In [50]:

```
# This Python 3 environment comes with many helpful analytics libraries installe
d
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/doc
ker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list
 all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets
preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved o
utside of the current session
```

In [51]:

```
## Setup
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

In [52]:

```
def sample_z(args):
    z_mean, z_log_var = args
    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 [53]:

```
latent_dim = 2
encoder_inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(32, 3, activation="relu", strides=2, padding="same")(encoder_i
nputs)
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 = layers.Lambda(sample_z)([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 #     Con
nected to
=====================================================================
==============================
input_7 (InputLayer)            [(None, 28, 28, 1)]  0
_____________________________________________________________________
______________________________
conv2d_6 (Conv2D)               (None, 14, 14, 32)   320         inp
ut_7[0][0]
_____________________________________________________________________
______________________________
conv2d_7 (Conv2D)               (None, 7, 7, 64)     18496       con
v2d_6[0][0]
_____________________________________________________________________
______________________________
flatten_3 (Flatten)             (None, 3136)         0           con
v2d_7[0][0]
_____________________________________________________________________
______________________________
dense_6 (Dense)                 (None, 16)           50192       fla
tten_3[0][0]
_____________________________________________________________________
______________________________
z_mean (Dense)                  (None, 2)            34          den
se_6[0][0]
_____________________________________________________________________
______________________________
z_log_var (Dense)               (None, 2)            34          den
se_6[0][0]
_____________________________________________________________________
______________________________
lambda_3 (Lambda)               (None, 2)            0           z_m
ean[0][0]
                                                                 z_l
og_var[0][0]
=====================================================================
==============================
Total params: 69,076
Trainable params: 69,076
Non-trainable params: 0
_____________________________________________________________________
______________________________
```

In [54]:

```
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="sa
me")(x)
decoder = keras.Model(latent_inputs, decoder_outputs, name="decoder")
decoder.summary()
```

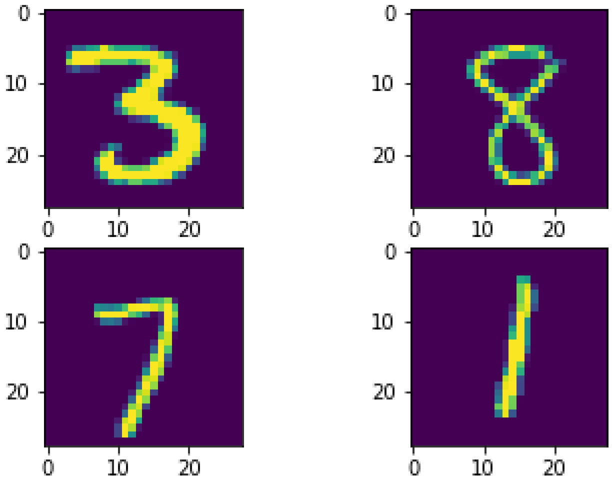
```
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_8 (InputLayer)         [(None, 2)]               0
_________________________________________________________________
dense_7 (Dense)              (None, 3136)              9408
_________________________________________________________________
reshape_3 (Reshape)          (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_transpose_9 (Conv2DTr (None, 14, 14, 64)        36928
_________________________________________________________________
conv2d_transpose_10 (Conv2DT (None, 28, 28, 32)        18464
_________________________________________________________________
conv2d_transpose_11 (Conv2DT (None, 28, 28, 1)         289
=================================================================
Total params: 65,089
Trainable params: 65,089
Non-trainable params: 0
_________________________________________________________________
```

In [55]:

```
class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder

    def train_step(self, data):
        if isinstance(data, tuple):
            data = data[0]
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            reconstruction = self.decoder(z)
            reconstruction_loss = tf.reduce_mean(
                keras.losses.binary_crossentropy(data, reconstruction)
            )
            reconstruction_loss *= 28 * 28
            kl_loss = 1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
            kl_loss = tf.reduce_mean(kl_loss)
            kl_loss *= -0.5
            total_loss = reconstruction_loss + kl_loss
        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
        return {
            "loss": total_loss,
            "reconstruction_loss": reconstruction_loss,
            "kl_loss": kl_loss,
        }
```

In [56]:

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
mnist_digits = np.concatenate([x_train, x_test], axis=0)
mnist_digits = np.expand_dims(mnist_digits, -1).astype("float32") / 255

plt.figure(1)
plt.subplot(221)
plt.imshow(mnist_digits[12][:,:,0])

plt.subplot(222)
plt.imshow(mnist_digits[144][:,:,0])

plt.subplot(223)
plt.imshow(mnist_digits[2666][:,:,0])

plt.subplot(224)
plt.imshow(mnist_digits[39888][:,:,0])
plt.show()
```



In [57]:

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

```
Epoch 1/30
547/547 [==============================] - 4s 7ms/step - loss: 207.3
724 - reconstruction_loss: 205.2293 - kl_loss: 2.1431
Epoch 2/30
547/547 [==============================] - 4s 7ms/step - loss: 184.1
933 - reconstruction_loss: 181.7391 - kl_loss: 2.4542
Epoch 3/30
547/547 [==============================] - 4s 7ms/step - loss: 166.4
575 - reconstruction_loss: 162.6039 - kl_loss: 3.8537
Epoch 4/30
547/547 [==============================] - 4s 7ms/step - loss: 161.1
563 - reconstruction_loss: 157.1913 - kl_loss: 3.9651
Epoch 5/30
547/547 [==============================] - 4s 7ms/step - loss: 158.1
563 - reconstruction_loss: 154.1365 - kl_loss: 4.0197
Epoch 6/30
547/547 [==============================] - 4s 7ms/step - loss: 156.2
655 - reconstruction_loss: 152.2232 - kl_loss: 4.0423
Epoch 7/30
547/547 [==============================] - 4s 7ms/step - loss: 154.7
405 - reconstruction_loss: 150.7202 - kl_loss: 4.0203
Epoch 8/30
547/547 [==============================] - 4s 7ms/step - loss: 153.5
280 - reconstruction_loss: 149.5612 - kl_loss: 3.9668
Epoch 9/30
547/547 [==============================] - 4s 7ms/step - loss: 152.4
241 - reconstruction_loss: 148.5057 - kl_loss: 3.9183
Epoch 10/30
547/547 [==============================] - 4s 7ms/step - loss: 151.7
395 - reconstruction_loss: 147.8407 - kl_loss: 3.8989
Epoch 11/30
547/547 [==============================] - 4s 7ms/step - loss: 151.0
939 - reconstruction_loss: 147.2184 - kl_loss: 3.8755
Epoch 12/30
547/547 [==============================] - 4s 7ms/step - loss: 150.5
595 - reconstruction_loss: 146.6921 - kl_loss: 3.8674
Epoch 13/30
547/547 [==============================] - 4s 7ms/step - loss: 150.1
192 - reconstruction_loss: 146.2761 - kl_loss: 3.8431
Epoch 14/30
547/547 [==============================] - 4s 7ms/step - loss: 149.6
743 - reconstruction_loss: 145.8305 - kl_loss: 3.8438
Epoch 15/30
547/547 [==============================] - 4s 7ms/step - loss: 149.3
014 - reconstruction_loss: 145.4608 - kl_loss: 3.8406
Epoch 16/30
547/547 [==============================] - 4s 7ms/step - loss: 148.9
757 - reconstruction_loss: 145.1367 - kl_loss: 3.8389
Epoch 17/30
547/547 [==============================] - 4s 7ms/step - loss: 148.7
016 - reconstruction_loss: 144.8715 - kl_loss: 3.8301
Epoch 18/30
547/547 [==============================] - 4s 7ms/step - loss: 148.4
469 - reconstruction_loss: 144.6082 - kl_loss: 3.8387
Epoch 19/30
547/547 [==============================] - 4s 7ms/step - loss: 148.0
821 - reconstruction_loss: 144.2517 - kl_loss: 3.8304
Epoch 20/30
547/547 [==============================] - 4s 7ms/step - loss: 147.9
485 - reconstruction_loss: 144.1212 - kl_loss: 3.8273
Epoch 21/30
```

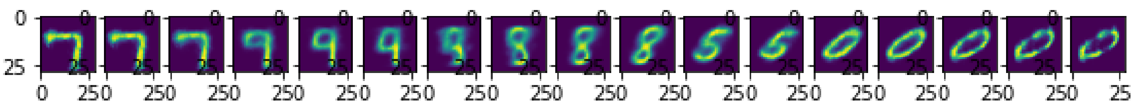
```
547/547 [==============================] - 4s 7ms/step - loss: 147.7
520 - reconstruction_loss: 143.9373 - kl_loss: 3.8147
Epoch 22/30
547/547 [==============================] - 4s 7ms/step - loss: 147.4
946 - reconstruction_loss: 143.6642 - kl_loss: 3.8304
Epoch 23/30
547/547 [==============================] - 4s 7ms/step - loss: 147.3
702 - reconstruction_loss: 143.5452 - kl_loss: 3.8249
Epoch 24/30
547/547 [==============================] - 4s 7ms/step - loss: 147.1
650 - reconstruction_loss: 143.3441 - kl_loss: 3.8209
Epoch 25/30
547/547 [==============================] - 4s 7ms/step - loss: 147.1
558 - reconstruction_loss: 143.3291 - kl_loss: 3.8267
Epoch 26/30
547/547 [==============================] - 4s 7ms/step - loss: 146.8
425 - reconstruction_loss: 142.9979 - kl_loss: 3.8446
Epoch 27/30
547/547 [==============================] - 4s 7ms/step - loss: 146.6
580 - reconstruction_loss: 142.8291 - kl_loss: 3.8289
Epoch 28/30
547/547 [==============================] - 4s 7ms/step - loss: 146.6
052 - reconstruction_loss: 142.7784 - kl_loss: 3.8269
Epoch 29/30
547/547 [==============================] - 4s 7ms/step - loss: 146.4
929 - reconstruction_loss: 142.6714 - kl_loss: 3.8215
Epoch 30/30
547/547 [==============================] - 4s 7ms/step - loss: 146.2
909 - reconstruction_loss: 142.4649 - kl_loss: 3.8259
```

Out[57]:

```
<tensorflow.python.keras.callbacks.History at 0x7f912d0b75d0>
```

In [95]:

```python
# Visualize images
#Single decoded image with random input latent vector (of size 1x2)
#Latent space range is about -5 to 5 so pick random values within this range
#Try starting with -1, 1 and slowly go up to -1.5,1.5 and see how it morphs from
#one image to the other.
plt.figure(figsize=(10, 8))
c = 1
r = [-4,-3.5,-3,-2.5,-2,-1.5,-1,-0.5, 0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4]
for i in r:
    plt.subplot(1, len(r), c)
    sample_vector = np.array([[i,-0.6]]) #mu_1, mu_2
    decoded_example = decoder.predict(sample_vector)
    decoded_example_reshaped = decoded_example.reshape(img_width, img_height)
    plt.imshow(decoded_example_reshaped)
    c = c + 1
plt.show()
```

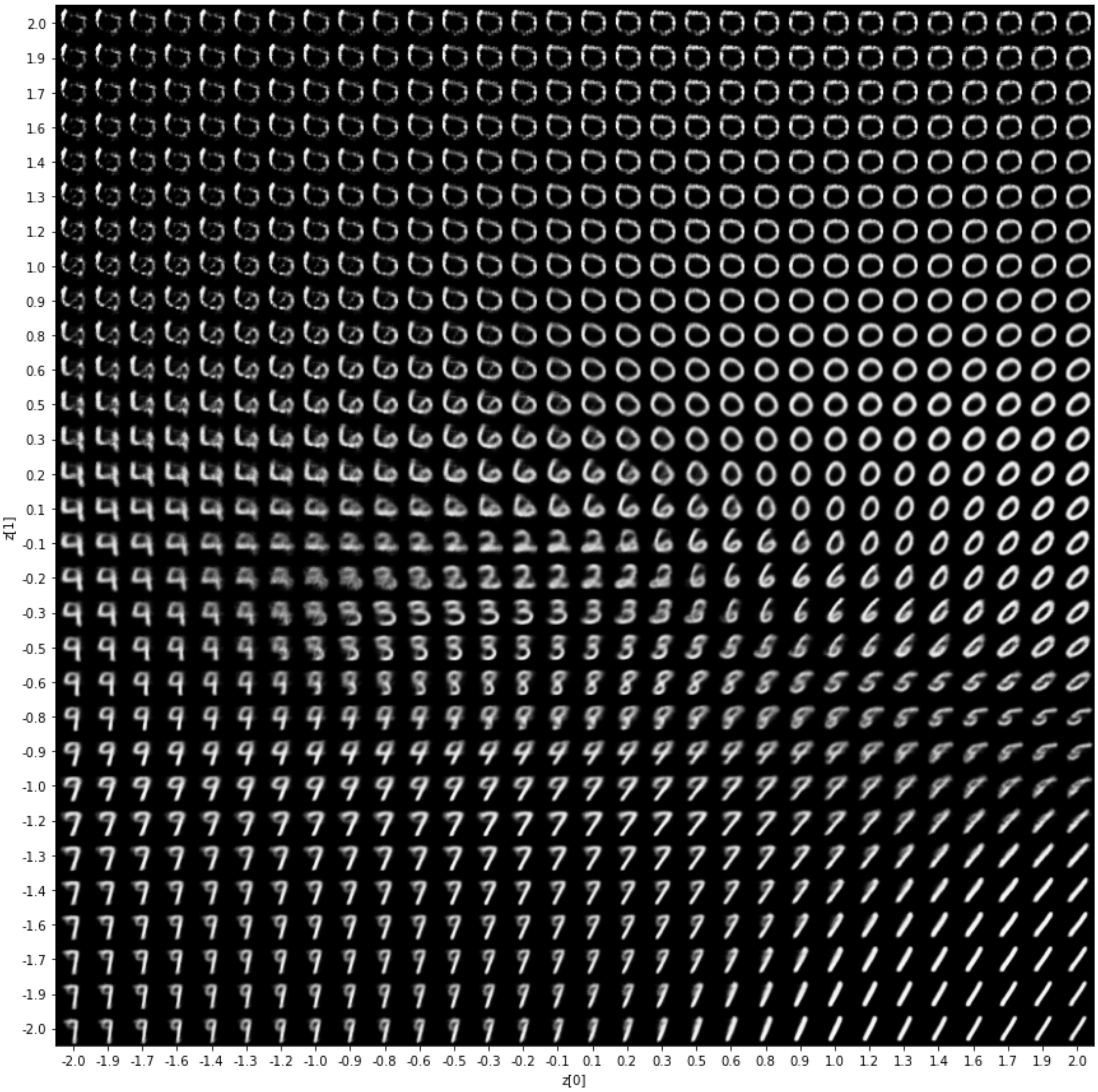


In [58]:

```
"""
## Display a grid of sampled digits
"""
import matplotlib.pyplot as plt
def plot_latent(encoder, decoder):
    # display a n*n 2D manifold of digits
    n = 30
    digit_size = 28
    scale = 2.0
    figsize = 15
    figure = np.zeros((digit_size * n, digit_size * n))
    # linearly spaced coordinates corresponding to the 2D plot
    # of digit classes in the latent space
    grid_x = np.linspace(-scale, scale, n)
    grid_y = np.linspace(-scale, scale, n)[::-1]

    for i, yi in enumerate(grid_y):
        for j, xi in enumerate(grid_x):
            z_sample = np.array([[xi, yi]])
            x_decoded = decoder.predict(z_sample)
            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=(figsize, figsize))
    start_range = digit_size // 2
    end_range = n * digit_size + start_range
    pixel_range = np.arange(start_range, end_range, digit_size)
    sample_range_x = np.round(grid_x, 1)
    sample_range_y = np.round(grid_y, 1)
    plt.xticks(pixel_range, sample_range_x)
    plt.yticks(pixel_range, sample_range_y)
    plt.xlabel("z[0]")
    plt.ylabel("z[1]")
    plt.imshow(figure, cmap="Greys_r")
    plt.show()
```

In [59]:

```
plot_latent(encoder, decoder)
```



In [60]:

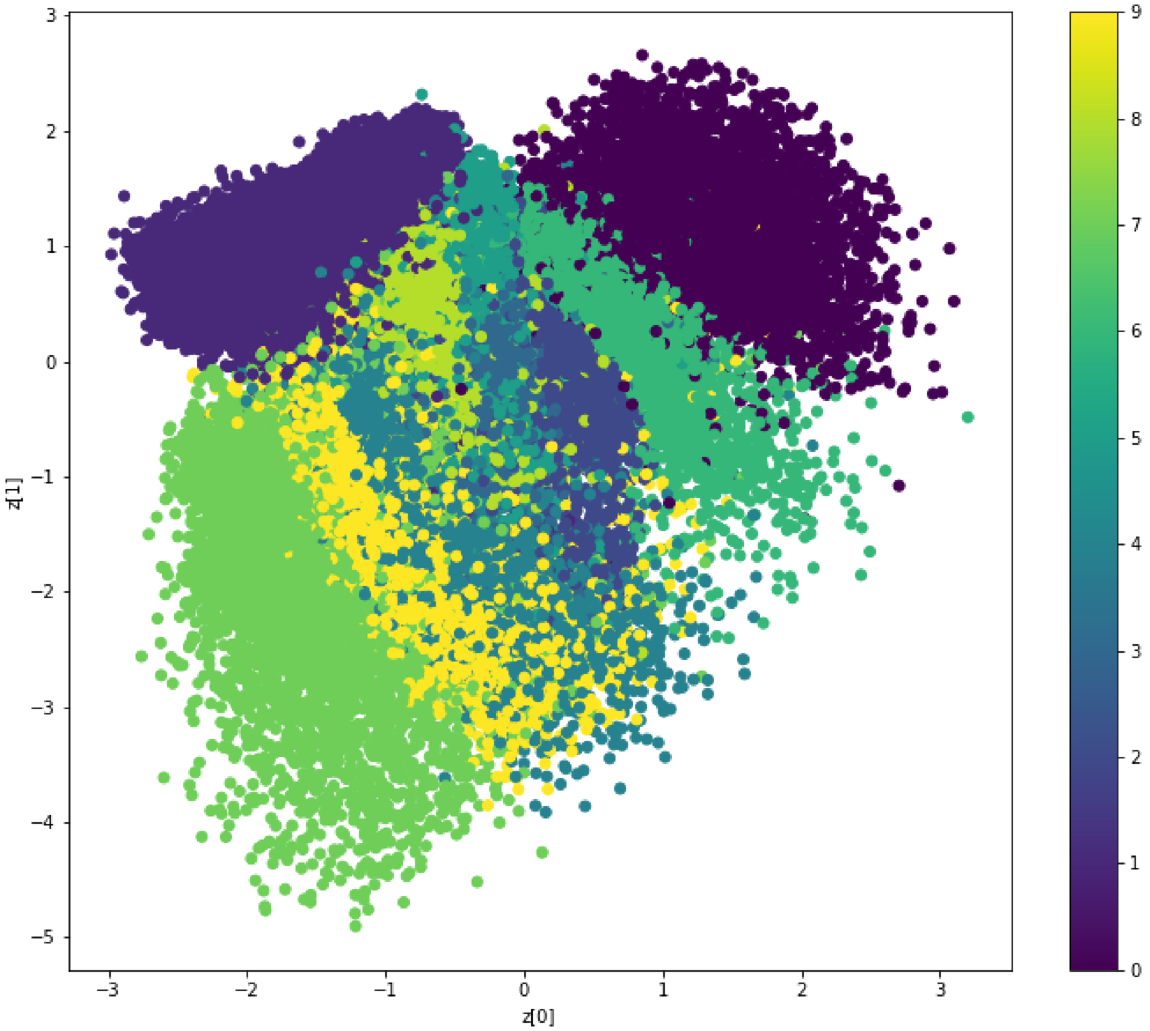
```
"""
## Display how the latent space clusters different digit classes
"""
def plot_label_clusters(encoder, decoder, data, labels):
    # display a 2D plot of the digit classes in the latent space
    z_mean, _, _ = encoder.predict(data)
    plt.figure(figsize=(12, 10))
    plt.scatter(z_mean[:, 0], z_mean[:, 1], c=labels)
    plt.colorbar()
    plt.xlabel("z[0]")
    plt.ylabel("z[1]")
    plt.show()
```

In [20]:

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train = np.expand_dims(x_train, -1).astype("float32") / 255
```

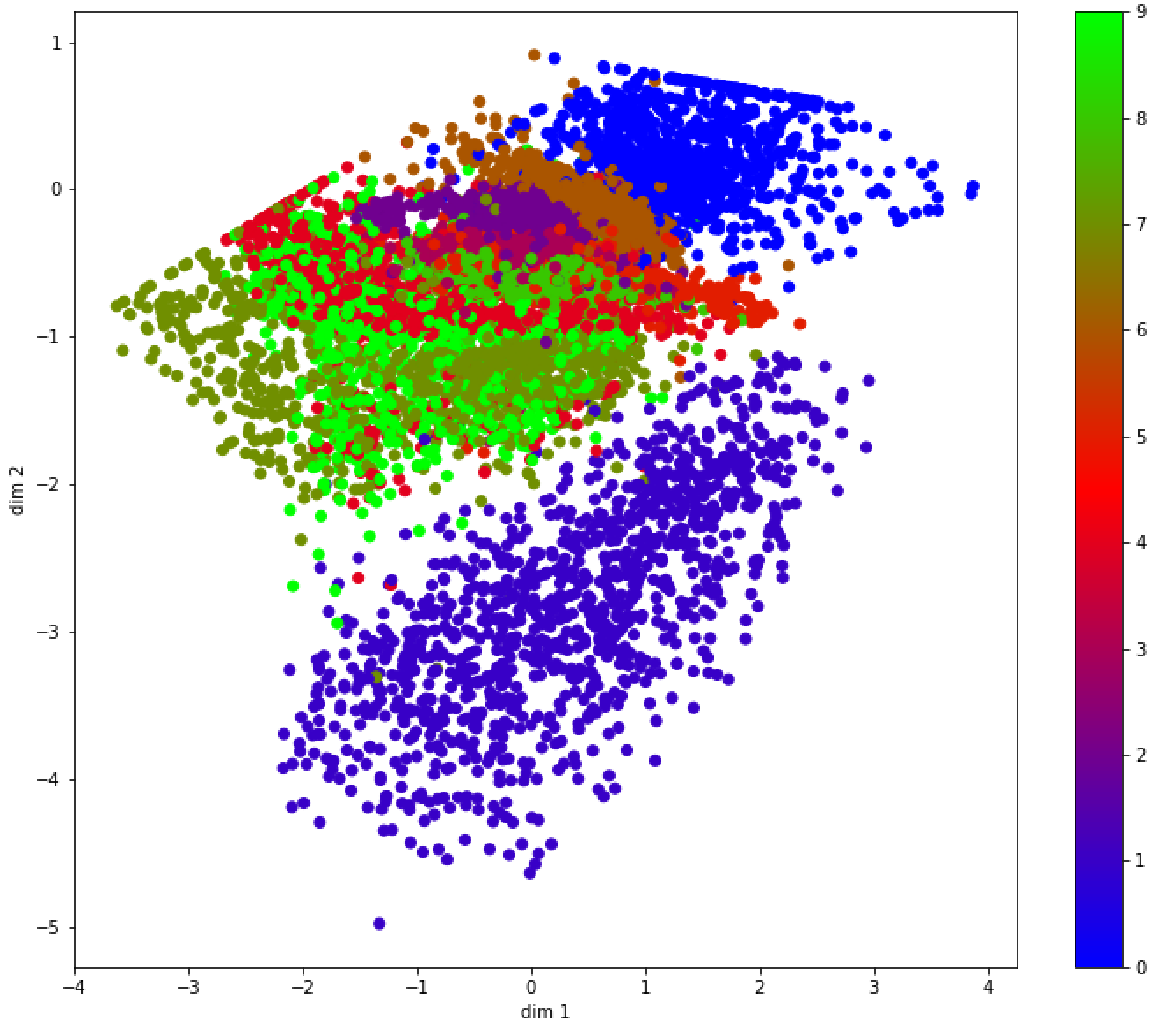
In [21]:

```
plot_label_clusters(encoder, decoder, x_train, y_train)
```



In [96]:

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_test = np.expand_dims(x_test, -1).astype("float32") / 255
mu, sig, _ = encoder.predict(x_test)
#Plot dim1 and dim2 for mu
plt.figure(figsize=(12, 10))
plt.scatter(mu[:, 0], mu[:, 1], c=y_test, cmap='brg')
plt.xlabel('dim 1')
plt.ylabel('dim 2')
plt.colorbar()
plt.show()
```

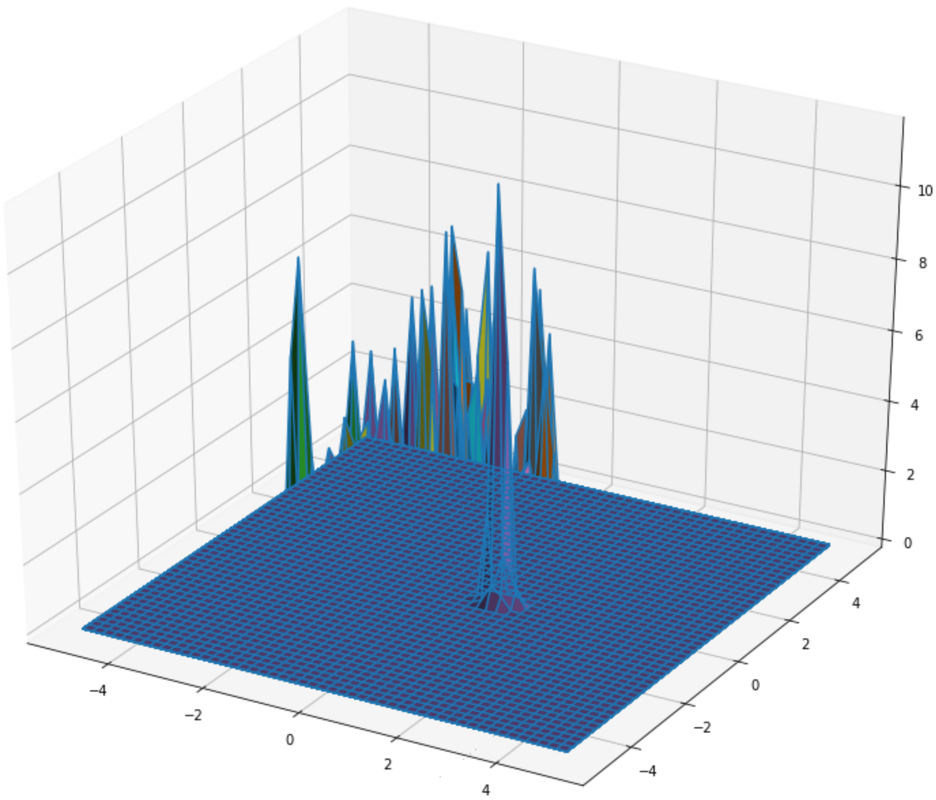


In [97]:

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train = np.expand_dims(x_train, -1).astype("float32") / 255
pred = encoder.predict(x_train)
```

