Assignment 12

In [7]:

```
# Import libraries
# 8.23

from tensorflow.keras import layers
from tensorflow.keras import backend as K
from tensorflow.keras.models import Model
import tensorflow.keras as keras

import numpy as np
import pandas as pd
```

In [5]:

```
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)
```

In [6]:

```
# Listing 8.24

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])
```

In [8]:

```
# 8.25

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 [9]:

```
# 8.26

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)
        k1_loss = -5e-4 * K.mean(
                                1 + z_log_var - K.square(z_mean) - K.exp(z_log_var), axi
s = -1
                                )
        return K.mean(xent_loss + k1_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))
```

In [10]:

```
# 8.27

from tensorflow.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))
```

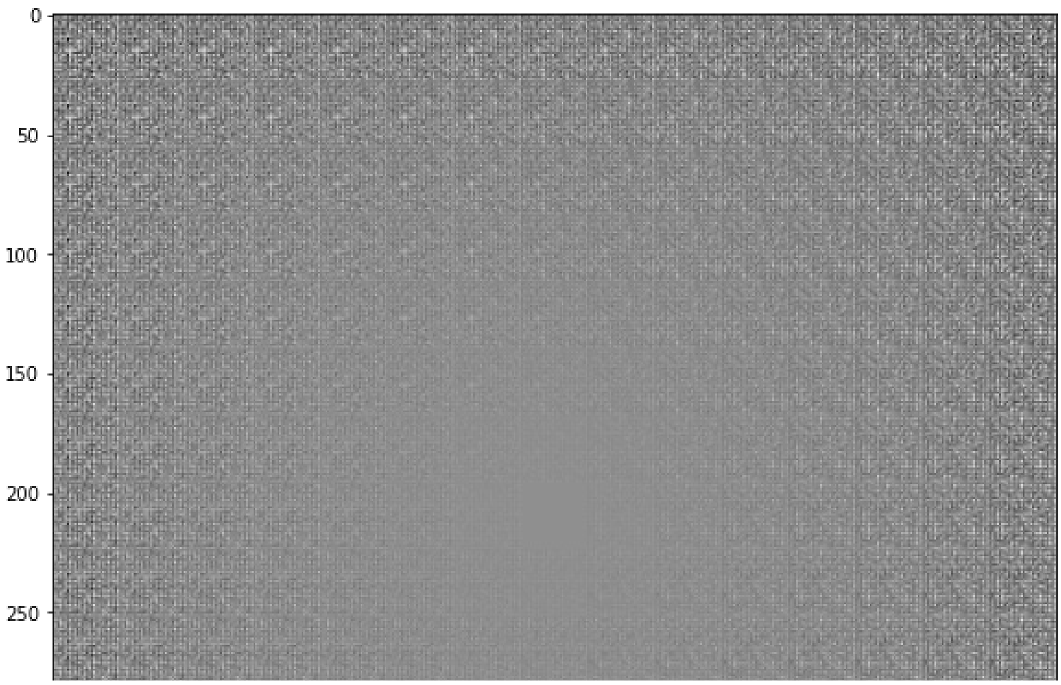
```
WARNING:tensorflow:Output custom_variational_layer missing from loss dictionary. We assum
e this was done on purpose. The fit and evaluate APIs will not be expecting any data to b
e passed to custom_variational_layer.
Model: "model_1"
_________________________________________________________________________________________
_________
Layer (type)                    Output Shape          Param #     Connected to

=========================================================================================
=========
input_2 (InputLayer)            [(None, 28, 28, 1)]  0


_________________________________________________________________________________________
_________
conv2d_2 (Conv2D)               (None, 28, 28, 32)   320         input_2[0][0]


_________________________________________________________________________________________
_________
conv2d_3 (Conv2D)               (None, 14, 14, 64)   18496       conv2d_2[0][0]


_________________________________________________________________________________________
_________
conv2d_4 (Conv2D)               (None, 14, 14, 64)   36928       conv2d_3[0][0]


_________________________________________________________________________________________
_________
conv2d_5 (Conv2D)               (None, 14, 14, 64)   36928       conv2d_4[0][0]


_________________________________________________________________________________________
```

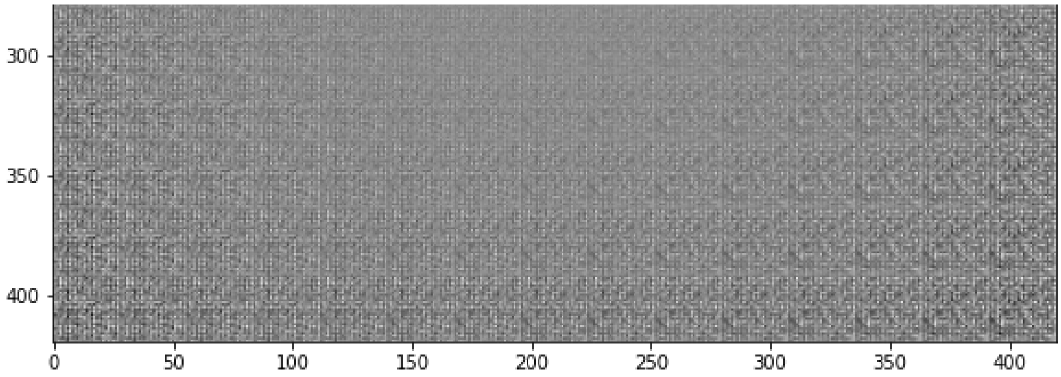
```
_________
flatten (Flatten)               (None, 12544)        0           conv2d_5[0][0]

_________________________________________________________________________________________
_________
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_2[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
   16/60000 [..............................] - ETA: 1:30:19

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\Anaconda3\lib\site-packages\tensorflow_core\python\eager\execute.py in quick_execute(op
_name, num_outputs, inputs, attrs, ctx, name)
     60                                                op_name, inputs, attrs,
---> 61                                                num_outputs)
     62   except core._NotOkStatusException as e:

TypeError: An op outside of the function building code is being passed
a "Graph" tensor. It is possible to have Graph tensors
leak out of the function building context by including a
tf.init_scope in your function building code.
For example, the following function will fail:
  @tf.function
  def has_init_scope():
    my_constant = tf.constant(1.)
    with tf.init_scope():
      added = my_constant * 2
The graph tensor has name: dense_2/Identity:0

During handling of the above exception, another exception occurred:

_SymbolicException                        Traceback (most recent call last)
<ipython-input-10-af2acba258c3> in <module>
     18         epochs=10,
     19         batch_size=batch_size,
---> 20         validation_data=(x_test, None))

~\Anaconda3\lib\site-packages\tensorflow_core\python\keras\engine\training.py in fit(self
, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffl
```

```
e, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, validat
ion_freq, max_queue_size, workers, use_multiprocessing, **kwargs)
    726         max_queue_size=max_queue_size,
    727         workers=workers,
--> 728         use_multiprocessing=use_multiprocessing)
    729 
    730   def evaluate(self,

~\Anaconda3\lib\site-packages\tensorflow_core\python\keras\engine\training_v2.py in fit(s
elf, model, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_da
ta, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_step
s, validation_freq, **kwargs)
    322                 mode=ModeKeys.TRAIN,
    323                 training_context=training_context,
--> 324                 total_epochs=epochs)
    325             cbks.make_logs(model, epoch_logs, training_result, ModeKeys.TRAIN)
    326 

~\Anaconda3\lib\site-packages\tensorflow_core\python\keras\engine\training_v2.py in run_o
ne_epoch(model, iterator, execution_function, dataset_size, batch_size, strategy, steps_p
er_epoch, num_samples, mode, training_context, total_epochs)
    121         step=step, mode=mode, size=current_batch_size) as batch_logs:
    122       try:
--> 123         batch_outs = execution_function(iterator)
    124       except (StopIteration, errors.OutOfRangeError):
    125         # TODO(kaftan): File bug about tf function and errors.OutOfRangeError?

~\Anaconda3\lib\site-packages\tensorflow_core\python\keras\engine\training_v2_utils.py in
execution_function(input_fn)
     84     # `numpy` translates Tensors to values in Eager mode.
     85     return nest.map_structure(_non_none_constant_value,
---> 86                               distributed_function(input_fn))
     87 
     88   return execution_function

~\Anaconda3\lib\site-packages\tensorflow_core\python\eager\def_function.py in __call__(se
lf, *args, **kwds)
    455 
    456     tracing_count = self._get_tracing_count()
--> 457     result = self._call(*args, **kwds)
    458     if tracing_count == self._get_tracing_count():
    459       self._call_counter.called_without_tracing()

~\Anaconda3\lib\site-packages\tensorflow_core\python\eager\def_function.py in _call(self,
*args, **kwds)
    518         # Lifting succeeded, so variables are initialized and we can run the
    519         # stateless function.
--> 520         return self._stateless_fn(*args, **kwds)
    521     else:
    522       canon_args, canon_kwds = \

~\Anaconda3\lib\site-packages\tensorflow_core\python\eager\function.py in __call__(self,
*args, **kwargs)
   1821     """Calls a graph function specialized to the inputs."""
   1822     graph_function, args, kwargs = self._maybe_define_function(args, kwargs)
-> 1823     return graph_function._filtered_call(args, kwargs)  # pylint: disable=protect
ed-access
   1824 
   1825   @property

~\Anaconda3\lib\site-packages\tensorflow_core\python\eager\function.py in _filtered_call(
self, args, kwargs)
   1139          if isinstance(t, (ops.Tensor,
   1140                            resource_variable_ops.BaseResourceVariable))),
-> 1141         self.captured_inputs)
   1142 
   1143   def _call_flat(self, args, captured_inputs, cancellation_manager=None):

~\Anaconda3\lib\site-packages\tensorflow_core\python\eager\function.py in _call_flat(self
, args, captured_inputs, cancellation_manager)
   1222     if executing_eagerly:
   1223       flat_outputs = forward_function.call(
-> 1224             ctx, args, cancellation_manager=cancellation_manager)
   1225     else:
   1226       gradient_name = self._delayed_rewrite_functions.register()

~\Anaconda3\lib\site-packages\tensorflow_core\python\eager\function.py in call(self, ctx, args, cancellation_manager)
    509               inputs=args,
    510               attrs=("executor_type", executor_type, "config_proto", config),
--> 511               ctx=ctx)
    512         else:
    513           outputs = execute.execute_with_cancellation(

~\Anaconda3\lib\site-packages\tensorflow_core\python\eager\execute.py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     73       raise core._SymbolicException(
     74           "Inputs to eager execution function cannot be Keras symbolic "
---> 75           "tensors, but found {}".format(keras_symbolic_tensors))
     76     raise e
     77   # pylint: enable=protected-access

_SymbolicException: Inputs to eager execution function cannot be Keras symbolic tensors, but found [<tf.Tensor 'dense_2/Identity:0' shape=(None, 2) dtype=float32>, <tf.Tensor 'dense_1/Identity:0' shape=(None, 2) dtype=float32>]
```

In [11]:

```
# 8.28

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

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()
```





In [ ]: