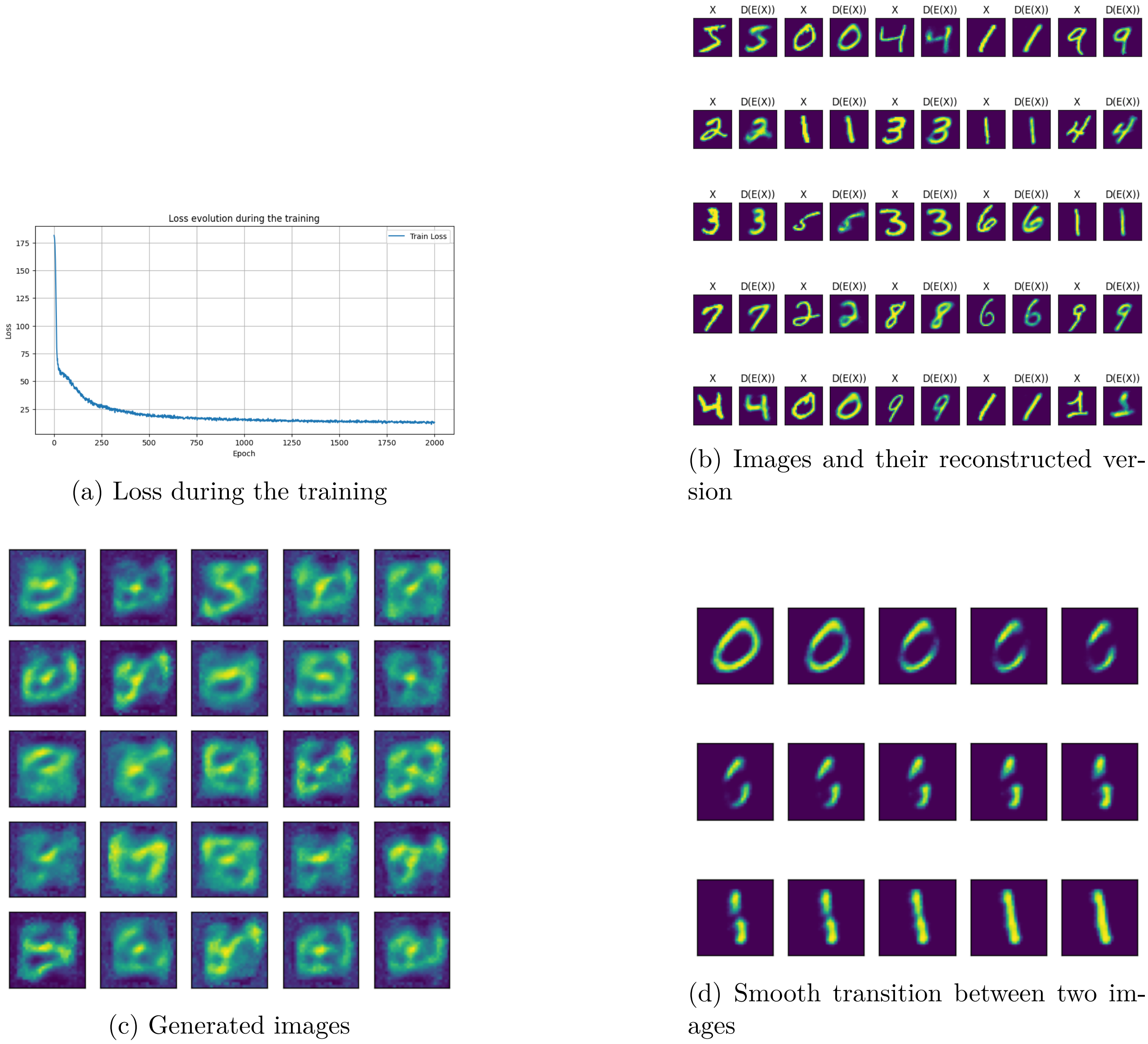
Figure 3: Test of the architecture and the parameters presented above for the MNIST dataset

.

4 Fourth test with the MNIST dataset (same parameters than in the previous question, but `lambda_factor` = 0)

Setting `lambda_factor` to 0 means that we ignore the latent loss.

