

**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
```

```
n11 = 9
```

```

n12 = 9
abn1 = 3
abn2 = 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 = train_size)

normal_data = x_train[(y_train == n11) | (y_train == n12)]          # Normal training data (Normal digits)
normal_labels = y_train[(y_train == n11) | (y_train == n12)]

valid_data = x_valid[(y_valid == abn1) | (y_valid == abn2) | (y_valid == n11) | (y_valid == n12)]  #
valid_labels = y_valid[(y_valid == abn1) | (y_valid == abn2) | (y_valid == n11) | (y_valid == n12)]

test_data = x_test[(y_test == abn1) | (y_test == abn2) | (y_test == n11) | (y_test == n12)]  # Test d
test_labels = y_test[(y_test == abn1) | (y_test == abn2) | (y_test == n11) | (y_test == n12)]

test_labels_T_F = np.where((test_labels == n11) | (test_labels == n12), 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 == n11) | (valid_labels == n12), 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, test_label

((5403, 28, 28), (5403,), (1238, 28, 28), (1238,), (2000, 28, 28), (2000,))

normal_test_data = test_data[(test_labels == n11) | (test_labels == n12)]          # The normal digit
abnormal_test_data = test_data[(test_labels == abn1) | (test_labels == abn2)]      # The abnormal dig
normal_test_labels = test_labels[(test_labels == n11) | (test_labels == n12)]      # Their labels
abnormal_test_labels = test_labels[(test_labels == abn1) | (test_labels == abn2)]  # Their labels

normal_test_data.shape, abnormal_test_data.shape

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

normal_valid_data = valid_data[(valid_labels == n11) | (valid_labels == n12)]      # The normal d
abnormal_valid_data = valid_data[(valid_labels == abn1) | (valid_labels == abn2)]  # The abnormal
normal_valid_labels = valid_labels[(valid_labels == n11) | (valid_labels == n12)]  # Their labels
abnormal_valid_labels = valid_labels[(valid_labels == abn1) | (valid_labels == abn1)] # Their labels

normal_valid_data.shape, abnormal_valid_data.shape

((597, 28, 28), (641, 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 backpropagation 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.random_normal(tf.shape(log_var_2)) * K.exp(log_var_2 / 2) + mean_2)*(1 - p)

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

# 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, codings_p)
variational_encoder = keras.models.Model(
    inputs=[inputs], outputs=[codings_mean_1, codings_log_var_1, codings_mean_2, codings_log_var_2, codings_p])

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)) # * codi
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="val_loss", s

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

```

```

Epoch 73/100
43/43 [=====] - 1s 13ms/step - loss: 0.2667 - val_loss: 0.2719
Epoch 74/100
41/43 [=====>..] - ETA: 0s - loss: 0.2665INFO:tensorflow:Assets written t
43/43 [=====] - 4s 104ms/step - loss: 0.2663 - val_loss: 0.2716
Epoch 75/100
43/43 [=====] - 1s 12ms/step - loss: 0.2661 - val_loss: 0.2747
Epoch 76/100
43/43 [=====] - 1s 12ms/step - loss: 0.2662 - val_loss: 0.2731
Epoch 77/100
43/43 [=====] - 1s 13ms/step - loss: 0.2659 - val_loss: 0.2717
Epoch 78/100
43/43 [=====] - 1s 12ms/step - loss: 0.2658 - val_loss: 0.2749
Epoch 79/100
43/43 [=====] - 1s 12ms/step - loss: 0.2656 - val_loss: 0.2744
Epoch 80/100
43/43 [=====] - ETA: 0s - loss: 0.2657INFO:tensorflow:Assets written t
43/43 [=====] - 4s 97ms/step - loss: 0.2657 - val_loss: 0.2711
Epoch 81/100
43/43 [=====] - 1s 14ms/step - loss: 0.2654 - val_loss: 0.2769
Epoch 82/100
43/43 [=====] - 1s 13ms/step - loss: 0.2653 - val_loss: 0.2745
Epoch 83/100
43/43 [=====] - 1s 13ms/step - loss: 0.2651 - val_loss: 0.2753
Epoch 84/100
43/43 [=====] - 1s 13ms/step - loss: 0.2651 - val_loss: 0.2717
Epoch 85/100
43/43 [=====] - 1s 13ms/step - loss: 0.2651 - val_loss: 0.2722
Epoch 86/100
40/43 [=====>...] - ETA: 0s - loss: 0.2643INFO:tensorflow:Assets written t

```

```

43/43 [=====] - 4s 102ms/step - loss: 0.2647 - val_loss: 0.2711
Epoch 87/100
43/43 [=====] - 1s 12ms/step - loss: 0.2650 - val_loss: 0.2749
Epoch 88/100
40/43 [=====>...] - ETA: 0s - loss: 0.2650INFO:tensorflow:Assets written t
43/43 [=====] - 4s 95ms/step - loss: 0.2649 - val_loss: 0.2707
Epoch 89/100
43/43 [=====] - 1s 13ms/step - loss: 0.2644 - val_loss: 0.2734
Epoch 90/100
43/43 [=====] - 1s 12ms/step - loss: 0.2646 - val_loss: 0.2711
Epoch 91/100
43/43 [=====] - 1s 12ms/step - loss: 0.2644 - val_loss: 0.2728
Epoch 92/100
43/43 [=====] - 1s 12ms/step - loss: 0.2643 - val_loss: 0.2756
Epoch 93/100

43/43 [=====] - 1s 13ms/step - loss: 0.2643 - val_loss: 0.2732
Epoch 94/100
43/43 [=====] - 1s 12ms/step - loss: 0.2639 - val_loss: 0.2753
Epoch 95/100
43/43 [=====] - 1s 13ms/step - loss: 0.2642 - val_loss: 0.2716
Epoch 96/100
43/43 [=====] - 1s 13ms/step - loss: 0.2639 - val_loss: 0.2723
Epoch 97/100
43/43 [=====] - 1s 13ms/step - loss: 0.2641 - val_loss: 0.2790
Epoch 98/100
43/43 [=====] - 1s 13ms/step - loss: 0.2638 - val_loss: 0.2741
Epoch 99/100

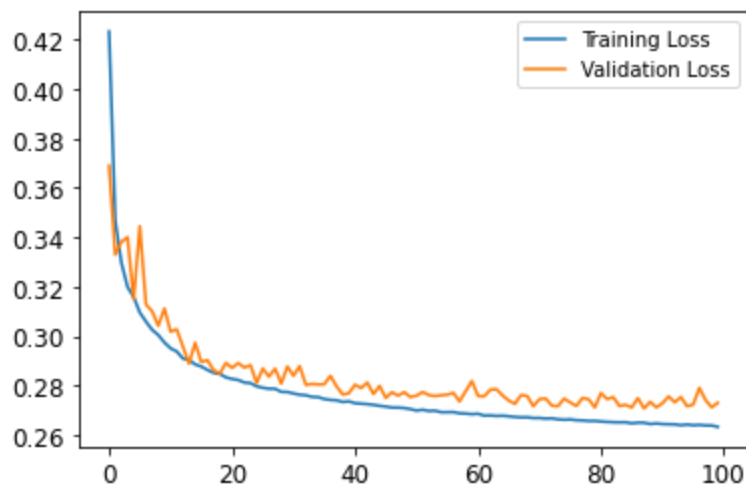
```

```

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

```

<matplotlib.legend.Legend at 0x7fe2e042ed10>



```

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

```

tf.math.subtract_10 (TFOpLambd (None, 16)	0	['dense_28[0][0]', 'tf.math.exp_4[0][0]']
a)		
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[0][0]', 'tf.math.square_4[0][0]']
tf.math.subtract_12 (TFOpLambd a)	( )	0	['tf.math.reduce_mean_4[0][0]']
tf.math.subtract_14 (TFOpLambd a)	(None, 16)	0	['tf.math.subtract_13[0][0]', 'tf.math.square_5[0][0]']
tf.math.multiply_8 (TFOpLambda )	(None, 16)	0	['tf.math.reduce_mean_4[0][0]', 'tf.math.subtract_11[0][0]']
tf.math.multiply_9 (TFOpLambda )	(None, 16)	0	['tf.math.subtract_12[0][0]', 'tf.math.subtract_14[0][0]']
tf.__operators__.add_6 (TFOpLa mbda)	(None, 16)	0	['tf.math.multiply_8[0][0]']
tf.__operators__.add_7 (TFOpLa mbda)	(None, 16)	0	['tf.math.multiply_9[0][0]']
tf.math.reduce_sum_4 (TFOpLamb da)	(None, )	0	['tf.__operators__.add_6[0][0] ']
tf.math.reduce_sum_5 (TFOpLamb da)	(None, )	0	['tf.__operators__.add_7[0][0] ']
tf.__operators__.add_8 (TFOpLa mbda)	(None, )	0	['tf.math.reduce_sum_4[0][0]', 'tf.math.reduce_sum_5[0][0]']
tf.math.multiply_10 (TFOpLambd a)	(None, )	0	['tf.__operators__.add_8[0][0] ']
tf.math.multiply_11 (TFOpLambd a)	(None, )	0	['tf.math.multiply_10[0][0]']
tf.math.reduce_mean_5 (TFOpLam bda)	( )	0	['tf.math.multiply_11[0][0]']
tf.math.truediv_2 (TFOpLambda)	( )	0	['tf.math.reduce_mean_5[0][0]']
add_loss_2 (AddLoss)	( )	0	['tf.math.truediv_2[0][0]']

=====  
Total params: 490,257

```
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)), axis = 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()

```



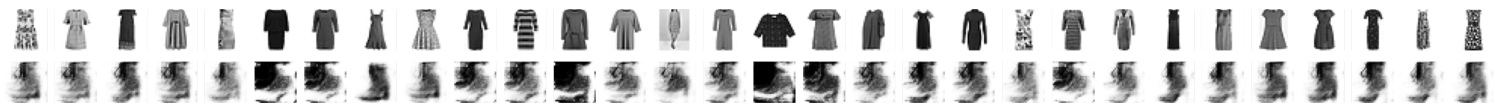
```

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()
```

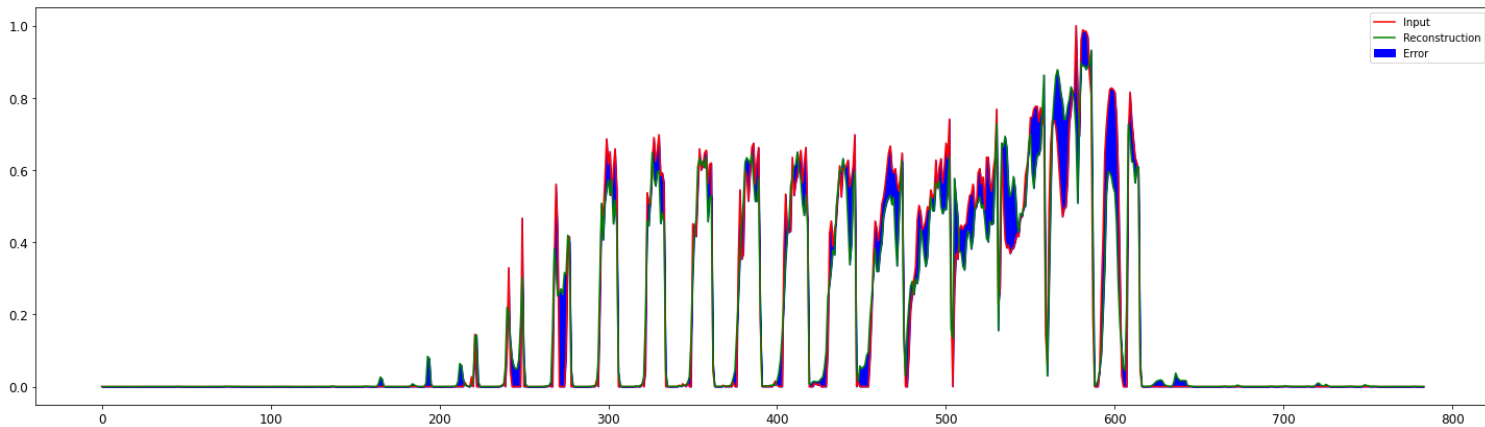


## 1-Dim plot of pixels of the first normal test data

```
reconstructions_n1_test = variational_ae.predict(normal_test_data)
```



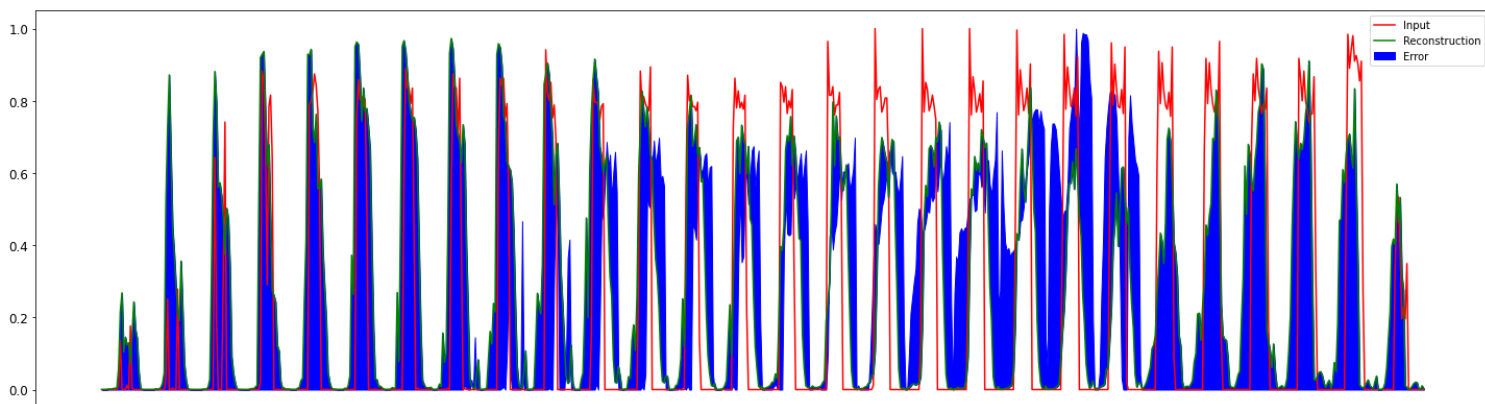
```
plt.figure(figsize=(25,7))
plt.plot(normal_test_data[0].ravel(), 'r')
plt.plot(reconstructions_nl_test[0].ravel(), 'g')
plt.fill_between(np.arange(28*28), reconstructions_nl_test[0].ravel(), normal_test_data[0].ravel(), color='b')
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()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



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

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

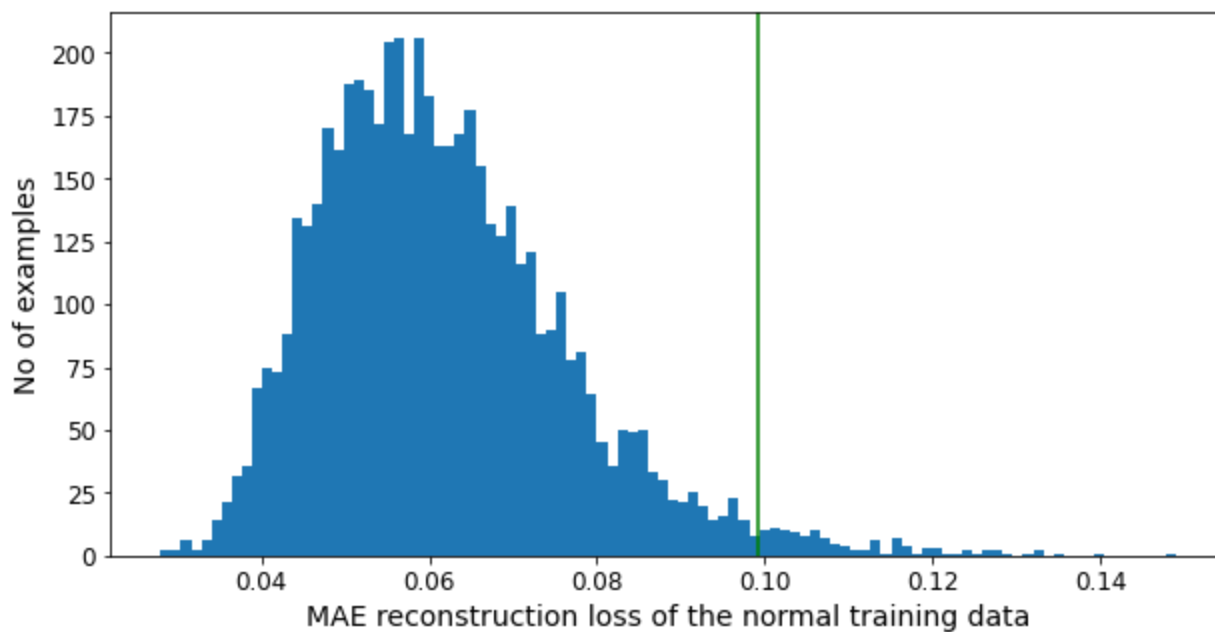
```
plt.figure(figsize=(25,7))
plt.plot(abnormal_test_data[0].ravel(), 'r')
plt.plot(reconstructions_abn_test[0].ravel(), 'g')
plt.fill_between(np.arange(28*28), reconstructions_abn_test[0].ravel(), abnormal_test_data[0].ravel(), color='b')
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()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



## ▼ 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()
```



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

```
Mean: 0.061750036
Std: 0.014999549
```

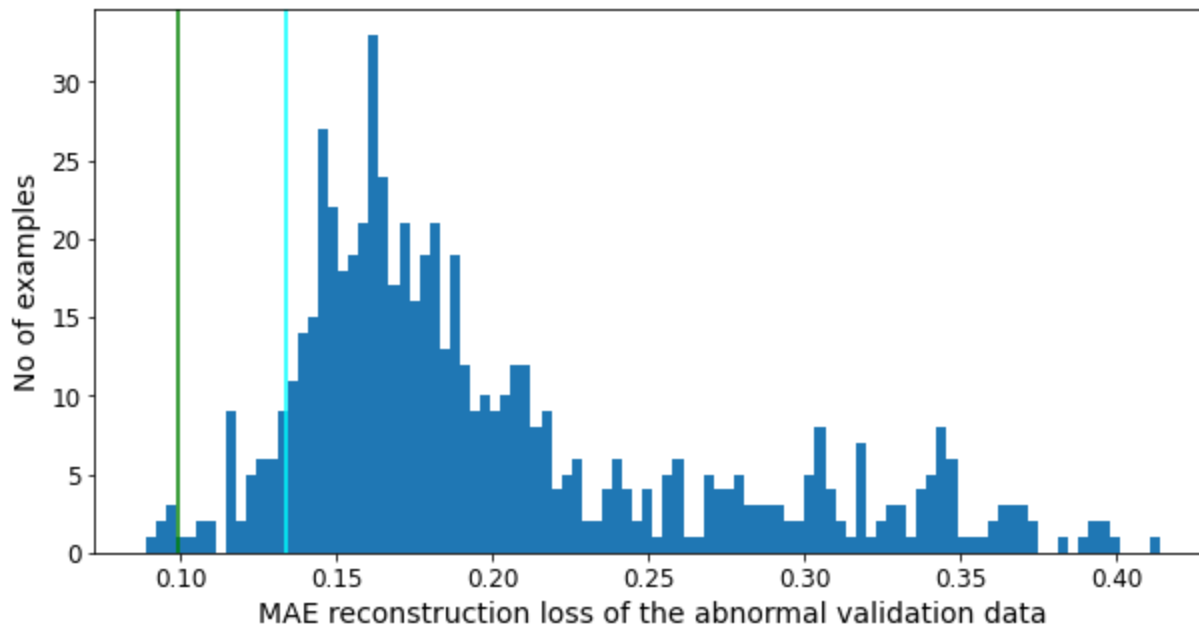
```
threshold_train_mean_2_5_std = np.mean(train_loss) + 2.5*np.std(train_loss)
print("Threshold based on the mean of the training data MAE reconstruction losses + 2.5 std: ", threshold_train_mean_2_5_std)
```

Threshold based on the mean of the training data MAE reconstruction losses + 2.5 std: 0.09924890

```
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)
```

```
(0.20084028, 0.06730281)
```

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

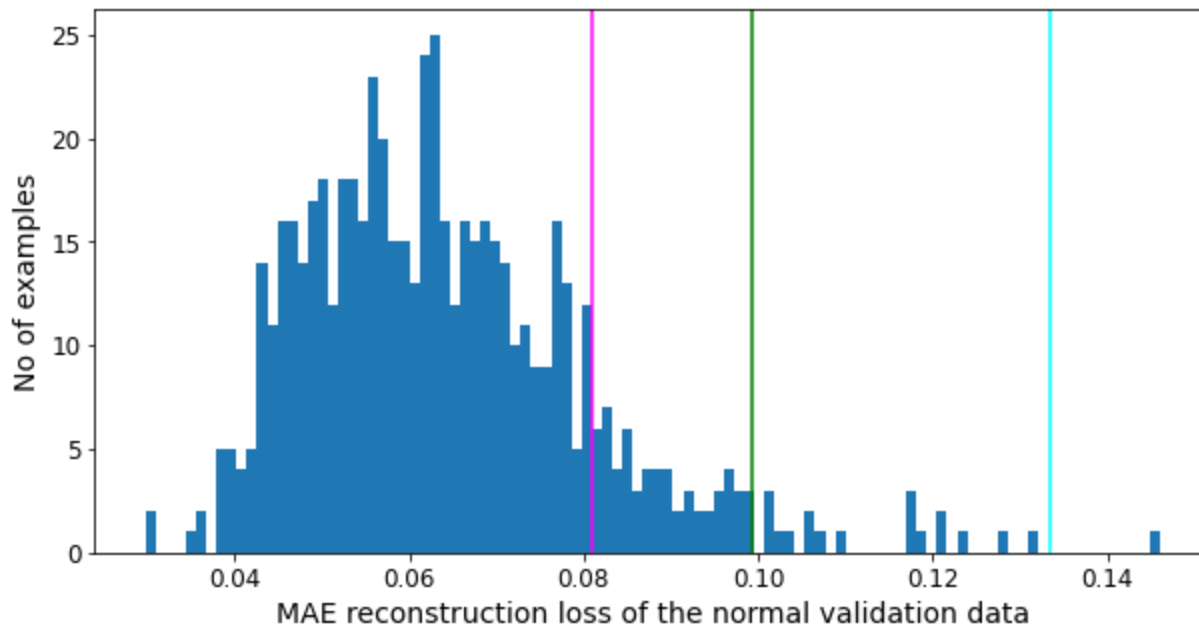
```
Threshold2: 0.13353747
```

### 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)

```

```

(0.064279325, 0.016569784)

```

```

threshold3 = normal_valid_mean_loss + np.std(nl_valid_loss)

```

```

print("Threshold3: ", threshold3)

```

```

Threshold3: 0.08084911

```

**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: 0.10719329118728638

```

```
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:  0.1052476093173027
```

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

    array([[0.10251639, 0.98142165],
           [0.103882  , 0.98384491],
           [0.10524761, 0.98465267],
           [0.10661322, 0.9822294  ],
           [0.10797883, 0.98303716]])
```

```
threshold5 = valid_data_best_threshold
```

```
threshold = threshold5
```

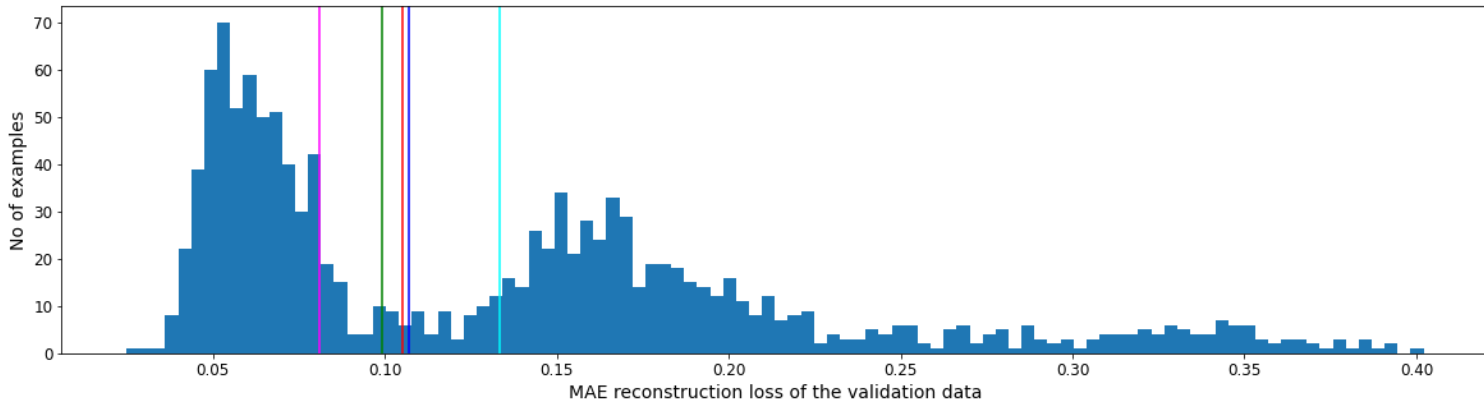
## ▼ Distribution of the reconstruction losses of all the validation data (normal and abnormal)

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()
```

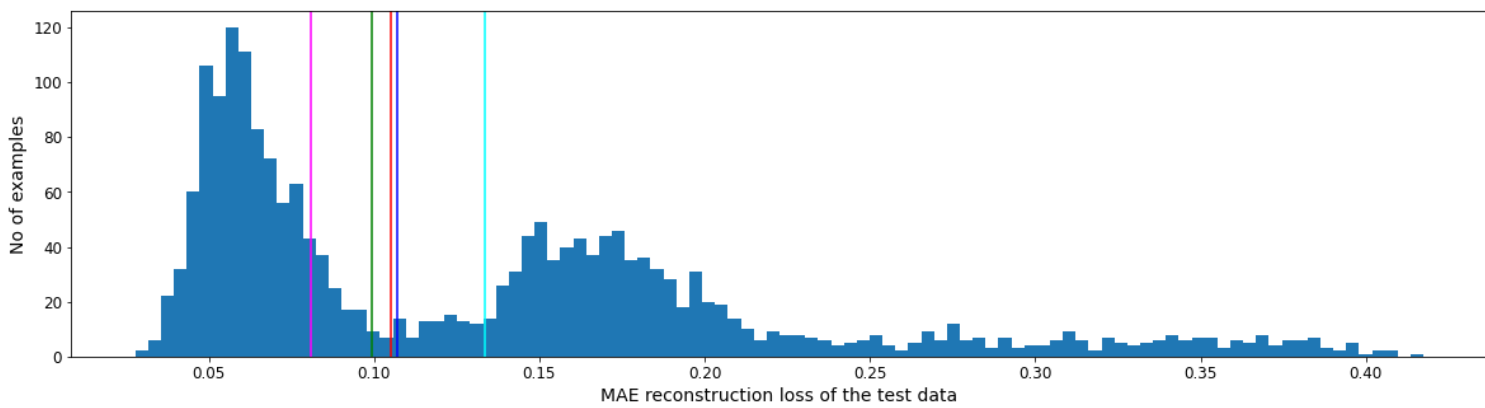


## ▼ Distribution of the reconstruction losses of the test data (normal and abnormal)

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(-1,784))
np.mean(nl_test_loss), np.std(nl_test_loss)
```

```
(0.064153805, 0.018380936)
```

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

```
(0.20320335, 0.071328335)
```

## ▼ 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].shape[0]))
    print(" label: F  [[{}  {}]  {}".format(cf[0,0], cf[0,1], test_labels_T_F[test_labels_T_F == False].shape[0]))
    print("      T  [[{}  {}]] {}".format(cf[1,0], cf[1,1], test_labels_T_F[test_labels_T_F == True].shape[0]))
    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.mean(abn_t
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
              1017  983
label: F      [[990  10]  1000
              T      [27  973]]  1000
Accuracy = 0.9815
Normal Test Data Mean = 0.06415380537509918
Normal Test Data Standard Deviation = 0.018380936235189438
Abnormal Test Data Mean = 0.20320335030555725
Abnormal Test Data Standard Deviation = 0.07132833451032639
Precision = 0.9898270600203459
Recall = 0.973
0.9815
0.064153805
0.018380936
0.20320335
0.071328335
0.9898270600203459
0.973
0.9815 0.064153805 0.018380936 0.20320335 0.071328335 0.9898270600203459 0.973

```

```

print("Threshold =", valid_data_best_threshold)

```

```

Threshold = 0.1052476093173027

```

```

print(confusion_matrix(test_labels_T_F, preds))

```

```

[[990  10]
 [ 27 973]]

```

## 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 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
           1015   985
label: F    [[991   9]   1000
           T    [24  976]]  1000
Accuracy = 0.9835
Normal Test Data Mean = 0.06415380537509918
Normal Test Data Standard Deviation = 0.018380936235189438
Abnormal Test Data Mean = 0.20320335030555725
Abnormal Test Data Standard Deviation = 0.07132833451032639
Precision = 0.9908629441624366
Recall = 0.976
0.9835
0.064153805
0.018380936
0.20320335
0.071328335
0.9908629441624366
0.976
0.9835 0.064153805 0.018380936 0.20320335 0.071328335 0.9908629441624366 0.976

```

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
           1039   961
label: F    [[997   3]   1000
           T    [42  958]]  1000
Accuracy = 0.9775
Normal Test Data Mean = 0.06415380537509918
Normal Test Data Standard Deviation = 0.018380936235189438
Abnormal Test Data Mean = 0.20320335030555725
Abnormal Test Data Standard Deviation = 0.07132833451032639
Precision = 0.9968782518210197
Recall = 0.958
0.9775
0.064153805
0.018380936
0.20320335
0.071328335
0.9968782518210197
0.958
0.9775 0.064153805 0.018380936 0.20320335 0.071328335 0.9968782518210197 0.958

```

## 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 4:00 PM

