

**References:** Chapter 17 of Geron's book. For 1-Dim plots, Keras tutorial :

<https://www.tensorflow.org/tutorials/generative/autoencoder>

This file trains a modified VAE (with a different sampling layer and a different loss function) with the instances of the normal digits 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 digits 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 sklearn
import tensorflow as tf
from tensorflow import keras
import numpy as np

import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix
from sklearn.model_selection import train_test_split
```

**Loading the MNIST data and forming arrays of the normal training data, the 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 = 5
nl2 = 7
nl3 = 9
abn1 = 4
abn2 = 4

(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
normal_labels = y_train[(y_train == nl1) | (y_train == nl2) | (y_train == nl3)]

valid_data = x_valid[(y_valid == abn1) | (y_valid == abn2) | (y_valid == nl1) | (y_valid == nl2)]
valid_labels = y_valid[(y_valid == abn1) | (y_valid == abn2) | (y_valid == nl1) | (y_valid == nl2)]

test_data = x_test[(y_test == abn1) | (y_test == abn2) | (y_test == nl1) | (y_test == nl2)]
test_labels = y_test[(y_test == abn1) | (y_test == abn2) | (y_test == nl1) | (y_test == nl2)]

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

((16232, 28, 28), (16232,), (2347, 28, 28), (2347,), (4000, 28, 28), (4000,))

normal_test_data = test_data[(test_labels == nl1) | (test_labels == nl2) | (test_labels == nl3)]
abnormal_test_data = test_data[(test_labels == abn1) | (test_labels == abn2)]           # The abnormal
normal_test_labels = test_labels[(test_labels == nl1) | (test_labels == nl2) | (test_labels == nl3)]
abnormal_test_labels = test_labels[(test_labels == abn1) | (test_labels == abn2)]      # Their

normal_test_data.shape, abnormal_test_data.shape

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

```

```

normal_valid_data = valid_data[(valid_labels == n11) | (valid_labels == n12) | (valid_labels == n13)]
abnormal_valid_data = valid_data[(valid_labels == abn1) | (valid_labels == abn2)] # Th
normal_valid_labels = valid_labels[(valid_labels == n11) | (valid_labels == n12) | (valid_labels == n13)]
abnormal_valid_labels = valid_labels[(valid_labels == abn1) | (valid_labels == abn2)] # Th

normal_valid_data.shape, abnormal_valid_data.shape

((1768, 28, 28), (579, 28, 28))

```

## ▼ Building and training the network

```

K = keras.backend
# def rounded_accuracy(y_true, y_pred):
#   return keras.metrics.binary_accuracy(tf.round(y_true), tf.round(y_pred))

# Modified sampling layer with the addition of mean_2, log_var_2, and fraction p, with
# the appropriate change in the reparametrization trick to do stochastic
# sampling from the superposition of the two MVN distributions, while allowing
# the 5 parallel layers containing the means and stds of the two MVNs and the fractions p's
# for each dimension to be trained via backpropogation of the error signal.
class Sampling(keras.layers.Layer):
    def call(self, inputs):
        mean_1, log_var_1, mean_2, log_var_2, p = inputs
        return (K.random_normal(tf.shape(log_var_1)) * K.exp(log_var_1 / 2) + mean_1)*p + (K.

# For details please see Geron's book.
codings_size = 16 # The number of dimensions of the two MVN distributions in the sampling la

inputs = keras.layers.Input(shape=[28, 28])
z = keras.layers.Flatten()(inputs)
z = keras.layers.Dense(256, activation="selu")(z)
z = keras.layers.Dense(128, activation="selu")(z)
z = keras.layers.Dense(64, activation="selu")(z)

# Adding output nodes (parallel layers) at the end of the encoder for means
# and standard deviations of a second Multivariate Normal (MVN) distribution
# in the dimensions of the coding size (here 32). In each of the dimensions,
# this first MVN is multiplied by a fraction p and added to the second MVN
# multiplied by 1 - p in each dimension.
# final distribution = p * first MVN + (1 - p) * second MVN
# Another parallel layer (set of nodes) is added to keep and train the fractions p's
# in each dimension
codings_mean_1 = keras.layers.Dense(codings_size)(z)
codings_log_var_1 = keras.layers.Dense(codings_size)(z)
codings_mean_2 = keras.layers.Dense(codings_size)(z)
codings_log_var_2 = keras.layers.Dense(codings_size)(z)
codings_p = keras.layers.Dense(1, activation='sigmoid')(z)

```

```

# Modified sampling layer at the end of the encoder
codings = Sampling()([codings_mean_1, codings_log_var_1, codings_mean_2, codings_log_var_2, c
variational_encoder = keras.models.Model(
    inputs=[inputs], outputs=[codings_mean_1, codings_log_var_1, codings_mean_2, codings_log_]

decoder_inputs = keras.layers.Input(shape=[codings_size])
x = keras.layers.Dense(64, activation="selu")(decoder_inputs)
x = keras.layers.Dense(128, activation="selu")(x)
x = keras.layers.Dense(256, activation="selu")(x)
x = keras.layers.Dense(28 * 28, activation="sigmoid")(x)
outputs = keras.layers.Reshape([28, 28])(x)
variational_decoder = keras.models.Model(inputs=[decoder_inputs], outputs=[outputs])

_, _, _, _, codings = variational_encoder(inputs)
reconstructions = variational_decoder(codings)
variational_ae = keras.models.Model(inputs=[inputs], outputs=[reconstructions])

# New latent loss function that will be added to the reconstruction binary cross-entropy loss
# The whole network (Encoder, sampling layer, and decoder) will train to minimize this loss
p_mean = K.mean(codings_p)
array1 = p_mean*(codings_log_var_1 - K.exp(codings_log_var_1) - K.square(codings_mean_1))
array2 = (1-p_mean)*(codings_log_var_2 - K.exp(codings_log_var_2) - K.square(codings_mean_2))
sum1 = K.sum(1 + array1, axis=-1)
sum2 = K.sum(1 + array2, axis=-1)

# latent_loss = -0.5 * tf.math.maximum(sum1, sum2)

latent_loss = -0.5 * (sum1 + sum2)

latent_loss = latent_loss * 0.5

# Add the latent loss to the reconstruction loss
variational_ae.add_loss(K.mean(latent_loss) / 784.)

# For the reconstruction loss binary cross-entropy loss is used (same as regular VAE).
# For details please see Chapter 17 of Geron's book (Stacked AE and VAE sections)
variational_ae.compile(loss="binary_crossentropy", optimizer="rmsprop")

checkpoint_cb = keras.callbacks.ModelCheckpoint("modVAE_latent_times_half_model", monitor="va

history = variational_ae.fit(normal_data, normal_data, epochs=100, batch_size=128, callbacks=
    validation_data=(normal_valid_data, normal_valid_data), shuffle=
    True)

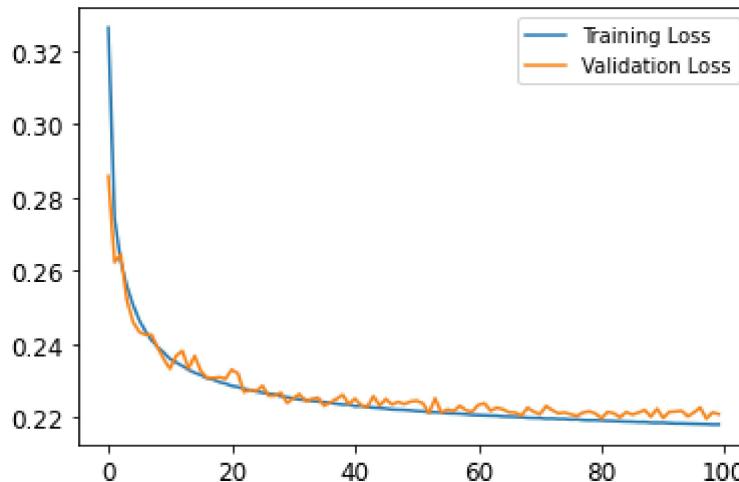
127/127 [=====] - 1s 8ms/step - loss: 0.2197 - val_loss: 0.2
Epoch 74/100
127/127 [=====] - 1s 8ms/step - loss: 0.2195 - val_loss: 0.2
Epoch 75/100
127/127 [=====] - 1s 8ms/step - loss: 0.2195 - val_loss: 0.2
Epoch 76/100
127/127 [=====] - 1s 8ms/step - loss: 0.2194 - val_loss: 0.2
Epoch 77/100
127/127 [=====] - 1s 8ms/step - loss: 0.2194 - val_loss: 0.2

```

```
127/127 [=====] - ETA: 0s - loss: 0.2194 - val_loss: 0.  
Epoch 78/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2192 - val_loss: 0.2  
Epoch 79/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2192 - val_loss: 0.2  
Epoch 80/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2192 - val_loss: 0.2  
Epoch 81/100  
125/127 [=====>.] - ETA: 0s - loss: 0.2191INFO:tensorflow:Asse  
127/127 [=====] - 4s 30ms/step - loss: 0.2191 - val_loss: 0.  
Epoch 82/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2190 - val_loss: 0.2  
Epoch 83/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2190 - val_loss: 0.2  
Epoch 84/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2189 - val_loss: 0.2  
Epoch 85/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2188 - val_loss: 0.2  
Epoch 86/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2188 - val_loss: 0.2  
Epoch 87/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2187 - val_loss: 0.2  
Epoch 88/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2187 - val_loss: 0.2  
Epoch 89/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2185 - val_loss: 0.2  
Epoch 90/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2185 - val_loss: 0.2  
Epoch 91/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2185 - val_loss: 0.2  
Epoch 92/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2184 - val_loss: 0.2  
Epoch 93/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2183 - val_loss: 0.2  
Epoch 94/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2183 - val_loss: 0.2  
Epoch 95/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2182 - val_loss: 0.2  
Epoch 96/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2182 - val_loss: 0.2  
Epoch 97/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2182 - val_loss: 0.2  
Epoch 98/100  
127/127 [=====] - ETA: 0s - loss: 0.2181INFO:tensorflow:Asse  
127/127 [=====] - 3s 27ms/step - loss: 0.2181 - val_loss: 0.  
Epoch 99/100  
127/127 [=====] - 1s 8ms/step - loss: 0.2180 - val_loss: 0.2  
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 0x7ff697cb9390>
```



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

tf.math.subtract_10 (TFOpLambd a)	(None, 16)	0	['dense_28[0][0]', 'tf.math.exp_4[0][0]
tf.math.square_4 (TFOpLambda)	(None, 16)	0	['dense_27[0][0]']
tf.math.subtract_13 (TFOpLambd a)	(None, 16)	0	['dense_30[0][0]', 'tf.math.exp_5[0][0]
tf.math.square_5 (TFOpLambda)	(None, 16)	0	['dense_29[0][0]']
tf.math.subtract_11 (TFOpLambd a)	(None, 16)	0	['tf.math.subtract_10', 'tf.math.square_4[0][0]
tf.math.subtract_12 (TFOpLambd a)	()	0	['tf.math.reduce_mean']
tf.math.subtract_14 (TFOpLambd a)	(None, 16)	0	['tf.math.subtract_11', 'tf.math.square_5[0][0]
tf.math.multiply_8 (TFOpLambda )	(None, 16)	0	['tf.math.reduce_mean', 'tf.math.subtract_14']
tf.math.multiply_9 (TFOpLambda )	(None, 16)	0	['tf.math.subtract_11', 'tf.math.subtract_14']
tf.__operators__.add_6 (TFOpLa mbda)	(None, 16)	0	['tf.math.multiply_8', 'tf.math.subtract_14']
tf.__operators__.add_7 (TFOpLa mbda)	(None, 16)	0	['tf.math.multiply_9', 'tf.math.subtract_14']
tf.math.reduce_sum_4 (TFOpLamb da)	(None, )	0	['tf.__operators__.a dd']
tf.math.reduce_sum_5 (TFOpLamb da)	(None, )	0	['tf.__operators__.a dd']

```

tf.__operators__.add_8 (TFOpLa (None, )
mbda) 0 ['tf.math.reduce_sum',
          'tf.math.reduce_sum']

tf.math.multiply_10 (TFOpLambd (None, )
a) 0 ['tf.__operators__.a']

tf.math.multiply_11 (TFOpLambd (None, )
a) 0 ['tf.math.multiply_10']

tf.math.reduce_mean_5 (TFOpLam () 0 ['tf.math.multiply_10']

bda)

tf.math.truediv_2 (TFOpLambda) () 0 ['tf.math.reduce_mean']

add_loss_2 (AddLoss) () 0 ['tf.math.truediv_2[0]']

=====
Total params: 490,257
Trainable params: 490,257
Non-trainable params: 0