Results



(a) Loss during the training

(b) Images and their reconstructed version

(c) Generated images

(d) Smooth transition between two images

Figure 4: Test of the architecture and the parameters presented above for the MNIST dataset

5 Test with the Fashion-MNIST dataset

Encoder Architecture

```
fashion_mnist_encoder = tf.keras.Sequential([
    InputLayer(input_shape = fashion_mnist_parameters['input_dim']),
    Conv2D(32, (3, 3), activation='relu'),
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(fashion_mnist_parameters['latent_dim'], activation=None)
])
```

Decoder Architecture

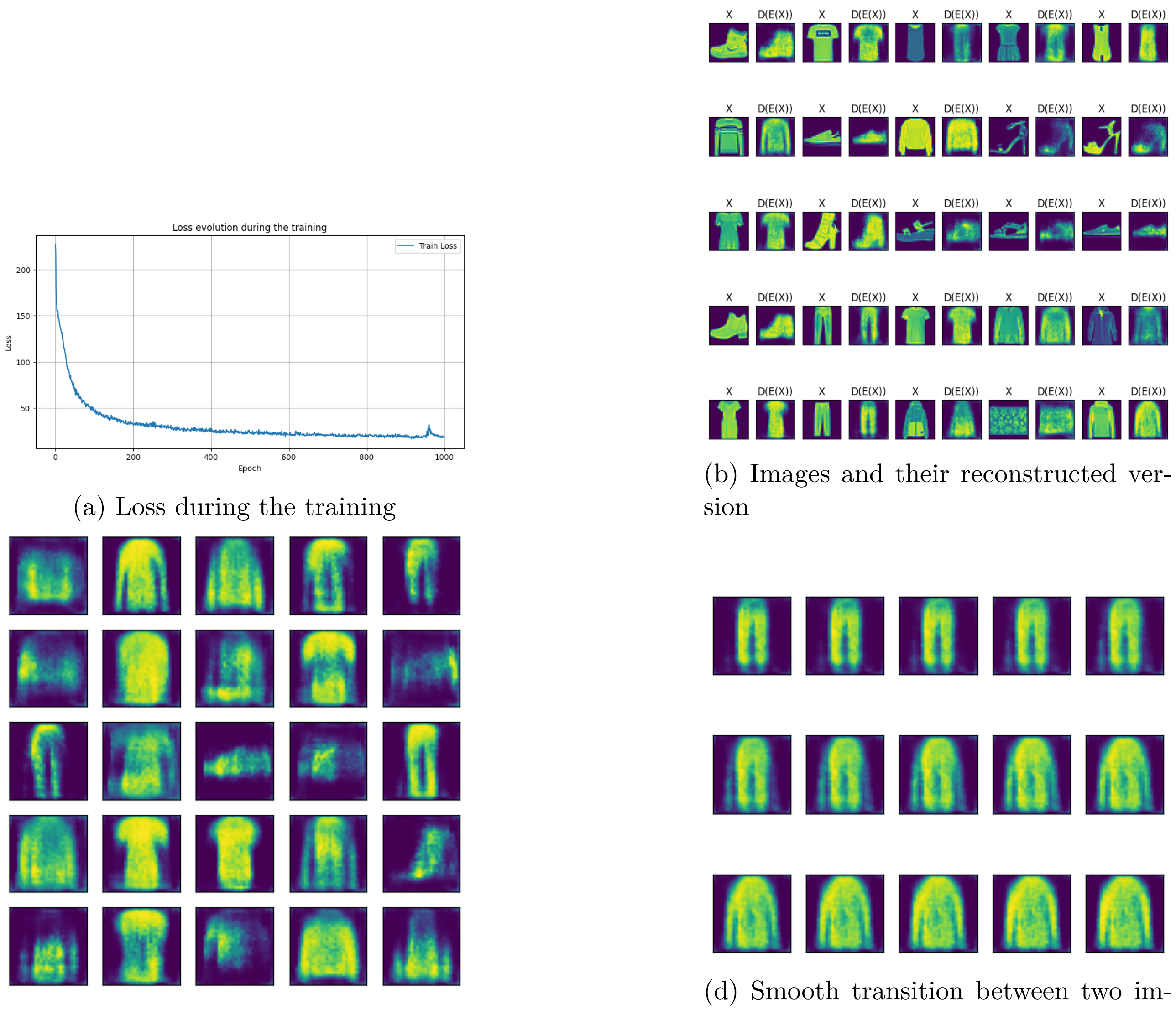
```
fashion_mnist_decoder = tf.keras.Sequential([
    InputLayer(input_shape = fashion_mnist_parameters['latent_dim']),
    Dense(256, activation='relu'),
    Dense(1024, activation='relu'),
    Dense(8*8*8, activation='relu'),
    Reshape((8,8,8)),
    Conv2DTranspose(16, (8, 8), activation='relu'),
    Conv2DTranspose(4, (8, 8), activation='relu'),
    Conv2DTranspose(1, (7, 7), activation='sigmoid')
])
```

Model Parameters

```
fashion_mnist_parameters = {
    'input_dim': (28, 28, 1),
    '1D_data': False,
    'latent_dim': 12
}

fashion_mnist_training_parameters = {
    'dataset': train_images_fashion_mnist / 255.0,
    'learning_rate': 0.0005,
    'epochs': 1000,
    'batch_size': 200
}
```

Results



(a) Loss during the training

(b) Images and their reconstructed version

(c) Generated images

(d) Smooth transition between two images

Figure 5: Test of the architecture and the parameters presented above for the Fashion-MNIST dataset

.

6 Test with the CIFAR-10 dataset

Encoder Architecture

```
cifar10_encoder = tf.keras.Sequential([
                InputLayer(input_shape = cifar10_parameters['input_dim']),
                Conv2D(3, (2, 2), activation='relu'),
                Conv2D(32, (2, 2), strides=(2,2), activation='relu'),
                Conv2D(32, (3, 3), activation='relu'),
                Conv2D(32, (3, 3), activation='relu'),
                Flatten(),
                Dense(128, activation='relu'),
                Dense(cifar10_parameters['latent_dim'], activation=None)
            ])
```

Decoder Architecture

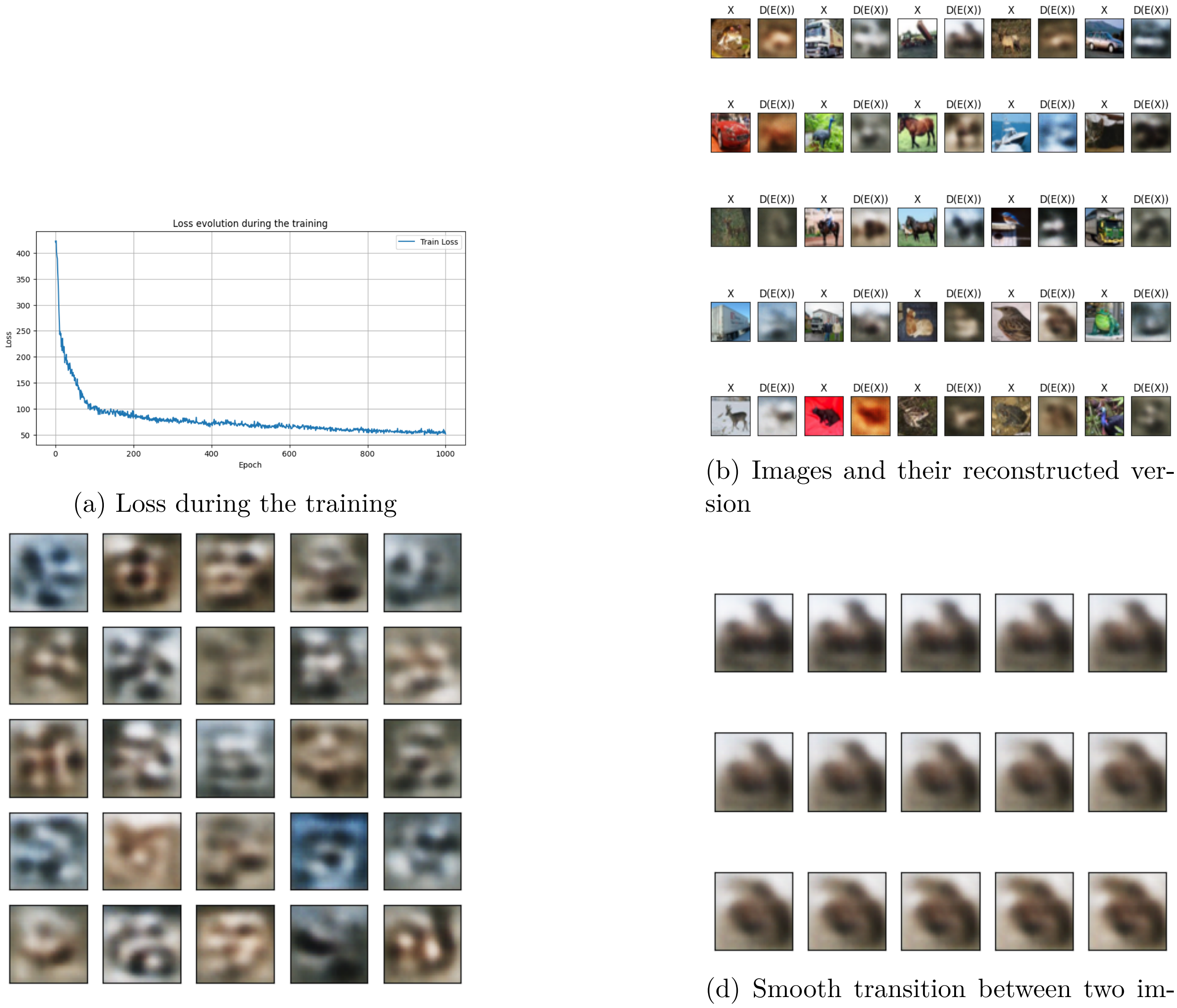
```
cifar10_decoder = tf.keras.Sequential([
                InputLayer(input_shape = cifar10_parameters['latent_dim']),
                Dense(128, activation='relu'),
                Dense(16*16*32, activation='relu'),
                Reshape((16,16,32)),
                Conv2DTranspose(32, (2, 2), padding = 'same', activation='relu'),
                Conv2DTranspose(32, (2, 2), padding = 'same', activation='relu'),
                Conv2DTranspose(32, (3, 3), strides = (2,2), activation='relu'),
                Conv2D(3, (2, 2), activation='sigmoid')
            ])
```

Model Parameters

```
cifar10_parameters = {'input_dim': train_images_cifar10[0].shape,
                      '1D data': False,
                      'latent_dim': 70}

cifar10_training_parameters = {'dataset': train_images_cifar10 / 255.0,
                               'learning_rate': 0.0005,
                               'epochs': 1000,
                               'batch_size': 200}
```

Results



(a) Loss during the training

(b) Images and their reconstructed version

(c) Generated images

(d) Smooth transition between two images

Figure 6: Test of the architecture and the parameters presented above for the CIFAR-10 dataset

.

7 Test with the Boston Housing dataset (1D data)

Encoder Architecture

```
boston_housing_encoder = tf.keras.Sequential([
                InputLayer(input_shape = boston_housing_parameters['
    input_dim']),
                Dense(256, activation='relu'),
                Dense(256, activation='relu'),
                Dense(256, activation='relu'),
                Dense(boston_housing_parameters['latent_dim'], activation=
    None)
            ])
```

Decoder Architecture

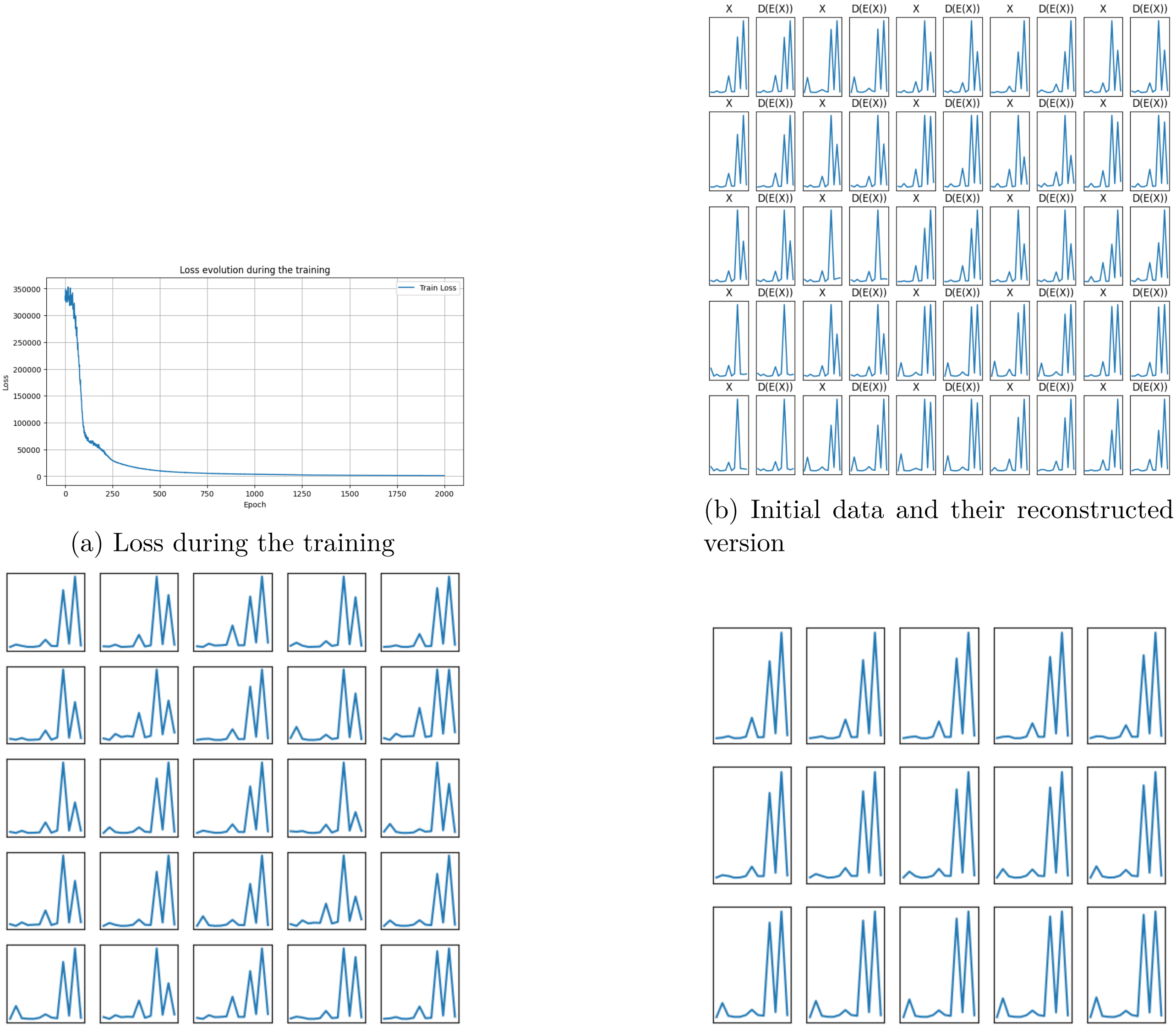
```
boston_housing_decoder = tf.keras.Sequential([
                InputLayer(input_shape = boston_housing_parameters['
    latent_dim']),
                Dense(256, activation='relu'),
                Dense(256, activation='relu'),
                Dense(256, activation='relu'),
                Dense(boston_housing_parameters['input_dim'], activation=
    None)
            ])
```

Model Parameters

```
boston_housing_parameters = {'input_dim': (train_data_boston_housing[0].
    shape)[0],
                             '1D data': True,
                             'latent_dim': 3}

boston_housing_training_parameters = {'dataset': train_data_boston_housing,
                                      'learning_rate': 0.00005,
                                      'epochs': 2000,
                                      'batch_size': 300}
```

Results



(a) Loss during the training

(b) Initial data and their reconstructed version

(c) Generated data

(d) Smooth transition between two data

Figure 7: Test of the architecture and the parameters presented above for 1D data of the Boston Housing data set