Bill Yerkes

CS5542 Big Data Apps and Analytics

In Class Programming –8
15th October 2020

Submit ICP Feedback in Class. : Lnik to Feed back Form

Variational Autoencoders:

Use the same data and source code but add two more layers to encoder path and their corresponding two layers to decoder path, run the new model and report your findings. In your report specify which 4 layers (2 layers in encoder path and 2 layers in decoder path) have you added and explain why you added those (their function).

Examples of layers that can be added Conv2D, Batchnorm, Conv2DTranspose etc.

ICP Requirements:

1) Successfully executing the code with new architecture for encoder and decoder path (75 points)
2) Explanation of new layers (5 points)
3) overall code quality (10 points)
4) Pdf Report quality, video explanation (10 points)

Submission Guidelines:

Same as previous ICPs.

ICP Report:

What I learned in the ICP:

I learned the basics of Variational Autoencoders. I learned a bit more about the Keras Layers. I got a better understanding of max pooling and average pooling.

Description of what task I was performing:

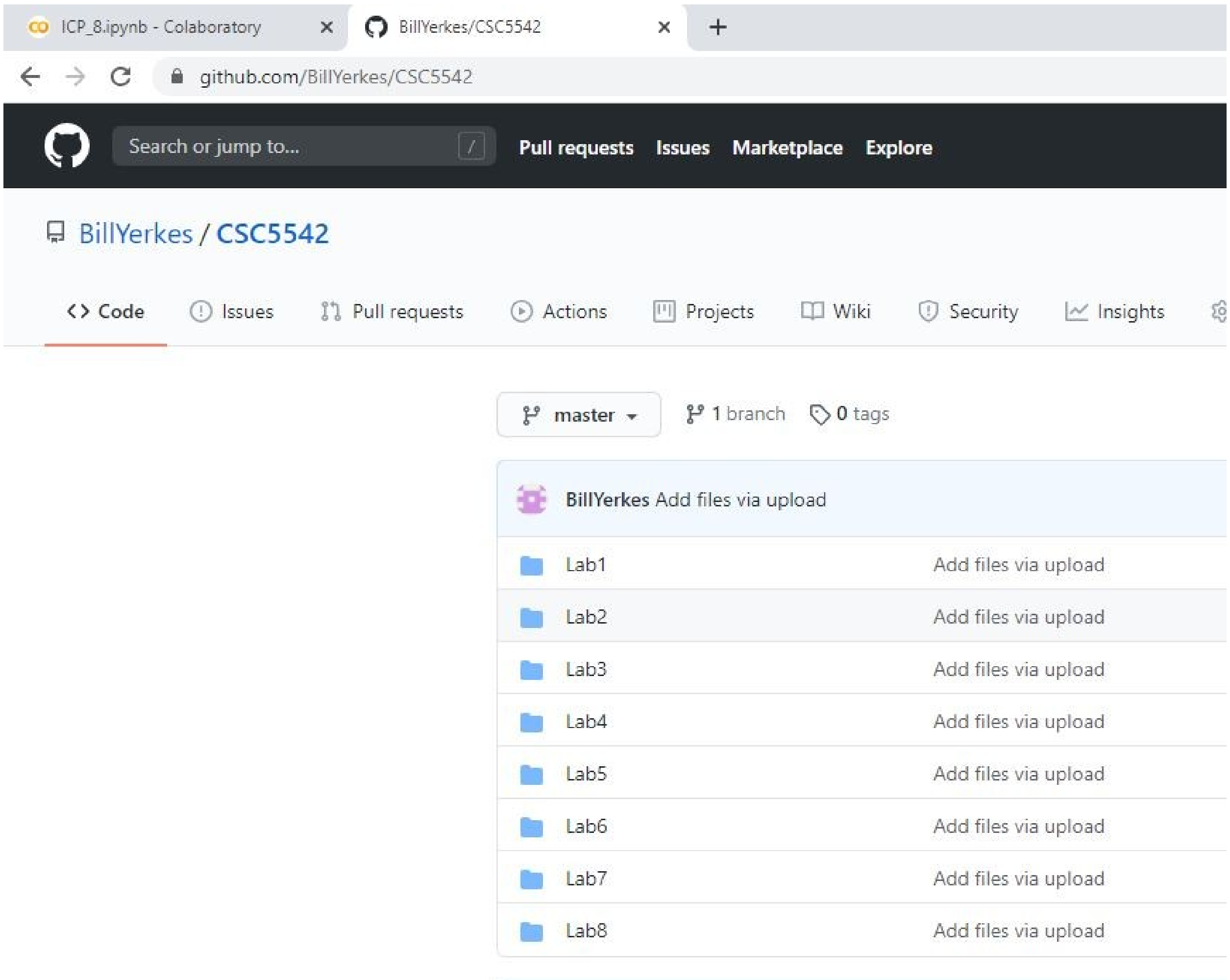
I added two layers each to the encoder and decoder section of the Autoencoder model.

Challenges I faced:

I once again ran into issues with making sure that the data size was correct when going between steps. Found that reducing from 7 to 4 in the encoder posed challenges when trying to do the inverse in the decoder as it would go from 4 to 8. I figured out how to keep the size of the data the same so as to be able to add additional layers. I would like to learn if it is possible to increase the image size from 28 x 28 to 32 x 32, and what implications that would have on the model.

Screen Shots

GitHub:



Initialize and Install Libraries, establish training and test data:

ICP_8.ipynb - Colaboratory

colab.research.google.com/drive/1GwN3t5JAZmS1R_N3D6j2CAdQ-uylTKNI#scrollTo=fwy9MsWHQp7m

ICP_8.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
#import the libraries
import keras
import tensorflow as tf
#Added MaxPooling and UpSampling
from keras.layers import Conv2D, Conv2DTranspose, Input, Flatten, Dense, Lambda, Reshape, MaxPooling2D, UpSampling2D
from keras.layers import BatchNormalization
from keras.models import Model
from keras.datasets import mnist
from keras.losses import binary_crossentropy
from keras import backend as K
import numpy as np
import matplotlib.pyplot as plt
```

The MNIST dataset will be used for training the autoencoder. This dataset contains thousands of 28 x 28 pixel images of handwritten digits. As such, our autoencoder will learn the distribution of handwritten digits across (two)dimensional latent space, which we can then use to manipulate samples into a format we like.

```
[2] # Load MNIST dataset
    (input_train_1, target_train_1), (input_test_1, target_test_1) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

Check the data shape and reduce if neccessary

```
[3] print(input_train_1.shape)
    print(input_test_1.shape)
    print(target_train_1.shape)
    print(target_test_1.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
```

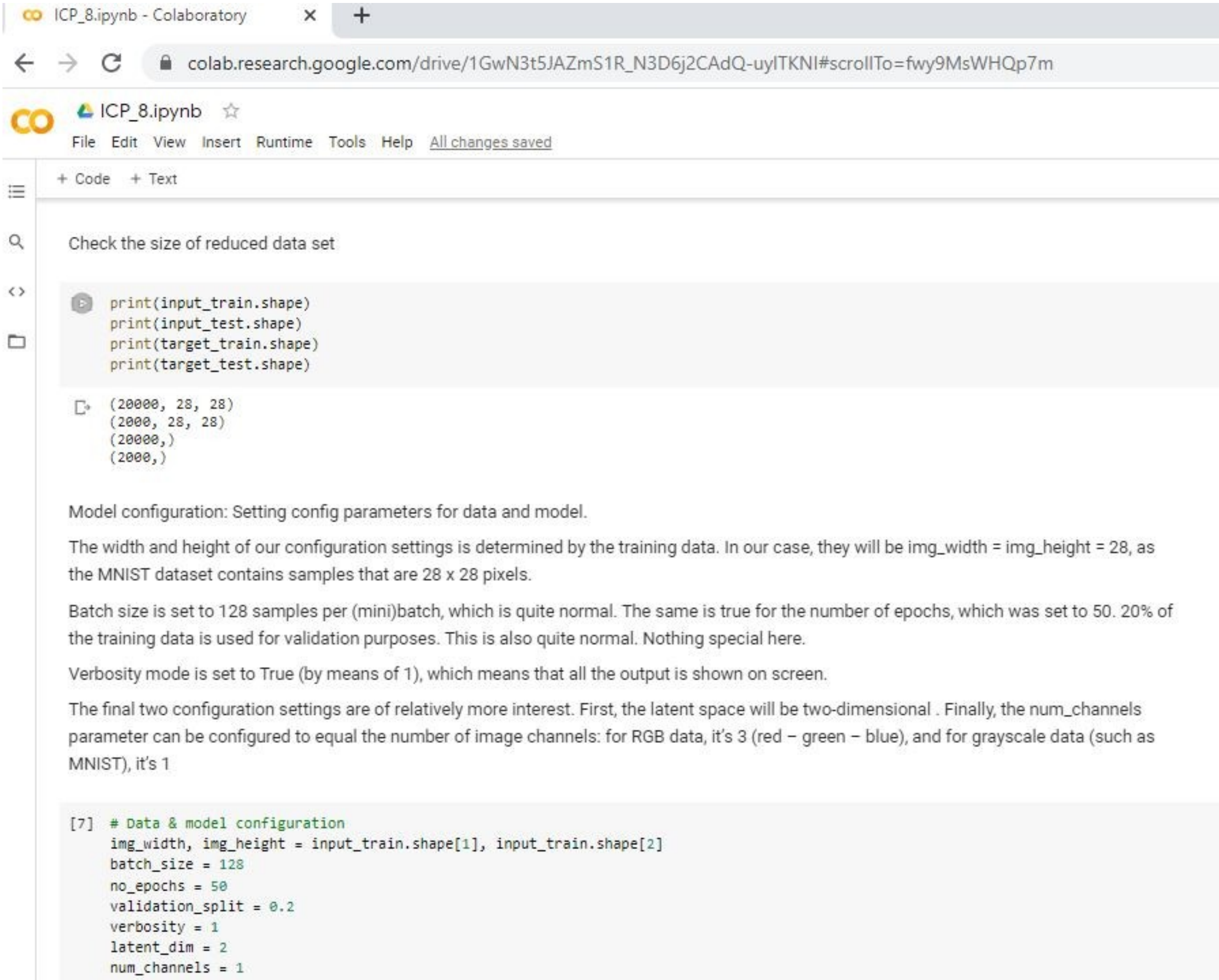
Since original data size was throwing out of memory error and required more GPU, I reduced the data size here (Training set is reduced from 60000 to 10000, and test set is reduced from 10000 to 1000 images.

```
[4] input_train=input_train_1[0:20000]
    target_train=target_train_1[0:20000]
    input_test=input_test_1[0:2000]
    target_test=target_test_1[0:2000]
```

