

Reference: Keras tutorial : <https://www.tensorflow.org/tutorials/generative/autoencoder> . Chapter 17 of Geron's book.

This file trains an autoencoder with the instances of the normal digit in the training data.

Then, it measures the reconstruction loss for the digits in the test data.

The reconstruction loss for the instances of the abnormal digit in the test data is higher.

A threshold is determined based on the distribution of the reconstruction losses of the normal training data (threshold = mean + 2.5*std of this distribution).

Then, if the reconstruction loss of a digit in the test data is higher than this threshold, it is classified as abnormal.

By comparing with the known labels of test data (with T for normal digit(s) and F for abnormal digit(s)), the confusion matrix and the accuracy is calculated.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras

from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix
from sklearn.model_selection import train_test_split
from keras import layers, losses
from keras.datasets import mnist
from keras.models import Model
# import cv2
```

Loading the MNIST data and forming arrays of the normal training data, validation data (normal and abnormal, and the test data (normal and abnormal))

```
#Labels
# 0 T-shirt/top
# 1 Trouser
# 2 Pullover
# 3 Dress
# 4 Coat
# 5 Sandal
# 6 Shirt
# 7 Sneaker
# 8 Bag
```

```
# 9 Ankle boot
```

```

nl1 = 0
nl2 = 2
nl3 = 6
abn = 7

(x_train_0, y_train_0), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()

x_train_0 = x_train_0.astype(np.float32) / 255
x_test = x_test.astype(np.float32) / 255

train_size = x_train_0.shape[0] * 9 // 10

x_train, x_valid, y_train, y_valid = train_test_split(x_train_0, y_train_0, train_size = train_size)

normal_data = x_train[(y_train == nl1) | (y_train == nl2) | (y_train == nl3)]      # Normal training data
normal_labels = y_train[(y_train == nl1) | (y_train == nl2) | (y_train == nl3)]

valid_data = x_valid[(y_valid == abn) | (y_valid == nl1) | (y_valid == nl2) | (y_valid == nl3)]
valid_labels = y_valid[(y_valid == abn) | (y_valid == nl1) | (y_valid == nl2) | (y_valid == nl3)]

test_data = x_test[(y_test == abn) | (y_test == nl1) | (y_test == nl2) | (y_test == nl3)]    # Test data
test_labels = y_test[(y_test == abn) | (y_test == nl1) | (y_test == nl2) | (y_test == nl3)]

test_labels_T_F = np.where((test_labels == nl1) | (test_labels == nl2) | (test_labels == nl3)) # Array of T and F, T where test digits are normal and F where test digits are abnormal

valid_labels_T_F = np.where((valid_labels == nl1) | (valid_labels == nl2) | (valid_labels == nl3)) # Array of T and F, T where test digits are normal and F where test digits are abnormal

normal_data.shape, normal_labels.shape, valid_data.shape, valid_labels.shape, test_data.shape

((16181, 28, 28), (16181,), (2417, 28, 28), (2417,), (4000, 28, 28), (4000,))

normal_test_data = test_data[(test_labels == nl1) | (test_labels == nl2) | (test_labels == nl3)] # Normal test data
abnormal_test_data = test_data[test_labels == abn]                                         # The abnormal digits
normal_test_labels = test_labels[(test_labels == nl1) | (test_labels == nl2) | (test_labels == nl3)]
abnormal_test_labels = test_labels[test_labels == abn]                                     # Their label

normal_test_data.shape, abnormal_test_data.shape

((3000, 28, 28), (1000, 28, 28))

normal_valid_data = valid_data[(valid_labels == nl1) | (valid_labels == nl2) | (valid_labels == nl3)] # Normal validation data
abnormal_valid_data = valid_data[valid_labels == abn]                                         # The abnormal digits
normal_valid_labels = valid_labels[(valid_labels == nl1) | (valid_labels == nl2) | (valid_labels == nl3)]
abnormal_valid_labels = valid_labels[valid_labels == abn]                                     # Their label

```

```
normal_valid_data.shape, abnormal_valid_data.shape
```

```
((1819, 28, 28), (598, 28, 28))
```

▼ Building and training the network

```
class AnomalyDetector(Model):
    def __init__(self):
        super(AnomalyDetector, self).__init__()
        self.encoder = tf.keras.Sequential([
            layers.Flatten(),
            layers.Dense(256, activation="selu"),
            layers.Dense(128, activation="selu"),
            layers.Dense(64, activation="selu"),
            layers.Dense(16, activation="selu")])

        self.decoder = tf.keras.Sequential([
            layers.Dense(64, activation="selu"),
            layers.Dense(128, activation="selu"),
            layers.Dense(256, activation="selu"),
            layers.Dense(28*28, activation="sigmoid"),
            layers.Reshape((28, 28))])

    def call(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

autoencoder = AnomalyDetector()

# autoencoder.compile(optimizer='adam', loss='mae')
autoencoder.compile(optimizer='rmsprop', loss='binary_crossentropy')

checkpoint_cb = keras.callbacks.ModelCheckpoint("AE_model", monitor="val_loss", save_best_only=True)

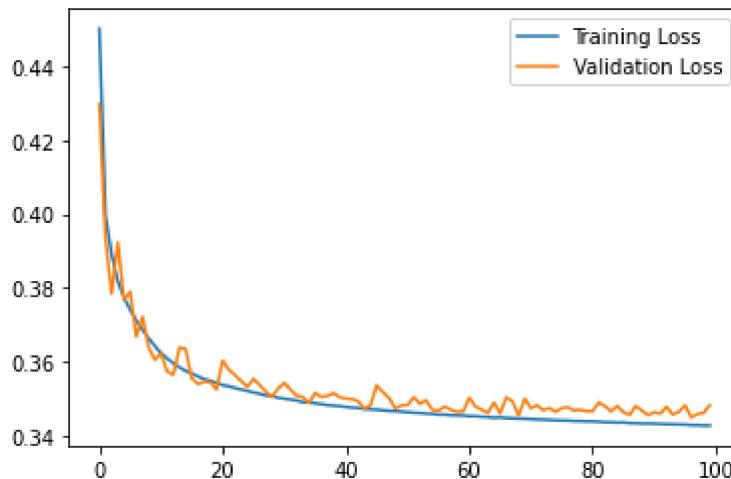
history = autoencoder.fit(normal_data, normal_data,
                           epochs=100,
                           batch_size=128,
                           validation_data=(normal_valid_data, normal_valid_data),
                           callbacks=[checkpoint_cb],
                           shuffle=True)

Epoch 1/100
127/127 [=====] - 1s 7ms/step - loss: 0.3444 - val_loss: 0.3444
Epoch 2/100
127/127 [=====] - 1s 7ms/step - loss: 0.3444 - val_loss: 0.3444
Epoch 3/100
127/127 [=====] - 1s 7ms/step - loss: 0.3444 - val_loss: 0.3444
Epoch 4/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 5/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 6/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 7/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 8/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 9/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 10/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 11/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 12/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 13/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 14/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 15/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 16/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 17/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 18/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 19/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 20/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 21/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 22/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 23/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 24/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 25/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 26/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 27/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 28/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 29/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 30/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 31/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 32/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 33/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 34/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 35/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 36/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 37/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 38/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 39/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 40/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 41/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 42/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 43/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 44/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 45/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 46/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 47/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 48/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 49/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 50/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 51/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 52/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 53/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 54/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 55/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 56/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 57/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 58/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 59/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 60/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 61/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 62/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 63/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 64/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 65/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 66/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 67/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 68/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 69/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 70/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 71/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 72/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 73/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 74/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 75/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 76/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 77/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 78/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 79/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 80/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 81/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 82/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 83/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 84/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 85/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 86/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 87/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 88/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 89/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 90/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 91/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 92/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 93/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 94/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 95/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 96/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 97/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 98/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 99/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
Epoch 100/100
127/127 [=====] - 1s 7ms/step - loss: 0.3443 - val_loss: 0.3443
```

```
127/127 [=====] - 1s 7ms/step - loss: 0.3442 - val_loss: 0.3442
Epoch 76/100
127/127 [=====] - 1s 7ms/step - loss: 0.3441 - val_loss: 0.3441
Epoch 77/100
127/127 [=====] - 1s 7ms/step - loss: 0.3440 - val_loss: 0.3440
Epoch 78/100
127/127 [=====] - 1s 7ms/step - loss: 0.3439 - val_loss: 0.3439
Epoch 79/100
127/127 [=====] - 1s 7ms/step - loss: 0.3439 - val_loss: 0.3439
Epoch 80/100
127/127 [=====] - 1s 7ms/step - loss: 0.3438 - val_loss: 0.3438
Epoch 81/100
127/127 [=====] - 1s 7ms/step - loss: 0.3437 - val_loss: 0.3437
Epoch 82/100
127/127 [=====] - 1s 7ms/step - loss: 0.3436 - val_loss: 0.3436
Epoch 83/100
127/127 [=====] - 1s 7ms/step - loss: 0.3436 - val_loss: 0.3436
Epoch 84/100
127/127 [=====] - 1s 7ms/step - loss: 0.3434 - val_loss: 0.3434
Epoch 85/100
127/127 [=====] - 1s 7ms/step - loss: 0.3434 - val_loss: 0.3434
Epoch 86/100
127/127 [=====] - 1s 7ms/step - loss: 0.3434 - val_loss: 0.3434
Epoch 87/100
127/127 [=====] - 1s 7ms/step - loss: 0.3433 - val_loss: 0.3433
Epoch 88/100
127/127 [=====] - 1s 7ms/step - loss: 0.3433 - val_loss: 0.3433
Epoch 89/100
127/127 [=====] - 1s 7ms/step - loss: 0.3433 - val_loss: 0.3433
Epoch 90/100
127/127 [=====] - 1s 7ms/step - loss: 0.3431 - val_loss: 0.3431
Epoch 91/100
127/127 [=====] - 1s 7ms/step - loss: 0.3431 - val_loss: 0.3431
Epoch 92/100
127/127 [=====] - 1s 7ms/step - loss: 0.3431 - val_loss: 0.3431
Epoch 93/100
127/127 [=====] - 1s 7ms/step - loss: 0.3431 - val_loss: 0.3431
Epoch 94/100
127/127 [=====] - 1s 7ms/step - loss: 0.3431 - val_loss: 0.3431
Epoch 95/100
127/127 [=====] - 1s 7ms/step - loss: 0.3429 - val_loss: 0.3429
INFO:tensorflow:Assets written to: /tmp/tensorflow2_1645344587/assets
127/127 [=====] - 3s 21ms/step - loss: 0.3429 - val_loss: 0.3429
Epoch 96/100
127/127 [=====] - 1s 7ms/step - loss: 0.3429 - val_loss: 0.3429
Epoch 97/100
127/127 [=====] - ETA: 0s - loss: 0.3429
127/127 [=====] - 3s 21ms/step - loss: 0.3429 - val_loss: 0.3429
Epoch 98/100
127/127 [=====] - 1s 7ms/step - loss: 0.3428 - val_loss: 0.3428
Epoch 99/100
127/127 [=====] - 1s 7ms/step - loss: 0.3428 - val_loss: 0.3428
Epoch 100/100
```

