

**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

((16227, 28, 28), (16227,), (2369, 28, 28), (2369,), (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
```

```
((1773, 28, 28), (596, 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)

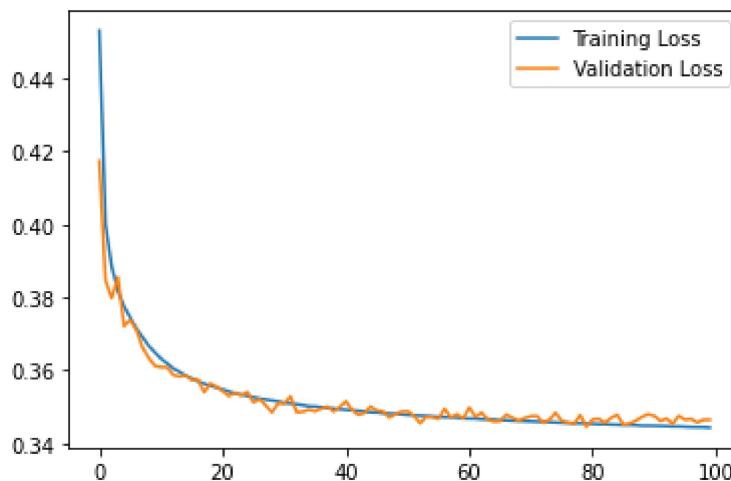
history = autoencoder.fit(normal_data, normal_data,
                           epochs=100,
                           batch_size=128,
                           validation_data=(normal_valid_data, normal_valid_data),
                           callbacks=[checkpoint_cb],
                           shuffle=True)

127/127 [=====] - 1s 7ms/step - loss: 0.3459 - val_loss: 0.3459
Epoch 73/100
127/127 [=====] - 1s 7ms/step - loss: 0.3459 - val_loss: 0.3459
Epoch 74/100
127/127 [=====] - 1s 7ms/step - loss: 0.3458 - val_loss: 0.3458
Epoch 75/100
127/127 [=====] - 1s 7ms/step - loss: 0.3457 - val_loss: 0.3457
Epoch 76/100
```

```
[GPU0] /usr/tensorflow/1.14.0/tensorflow/python/training/monitored_session.py:100: UserWarning: MonitoredSession: Session will be closed at epoch 100. If you want to close it earlier, call session.close() or session.stop().  
  warnings.warn("MonitoredSession: Session will be closed at epoch %d. If you want to close it earlier, call session.close() or session.stop()." % epoch)  
127/127 [=====] - 1s 7ms/step - loss: 0.3456 - val_loss: 0.3456  
Epoch 77/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3455 - val_loss: 0.3455  
Epoch 78/100  
120/127 [=====>...] - ETA: 0s - loss: 0.3455INFO:tensorflow:Assets created in this session can be saved by calling save().  
127/127 [=====] - 3s 21ms/step - loss: 0.3455 - val_loss: 0.3455  
Epoch 79/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3454 - val_loss: 0.3454  
Epoch 80/100  
124/127 [=====>...] - ETA: 0s - loss: 0.3455INFO:tensorflow:Assets created in this session can be saved by calling save().  
127/127 [=====] - 3s 21ms/step - loss: 0.3454 - val_loss: 0.3454  
Epoch 81/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3453 - val_loss: 0.3453  
Epoch 82/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3452 - val_loss: 0.3452  
Epoch 83/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3452 - val_loss: 0.3452  
Epoch 84/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3451 - val_loss: 0.3451  
Epoch 85/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3451 - val_loss: 0.3451  
Epoch 86/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3450 - val_loss: 0.3450  
Epoch 87/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3450 - val_loss: 0.3450  
Epoch 88/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3449 - val_loss: 0.3449  
Epoch 89/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3448 - val_loss: 0.3448  
Epoch 90/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3448 - val_loss: 0.3448  
Epoch 91/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3448 - val_loss: 0.3448  
Epoch 92/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3447 - val_loss: 0.3447  
Epoch 93/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3447 - val_loss: 0.3447  
Epoch 94/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3446 - val_loss: 0.3446  
Epoch 95/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3446 - val_loss: 0.3446  
Epoch 96/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3445 - val_loss: 0.3445  
Epoch 97/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3444 - val_loss: 0.3444  
Epoch 98/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3444 - val_loss: 0.3444  
Epoch 99/100  
127/127 [=====] - 1s 7ms/step - loss: 0.3444 - val_loss: 0.3444  
Epoch 100/100
```

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

&lt;matplotlib.legend.Legend at 0x7f5bc6ab90&gt;



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

Model: "anomaly\_detector\_3"

Layer (type)	Output Shape	Param #	Trainable
<hr/>			
sequential_6 (Sequential)	(None, 16)	243152	Y
-----			
flatten_3 (Flatten)	(None, 784)	0	Y
dense_24 (Dense)	(None, 256)	200960	Y
dense_25 (Dense)	(None, 128)	32896	Y
dense_26 (Dense)	(None, 64)	8256	Y
dense_27 (Dense)	(None, 16)	1040	Y
-----			
sequential_7 (Sequential)	(None, 28, 28)	243920	Y
-----			
dense_28 (Dense)	(None, 64)	1088	Y
dense_29 (Dense)	(None, 128)	8320	Y
dense_30 (Dense)	(None, 256)	33024	Y
dense_31 (Dense)	(None, 784)	201488	Y
reshape_3 (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()
```

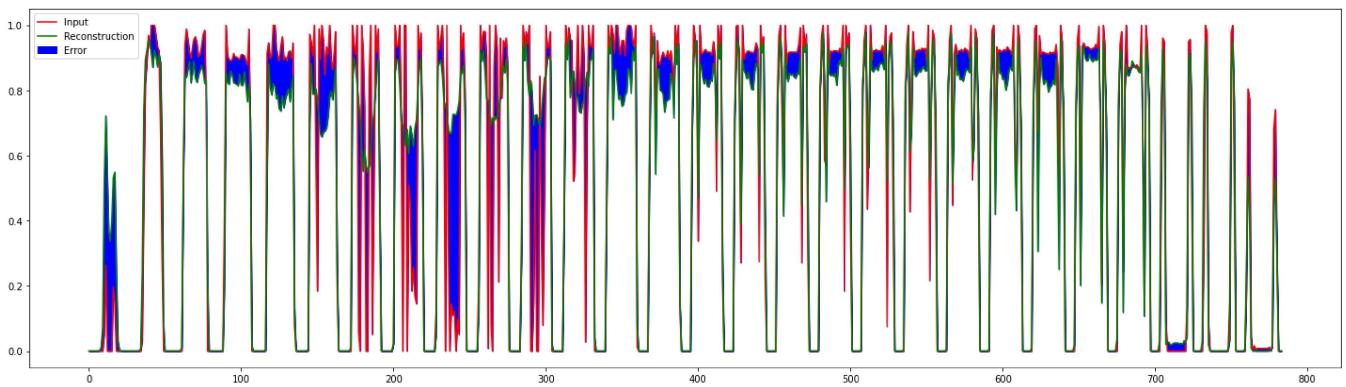


```
encoded_data = autoencoder.encoder(normal_test_data).numpy()
```

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

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

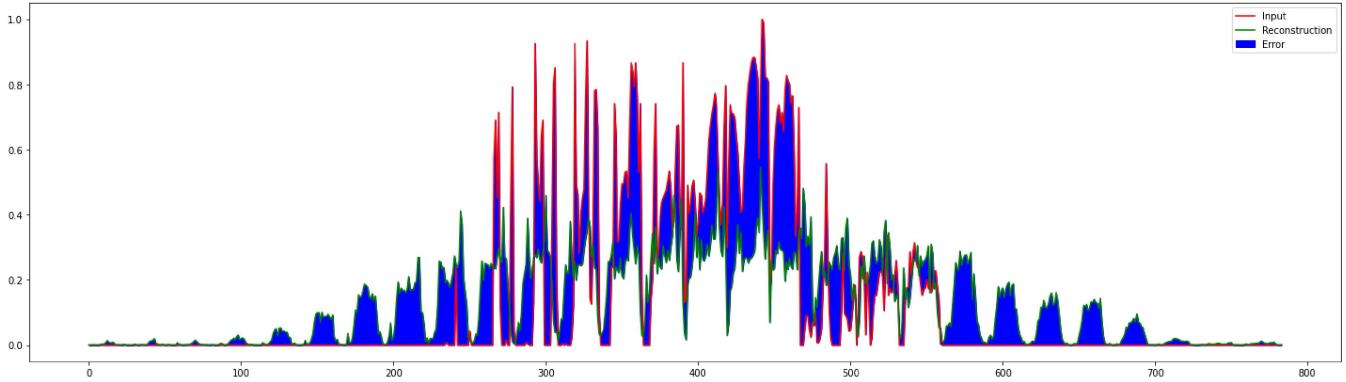
```
plt.figure(figsize=(25,7))
plt.plot(normal_test_data[0].ravel(), 'r')
plt.plot(decoded_data[0].ravel(), 'g')
plt.fill_between(np.arange(28*28), decoded_data[0].ravel(), normal_test_data[0].ravel(), color='blue')
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```



```
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()
```

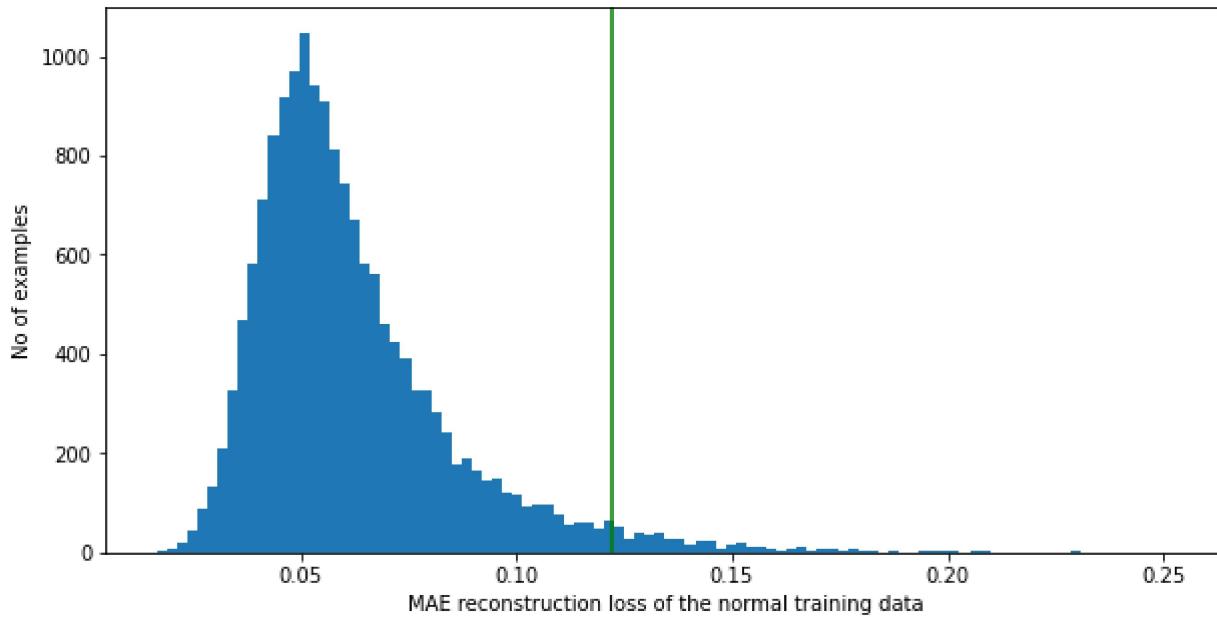


## ▼ Distributions of the reconstruction losses and the calculation of the threshold.

### 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:  0.062109392
Std:  0.024056483
```

```
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:
```

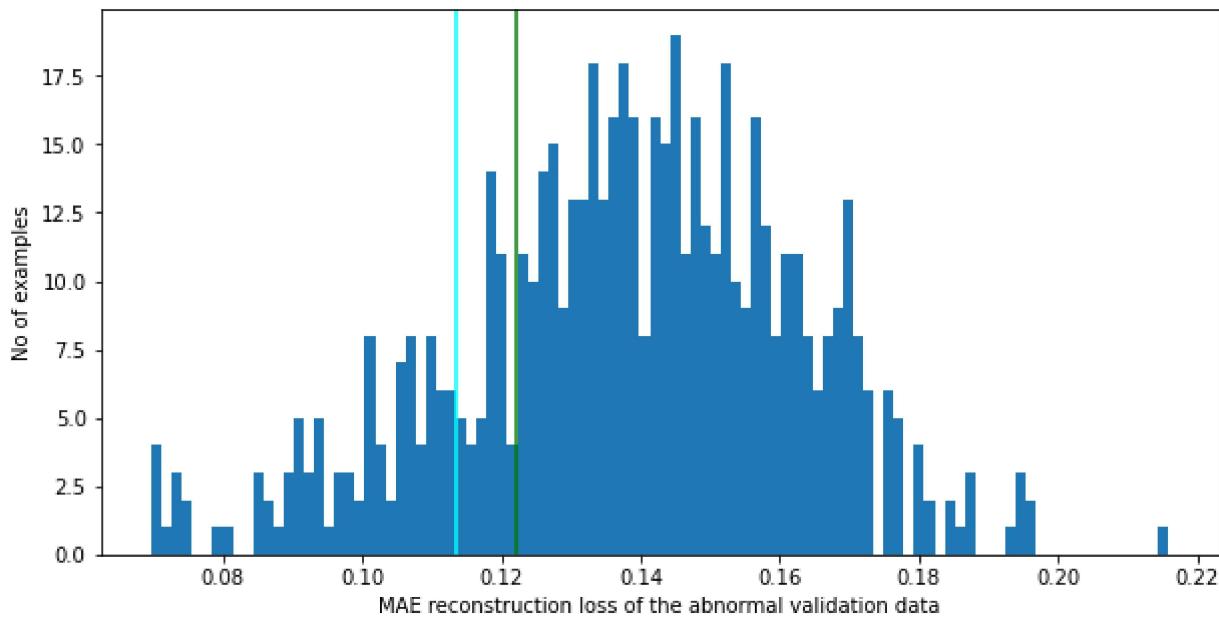
```
Threshold based on the mean of the training data MAE reconstruction losses + 2.5 std:  0.145
```

```
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)
```

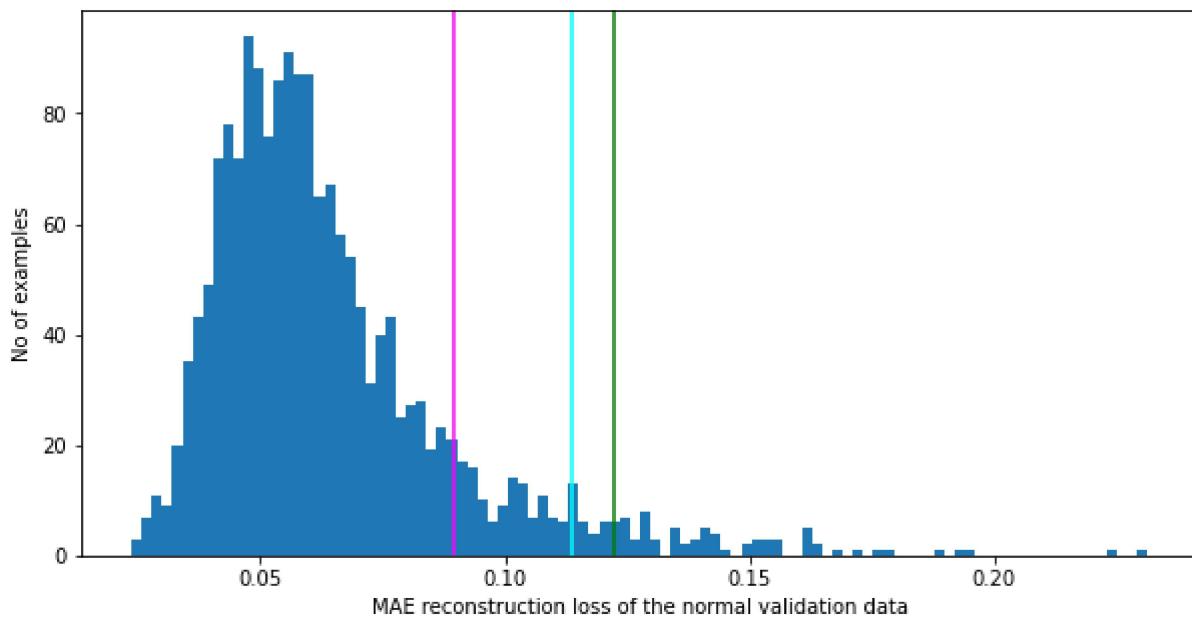
```
(0.13812023, 0.02449026)
```

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

```
Threshold2: 0.113629974
```

## 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)
```

```
(0.06424598, 0.025472915)
```

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

```
Threshold3: 0.08971889
```

**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: 0.10167443752288818
```

```
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: 0.1041380755603312

```

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

array([[0.10266059, 0.91599831],
       [0.10339933, 0.91895314],
       [0.10413808, 0.91937526],
       [0.10487682, 0.91937526],
       [0.10561556, 0.91810891]])

```

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.

```

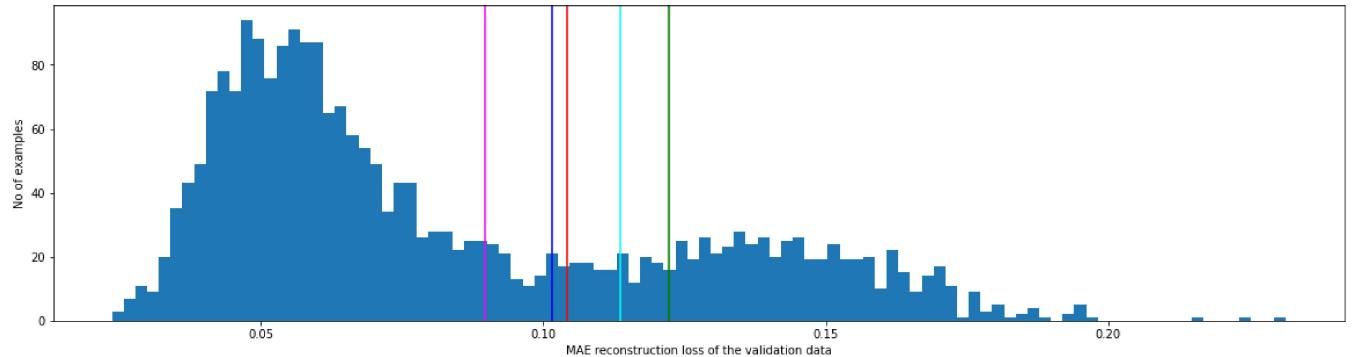
reconstructions = autoencoder.predict(valid_data)
valid_loss = tf.keras.losses.mae(reconstructions.reshape(-1,784), valid_data.reshape(-1,784))
plt.figure(figsize=(20,5))
plt.hist(valid_loss[None, :], bins=100)
plt.axvline(threshold, c='r')
plt.axvline(threshold4, c='b')

```

```

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))

(0.064862005, 0.025840804)

reconstructions = autoencoder.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))

(0.13897489, 0.022689356)

```

## 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].shape[0]))
    print("label: F [[{} {}] {}] {}".format(cf[0,0], cf[0,1], test_labels_T_F[test_labels_T_F == False].shape[0], cf[0,1]))
    print("      T [[{} {}] {}] {}".format(cf[1,0], cf[1,1], test_labels_T_F[test_labels_T_F == True].shape[0], cf[1,1]))
    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, Avg_of_threshold_2_3)
print_stats(preds, test_labels_T_F)
```

```
mean_nl_test_loss: 0.06486200541257858
std_nl_test_loss: 0.02584080398082733
mean_abn_test_loss: 0.1389748901128769
std_abn_test_loss: 0.02268935553729534
Confusion Matrix:
 prediction: F      T
               1142   2858
label: F    [[921   79]    1000
             [221  2779]]   3000
Accuracy = 0.925
Precision = 0.9723582925122464
Recall = 0.9263333333333333
```

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

```
[[ 921   79]
 [ 221  2779]]
```

## 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: 0.06486200541257858
std_nl_test_loss: 0.02584080398082733
mean_abn_test_loss: 0.1389748901128769
std_abn_test_loss: 0.02268935553729534
Confusion Matrix:
 prediction: F      T
               1183   2817
label: F    [[934   66]    1000
             [249  2751]]   3000
Accuracy = 0.92125
Precision = 0.9765708200212992
Recall = 0.917
```

## 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: 0.06486200541257858
std_nl_test_loss: 0.02584080398082733
mean_abn_test_loss: 0.1389748901128769
std_abn_test_loss: 0.02268935553729534
Confusion Matrix:
prediction: F      T
            888    3112
label: F   [[775    225]    1000
           T   [113    2887]]   3000
Accuracy = 0.9155
Precision = 0.9276992287917738
Recall = 0.9623333333333334
```

---

✓ 0s completed at 11:23 AM

✖