Assignment 12

Name: Kesav Adithya Venkidusamy

Course: DSC650 - Big Data

Instructor: Amirfarrokh Iranitalab

Using section 8.4 in Deep Learning with Python as a guide, implement a variational autoencoder using the MNIST data set and save a grid of 15 x 15 digits to the `results/vae` directory. If you would rather work on a more interesting dataset, you can use the **CelebFaces Attributes** Dataset instead.

In [1]:

```
## Importing libraries required for this activity
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

import keras
from keras import layers
from keras import backend as K
from keras.models import Model
import numpy as np
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.8/site-packages/tensorflow/python/compat/v2_compat.py:96: disable_resource
_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

In [2]:

```
## Defining the parameters and input image
img_shape = (28,28, 1)
batch_size = 16
latent_dim = 2

input_img = keras.Input(shape=img_shape)

x = layers.Conv2D(32, 3, padding ='same', activation='relu')(input_img)
x = layers.Conv2D(64, 3, padding = 'same', activation='relu', strides=(2, 2))(x)
x = layers.Conv2D(64, 3, padding = 'same', activation='relu')(x)
x = layers.Conv2D(64, 3, padding = 'same', activation='relu')(x)
shape_before_flattening = K.int_shape(x)

x = layers.Flatten()(x)
x = layers.Dense(32, activation='relu')(x)
```

```
z_mean = layers.Dense(latent_dim)(x)
z_log_var = layers.Dense(latent_dim)(x)
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.8/site-packages/tensorflow/python/ops/resource_variable_ops.py:1659: calli
ng BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and wil
l be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

Latent-space-sampling function

In [3]:

```
## Defining the sampling function

def sampling(args):
    z_mean, z_log_var = args
    epsilon = K.random_normal(shape=(K.shape(z_mean)[0], latent_dim), mean=0., stddev=1.)
    return z_mean + K.exp(z_log_var) * epsilon

z = layers.Lambda(sampling)([z_mean, z_log_var])
```

VAE decoder network, mapping latent space points to images

In [4]:

```
decoder_input = layers.Input(K.int_shape(z)[1:])
x = layers.Dense(np.prod(shape_before_flattening[1:]), activation='relu')(decoder_input)

x = layers.Reshape(shape_before_flattening[1:])(x)
x = layers.Conv2DTranspose(32, 3, padding='same', activation='relu', strides=(2, 2))(x)
x = layers.Conv2D(1, 3, padding='same', activation='sigmoid')(x)

decoder = Model(decoder_input, x)
z_decoded = decoder(z)
```

In [5]:

```
# View summary of decoder
decoder.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         [(None, 2)]               0         
_________________________________________________________________
dense_3 (Dense)              (None, 12544)             37632     
```

```
_________________________________________________________________
reshape (Reshape)            (None, 14, 14, 64)        0         
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 28, 28, 32)        18464     
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 1)         289       
=================================================================
Total params: 56,385
Trainable params: 56,385
Non-trainable params: 0
_________________________________________________________________
```

Custom layer used to compute the VAE loss

In [6]:

```
class CustomVariationalLayer(keras.layers.Layer):

    def vae_loss(self, x, z_decoded):
        x = K.flatten(x)
        z_decoded = K.flatten(z_decoded)
        xent_loss = keras.metrics.binary_crossentropy(x, z_decoded)
        kl_loss = -5e-4 * K.mean(1 + z_log_var - K.square(z_mean) - K.exp(z_log_var), axis=-1)
        return K.mean(xent_loss + kl_loss)

    def call(self, inputs):
        x = inputs[0]
        z_decoded = inputs[1]
        loss = self.vae_loss(x, z_decoded)
        self.add_loss(loss, inputs=inputs)
        return x

y = CustomVariationalLayer()([input_img, z_decoded])
```

Training the MNIST VAE

In [7]:

```
from keras.datasets import mnist
vae = Model(input_img, y)
vae.compile(optimizer='rmsprop', loss=None)
vae.summary()

(x_train, _), (x_test, y_test) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_train = x_train.reshape(x_train.shape + (1,))

x_test = x_test.astype('float32') / 255.
x_test = x_test.reshape(x_test.shape + (1,))
```

```
vae.fit(x=x_train, y=None, shuffle=True, epochs=10, batch_size=batch_size, validation_data=(x_test, None))
```

```
WARNING:tensorflow:Output custom_variational_layer missing from loss dictionary. We assume this was done on purpose. The
fit and evaluate APIs will not be expecting any data to be passed to custom_variational_layer.
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 28, 28, 1)]  0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 28, 28, 32)   320         input_1[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 14, 14, 64)   18496       conv2d[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 14, 14, 64)   36928       conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 14, 14, 64)   36928       conv2d_2[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 12544)        0           conv2d_3[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 32)           401440      flatten[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 2)            66          dense[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 2)            66          dense[0][0]
__________________________________________________________________________________________________
lambda (Lambda)                 (None, 2)            0           dense_1[0][0]
                                                                 dense_2[0][0]
__________________________________________________________________________________________________
model (Model)                   (None, 28, 28, 1)    56385       lambda[0][0]
__________________________________________________________________________________________________
custom_variational_layer (Custo (None, 28, 28, 1)    0           input_1[0][0]
                                                                 model[1][0]
==================================================================================================
Total params: 550,629
Trainable params: 550,629
Non-trainable params: 0
__________________________________________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 91s 2ms/sample - loss: 0.2164 - val_loss: 0.1982
Epoch 2/10
60000/60000 [==============================] - 88s 1ms/sample - loss: 0.1960 - val_loss: 0.1926
Epoch 3/10
60000/60000 [==============================] - 89s 1ms/sample - loss: 0.1913 - val_loss: 0.1884
Epoch 4/10
60000/60000 [==============================] - 88s 1ms/sample - loss: 0.1886 - val_loss: 0.1906
```

```
Epoch 5/10
60000/60000 [==============================] - 89s 1ms/sample - loss: 0.1867 - val_loss: 0.1861
Epoch 6/10
60000/60000 [==============================] - 88s 1ms/sample - loss: 0.1852 - val_loss: 0.1840
Epoch 7/10
60000/60000 [==============================] - 88s 1ms/sample - loss: 0.1841 - val_loss: 0.1843
Epoch 8/10
60000/60000 [==============================] - 88s 1ms/sample - loss: 0.1832 - val_loss: 0.1827
Epoch 9/10
60000/60000 [==============================] - 89s 1ms/sample - loss: 0.1822 - val_loss: 0.1825
Epoch 10/10
60000/60000 [==============================] - 88s 1ms/sample - loss: 0.1817 - val_loss: 0.1819
```

Out[7]:

```
<tensorflow.python.keras.callbacks.History at 0x7f3673688580>
```

Sampling a grid of points from the 2D latent space and decoding them to images

In [11]:

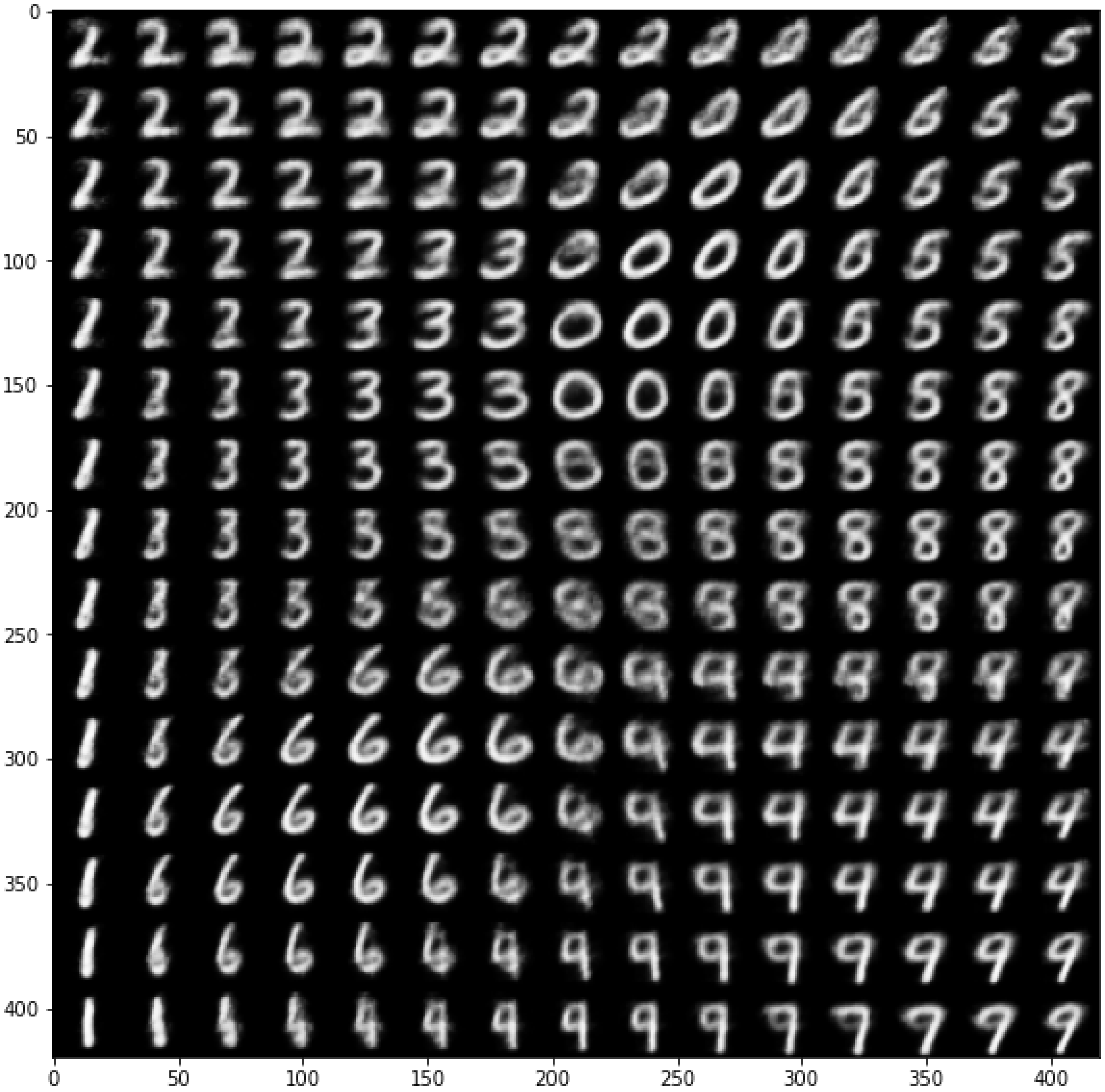
```python
import matplotlib.pyplot as plt
from scipy.stats import norm
from pathlib import Path

results_dir = Path('/home/jovyan/dsc650/dsc650/assignments/assignment12/results/vae')

n = 15
digit_size = 28
figure = np.zeros((digit_size * n, digit_size * n))
grid_x = norm.ppf(np.linspace(0.05, 0.95, n))
print("grid_x")
print(grid_x)
grid_y = norm.ppf(np.linspace(0.05, 0.95, n))
print("grid_y")
print(grid_y)

for i, yi in enumerate(grid_x):
    for j, xi in enumerate(grid_y):
        z_sample = np.array([[xi, yi]])
        z_sample = np.tile(z_sample, batch_size).reshape(batch_size, 2)
        x_decoded = decoder.predict(z_sample, batch_size=batch_size)
        digit = x_decoded[0].reshape(digit_size, digit_size)
        figure[i * digit_size: (i + 1) * digit_size,
               j * digit_size: (j + 1) * digit_size] = digit
plt.figure(figsize=(10, 10))
plt.imshow(figure, cmap='Greys_r')
img_file = results_dir.joinpath('Assignment_12_15x15_Grid.png')
plt.savefig(img_file)
plt.show()
```

```
grid_x
[-1.64485363e+00 -1.20404696e+00 -9.20822976e-01 -6.97141435e-01
 -5.03965367e-01 -3.28072108e-01 -1.61844167e-01 -1.39145821e-16
  1.61844167e-01  3.28072108e-01  5.03965367e-01  6.97141435e-01
  9.20822976e-01  1.20404696e+00  1.64485363e+00]
grid_y
[-1.64485363e+00 -1.20404696e+00 -9.20822976e-01 -6.97141435e-01
 -5.03965367e-01 -3.28072108e-01 -1.61844167e-01 -1.39145821e-16
  1.61844167e-01  3.28072108e-01  5.03965367e-01  6.97141435e-01
  9.20822976e-01  1.20404696e+00  1.64485363e+00]
```



```
In [ ]:
```