```

```

model_encoder = variational_encoder
# model_encoder.summary(expand_nested=True, show_trainable=True)

model_decoder = variational_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(model, images=test_data, n_images=5):
    reconstructions = model.predict(images[:n_images])
    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(variational_ae, normal_data, 30)
plt.show()
```



```
show_reconstructions(variational_ae, valid_data, 30)
plt.show()
```



```
show_reconstructions(variational_ae, normal_valid_data, 30)
plt.show()
```



```
show_reconstructions(variational_ae, abnormal_valid_data, 30)
plt.show()
```



```
show_reconstructions(variational_ae, test_data, 30)
plt.show()
```



```
show_reconstructions(variational_ae, normal_test_data, 30)
plt.show()
```



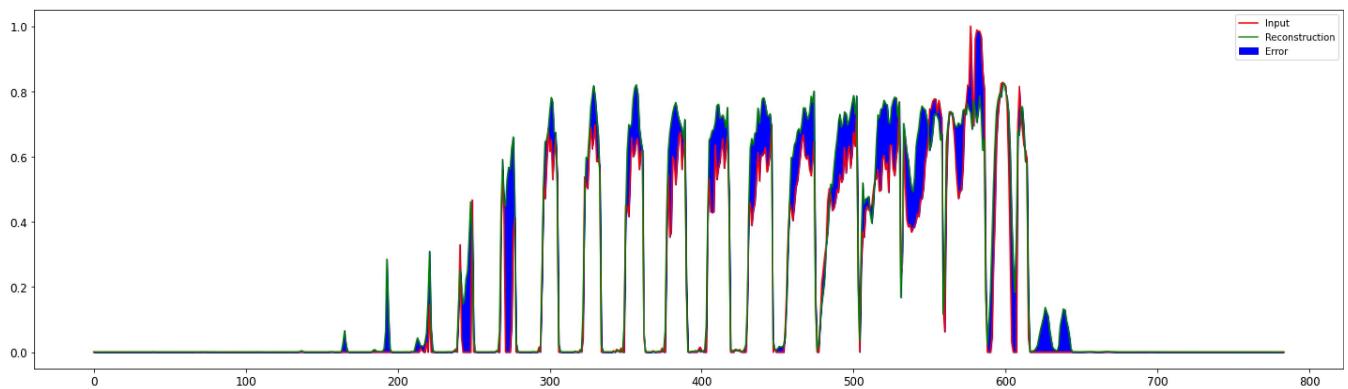
```
show_reconstructions(variational_ae, abnormal_test_data, 30)
plt.show()
```



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

```
reconstructions_nl_test = variational_ae.predict(normal_test_data)
```

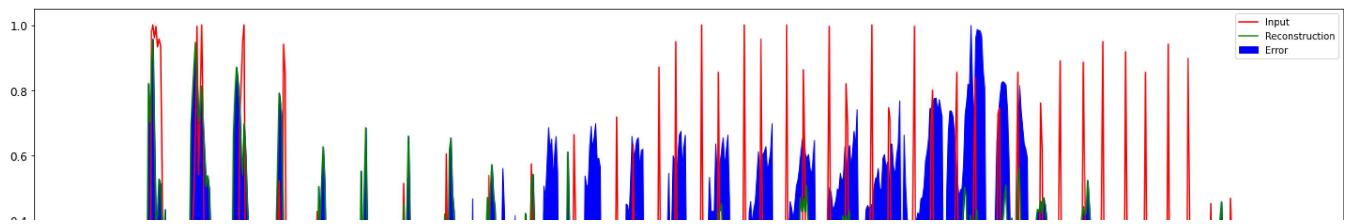
```
plt.figure(figsize=(25,7))
plt.plot(normal_test_data[0].ravel(), 'r')
plt.plot(reconstructions_nl_test[0].ravel(), 'g')
plt.fill_between(np.arange(28*28), reconstructions_nl_test[0].ravel(), normal_test_data[0].ravel())
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```



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

