```
'''This script demonstrates how to build a variational autoencoder with Ke

 #Reference

 - Auto-Encoding Variational Bayes
   https://arxiv.org/abs/1312.6114
'''
from __future__ import print_function

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

from keras.layers import Input, Dense, Lambda
from keras.models import Model
from keras import backend as K
from keras import metrics
from keras.datasets import mnist
from functools import reduce

batch_size = 100
original_dim = 784
latent_dim = 2
intermediate_dim = 16
epochs = 50
epsilon_std = 1.0


input_shape = (28, 28, 1)
inputs = Input(shape=input_shape, name='encoder_input')
x = inputs
x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
# shape info needed to build decoder model
shape = K.int_shape(x)
# generate latent vector Q(z|X)
x = Flatten()(x)
x = Dense(intermediate_dim, activation='relu')(x)
z_mean = Dense(latent_dim, name='z_mean')(x)
z_log_var = Dense(latent_dim, name='z_log_var')(x)


def sampling(args):
    z_mean, z_log_var = args
    epsilon = K.random_normal(shape=(K.shape(z_mean)[0], latent_dim), mean
                              stddev=epsilon_std)
    return z_mean + K.exp(z_log_var / 2) * epsilon

z = Lambda(sampling, output_shape=(latent_dim,))([z_mean, z_log_var])

# we instantiate these layers separately so as to reuse them later
decode_layers = [Dense(intermediate_dim, activation='relu'),
```

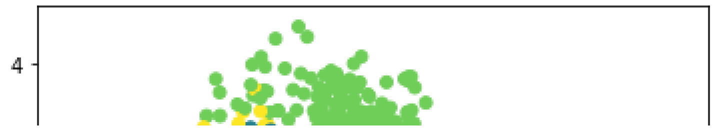
```
decode_layers = [Dense(intermediate_dim, activation='relu'),
          Dense(shape[1] * shape[2] * shape[3], activation='relu'),
          Reshape((shape[1], shape[2], shape[3])),
          Conv2D(8, (3, 3), activation='relu', padding='same'),
          UpSampling2D((2, 2)),
          Conv2D(16, (3, 3), activation='relu', padding='same'),
          UpSampling2D((2, 2)),
          Conv2D(1, (3, 3), activation='sigmoid', padding='same')]

outputs = reduce(lambda x, f: f(x), decode_layers, z)

# instantiate VAE model
vae = Model(inputs, outputs, name='vae')

# Compute VAE loss
xent_loss = original_dim * metrics.binary_crossentropy(K.flatten(inputs),
kl_loss = - 0.5 * K.sum(1 + z_log_var - K.square(z_mean) - K.exp(z_log_var
vae_loss = K.mean(xent_loss + kl_loss)

vae.add_loss(vae_loss)
vae.compile(optimizer='rmsprop')
vae.summary()


# train the VAE on MNIST digits
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))

vae.fit(x_train,
        shuffle=True,
        epochs=epochs,
        batch_size=batch_size,
        validation_data=(x_test, None))
```

```
Epoch 22/50
60000/60000 [==============================] - 9s 151us/step - loss: 151.8245 - val
Epoch 23/50
60000/60000 [==============================] - 9s 152us/step - loss: 151.5922 - val
Epoch 24/50
60000/60000 [==============================] - 9s 153us/step - loss: 151.4303 - val
Epoch 25/50
60000/60000 [==============================] - 9s 153us/step - loss: 151.2518 - val
Epoch 26/50
60000/60000 [==============================] - 9s 153us/step - loss: 151.0843 - val
Epoch 27/50
60000/60000 [==============================] - 9s 154us/step - loss: 150.9169 - val
Epoch 28/50
60000/60000 [==============================] - 9s 152us/step - loss: 150.7665 - val
Epoch 29/50
60000/60000 [==============================] - 9s 153us/step - loss: 150.6571 - val
Epoch 30/50
60000/60000 [==============================] - 9s 153us/step - loss: 150.5704 - val
Epoch 31/50
60000/60000 [==============================] - 9s 154us/step - loss: 150.4213 - val
Epoch 32/50
60000/60000 [==============================] - 9s 151us/step - loss: 150.2776 - val
Epoch 33/50
60000/60000 [==============================] - 9s 150us/step - loss: 150.1554 - val
Epoch 34/50
60000/60000 [==============================] - 9s 151us/step - loss: 150.0152 - val
Epoch 35/50
60000/60000 [==============================] - 9s 151us/step - loss: 149.8994 - val
Epoch 36/50
60000/60000 [==============================] - 9s 152us/step - loss: 149.8041 - val
Epoch 37/50
60000/60000 [==============================] - 9s 153us/step - loss: 149.7141 - val
Epoch 38/50
60000/60000 [==============================] - 9s 153us/step - loss: 149.6008 - val
Epoch 39/50
60000/60000 [==============================] - 9s 153us/step - loss: 149.5477 - val
Epoch 40/50
60000/60000 [==============================] - 9s 152us/step - loss: 149.3931 - val
Epoch 41/50
60000/60000 [==============================] - 9s 154us/step - loss: 149.3262 - val
Epoch 42/50
60000/60000 [==============================] - 9s 154us/step - loss: 149.2467 - val
Epoch 43/50
60000/60000 [==============================] - 9s 151us/step - loss: 149.1884 - val
Epoch 44/50
60000/60000 [==============================] - 9s 152us/step - loss: 149.1181 - val
Epoch 45/50
60000/60000 [==============================] - 9s 152us/step - loss: 149.0389 - val
Epoch 46/50
60000/60000 [==============================] - 9s 152us/step - loss: 148.9470 - val
Epoch 47/50
60000/60000 [==============================] - 9s 153us/step - loss: 148.9101 - val
Epoch 48/50
60000/60000 [==============================] - 9s 153us/step - loss: 148.7883 - val
Epoch 49/50
60000/60000 [==============================] - 9s 153us/step - loss: 148.7192 - val
Epoch 50/50
60000/60000 [==============================] - 9s 152us/step - loss: 148.7282 - val
<keras.callbacks.History at 0x7fbf7175b940>
```