```
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.legend()
```

<matplotlib.legend.Legend at 0x7f5bcd9c4d50>



```
model = autoencoder
model.summary(expand_nested=True, show_trainable=True)
```

Model: "anomaly_detector_4"

Layer (type)	Output Shape	Param #	Trainable
<hr/>			
sequential_8 (Sequential)	(None, 16)	243152	Y

flatten_4 (Flatten)	(None, 784)	0	Y
dense_32 (Dense)	(None, 256)	200960	Y
dense_33 (Dense)	(None, 128)	32896	Y
dense_34 (Dense)	(None, 64)	8256	Y
dense_35 (Dense)	(None, 16)	1040	Y

sequential_9 (Sequential)	(None, 28, 28)	243920	Y

dense_36 (Dense)	(None, 64)	1088	Y
dense_37 (Dense)	(None, 128)	8320	Y
dense_38 (Dense)	(None, 256)	33024	Y
dense_39 (Dense)	(None, 784)	201488	Y
reshape_4 (Reshape)	(None, 28, 28)	0	Y

<hr/>			
Total params:	487,072		
Trainable params:	487,072		
Non-trainable params:	0		

```

model_encoder = autoencoder.encoder
# model_encoder.summary(expand_nested=True, show_trainable=True)

model_decoder = autoencoder.decoder
# model_decoder.summary(expand_nested=True, show_trainable=True)

model_layers = np.array(model.layers)
n_layers = model_layers.shape[0]
# np.concatenate((np.arange(n_layers).reshape(n_layers,1), model_layers.reshape(n_layers,1)),

```

The original and reconstructed images for the first 30 instances of the normal training data, validation data, normal validation data, abnormal validation data, test data, normal test data, and abnormal test data

```

def plot_image(image):
    plt.imshow(image, cmap="binary")
    plt.axis("off")

def show_reconstructions(autoencoder, images, n_images=5):
    encoded_data = autoencoder.encoder(images[:n_images]).numpy()
    decoded_data = autoencoder.decoder(encoded_data).numpy()
    reconstructions = decoded_data
    fig = plt.figure(figsize=(n_images * 1.5, 3))
    for image_index in range(n_images):
        plt.subplot(2, n_images, 1 + image_index)
        plot_image(images[image_index])
        plt.subplot(2, n_images, 1 + n_images + image_index)
        plot_image(reconstructions[image_index])

show_reconstructions(autoencoder, normal_data, 30)
plt.show()

```



```

show_reconstructions(autoencoder, valid_data, 30)
plt.show()

```



```

show_reconstructions(autoencoder, normal_valid_data, 30)

```

```
plt.show()
```



```
show_reconstructions(autoencoder, abnormal_valid_data, 30)
```

```
plt.show()
```



```
show_reconstructions(autoencoder, test_data, 30)
```

```
plt.show()
```



```
show_reconstructions(autoencoder, normal_test_data, 30)
```

```
plt.show()
```



```
show_reconstructions(autoencoder, abnormal_test_data, 30)
```

```
plt.show()
```

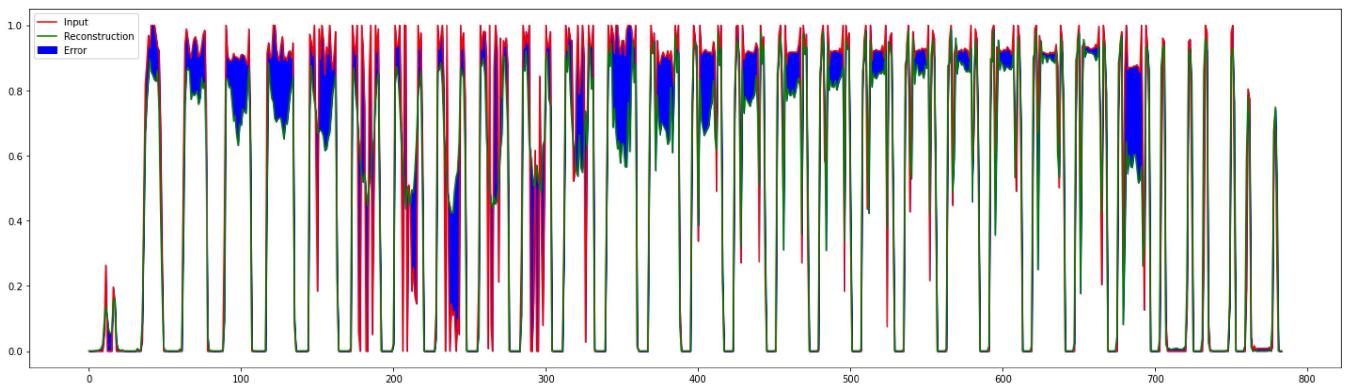


```
encoded_data = autoencoder.encoder(normal_test_data).numpy()
```

```
decoded_data = autoencoder.decoder(encoded_data).numpy()
```

1-Dim plot of pixels of the first normal test data

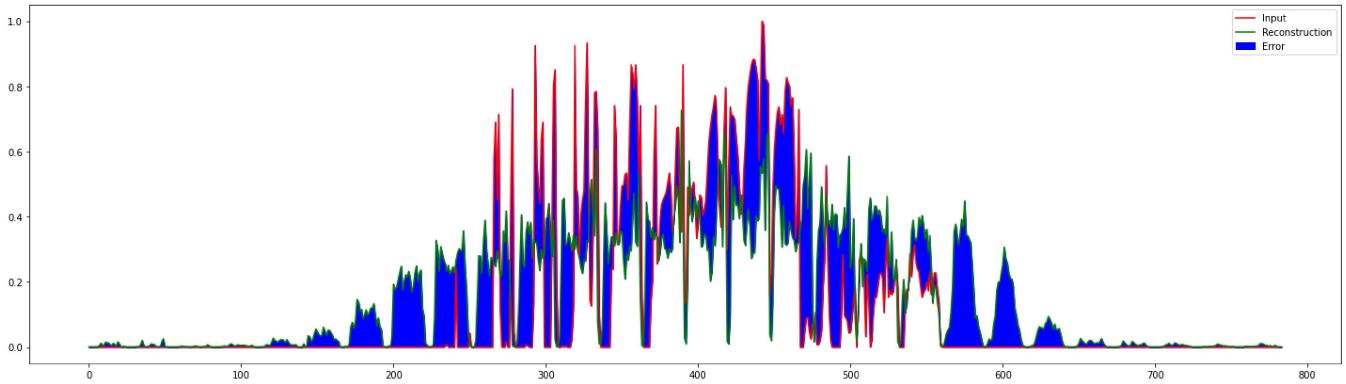
```
plt.figure(figsize=(25,7))
plt.plot(normal_test_data[0].ravel(), 'r')
plt.plot(decoded_data[0].ravel(), 'g')
plt.fill_between(np.arange(28*28), decoded_data[0].ravel(), normal_test_data[0].ravel(), colo
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```



```
encoded_abn_data = autoencoder.encoder(abnormal_test_data).numpy()
decoded_abn_data = autoencoder.decoder(encoded_abn_data).numpy()
```

1-Dim plot of pixels of the first abnormal test data

```
plt.figure(figsize=(25,7))
plt.plot(abnormal_test_data[0].ravel(), 'r')
plt.plot(decoded_abn_data[0].ravel(), 'g')
plt.fill_between(np.arange(28*28), decoded_abn_data[0].ravel(), abnormal_test_data[0].ravel())
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```

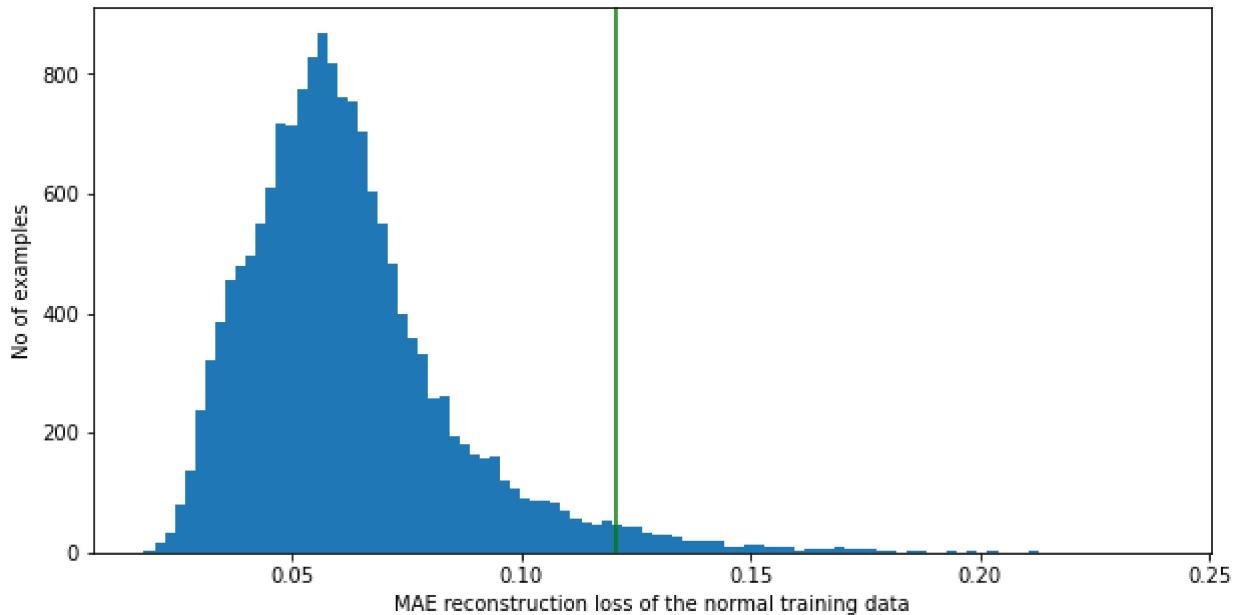


▼ Distributions of the reconstruction losses and the calculation of the threshold.

Distribution of the reconstruction losses of the normal training data

```
reconstructions = autoencoder.predict(normal_data)
train_loss = tf.keras.losses.mae(reconstructions.reshape(-1, 784), normal_data.reshape(-1, 784))
plt.figure(figsize=(10,5))
plt.hist(train_loss[None,:], bins=100)
```

```
threshold1 = np.mean(train_loss) + 2.5*np.std(train_loss)
plt.axvline(threshold1,c='g')
plt.xlabel("MAE reconstruction loss of the normal training data")
plt.ylabel("No of examples")
plt.show()
```



```
mean1 = np.mean(train_loss)
std1 = np.std(train_loss)
print("Mean: ", mean1)
print("Std: ", std1)
```

```
Mean:  0.062465515
Std:  0.023326473
```

```
threshold_train_mean_2_5_std = np.mean(train_loss) + 2.5*np.std(train_loss)
print("Threshold based on the mean of the training data MAE reconstruction losses + 2.5 std:
```

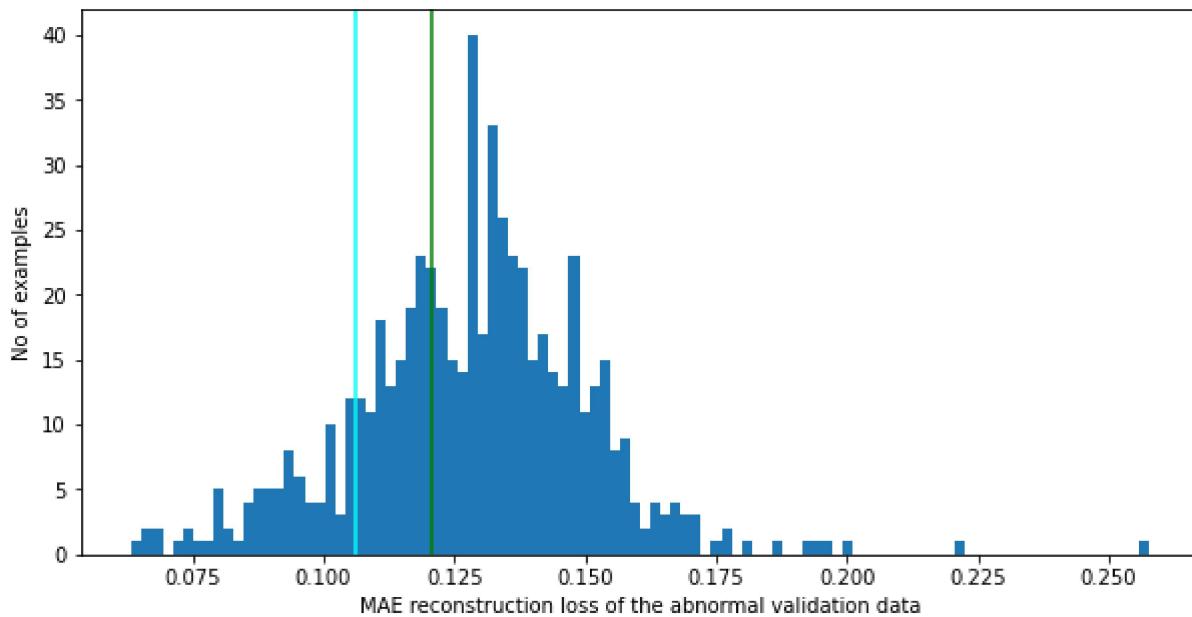
```
Threshold based on the mean of the training data MAE reconstruction losses + 2.5 std: 6
```

```
threshold1 = threshold_train_mean_2_5_std
```

Distribution of the reconstruction losses of the abnormal validation data

```
reconstructions = autoencoder.predict(abnormal_valid_data)
abn_valid_loss = tf.keras.losses.mae(reconstructions.reshape(-1,784), abnormal_valid_data.reshape(-1,784))
plt.figure(figsize=(10,5))
plt.hist(abn_valid_loss[None, :], bins=100)
threshold2 = np.mean(abn_valid_loss) - np.std(abn_valid_loss)
plt.axvline(threshold2,c='cyan')
```

```
plt.axvline(threshold1,c='g')
plt.xlabel("MAE reconstruction loss of the abnormal validation data")
plt.ylabel("No of examples")
plt.show()
```



```
abnormal_valid_mean_loss = np.mean(abn_valid_loss)
```

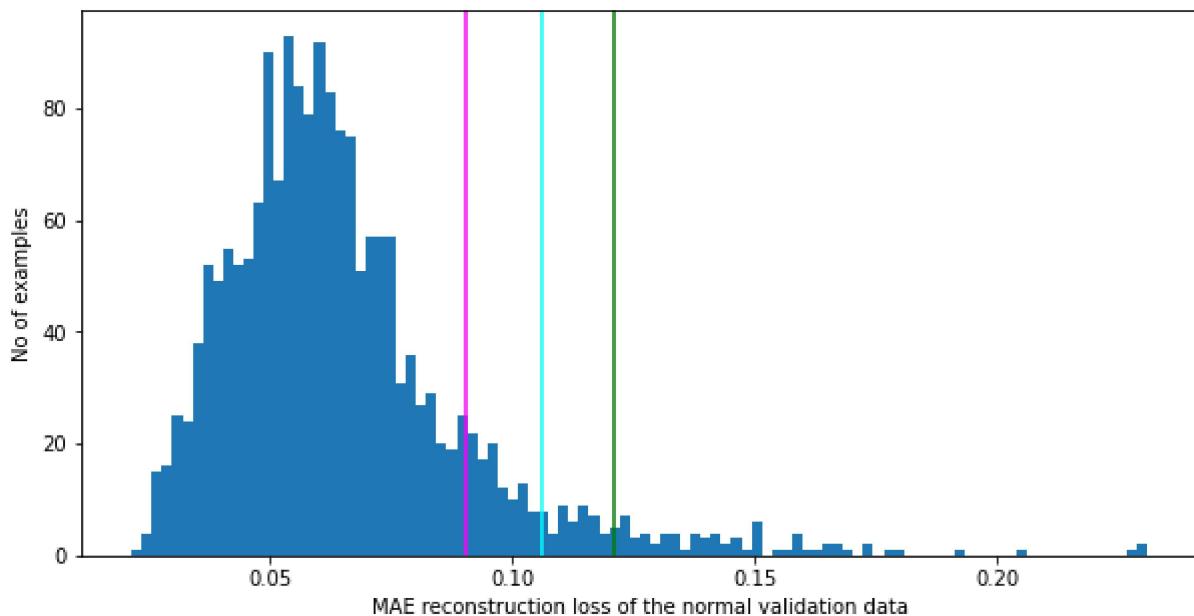
```
abnormal_valid_mean_loss , np.std(abn_valid_loss)
(0.12827927, 0.022336965)
```

```
threshold2 = abnormal_valid_mean_loss - np.std(abn_valid_loss)
print("Threshold2: ", threshold2)
```

```
Threshold2: 0.1059423
```

Distribution of the reconstruction losses of the normal validation data

```
reconstructions = autoencoder.predict(normal_valid_data)
nl_valid_loss = tf.keras.losses.mae(reconstructions.reshape(-1,784), normal_valid_data.reshape(-1,784))
plt.figure(figsize=(10,5))
plt.hist(nl_valid_loss[None, :], bins=100)
threshold3 = np.mean(nl_valid_loss) + np.std(nl_valid_loss)
plt.axvline(threshold3, c='magenta')
plt.axvline(threshold2, c='cyan')
plt.axvline(threshold1, c='g')
plt.xlabel("MAE reconstruction loss of the normal validation data")
plt.ylabel("No of examples")
plt.show()
```



```
normal_valid_mean_loss = np.mean(nl_valid_loss)
```

```
normal_valid_mean_loss , np.std(nl_valid_loss)
```

```
(0.06484259, 0.02563468)
```

```
threshold3 = normal_valid_mean_loss + np.std(nl_valid_loss)
print("Threshold3: ", threshold3)
```

```
Threshold3: 0.09047727
```

Calculation of a preliminary threshold based on $(\text{threshold2} + \text{threshold3}) / 2$ = Average of (mean + std of the distribution of the reconstruction losses of the normal validation data) and (mean - std of the distribution of the reconstruction losses of the abnormal validation data)

```
Avg_of_threshold_2_3 = (threshold2 + threshold3)/2
print("Average of threshold 2 and 3: ", Avg_of_threshold_2_3)
```

```
Average of threshold 2 and 3: 0.09820978343486786
```

```
threshold4 = Avg_of_threshold_2_3
```

Calculation of the threshold that gives the best accuracy on the validation data and set this as the threshold.

```
def predict(model, data, threshold):
```

```

reconstructions = model.predict(data)
loss = tf.keras.losses.mae(reconstructions.reshape(-1, 784), data.reshape(-1, 784))
return tf.math.less(loss, threshold)

```

```

increment = (abnormal_valid_mean_loss - normal_valid_mean_loss)/100
thresholds = np.arange(normal_valid_mean_loss, abnormal_valid_mean_loss, increment)
thrs_size = thresholds.shape[0]
accuracies = np.zeros(thrs_size)
for i in range(thrs_size):
    preds = predict(autoencoder, valid_data, thresholds[i])
    accuracies[i] = accuracy_score(preds, valid_labels_T_F)
argmax = np.argmax(accuracies)
valid_data_best_threshold = thresholds[argmax]
print("The best threshold based on validation data: ", valid_data_best_threshold)

```

The best threshold based on validation data: 0.10417333051562289

```

thr_acc = np.zeros((thrs_size, 2))
thr_acc[:, 0] = thresholds
thr_acc[:, 1] = accuracies
thr_acc[argmax-2:argmax+3]

array([[0.1029046 , 0.91642532],
       [0.10353896, 0.91683906],
       [0.10417333, 0.91808026],
       [0.1048077 , 0.91766653],
       [0.10544206, 0.91683906]])

```

threshold5 = valid_data_best_threshold

threshold = threshold5

Distribution of the reconstruction losses of all the validation data (normal and abnormal)

The blue line is threshold4 (= the average of threshold3 [magenta] and threshold2 [cyan]).

The red line is the threshold that gives the best accuracy for the validation data.

```

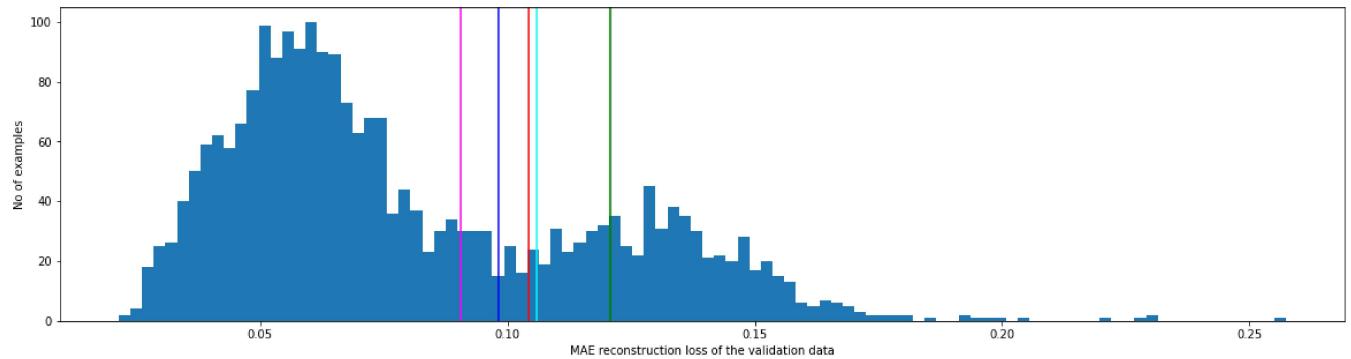
reconstructions = autoencoder.predict(valid_data)
valid_loss = tf.keras.losses.mae(reconstructions.reshape(-1,784), valid_data.reshape(-1,784))
plt.figure(figsize=(20,5))
plt.hist(valid_loss[None, :], bins=100)
plt.axvline(threshold, c='r')
plt.axvline(threshold4, c='b')

```

```

plt.axvline(threshold2, c='cyan')
plt.axvline(threshold3, c='magenta')
plt.axvline(threshold1, c='green')
plt.xlabel("MAE reconstruction loss of the validation data")
plt.ylabel("No of examples")
plt.show()

```



▼ Distribution of the reconstruction losses of the test data (normal and abnormal)

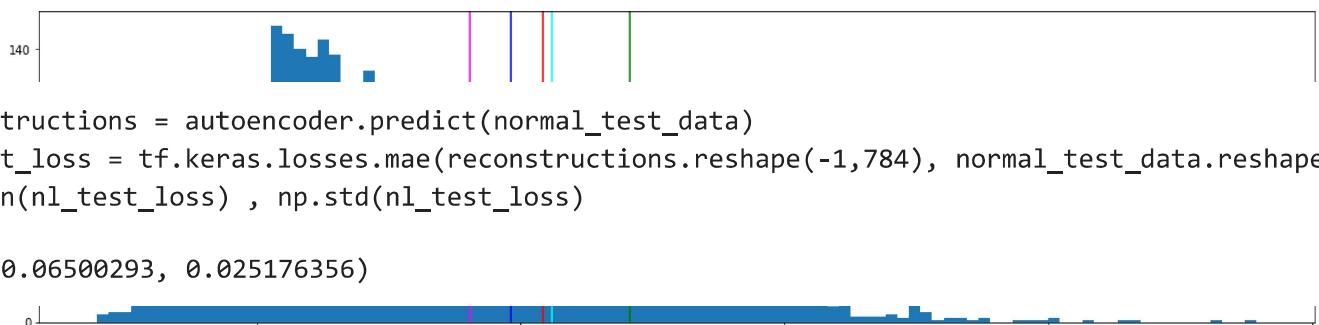
The blue line is threshold4 (= the average of threshold3 [magenta] and threshold2 [cyan]).

The red line is the threshold that gives the best accuracy for the validation data.

```

reconstructions = autoencoder.predict(test_data)
test_loss = tf.keras.losses.mae(reconstructions.reshape(-1,784), test_data.reshape(-1,784))
plt.figure(figsize=(20,5))
plt.hist(test_loss[None, :], bins=100)
plt.axvline(threshold, c='r')
plt.axvline(threshold4, c='b')
plt.axvline(threshold2, c='cyan')
plt.axvline(threshold3, c='magenta')
plt.axvline(threshold1, c='green')
plt.xlabel("MAE reconstruction loss of the test data")
plt.ylabel("No of examples")
plt.show()

```



```

reconstructions = autoencoder.predict(normal_test_data)
nl_test_loss = tf.keras.losses.mae(reconstructions.reshape(-1,784), normal_test_data.reshape(
np.mean(nl_test_loss) , np.std(nl_test_loss)

(0.06500293, 0.025176356)

reconstructions = autoencoder.predict(abnormal_test_data)
abn_test_loss = tf.keras.losses.mae(reconstructions.reshape(-1,784), abnormal_test_data.resha
np.mean(abn_test_loss) , np.std(abn_test_loss)

(0.12834202, 0.018991478)

```

Calculation of the accuracy and the confusion matrix on the test data with threshold set based on the best threshold from the validation data

```

# def predict(model, data, threshold):
#   reconstructions = model.predict(data)
#   loss = tf.keras.losses.mae(reconstructions.reshape(-1, 784), data.reshape(-1, 784))
#   return tf.math.less(loss, threshold)

mean_nl_test_loss = np.mean(nl_test_loss)
std_nl_test_loss = np.std(nl_test_loss)

mean_abn_test_loss = np.mean(abn_test_loss)
std_abn_test_loss = np.std(abn_test_loss)

def print_stats(predictions, labels):
    cf = confusion_matrix(labels, predictions)

    print("mean_nl_test_loss: {}".format(mean_nl_test_loss))
    print("std_nl_test_loss: {}".format(std_nl_test_loss))
    print("mean_abn_test_loss: {}".format(mean_abn_test_loss))
    print("std_abn_test_loss: {}".format(std_abn_test_loss))

    print("Confusion Matrix: \n prediction: F      T ")
    print("          {}  {}".format(preds[preds == False].shape[0], preds[preds == True].sh
    print(" label: F  [[{}  {}]  {}]".format(cf[0,0], cf[0,1], test_labels_T_F[test_labels_T_
    print("          T  [{}  {}]]  {}".format(cf[1,0], cf[1,1], test_labels_T_F[test_labels_T_
    print("Accuracy = {}".format(accuracy_score(labels, predictions)))
    print("Precision = {}".format(precision_score(labels, predictions)))
    print("Recall = {}".format(recall_score(labels, predictions)))

```

```
preds = predict(autoencoder, test_data, validation_data=validation)
print_stats(preds, test_labels_T_F)
```

```
↳ mean_nl_test_loss: 0.0650029331445694
  std_nl_test_loss: 0.025176355615258217
  mean_abn_test_loss: 0.12834201753139496
  std_abn_test_loss: 0.01899147778749466
  Confusion Matrix:
    prediction: F      T
                  1087   2913
    label: F   [[891   109]     1000
                  T   [196   2804]]     3000
  Accuracy = 0.92375
  Precision = 0.9625815310676279
  Recall = 0.9346666666666666
```

```
print(confusion_matrix(test_labels_T_F, preds))
```

```
[[ 891  109]
 [ 196  2804]]
```

Extra accuracy info

Just informative. Please record the above accuracy.

Accuracy on the test data with threshold set based on (threshold2 + threshold3) / 2 = Average of (mean + std of the distribution of the reconstruction losses of the normal validation data) and (mean - std of the distribution of the reconstruction losses of the abnormal validation data)

```
preds = predict(autoencoder, test_data, Avg_of_threshold_2_3)
print_stats(preds, test_labels_T_F)
```

```
mean_nl_test_loss: 0.0650029331445694
  std_nl_test_loss: 0.025176355615258217
  mean_abn_test_loss: 0.12834201753139496
  std_abn_test_loss: 0.01899147778749466
  Confusion Matrix:
    prediction: F      T
                  1201   2799
    label: F   [[939   61]     1000
                  T   [262   2738]]     3000
  Accuracy = 0.91925
  Precision = 0.9782065023222579
  Recall = 0.9126666666666666
```

Accuracy on the test data with threshold set based on the mean of the training data MAE

```
preds = predict(autoencoder, test_data, threshold_train_mean_2_5_std)
print_stats(preds, test_labels_T_F)

mean_nl_test_loss: 0.0650029331445694
std_nl_test_loss: 0.025176355615258217
mean_abn_test_loss: 0.12834201753139496
std_abn_test_loss: 0.01899147778749466
Confusion Matrix:
prediction: F      T
            799    3201
label: F   [[691    309]    1000
            T   [108    2892]]   3000
Accuracy = 0.89575
Precision = 0.9034676663542643
Recall = 0.964
```

✓ 0s completed at 11:28 AM

● ✘