View Images:


ICP_8.ipynb - Colaboratory
colab.research.google.com/drive/1GwN3t5JAZmS1R_N3D6j2
ICP_8.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Print the first 10 images in training set
for i in range(10):
pr_image = input_train[i]
pr_image = np.array(pr_image, dtype='float')
pixels = pr_image.reshape((28, 28))
plt.imshow(pixels)
plt.show()
0 5 10 15 20 25

Check the size of reduce data set:



Check the size of reduced data set

```
print(input_train.shape)
print(input_test.shape)
print(target_train.shape)
print(target_test.shape)
```

```
(20000, 28, 28)
(2000, 28, 28)
(20000,)
(2000,)
```

Model configuration: Setting config parameters for data and model.

The width and height of our configuration settings is determined by the training data. In our case, they will be img_width = img_height = 28, as the MNIST dataset contains samples that are 28 x 28 pixels.

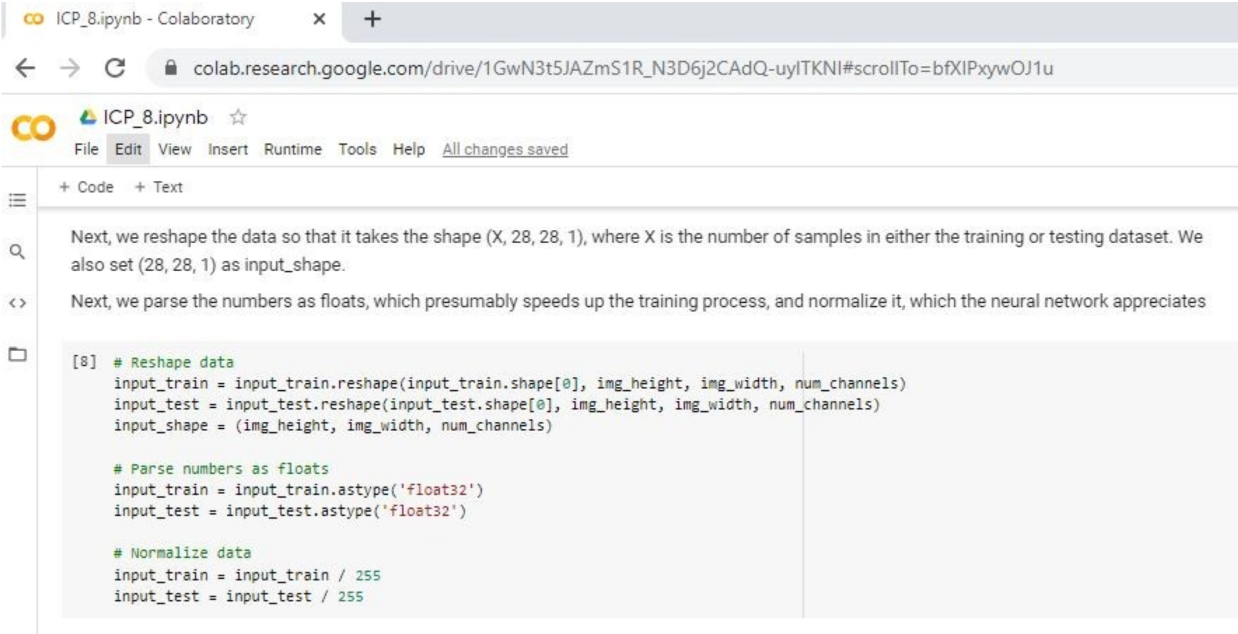
Batch size is set to 128 samples per (mini)batch, which is quite normal. The same is true for the number of epochs, which was set to 50. 20% of the training data is used for validation purposes. This is also quite normal. Nothing special here.