Here we start writing our code

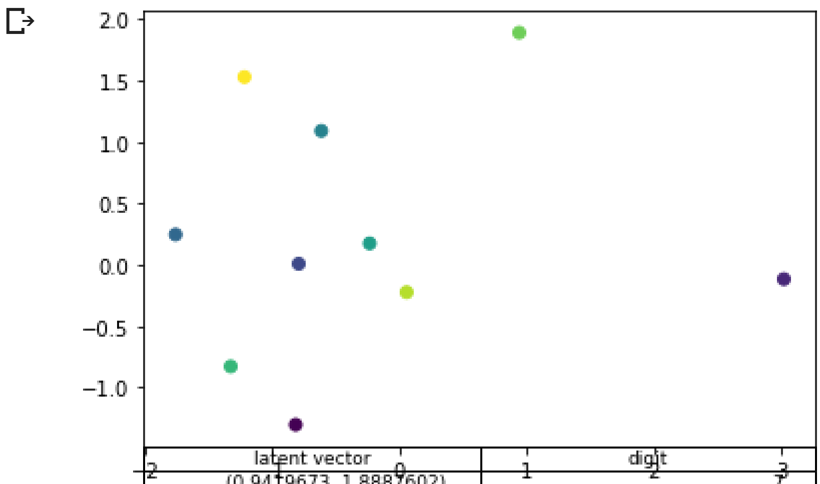
Section C - Add an encoder which maps MNIST digits to the latent space. Using this encoder, visualize the test set in the latent space

```
# encoder which maps MNIST digits to the latent space
encoder = Model(inputs, z_mean)

# visualize the test set in the latent space
x_test_encoded = encoder.predict(x_test, batch_size=batch_size)
plt.scatter(x_test_encoded[:, 0], x_test_encoded[:, 1], c=y_test)
plt.show()
```



Section C - Take one image per digit and print its corresponding mapping coordinates in the latent space, present the answer as a table.

```
# Take one image per digit and print its corresponding mapping coordinates
# in the latent space, present the answer as a table
NUM_OF_DIGITS = 10
digits_to_latent = {}
for i, digit in enumerate(y_test):
    if len(digits_to_latent) == NUM_OF_DIGITS:
        break
    if digit in digits_to_latent:
        continue
    digits_to_latent[digit] = (x_test_encoded[i, 0], x_test_encoded[i, 1])
assert len(digits_to_latent) == NUM_OF_DIGITS
plt.scatter([latent[0] for latent in digits_to_latent.values()], [latent[1
plt.table(colLabels=('latent vector', 'digit'),
                cellText=[[latent, digit] for digit, latent in digits_to_l
plt.show()
```

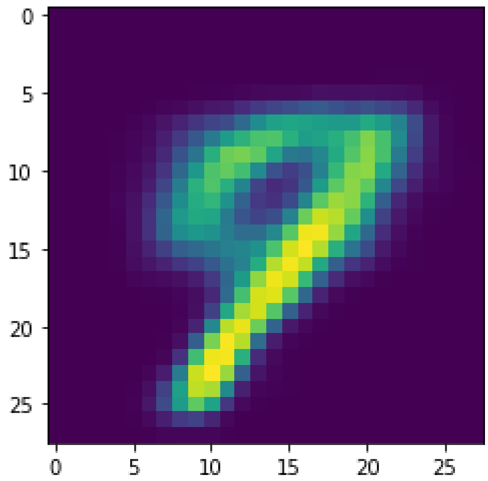


latent vector	digit
(0.9419673, 1.8887602)	7
(-0.79134905, 0.006724164)	2
(3.022952, -0.11835402)	1
(-0.815526, -1.3063334)	0
(-0.61316264, 1.0872365)	4
(-1.2191329, 1.5268204)	9
(-0.23251958, 0.17312841)	5
(-1.3270817, -0.83090216)	6
(-1.7577021, 0.24547064)	3
(0.05874099, -0.225311)	8

Section D - Use the following code to define a generator that based on a sample from the latent

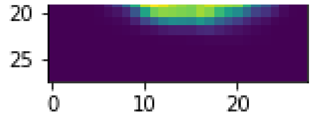
```
# Use the following code to define a generator that based on a sample from
# the latent space, generates a digits.
gen_input = Input(shape=(latent_dim,))
gen_output = reduce(lambda x, f: f(x), decode_layers, gen_input)
generator = Model(inputs=gen_input, outputs=gen_output)
z_sample = np.array([[0.5, 0.2]])
x_decoded = generator.predict(z_sample)


plt.imshow(x_decoded.reshape(28,28))
```

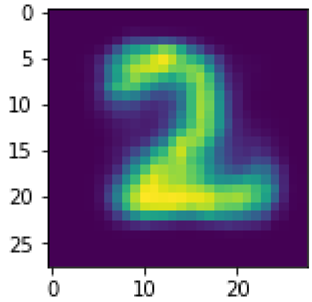
```
<matplotlib.image.AxesImage at 0x7fbf715085f8>
```

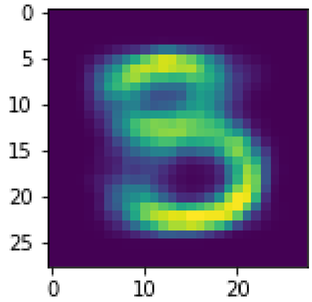


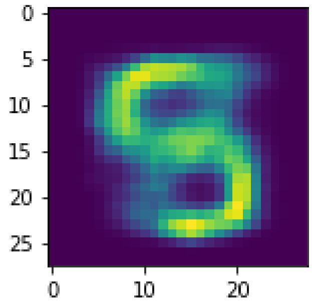
Section E - Take two original images from MNIST of different digits. Sample 10 points from the line connecting the two representations in the latent space and generate their images

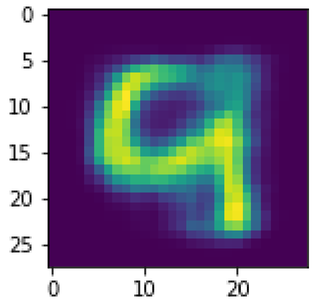
```
# Take two original images from MNIST of different digits
FIRST_DIGIT = 0
SECOND_DIGIT = 9
first_z = digits_to_latent[FIRST_DIGIT]
second_z = digits_to_latent[SECOND_DIGIT]

# Sample 10 points from the line connecting the two representations
# in the latent space and generate their images
SAMPLE_AMOUNT = 10
sampled = list(zip(np.linspace(first_z[0], second_z[0], SAMPLE_AMOUNT), np
fig=plt.figure(figsize=(28, 28))
columns = 1
rows = 10
for i, z_sample in enumerate(sampled):
  x_sample = generator.predict(np.array([list(z_sample)]))
  fig.add_subplot(rows, columns, i+1)
  plt.imshow(x_sample.reshape(28,28))
plt.show()
```

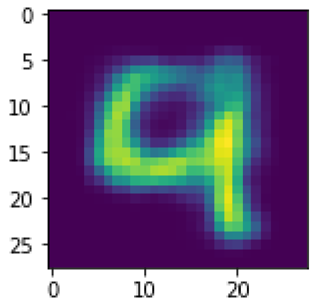

20
25
0
10
20

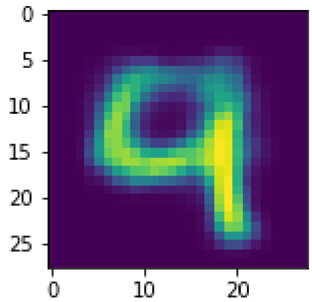

0
5
10
15
20
25
0
10
20


0
5
10
15
20
25
0
10
20


0
5
10
15
20
25
0
10
20


0
5
10
15
20
25
0
10
20


0
5
10
15
20
25
0
10
20


0
5
10
15
20
25
0
10
20