In [1]:

```
# z_mean, z_log_variance = encoder(input_img)
# z = z_mean + exp(z_log_variance) * epsilon
# reconstructed_img = decoder(z)
# model = Model(input_img, reconstructed_img
```

In [2]:

```
import keras
from keras import layers
from keras import backend as K
from keras.models import Model
import numpy as np
import tensorflow as tf
img_shape = (28, 28, 1)
batch_size = 16
latent_dim = 2
input_img = keras.Input(shape=img_shape)
```

In [3]:

```
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 [4]:

```
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 [5]:

```
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 [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])
```

In [7]:

```
class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.total_loss_tracker = keras.metrics.Mean(name="total_loss")
        self.reconstruction_loss_tracker = keras.metrics.Mean(
            name="reconstruction_loss"
        )
        self.kl_loss_tracker = keras.metrics.Mean(name="kl_loss")

    @property
    def metrics(self):
        return [
            self.total_loss_tracker,
            self.reconstruction_loss_tracker,
            self.kl_loss_tracker,
        ]

    def train_step(self, data):
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            reconstruction = self.decoder(z)
            reconstruction_loss = tf.reduce_mean(
                tf.reduce_sum(
                    keras.losses.binary_crossentropy(data, reconstruction), axis=(1, 2)
                )
            )
            kl_loss = -0.5 * (1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var))
            kl_loss = tf.reduce_mean(tf.reduce_sum(kl_loss, axis=1))
            total_loss = reconstruction_loss + kl_loss
        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
        self.total_loss_tracker.update_state(total_loss)
        self.reconstruction_loss_tracker.update_state(reconstruction_loss)
        self.kl_loss_tracker.update_state(kl_loss)
        return {
            "loss": self.total_loss_tracker.result(),
            "reconstruction_loss": self.reconstruction_loss_tracker.result(),
            "kl_loss": self.kl_loss_tracker.result(),
        }
```

In [8]:

```
class Sampling(layers.Layer):
    """Uses (z_mean, z_log_var) to sample z, the vector encoding a digit."""

    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.keras.backend.random_normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon
```

In [9]:

```
latent_dim = 2

encoder_inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(32, 3, activation="relu", strides=2, padding="same")(encoder_inputs)
x = layers.Conv2D(64, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Flatten()(x)
x = layers.Dense(16, activation="relu")(x)
z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)
z = Sampling()([z_mean, z_log_var])
encoder = keras.Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")
encoder.summary()
```

```
Model: "encoder"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_3 (InputLayer)           [(None, 28, 28, 1)]  0           []

 conv2d_5 (Conv2D)              (None, 14, 14, 32)   320         ['input_3[0][0]']

 conv2d_6 (Conv2D)              (None, 7, 7, 64)     18496       ['conv2d_5[0][0]']

 flatten_1 (Flatten)            (None, 3136)         0           ['conv2d_6[0][0]']

 dense_4 (Dense)                (None, 16)           50192       ['flatten_1[0][0]']

 z_mean (Dense)                 (None, 2)            34          ['dense_4[0][0]']

 z_log_var (Dense)              (None, 2)            34          ['dense_4[0][0]']

 sampling (Sampling)            (None, 2)            0           ['z_mean[0][0]',
                                                                  'z_log_var[0][0]']

==================================================================================================
Total params: 69,076
Trainable params: 69,076
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [10]:

```
latent_inputs = keras.Input(shape=(latent_dim,))
x = layers.Dense(7 * 7 * 64, activation="relu")(latent_inputs)
x = layers.Reshape((7, 7, 64))(x)
x = layers.Conv2DTranspose(64, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Conv2DTranspose(32, 3, activation="relu", strides=2, padding="same")(x)
decoder_outputs = layers.Conv2DTranspose(1, 3, activation="sigmoid", padding="same")(x)
decoder = keras.Model(latent_inputs, decoder_outputs, name="decoder")
decoder.summary()
```

```
Model: "decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_4 (InputLayer)        [(None, 2)]               0         
                                                                 
 dense_5 (Dense)             (None, 3136)              9408      
                                                                 
 reshape_1 (Reshape)         (None, 7, 7, 64)          0         
                                                                 
 conv2d_transpose_1 (Conv2DT  (None, 14, 14, 64)       36928     
 ranspose)                                                       
                                                                 
 conv2d_transpose_2 (Conv2DT  (None, 28, 28, 32)       18464     
 ranspose)                                                       
                                                                 
 conv2d_transpose_3 (Conv2DT  (None, 28, 28, 1)        289       
 ranspose)                                                       
                                                                 
=================================================================
Total params: 65,089
Trainable params: 65,089
Non-trainable params: 0
_________________________________________________________________
```

In [11]:
```
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()
mnist_digits = np.concatenate([x_train, x_test], axis=0)
mnist_digits = np.expand_dims(mnist_digits, -1).astype("float32") / 255

vae = VAE(encoder, decoder)
vae.compile(optimizer=keras.optimizers.Adam())
vae.fit(mnist_digits, epochs=30, batch_size=128)
```

```
Epoch 25/30
547/547 [==============================] - 5s 10ms/step - loss: 151.2329 - reconstruction_loss: 145.1587 - k
l_loss: 6.0805
Epoch 26/30
547/547 [==============================] - 5s 10ms/step - loss: 150.9731 - reconstruction_loss: 144.9113 - k
l_loss: 6.0912
Epoch 27/30
547/547 [==============================] - 5s 10ms/step - loss: 150.6014 - reconstruction_loss: 144.7715 - k
l_loss: 6.1031

Epoch 28/30
547/547 [==============================] - 5s 10ms/step - loss: 150.9017 - reconstruction_loss: 144.5966 - k
l_loss: 6.1232
Epoch 29/30
547/547 [==============================] - 5s 10ms/step - loss: 150.1648 - reconstruction_loss: 144.4724 - k
l_loss: 6.1470
Epoch 30/30
547/547 [==============================] - 6s 10ms/step - loss: 150.5159 - reconstruction_loss: 144.3404 - k
l_loss: 6.1667
```

Out[11]:
```
<keras.callbacks.History at 0x2b4c0ffab50>
```

In [12]:

```
# 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))
```

In [13]:

```python
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()
```

```
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 11ms/step
```

In [ ]: