References: Chapter 17 of Geron's book. For 1-Dim plots, Keras tutorial : https://www.tensorflow.org/tutorials/generative/autoencoder

This file trains a VAE 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 = 6
nl2 = 6
abn1 = 7
abn2 = 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 = trai

normal_data = x_train[(y_train == nl1) | (y_train == nl2)]        # Normal training data (Nor
normal_labels = y_train[(y_train == nl1) | (y_train == nl2)]

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

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), 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
```

```
((5401, 28, 28), (5401,), (1177, 28, 28), (1177,), (2000, 28, 28), (2000,))
```

```
normal_test_data = test_data[(test_labels == nl1) | (test_labels == nl2)]            # The no
abnormal_test_data = test_data[(test_labels == abn1) | (test_labels == abn2)]        # The ab
normal_test_labels = test_labels[(test_labels == nl1) | (test_labels == nl2)]        # Their
abnormal_test_labels = test_labels[(test_labels == abn1) | (test_labels == abn2)]    # Their


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)]            # Th
abnormal_valid_data = valid_data[(valid_labels == abn1) | (valid_labels == abn2)]        # Th
normal_valid_labels = valid_labels[(valid_labels == nl1) | (valid_labels == nl2)]        # Th
abnormal_valid_labels = valid_labels[(valid_labels == abn1) | (valid_labels == abn1)]    # Th
```

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

```
((599, 28, 28), (578, 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))


# For details please see Geron's book. Uses the reparametrization trick to do stochastic
# sampling from the MVN distribution, while allowing the 2 parallel layers containing the
# means and stds of the MVN distribution for each dimension to be trained via
# backpropogation of the error signal.
class Sampling(keras.layers.Layer):
    def call(self, inputs):
        mean, log_var = inputs
        return K.random_normal(tf.shape(log_var)) * K.exp(log_var / 2) + mean


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

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)

# Parallel layers at the end of the encoder for means
# and standard deviations of the Multivariate Normal (MVN) distribution
# in the dimensions of the coding size (here 32).
codings_mean = keras.layers.Dense(codings_size)(z)
codings_log_var = keras.layers.Dense(codings_size)(z)

# Sampling layer at the end of the encoder
codings = Sampling()([codings_mean, codings_log_var])
variational_encoder = keras.models.Model(
    inputs=[inputs], outputs=[codings_mean, codings_log_var, codings])

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])
```

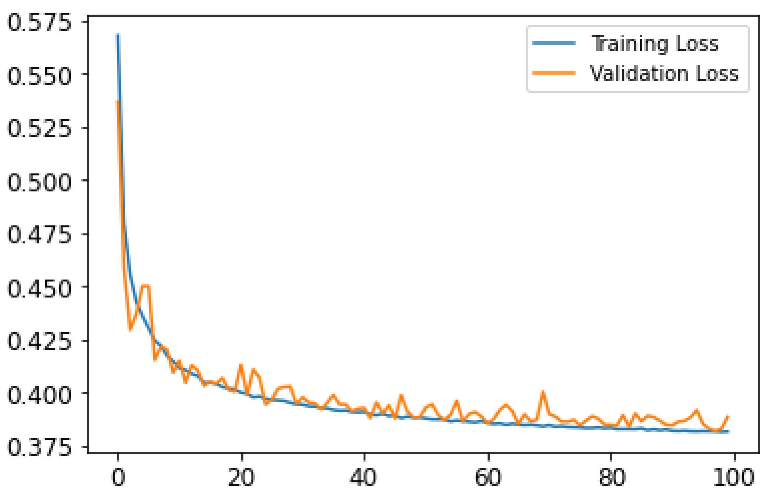
```
_, _, codings = variational_encoder(inputs)
reconstructions = variational_decoder(codings)
variational_ae = keras.models.Model(inputs=[inputs], outputs=[reconstructions])

# The latent loss function
latent_loss = -0.5 * K.sum(
    1 + codings_log_var - K.exp(codings_log_var) - K.square(codings_mean),
    axis=-1)

# 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.
# 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("VAE_model", monitor="val_loss", save_best_on

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

```
Epoch 74/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3839 - val_loss: 0.38
Epoch 75/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3838 - val_loss: 0.38
Epoch 76/100
43/43 [==============================] - ETA: 0s - loss: 0.3837INFO:tensorflow:Assets
43/43 [==============================] - 3s 80ms/step - loss: 0.3837 - val_loss: 0.38
Epoch 77/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3834 - val_loss: 0.38
Epoch 78/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3835 - val_loss: 0.38
Epoch 79/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3837 - val_loss: 0.38
Epoch 80/100
43/43 [==============================] - 1s 12ms/step - loss: 0.3833 - val_loss: 0.38
Epoch 81/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3835 - val_loss: 0.38
Epoch 82/100
40/43 [==========================>...] - ETA: 0s - loss: 0.3831INFO:tensorflow:Assets
43/43 [==============================] - 3s 74ms/step - loss: 0.3829 - val_loss: 0.38
Epoch 83/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3830 - val_loss: 0.38
Epoch 84/100
43/43 [==============================] - ETA: 0s - loss: 0.3830INFO:tensorflow:Assets
43/43 [==============================] - 3s 80ms/step - loss: 0.3830 - val_loss: 0.38
Epoch 85/100
43/43 [==============================] - 0s 11ms/step - loss: 0.3829 - val_loss: 0.39
Epoch 86/100
43/43 [==============================] - 0s 11ms/step - loss: 0.3833 - val_loss: 0.38
Epoch 87/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3824 - val_loss: 0.38
Epoch 88/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3829 - val_loss: 0.38
Epoch 89/100
```

```
Epoch 89/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3824 - val_loss: 0.38
Epoch 90/100
43/43 [==============================] - 0s 11ms/step - loss: 0.3828 - val_loss: 0.38
Epoch 91/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3822 - val_loss: 0.38
Epoch 92/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3820 - val_loss: 0.38
Epoch 93/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3822 - val_loss: 0.38
Epoch 94/100
43/43 [==============================] - 0s 11ms/step - loss: 0.3819 - val_loss: 0.38
Epoch 95/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3819 - val_loss: 0.39
Epoch 96/100
43/43 [==============================] - 0s 10ms/step - loss: 0.3820 - val_loss: 0.38
Epoch 97/100
42/43 [============================>.] - ETA: 0s - loss: 0.3819INFO:tensorflow:Assets
43/43 [==============================] - 3s 72ms/step - loss: 0.3819 - val_loss: 0.38
Epoch 98/100
42/43 [============================>.] - ETA: 0s - loss: 0.3819INFO:tensorflow:Assets
43/43 [==============================] - 3s 81ms/step - loss: 0.3819 - val_loss: 0.38
Epoch 99/100
43/43 [==============================] - 0s 11ms/step - loss: 0.3816 - val_loss: 0.38
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 0x7fa55ae44710>




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

```
 model_10 (Functional)         (None, 28, 28)       243920      ['model_9[0][2]']
|---------------------------------------------------------------------------------
| input_8 (InputLayer)        [(None, 16)]         0           []
|
| dense_32 (Dense)            (None, 64)           1088        []
```

```
| dense_32 (Dense)            (None, 64)          1088         []
|
| dense_33 (Dense)            (None, 128)         8320         []
|
| dense_34 (Dense)            (None, 256)         33024        []
|
| dense_35 (Dense)            (None, 784)         201488       []
|
| reshape_3 (Reshape)         (None, 28, 28)      0            []
-------------------------------------------------------------------------------------
 flatten_3 (Flatten)            (None, 784)         0           ['input_7[0][0]']

 dense_27 (Dense)               (None, 256)         200960      ['flatten_3[0][0]']

 dense_28 (Dense)               (None, 128)         32896       ['dense_27[0][0]']

 dense_29 (Dense)               (None, 64)          8256        ['dense_28[0][0]']

 dense_31 (Dense)               (None, 16)          1040        ['dense_29[0][0]']

 tf.__operators__.add_3 (TFOpLa  (None, 16)         0           ['dense_31[0][0]']
 mbda)

 tf.math.exp_3 (TFOpLambda)     (None, 16)          0           ['dense_31[0][0]']

 dense_30 (Dense)               (None, 16)          1040        ['dense_29[0][0]']

 tf.math.subtract_6 (TFOpLambda  (None, 16)         0           ['tf.__operators__.a
 )                                                              , 'tf.math.exp_3[0][

 tf.math.square_3 (TFOpLambda)  (None, 16)          0           ['dense_30[0][0]']

 tf.math.subtract_7 (TFOpLambda  (None, 16)         0           ['tf.math.subtract_6
 )                                                               'tf.math.square_3[0

 tf.math.reduce_sum_3 (TFOpLamb  (None,)            0           ['tf.math.subtract_7
 da)

 tf.math.multiply_3 (TFOpLambda  (None,)            0           ['tf.math.reduce_sum_
 )

 tf.math.reduce_mean_3 (TFOpLam  ()                 0           ['tf.math.multiply_3
 bda)

 tf.math.truediv_3 (TFOpLambda)  ()                 0           ['tf.math.reduce_mea

 add_loss_3 (AddLoss)           ()                  0           ['tf.math.truediv_3[

=====================================================================================
Total params: 488,112
Trainable params: 488,112
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, 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()
```

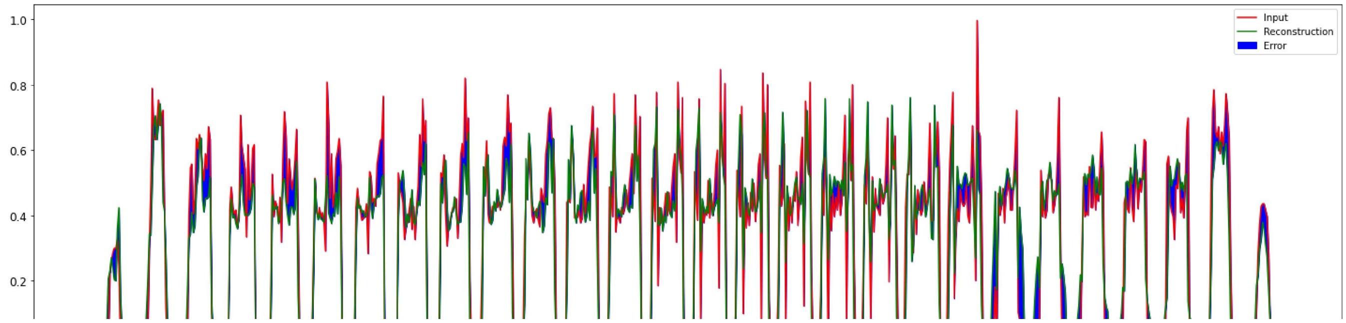
```
show_reconstructions(variational_ae, normal_test_data, 30)
plt.show()
```

```
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)
```

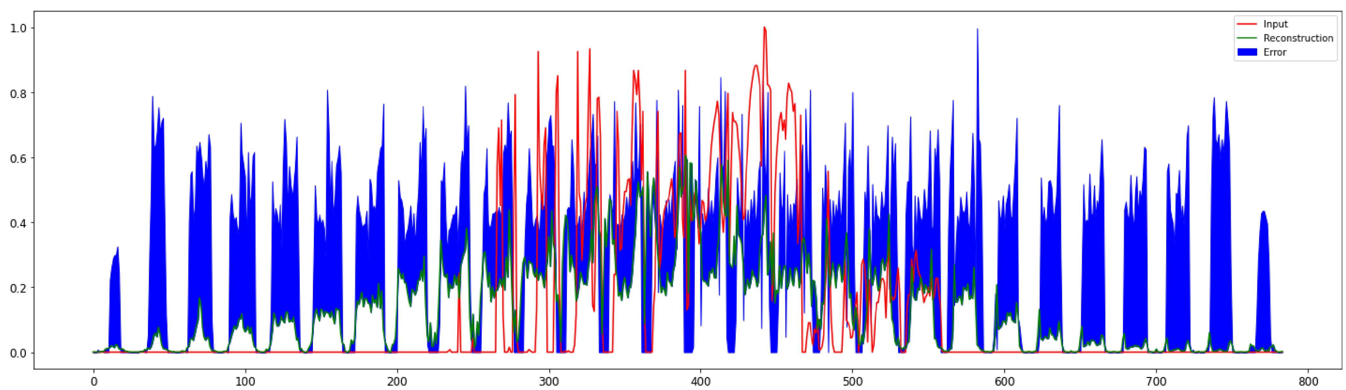
```
plt.figure(figsize=(25,7))
plt.plot(normal_test_data[0].ravel(), 'r')
plt.plot(reconstructions_nl_test[0].ravel(), 'g')
plt.fill_between(np.arange(28*28), reconstructions_nl_test[0].ravel(), normal_test_data[0].ra
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```



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].r
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```

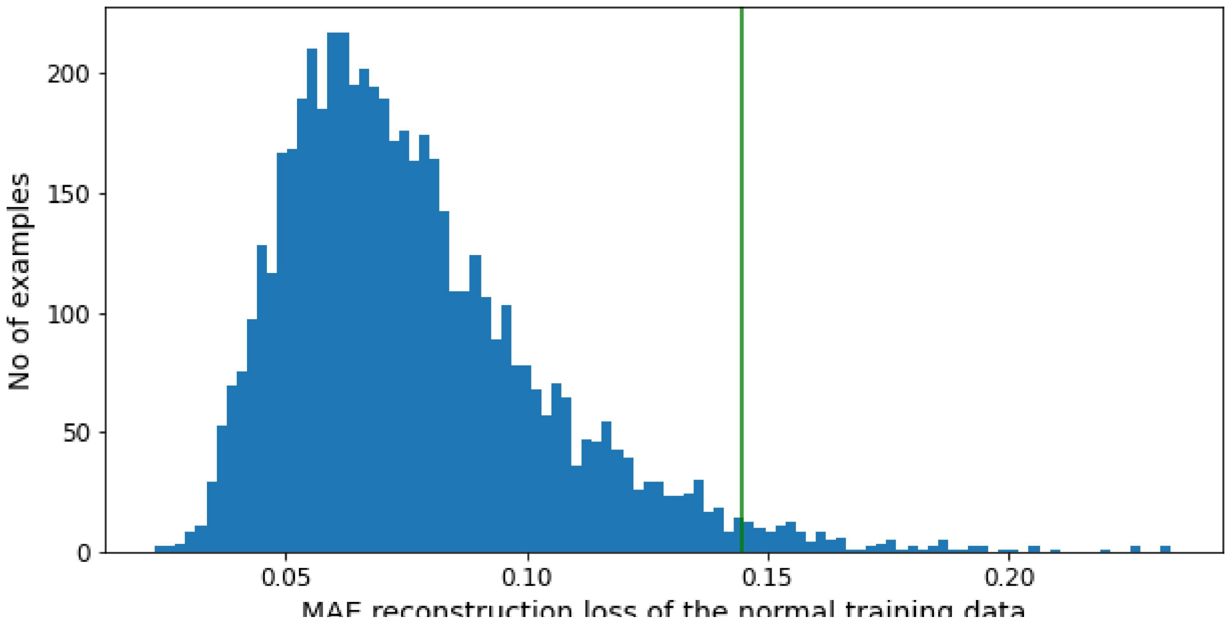


Distributions of the reconstruction losses and the calculation of the threshold.

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, 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:  0.076522365
Std:  0.027244987
```

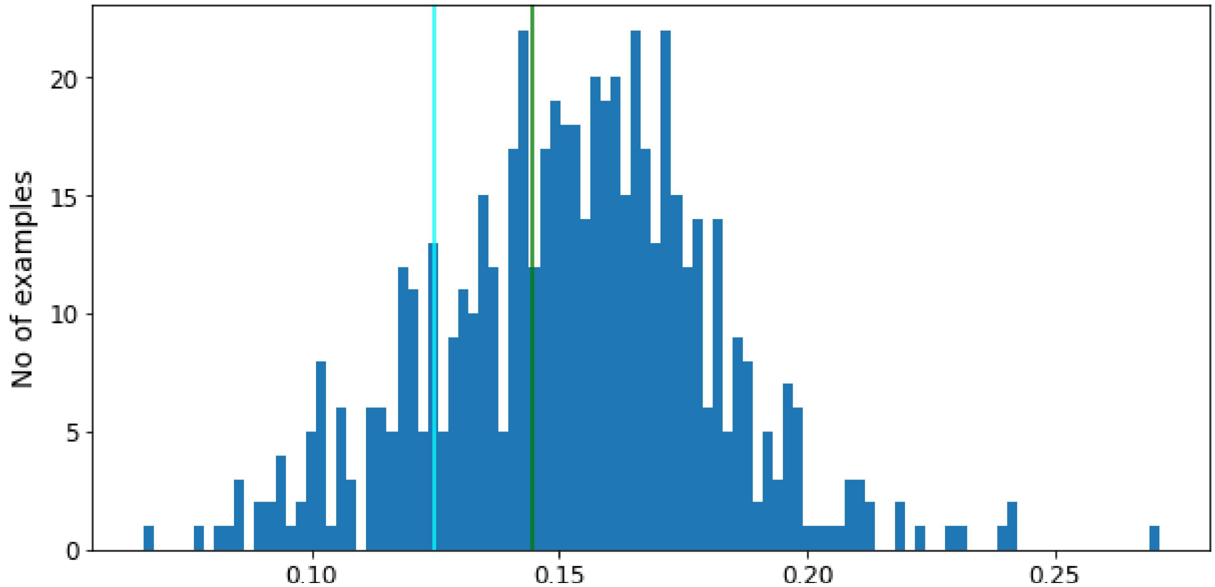
```
threshold_train_mean_2_5_std = np.mean(train_loss) + 2.5*np.std(train_loss)
print("Threshold based on the mean of the training data MAE reconstruction losses + 2.5 std:
```

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

```
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.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)
```

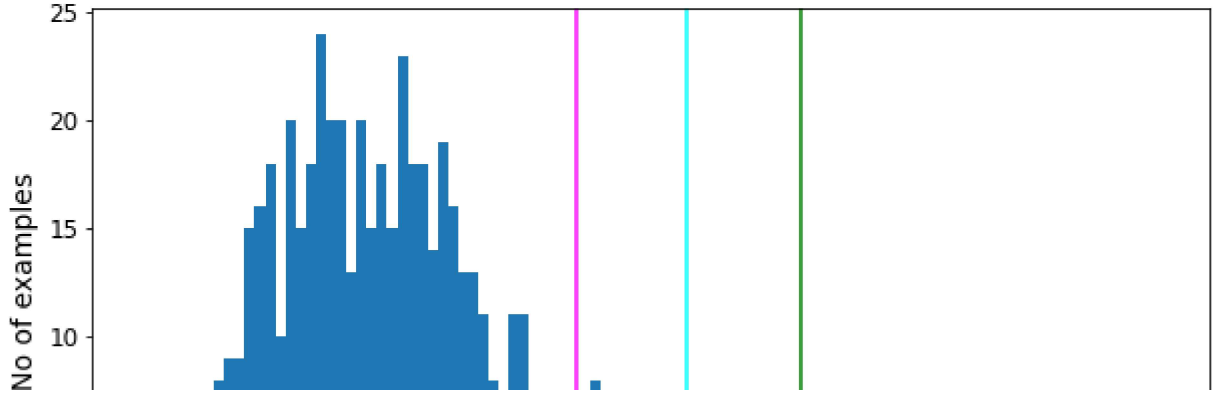
```
(0.15295188, 0.028124807)
```

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

```
Threshold2:  0.12482707
```

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.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)
```

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

```
(0.07834342, 0.02735705)
```

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

```
Threshold3:  0.10570047
```

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:  0.11526377499103546
```

```
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:  0.11117114365100839
```

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

```
array([[0.10967897, 0.89719626],
       [0.11042506, 0.89804588],
       [0.11117114, 0.9039932 ],
       [0.11191723, 0.89974511],
       [0.11266331, 0.89804588]])
```

```
threshold5 = valid_data_best_threshold
```

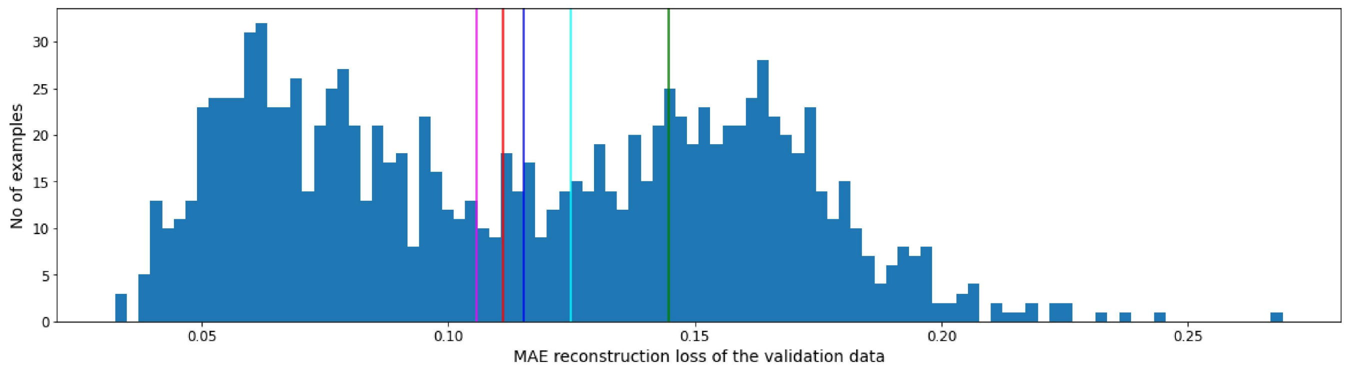
```
threshold = threshold5
```

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

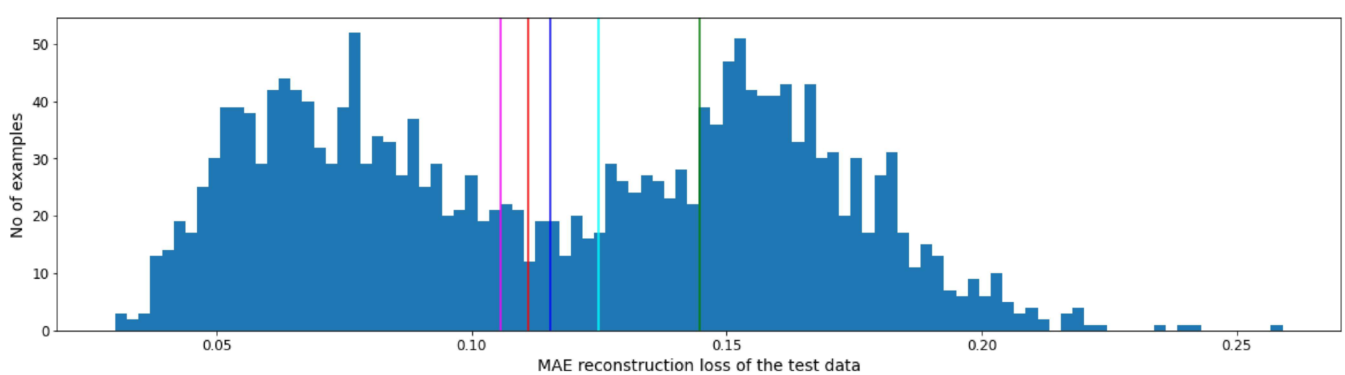


Distribution of the reconstruction losses of the test data (normal and abnormal)

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 = 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)
```

```
(0.07783427, 0.026862532)
```

```
reconstructions = variational_ae.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)
```

```
(0.15448362, 0.026181856)
```

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

```
Confusion Matrix:
```

```
 prediction: F      T
             1043   957
 label: F   [[941   59]    1000
       T    [102   898]]   1000
Accuracy = 0.9195
Normal Test Data Mean = 0.07783427089452744
Normal Test Data Standard Deviation = 0.026862531900405884
Abnormal Test Data Mean = 0.1544836163520813
Abnormal Test Data Standard Deviation = 0.02618185617029667
Precision = 0.9383490073145245
Recall = 0.898
0.9195
0.07783427
0.026862532
0.15448362
0.026181856
0.9383490073145245
0.898
0.9195 0.07783427 0.026862532 0.15448362 0.026181856 0.9383490073145245 0.898
```

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

```
Threshold = 0.11117114365100839
```

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

```
[[941  59]
 [102 898]]
```

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

```
Confusion Matrix:
 prediction: F      T
             1011   989
 label: F   [[926   74]    1000
       T    [85   915]]   1000
Accuracy = 0.9205
Normal Test Data Mean = 0.07783427089452744
```

```
Normal Test Data Standard Deviation = 0.026862531900405884
Abnormal Test Data Mean = 0.1544836163520813
Abnormal Test Data Standard Deviation = 0.02618185617029667
Precision = 0.9251769464105156
Recall = 0.915
0.9205
0.07783427
0.026862532
0.15448362
0.026181856
0.9251769464105156
0.915
0.9205 0.07783427 0.026862532 0.15448362 0.026181856 0.9251769464105156 0.915
```

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)
```

```
Confusion Matrix:
 prediction: F      T
             707   1293
 label: F   [[680   320]    1000
        T    [27   973]]   1000
Accuracy = 0.8265
Normal Test Data Mean = 0.07783427089452744
Normal Test Data Standard Deviation = 0.026862531900405884
Abnormal Test Data Mean = 0.1544836163520813
Abnormal Test Data Standard Deviation = 0.02618185617029667
Precision = 0.7525135344160866
Recall = 0.973
0.8265
0.07783427
0.026862532
0.15448362
0.026181856
0.7525135344160866
0.973
0.8265 0.07783427 0.026862532 0.15448362 0.026181856 0.7525135344160866 0.973
```

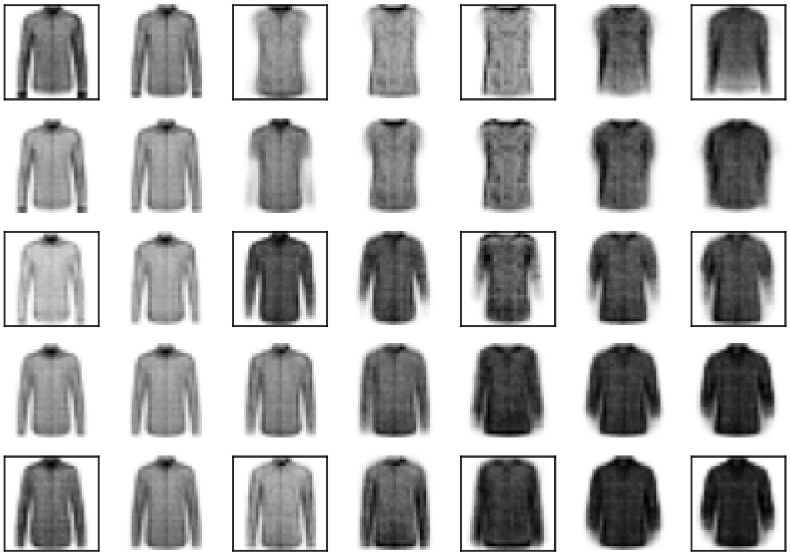
Extra Info

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 1:40 PM