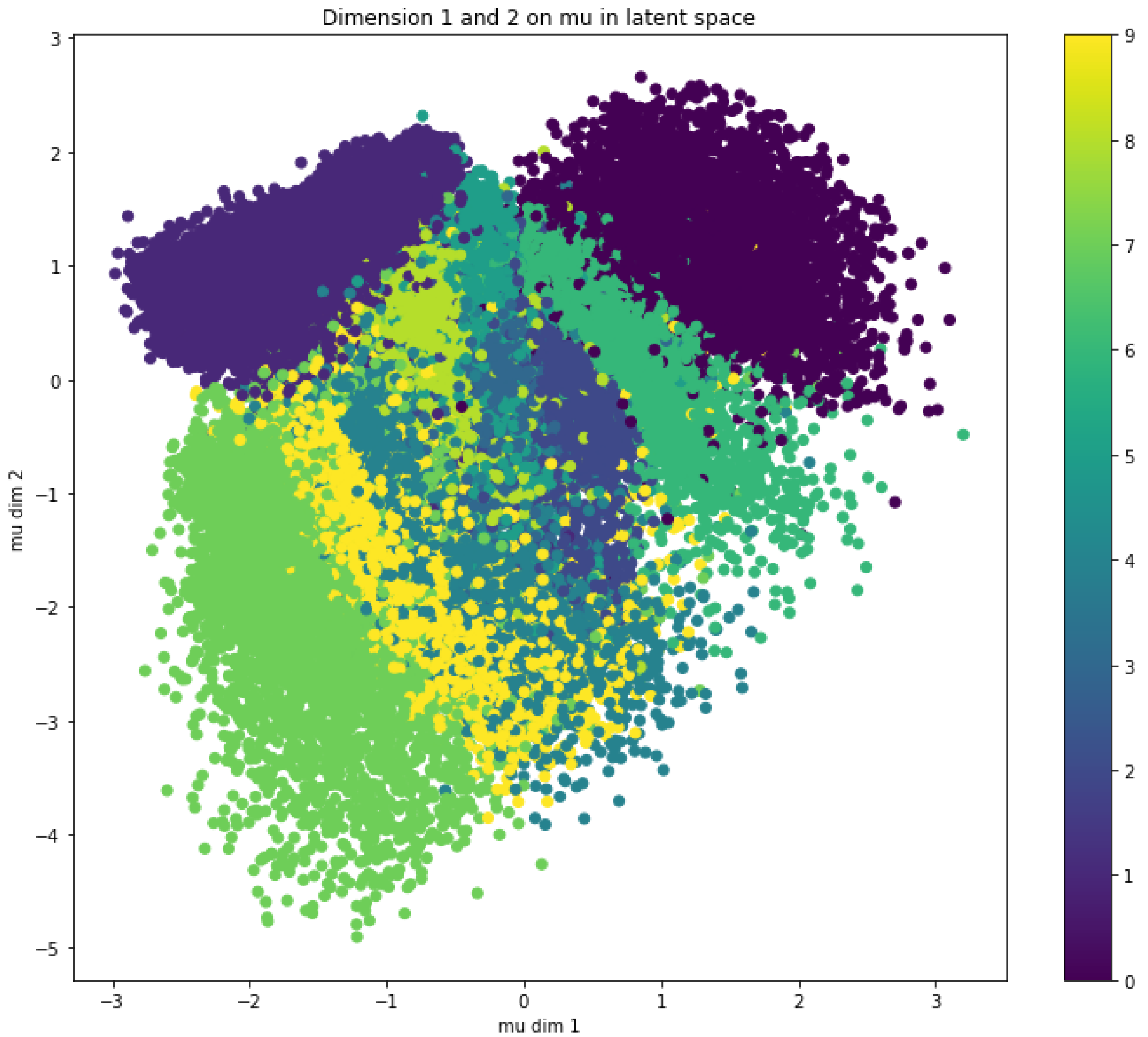
In [98]:

```
import numpy as np
from scipy.stats import multivariate_normal

x, y = np.mgrid[-5.0:5.0:50j, -5.0:5.0:50j]

# Need an (N, 2) array of (x, y) pairs.
xy = np.column_stack([x.flat, y.flat])

fig = plt.figure(figsize=(15, 12))
ax = fig.add_subplot(111, projection='3d')

for i in range(60):
    mu = np.array(pred[0][i])
    sigma = np.array(pred[1][i])
    covariance = np.diag(np.sqrt(np.exp(sigma)))
    z = multivariate_normal.pdf(xy, mean=mu, cov=covariance)
    z = z.reshape(x.shape)
    ax.plot_surface(x,y,z)
    ax.plot_wireframe(x,y,z)
plt.show()
```



In [29]:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st
n_components = 3
fig = plt.figure(figsize=(12, 10))
plt.scatter(mu_1, mu_2, c = y_train)
plt.title(f"Dimension 1 and 2 on mu in latent space")
plt.colorbar()
plt.xlabel("mu dim 1")
plt.ylabel("mu dim 2");
```



In [ ]:

```
```