```
reconstructions_abn_test = variational_ae.predict(abnormal_test_data)
```

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



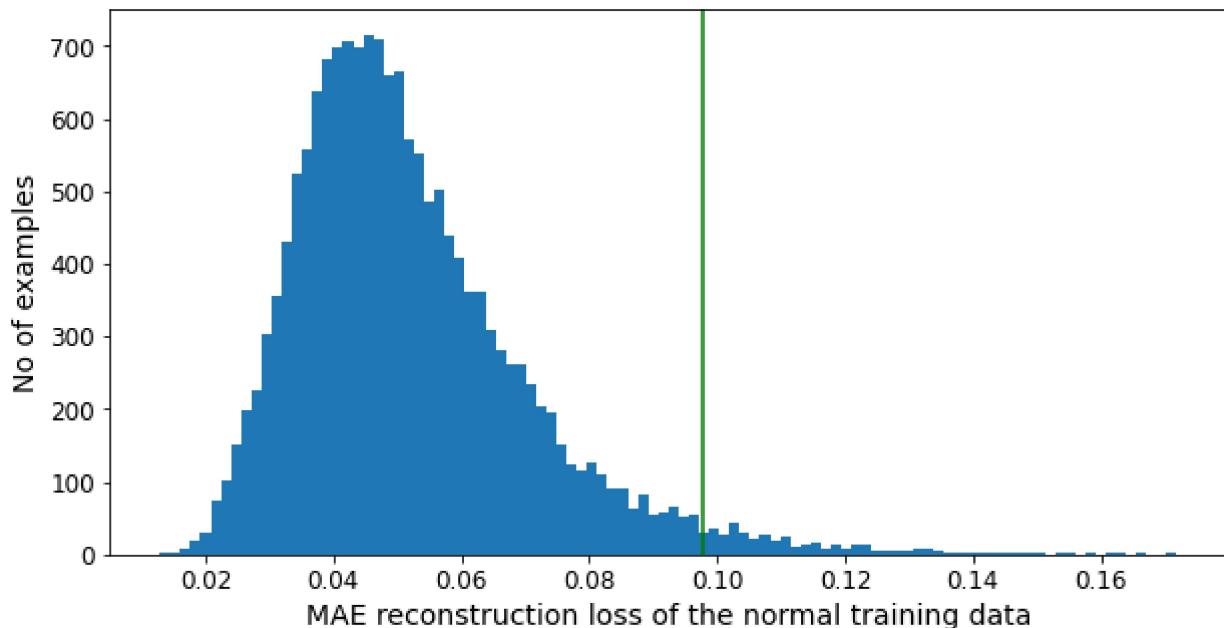
## ▼ Distributions of the reconstruction losses and the calculation of the threshold.

### Distribution of the reconstruction losses of the normal training data

```

reconstructions = variational_ae.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: 0.05193515  
Std: 0.018269068

```
threshold_train_mean_2_5_std = np.mean(train_loss) + 2.5*np.std(train_loss)
```

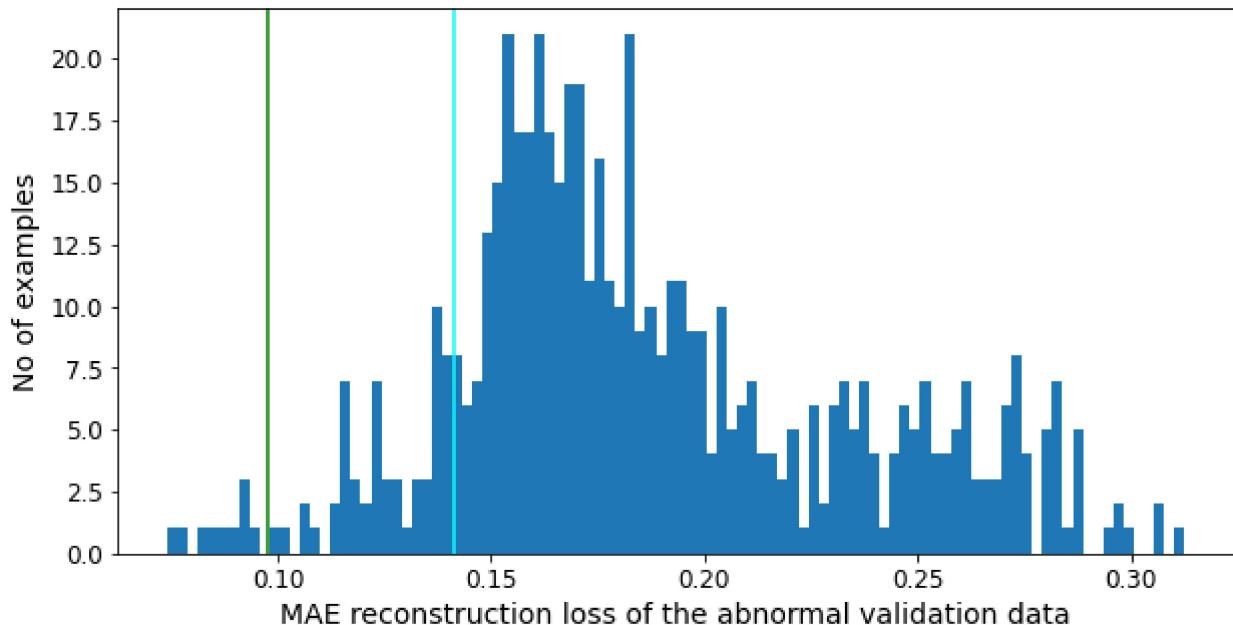
```
print("Threshold based on the mean of the training data MAE reconstruction losses + 2.5 std:
```

