

# Assignment12

June 1, 2022

```
[1]: import keras
      from keras import layers
      from keras.models import Model
      import numpy as np
      from tensorflow.keras import backend as K
      import tensorflow as tf
      tf.compat.v1.disable_eager_execution()
```

```
[2]: 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: calling BaseResourceVariable.\_\_init\_\_ (from tensorflow.python.ops.resource\_variable\_ops) with constraint is deprecated and will be removed in a future version.  
Instructions for updating:  
If using Keras pass \*\_constraint arguments to layers.

```
[3]: 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])
```

```
[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)
```

```
[5]: class CustomVariationLayer(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 = CustomVariationLayer()([input_img,z_decoded])
```

```
[6]: from keras.datasets import mnist
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt
from scipy.stats import norm

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,))
#y = layers.Conv2D(5, kernel_size=3)(x)
```

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

WARNING:tensorflow:Output custom\_variation\_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\_variation\_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_variation_layer (CustomV (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 [=====] - 90s 1ms/sample - loss: 4297.4216 -
- val_loss: 0.1998
Epoch 2/10
60000/60000 [=====] - 87s 1ms/sample - loss: 0.1949 -
val_loss: 0.1910
Epoch 3/10
60000/60000 [=====] - 87s 1ms/sample - loss: 0.1893 -
val_loss: 0.1884
Epoch 4/10
60000/60000 [=====] - 87s 1ms/sample - loss: 0.1862 -
val_loss: 0.1871
Epoch 5/10
60000/60000 [=====] - 87s 1ms/sample - loss: 0.1842 -
val_loss: 0.1845
Epoch 6/10
60000/60000 [=====] - 87s 1ms/sample - loss: 0.1830 -
val_loss: 0.1825
Epoch 7/10
60000/60000 [=====] - 87s 1ms/sample - loss: 0.1820 -
val_loss: 0.1822
Epoch 8/10
60000/60000 [=====] - 87s 1ms/sample - loss: 0.1811 -
val_loss: 0.1835
Epoch 9/10
60000/60000 [=====] - 87s 1ms/sample - loss: 0.1804 -
val_loss: 0.1817
Epoch 10/10
60000/60000 [=====] - 87s 1ms/sample - loss: 0.1799 -
val_loss: 0.1811

```

[6]: <tensorflow.python.keras.callbacks.History at 0x7f9a6c81adc0>

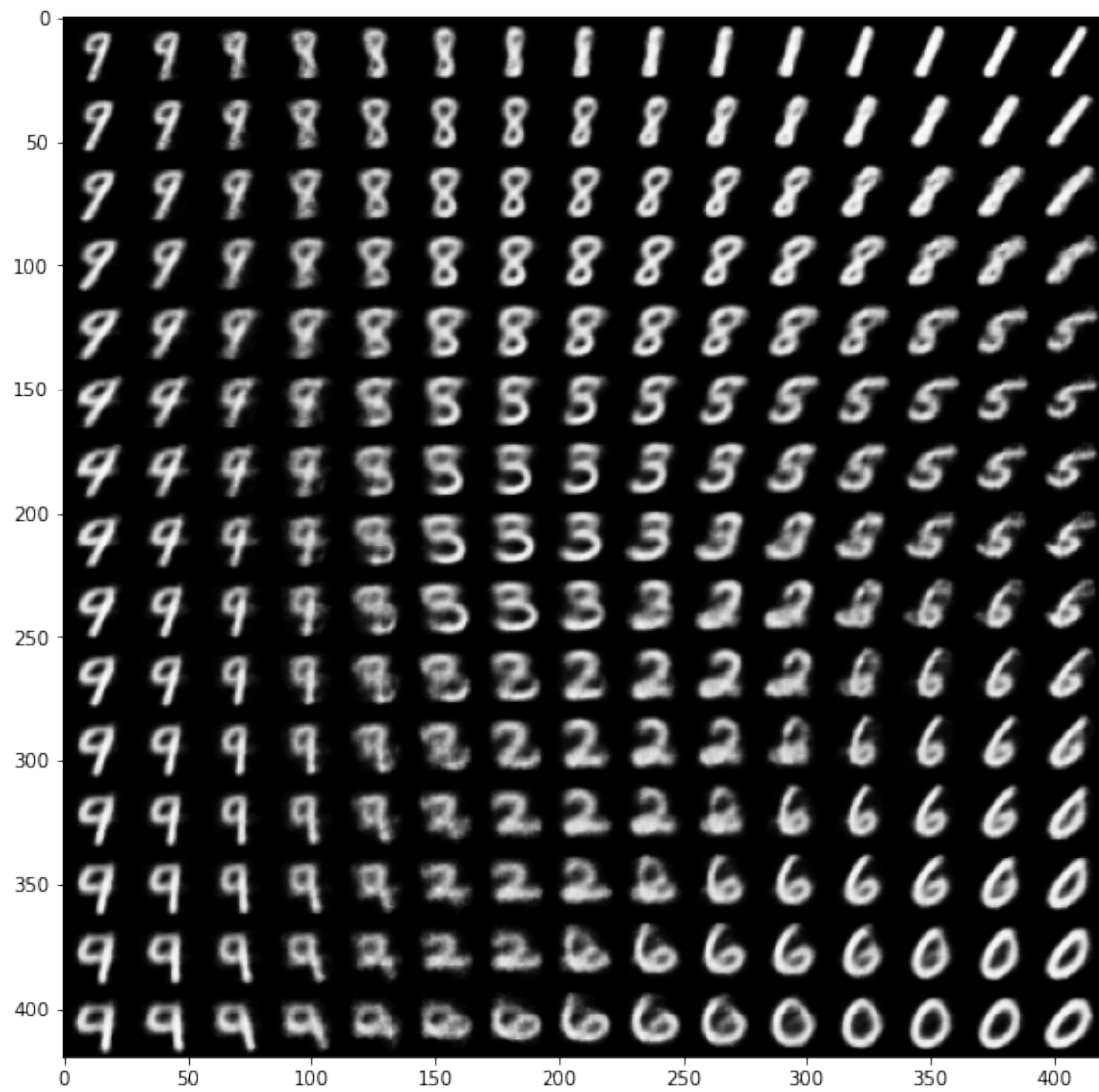
```

[9]: 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))
    grid_y = norm.ppf(np.linspace(0.05,0.95,n))

    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')
    plt.show()

```



[ ]: