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
```

```
# 9 Ankle Boot

nl1 = 9
nl2 = 9
abn = 3

(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 = trai

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

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

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

test_labels_T_F = np.where((test_labels == nl1) | (test_labels == nl2), True, False)
# 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), True, False)
# 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
```

```
((5405, 28, 28), (5405,), (1185, 28, 28), (1185,), (2000, 28, 28), (2000,))
```

```
normal_test_data = test_data[(test_labels == nl1) | (test_labels == nl2)]   # The normal digi
abnormal_test_data = test_data[test_labels == abn]                          # The abnormal di
normal_test_labels = test_labels[(test_labels == nl1) | (test_labels == nl2)]   # Their label
abnormal_test_labels = test_labels[test_labels == abn]                          # Their label


normal_test_data.shape, abnormal_test_data.shape
```

```
((1000, 28, 28), (1000, 28, 28))
```

```
normal_valid_data = valid_data[(valid_labels == nl1) | (valid_labels == nl2)]   # The normal
abnormal_valid_data = valid_data[valid_labels == abn]                           # The abnorma
normal_valid_labels = valid_labels[(valid_labels == nl1) | (valid_labels == nl2)]   # Their l
abnormal_valid_labels = valid_labels[valid_labels == abn]                           # Their l


normal_valid_data.shape, abnormal_valid_data.shape
```

```
((595, 28, 28), (590, 28, 28))
```

Building and training the network

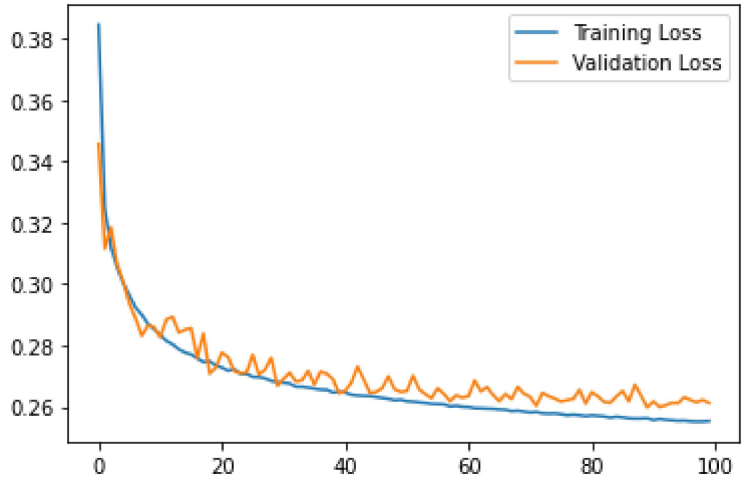
```
class AnomalyDetector(Model):
  def __init__(self):
    super(AnomalyDetector, self).__init__()
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(256, activation="selu"),
      layers.Dense(128, activation="selu"),
      layers.Dense(64, activation="selu"),
    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

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)
```

```
36/43 [========================>.....] - ETA: 0s - loss: 0.2584INFO:tensorflow:Assets
43/43 [==============================] - 2s 37ms/step - loss: 0.2583 - val_loss: 0.26
Epoch 73/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2579 - val_loss: 0.264
Epoch 74/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2578 - val_loss: 0.263
Epoch 75/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2579 - val_loss: 0.262
Epoch 76/100
```

```
Epoch 76/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2576 - val_loss: 0.261
Epoch 77/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2572 - val_loss: 0.262
Epoch 78/100
43/43 [==============================] - 0s 6ms/step - loss: 0.2574 - val_loss: 0.262
Epoch 79/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2572 - val_loss: 0.265
Epoch 80/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2570 - val_loss: 0.261
Epoch 81/100
43/43 [==============================] - 0s 6ms/step - loss: 0.2572 - val_loss: 0.264
Epoch 82/100
43/43 [==============================] - 0s 6ms/step - loss: 0.2570 - val_loss: 0.263
Epoch 83/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2569 - val_loss: 0.261
Epoch 84/100
43/43 [==============================] - 0s 6ms/step - loss: 0.2565 - val_loss: 0.261
Epoch 85/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2568 - val_loss: 0.263
Epoch 86/100
43/43 [==============================] - 0s 6ms/step - loss: 0.2566 - val_loss: 0.265
Epoch 87/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2563 - val_loss: 0.261
Epoch 88/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2562 - val_loss: 0.267
Epoch 89/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2562 - val_loss: 0.263
Epoch 90/100
35/43 [=======================>......] - ETA: 0s - loss: 0.2559INFO:tensorflow:Assets
43/43 [==============================] - 2s 43ms/step - loss: 0.2563 - val_loss: 0.25
Epoch 91/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2557 - val_loss: 0.261
Epoch 92/100
43/43 [==============================] - 0s 6ms/step - loss: 0.2560 - val_loss: 0.260
Epoch 93/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2558 - val_loss: 0.260
Epoch 94/100
43/43 [==============================] - 0s 6ms/step - loss: 0.2556 - val_loss: 0.261
Epoch 95/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2554 - val_loss: 0.261
Epoch 96/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2555 - val_loss: 0.263
Epoch 97/100
43/43 [==============================] - 0s 6ms/step - loss: 0.2553 - val_loss: 0.262
Epoch 98/100
43/43 [==============================] - 0s 5ms/step - loss: 0.2552 - val_loss: 0.261
Epoch 99/100
43/43 [==============================] - 0s 6ms/step - loss: 0.2552 - val_loss: 0.262
```

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

```
<matplotlib.legend.Legend at 0x7fb1e03e5510>
```




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

```
Model: "anomaly_detector_3"
_________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Trainable  
=================================================================================
 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          |
---------------------------------------------------------------------------

=================================================================================
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()
```

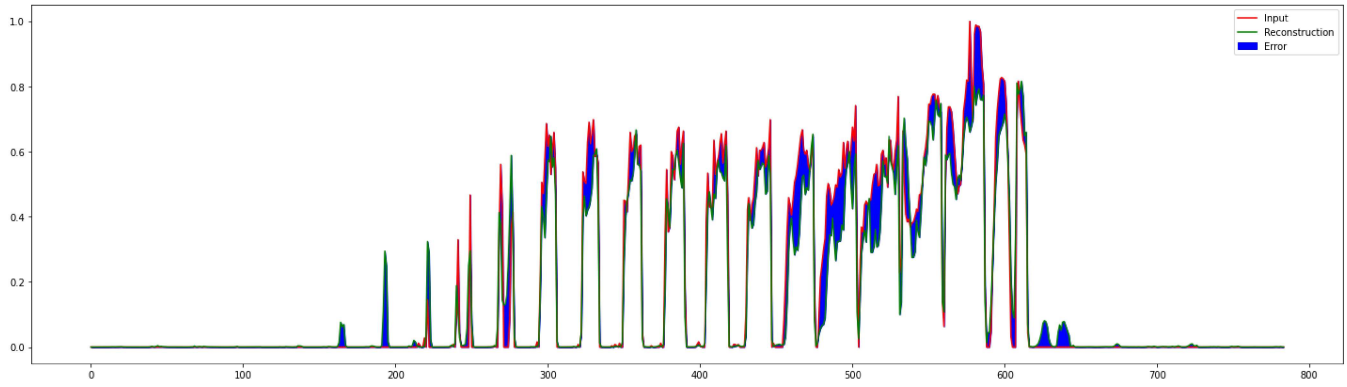
```
show_reconstructions(autoencoder, normal_test_data, 30)
plt.show()
```

```
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

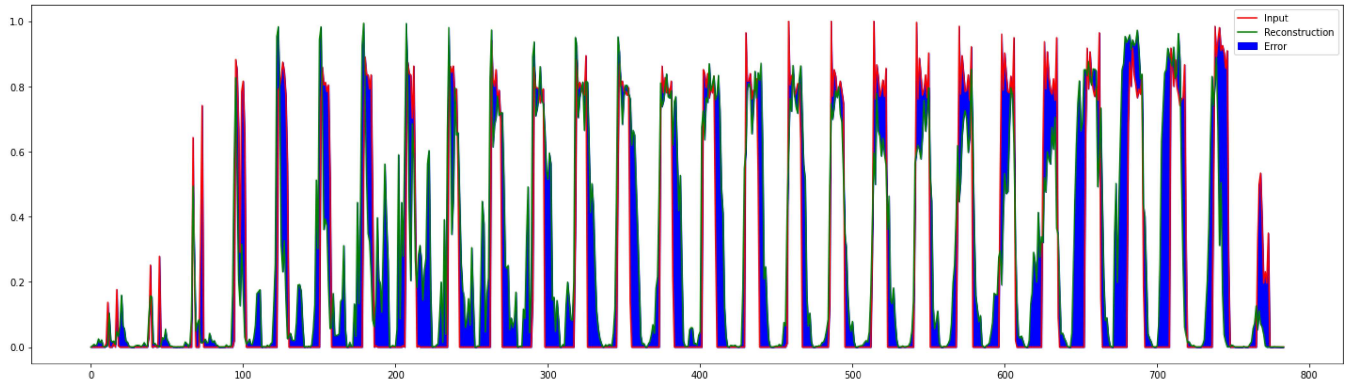
```
plt.figure(figsize=(25,7))
plt.plot(normal_test_data[0].ravel(), 'r')
plt.plot(decoded_data[0].ravel(), 'g')
plt.fill_between(np.arange(28*28), decoded_data[0].ravel(), normal_test_data[0].ravel(), colo
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```




```
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()
```

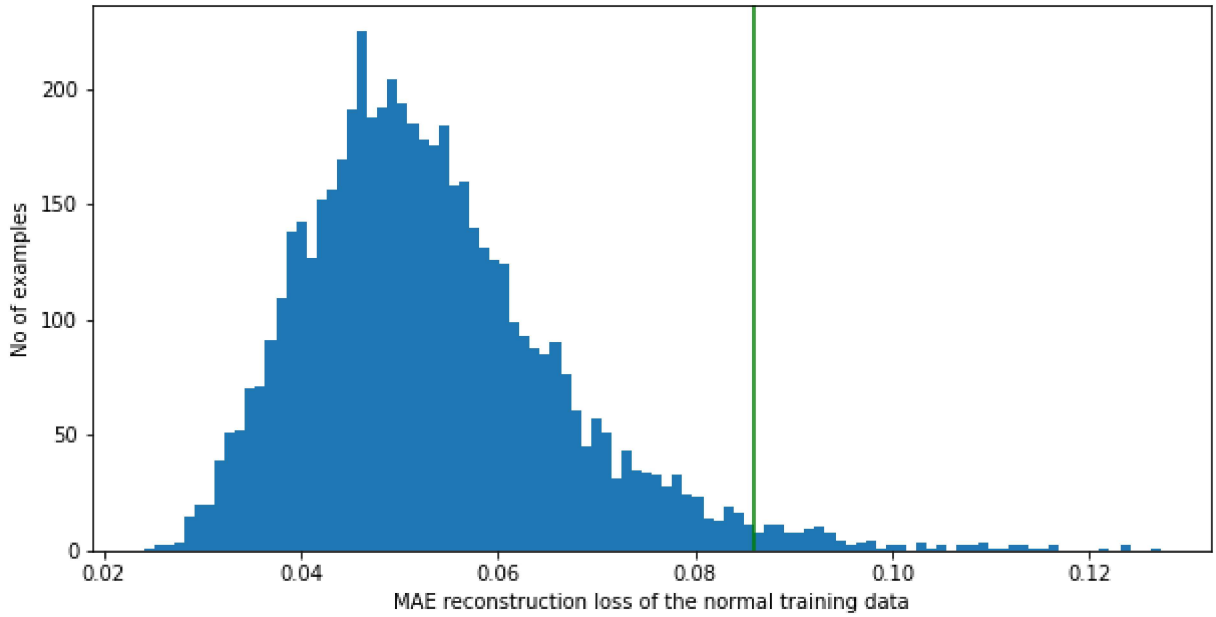


Distributions of the reconstruction losses and the calculation of the threshold.

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, 78
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()
```




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

```
Mean:  0.05321313
Std:  0.013122587
```

```
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:
```

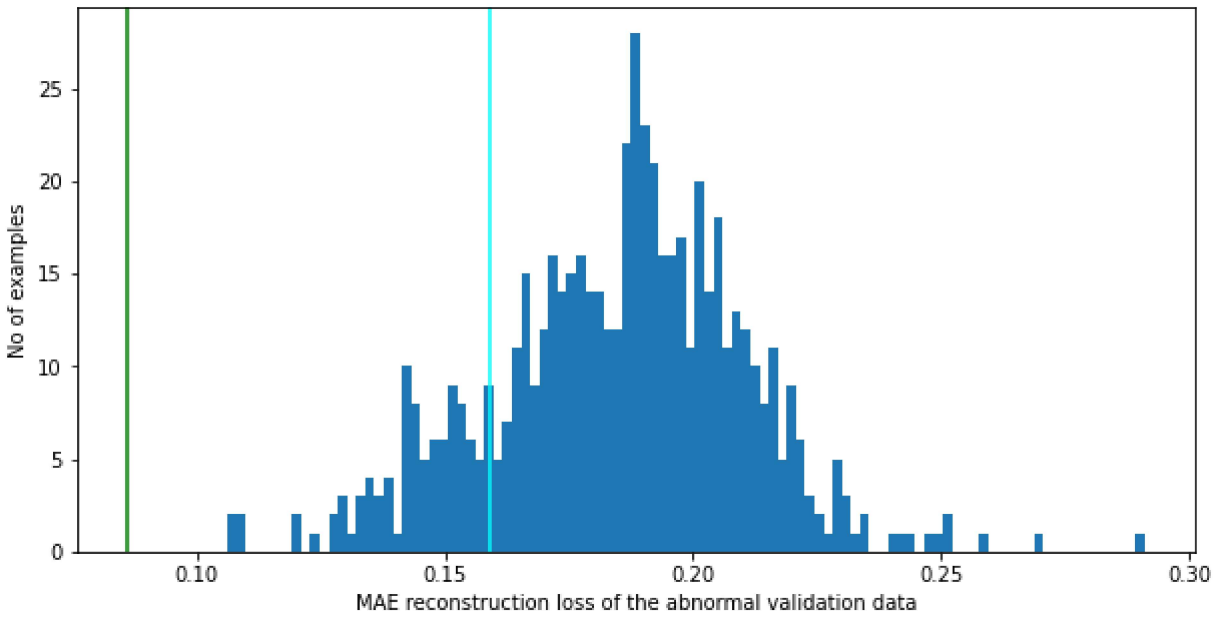
```
Threshold based on the mean of the training data MAE reconstruction losses + 2.5 std:  0
```

```
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.res
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)
```

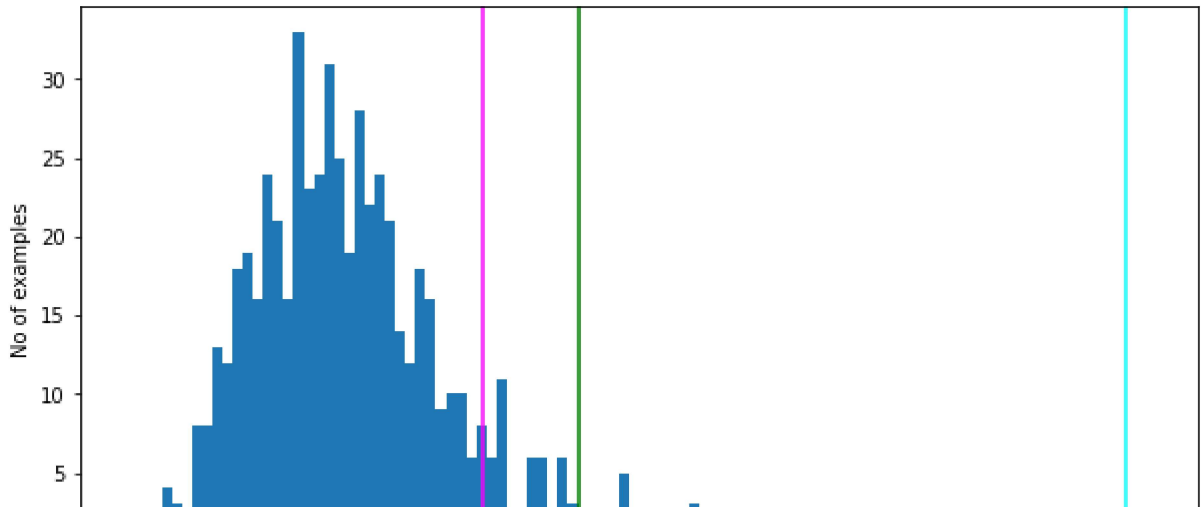
```
(0.18480349, 0.025693104)
```

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

```
Threshold2:  0.15911038
```

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.reshap
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)
```

MAE reconstruction loss of the normal validation data

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

```
     (0.057034846, 0.016075548)
```

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

```
     Threshold3:  0.073110394
```

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:  0.11611038446426392
```

```
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)
```

Double-click (or enter) to edit

```
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:  0.11964147791266447
```

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

```
array([[0.11708611, 0.99324895],
       [0.11836379, 0.99409283],
       [0.11964148, 0.99493671],
       [0.12091916, 0.99409283],
       [0.12219685, 0.99409283]])
```

```
threshold5 = valid_data_best_threshold
```

```
threshold = threshold5
```

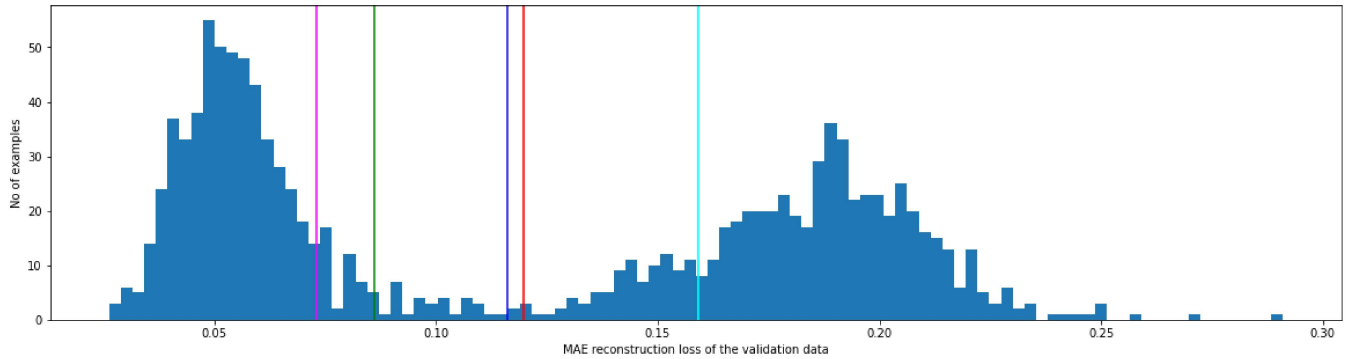
Distribution of the reconstruction losses of all the validation data (normal and abnormal)

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)
```

```
(0.05696041, 0.017429013)
```


```
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)
```

```
(0.18634458, 0.027400272)
```

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)

def print_stats(predictions, labels):
  cf = confusion_matrix(labels, predictions)
  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("Normal Test Data Mean = {}".format(np.mean(nl_test_loss)))
  print("Normal Test Data Standard Deviation = {}".format(np.std(nl_test_loss)))
  print("Abnormal Test Data Mean = {}".format(np.mean(abn_test_loss)))
  print("Abnormal Test Data Standard Deviation = {}".format(np.std(abn_test_loss)))
  print("Precision = {}".format(precision_score(labels, predictions)))
  print("Recall = {}".format(recall_score(labels, predictions)))
  print(accuracy_score(labels, predictions))
  print(np.mean(nl_test_loss))
  print(np.std(nl_test_loss))
  print(np.mean(abn_test_loss))
  print(np.std(abn_test_loss))
  print(precision_score(labels, predictions))
  print(recall_score(labels, predictions))
  print(accuracy_score(labels, predictions), np.mean(nl_test_loss), np.std(nl_test_loss), np.
        precision_score(labels, predictions), recall_score(labels, predictions))


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

```
Confusion Matrix:
 prediction: F      T
             992   1008
 label: F   [[989   11]    1000
        T    [3   997]]   1000
Accuracy = 0.993
Normal Test Data Mean = 0.05696041136980057
Normal Test Data Standard Deviation = 0.017429012805223465
Abnormal Test Data Mean = 0.18634457886219025
Abnormal Test Data Standard Deviation = 0.027400271967053413
Precision = 0.9890873015873016
Recall = 0.997
0.993
0.05696041
0.017429013
0.18634458
0.027400272
0.9890873015873016
0.997
0.993 0.05696041 0.017429013 0.18634458 0.027400272 0.9890873015873016 0.997
```

```
print("Threshold =", valid_data_best_threshold)
```

```
Threshold = 0.11964147791266447
```

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

```
[[989  11]
 [  3 997]]
```

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)
```

```
Confusion Matrix:
 prediction: F      T
             997   1003
 label: F   [[993   7]    1000
        T    [4   996]]   1000
```

```
Accuracy = 0.9945
Normal Test Data Mean = 0.05696041136980057
Normal Test Data Standard Deviation = 0.017429012805223465
Abnormal Test Data Mean = 0.18634457886219025
Abnormal Test Data Standard Deviation = 0.027400271967053413
Precision = 0.9930209371884346
Recall = 0.996
0.9945
0.05696041
0.017429013
0.18634458
0.027400272
0.9930209371884346
0.996
0.9945 0.05696041 0.017429013 0.18634458 0.027400272 0.9930209371884346 0.996
```

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

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

```
Confusion Matrix:
 prediction: F      T
             1043   957
 label: F   [[1000   0]    1000
        T    [43   957]]   1000
Accuracy = 0.9785
Normal Test Data Mean = 0.05696041136980057
Normal Test Data Standard Deviation = 0.017429012805223465
Abnormal Test Data Mean = 0.18634457886219025
Abnormal Test Data Standard Deviation = 0.027400271967053413
Precision = 1.0
Recall = 0.957
0.9785
0.05696041
0.017429013
0.18634458
0.027400272
1.0
0.957
0.9785 0.05696041 0.017429013 0.18634458 0.027400272 1.0 0.957
```

✓ 0s completed at 3:39 PM