Threshold based on the mean of the training data MAE reconstruction losses + 2.5 std: 0.18740454

```
threshold1 = threshold_train_mean_2_5_std
```

## Distribution of the reconstruction losses of the abnormal validation data

```
reconstructions = variational_ae.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)
```

```
(0.18740454, 0.046277314)
```

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

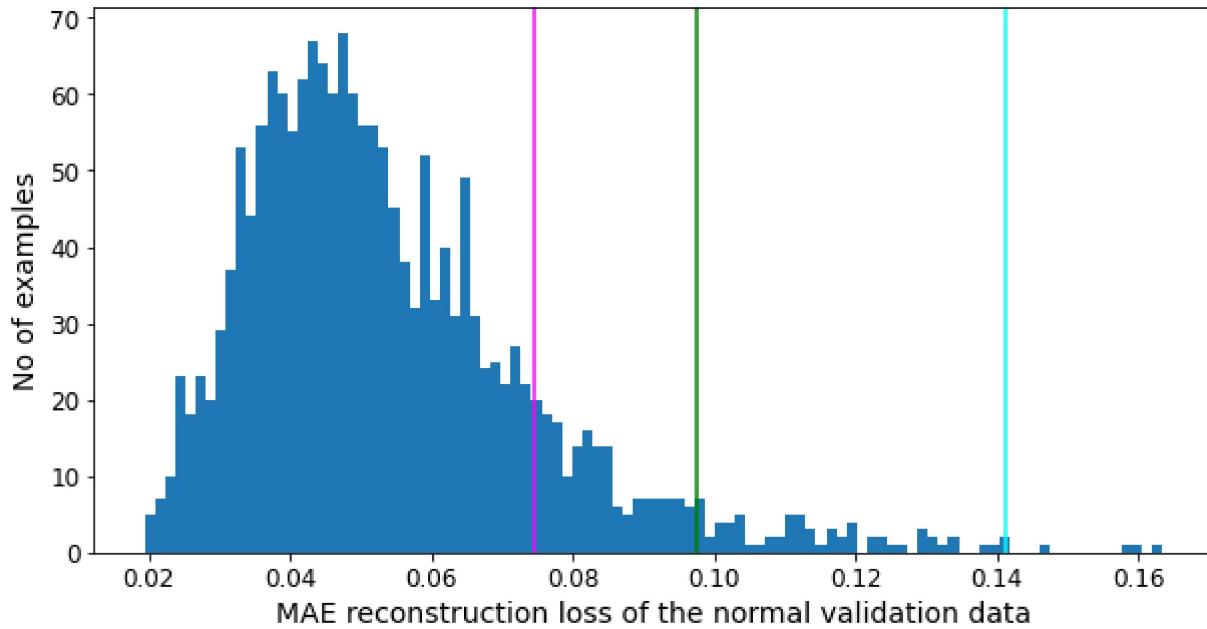
Threshold2: 0.14112723

## Distribution of the reconstruction losses of the normal validation data

```

reconstructions = variational_ae.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)
```

(0.05394568, 0.020706058)

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

Threshold3: 0.07465173

**Calculation of a preliminary threshold 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)**

```
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: 0.10788948088884354

```
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(variational_ae, 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: 0.12200970172882078

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

array([[0.11934052, 0.97784406],
       [0.12067511, 0.97827013],
       [0.1220097 , 0.97912228],
       [0.12334429, 0.97741798],
       [0.12467888, 0.97912228]])
```

```
threshold5 = valid_data_best_threshold
```

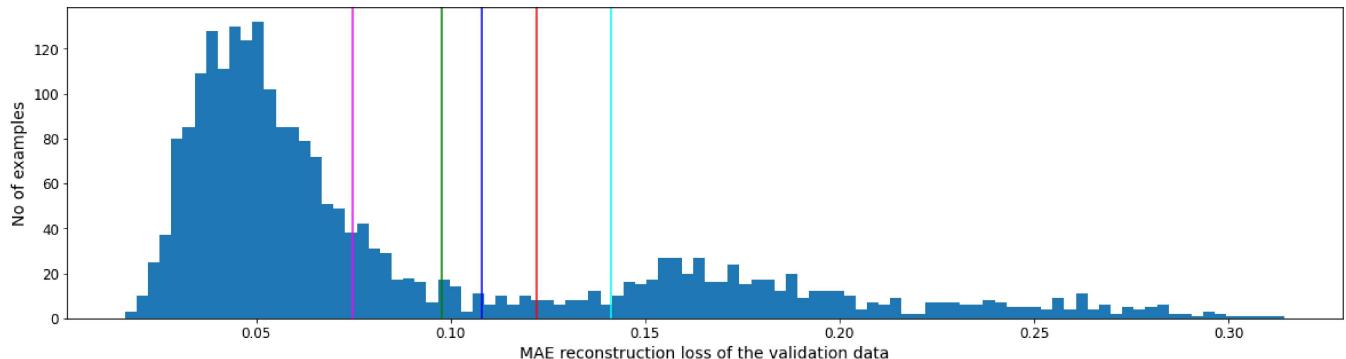
```
threshold = threshold5
```

## 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 = variational_ae.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.xlabel("MAE reconstruction loss of the validation data")
plt.ylabel("No of examples")
plt.show()
```



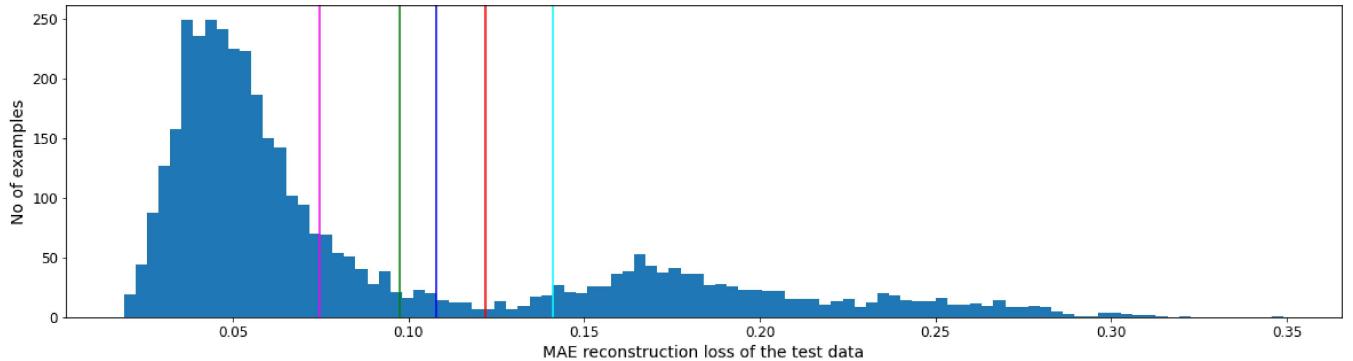
## Distribution of the reconstruction losses of the test data (normal and abnormal)

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 = variational_ae.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()
```



## Mean and standard deviation of reconstruction losses for normal and abnormal test data

```
reconstructions = variational_ae.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))

(0.053806253, 0.02048242)
```

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

(0.19223505, 0.04522203)
```

## 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(variational_ae, test_data, threshold)
print_stats(preds, test_labels_T_F)

→ mean_nl_test_loss: 0.05380625277757645
    std_nl_test_loss: 0.020482420921325684
    mean_abn_test_loss: 0.1922350525856018
    std_abn_test_loss: 0.04522202908992767
    Confusion Matrix:
        prediction: F      T
                    993   3007
        label: F  [[965   35]   1000
                    T  [28   2972]]   3000
    Accuracy = 0.98425
    Precision = 0.9883604921849019
    Recall = 0.9906666666666667

print(confusion_matrix(test_labels_T_F, preds))

[[ 965   35]
 [ 28 2972]]

```

## Extra accuracy info

**Just informative. Please record the above accuracy.**

Accuracy on the test data with threshold set 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

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

mean_nl_test_loss: 0.05380625277757645
std_nl_test_loss: 0.020482420921325684
mean_abn_test_loss: 0.1922350525856018
std_abn_test_loss: 0.04522202908992767
Confusion Matrix:
prediction: F      T
            1047   2953
label: F    [[982   18]    1000
            T    [65   2935]]   3000
Accuracy = 0.97925
Precision = 0.9939045038943447
Recall = 0.9783333333333334
```

Accuracy on the test data with threshold set based on the mean of the training data MAE  
reconstruction losses + 2.5 std

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

mean_nl_test_loss: 0.05380625277757645
std_nl_test_loss: 0.020482420921325684
mean_abn_test_loss: 0.1922350525856018
std_abn_test_loss: 0.04522202908992767
Confusion Matrix:
prediction: F      T
            1096   2904
label: F    [[988   12]    1000
            T    [108   2892]]   3000
Accuracy = 0.97
Precision = 0.9958677685950413
Recall = 0.964
```

## Extra Info

Giving the VAE codings (please see book) (Just informative, not the goal here)

```
def plot_multiple_images(images, n_cols=None):
    n_cols = n_cols or len(images)
    n_rows = (len(images) - 1) // n_cols + 1
    if images.shape[-1] == 1:
        images = np.squeeze(images, axis=-1)
    plt.figure(figsize=(n_cols, n_rows))
    for index, image in enumerate(images):
        plt.subplot(n_rows, n_cols, index + 1)
```

```

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

codings = tf.random.normal(shape=[12, codings_size])
images = variational_decoder(codings).numpy()
plot_multiple_images(images, 4)
# save_fig("vae_generated_images_plot", tight_layout=False)

```

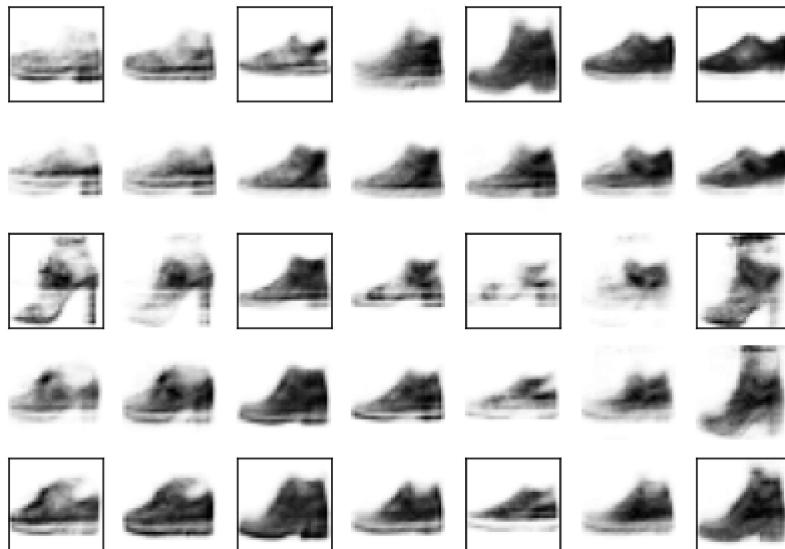


```

codings_grid = tf.reshape(codings, [1, 3, 4, codings_size])
larger_grid = tf.image.resize(codings_grid, size=[5, 7])
interpolated_codings = tf.reshape(larger_grid, [-1, codings_size])
images = variational_decoder(interpolated_codings).numpy()

plt.figure(figsize=(7, 5))
for index, image in enumerate(images):
    plt.subplot(5, 7, index + 1)
    if index%7%2==0 and index//7%2==0:
        plt.gca().get_xaxis().set_visible(False)
        plt.gca().get_yaxis().set_visible(False)
    else:
        plt.axis("off")
    plt.imshow(image, cmap="binary")
# save_fig("semantic_interpolation_plot", tight_layout=False)

```



---

✓ 1s completed at 3:35 PM