Verbosity mode is set to True (by means of 1), which means that all the output is shown on screen.

The final two configuration settings are of relatively more interest. First, the latent space will be two-dimensional . Finally, the num_channels parameter can be configured to equal the number of image channels: for RGB data, it's 3 (red – green – blue), and for grayscale data (such as MNIST), it's 1

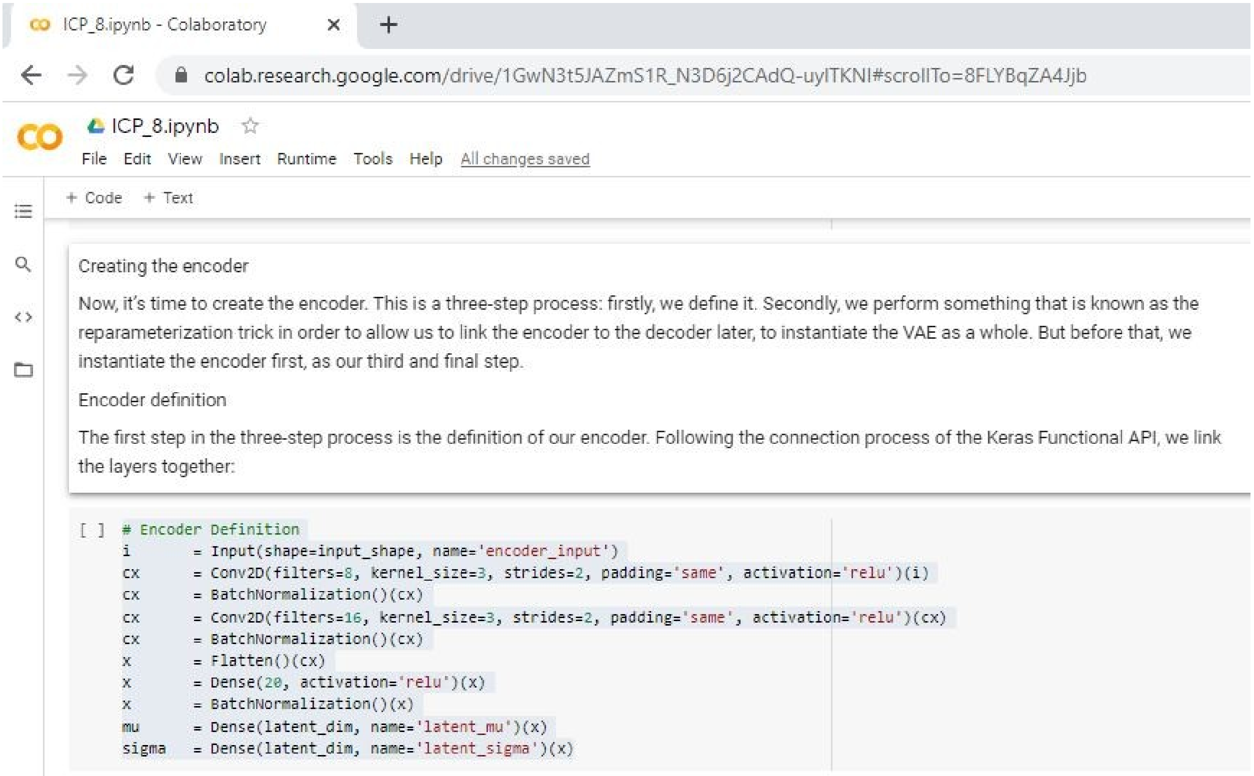
```
[7] # Data & model configuration
img_width, img_height = input_train.shape[1], input_train.shape[2]
batch_size = 128
no_epochs = 50
validation_split = 0.2
verbosity = 1
latent_dim = 2
num_channels = 1
```

Reshape and Normalize the data:



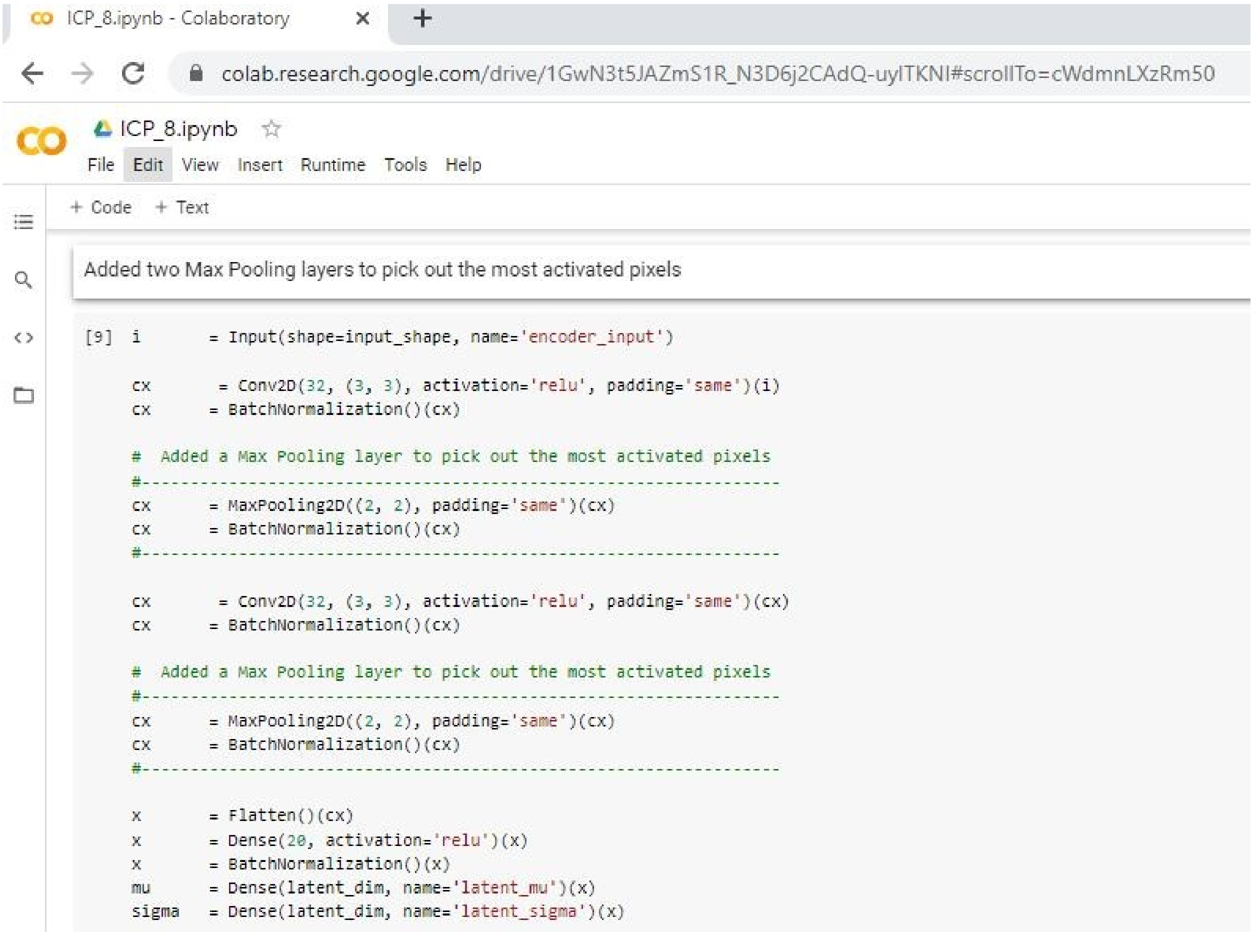

```
# Reshape data
input_train = input_train.reshape(input_train.shape[0], img_height, img_width, num_channels)
input_test = input_test.reshape(input_test.shape[0], img_height, img_width, num_channels)
input_shape = (img_height, img_width, num_channels)

# Parse numbers as floats
input_train = input_train.astype('float32')
input_test = input_test.astype('float32')

# Normalize data
input_train = input_train / 255
input_test = input_test / 255
```

Encoder Definition from Class:

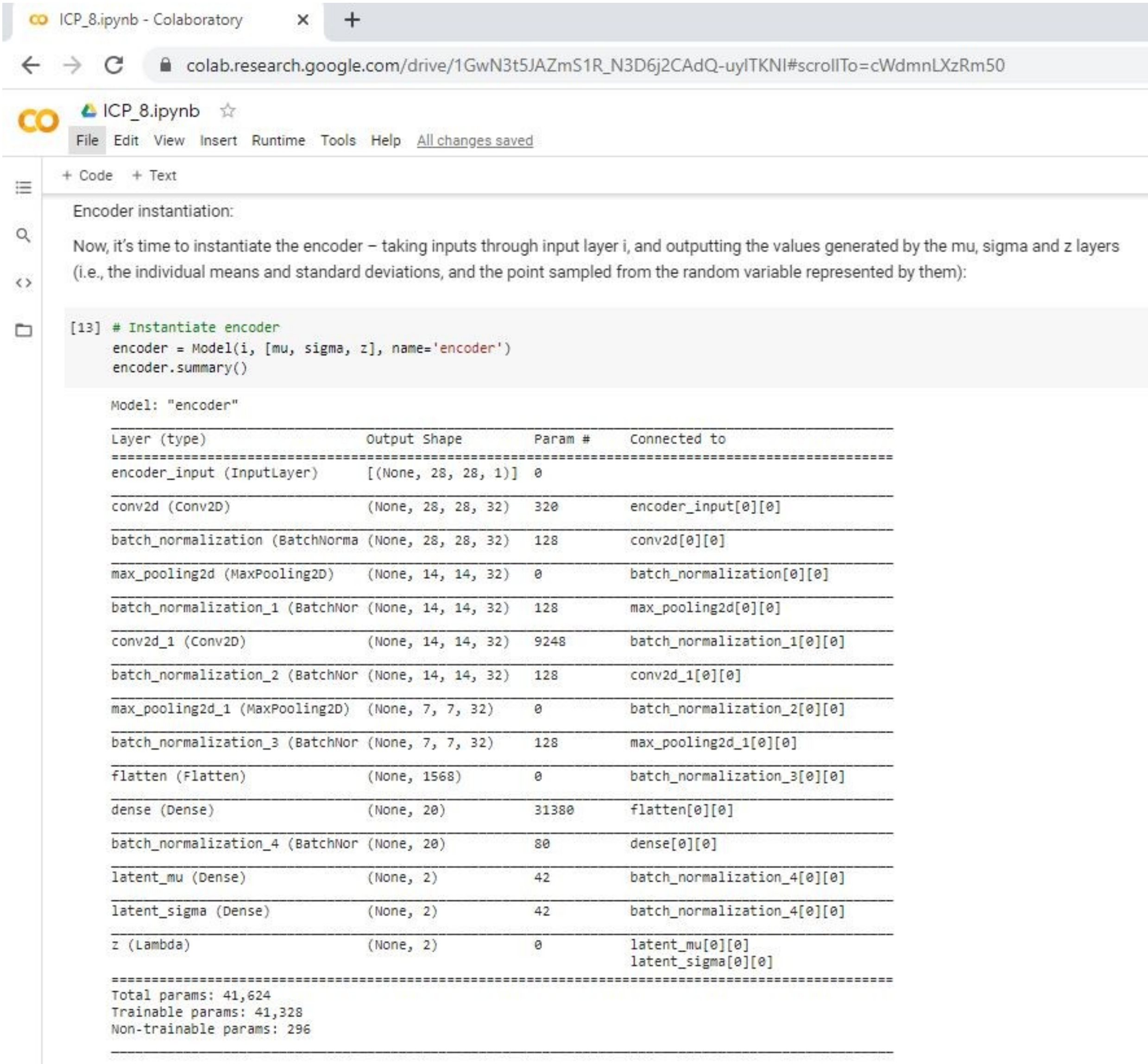



```
# Encoder Definition
i       = Input(shape=input_shape, name='encoder_input')
cx      = Conv2D(filters=8, kernel_size=3, strides=2, padding='same', activation='relu')(i)
cx      = BatchNormalization()(cx)
cx      = Conv2D(filters=16, kernel_size=3, strides=2, padding='same', activation='relu')(cx)
cx      = BatchNormalization()(cx)
x       = Flatten()(cx)
x       = Dense(20, activation='relu')(x)
x       = BatchNormalization()(x)
mu      = Dense(latent_dim, name='latent_mu')(x)
sigma   = Dense(latent_dim, name='latent_sigma')(x)
```

Updated Encoder Definition, with Max Pooling added:



Encoder Summary (With Max Pooling):



ICP_8.ipynb - Colaboratory

colab.research.google.com/drive/1GwN3t5JAZmS1R_N3D6j2CAdQ-uyITKNI#scrollTo=cWdmnLXzRm50

ICP_8.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

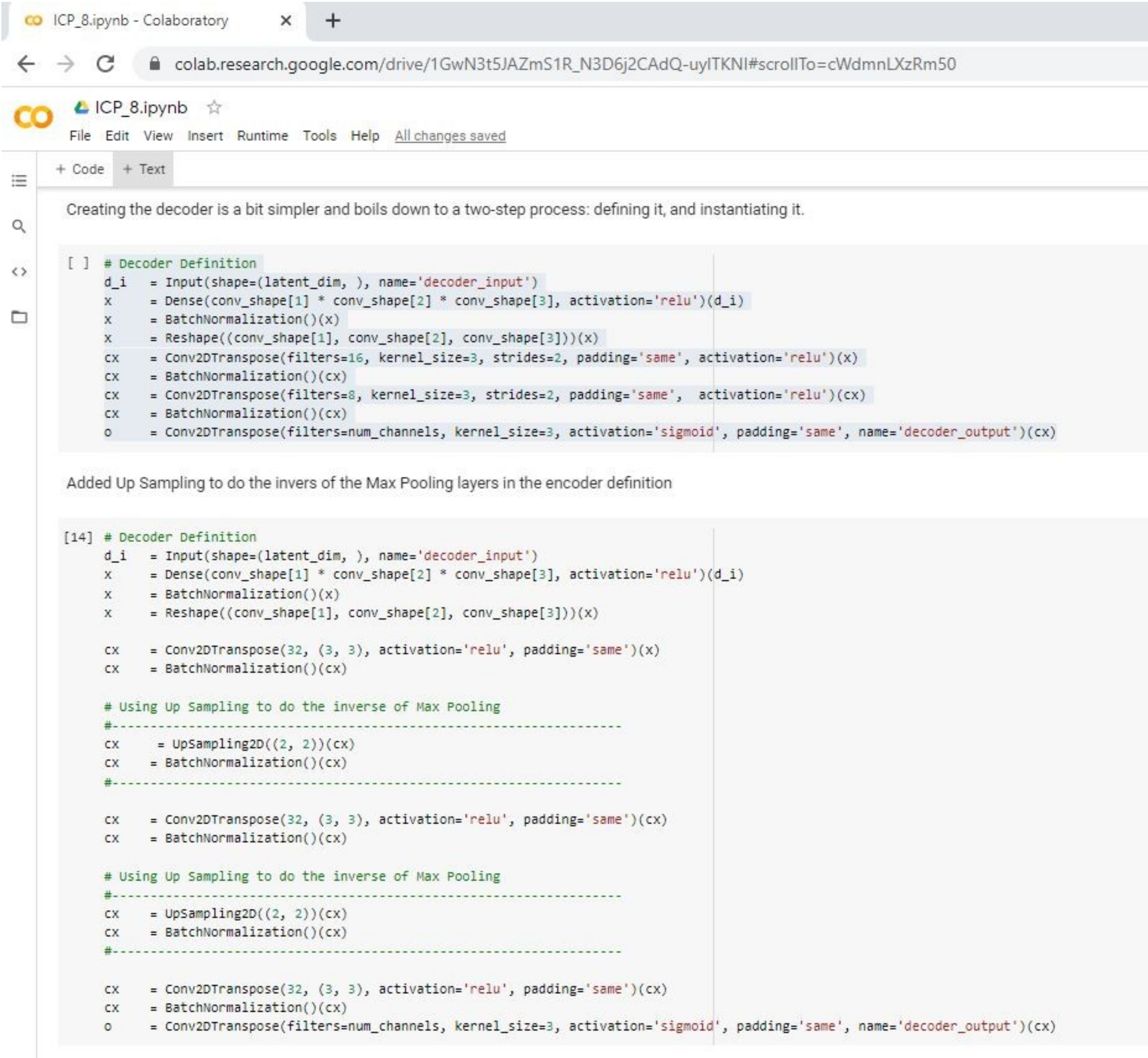
Encoder instantiation:

Now, it's time to instantiate the encoder – taking inputs through input layer i, and outputting the values generated by the mu, sigma and z layers (i.e., the individual means and standard deviations, and the point sampled from the random variable represented by them):

```
[13] # Instantiate encoder
     encoder = Model(i, [mu, sigma, z], name='encoder')
     encoder.summary()
```

```
Model: "encoder"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
encoder_input (InputLayer)      [(None, 28, 28, 1)]  0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 28, 28, 32)   320         encoder_input[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 28, 28, 32)   128         conv2d[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 14, 14, 32)   0           batch_normalization[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 14, 14, 32)   128         max_pooling2d[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 14, 14, 32)   9248        batch_normalization_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 14, 14, 32)   128         conv2d_1[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 7, 7, 32)     0           batch_normalization_2[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 7, 7, 32)     128         max_pooling2d_1[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 1568)         0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 20)           31380       flatten[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 20)           80          dense[0][0]
__________________________________________________________________________________________________
latent_mu (Dense)               (None, 2)            42          batch_normalization_4[0][0]
__________________________________________________________________________________________________
latent_sigma (Dense)            (None, 2)            42          batch_normalization_4[0][0]
__________________________________________________________________________________________________
z (Lambda)                      (None, 2)            0           latent_mu[0][0]
                                                                 latent_sigma[0][0]
==================================================================================================
Total params: 41,624
Trainable params: 41,328
Non-trainable params: 296
__________________________________________________________________________________________________
```

Decoder Definition from Class and the updated definition with Up Sampling:


ICP_8.ipynb - Colaboratory

colab.research.google.com/drive/1GwN3t5JAZmS1R_N3D6j2CAdQ-uyITKNI#scrollTo=cWdmnLXzRm50

ICP_8.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Creating the decoder is a bit simpler and boils down to a two-step process: defining it, and instantiating it.

```
[ ] # Decoder Definition
    d_i   = Input(shape=(latent_dim, ), name='decoder_input')
    x     = Dense(conv_shape[1] * conv_shape[2] * conv_shape[3], activation='relu')(d_i)
    x     = BatchNormalization()(x)
    x     = Reshape((conv_shape[1], conv_shape[2], conv_shape[3]))(x)
    cx    = Conv2DTranspose(filters=16, kernel_size=3, strides=2, padding='same', activation='relu')(x)
    cx    = BatchNormalization()(cx)
    cx    = Conv2DTranspose(filters=8, kernel_size=3, strides=2, padding='same',  activation='relu')(cx)
    cx    = BatchNormalization()(cx)
    o     = Conv2DTranspose(filters=num_channels, kernel_size=3, activation='sigmoid', padding='same', name='decoder_output')(cx)
```

Added Up Sampling to do the invers of the Max Pooling layers in the encoder definition

```
[14] # Decoder Definition
    d_i   = Input(shape=(latent_dim, ), name='decoder_input')
    x     = Dense(conv_shape[1] * conv_shape[2] * conv_shape[3], activation='relu')(d_i)
    x     = BatchNormalization()(x)
    x     = Reshape((conv_shape[1], conv_shape[2], conv_shape[3]))(x)

    cx    = Conv2DTranspose(32, (3, 3), activation='relu', padding='same')(x)
    cx    = BatchNormalization()(cx)

    # Using Up Sampling to do the inverse of Max Pooling
    #-------------------------------------------------------------------
    cx     = UpSampling2D((2, 2))(cx)
    cx    = BatchNormalization()(cx)
    #-------------------------------------------------------------------

    cx    = Conv2DTranspose(32, (3, 3), activation='relu', padding='same')(cx)
    cx    = BatchNormalization()(cx)

    # Using Up Sampling to do the inverse of Max Pooling
    #-------------------------------------------------------------------
    cx    = UpSampling2D((2, 2))(cx)
    cx    = BatchNormalization()(cx)
    #-------------------------------------------------------------------

    cx    = Conv2DTranspose(32, (3, 3), activation='relu', padding='same')(cx)
    cx    = BatchNormalization()(cx)
    o     = Conv2DTranspose(filters=num_channels, kernel_size=3, activation='sigmoid', padding='same', name='decoder_output')(cx)
```

Decoder Summary (With Up Sampling):

ICP_8.ipynb - Colaboratory

colab.research.google.com/drive/1GwN3t5JAZmS1R_N3D6j2CAdQ-uylTKNI#scrollTo=cWdmnLXzRm50

ICP_8.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Decoder instantiation

The next thing we do is instantiate the decoder:

It takes the inputs from the decoder input layer d_i and outputs whatever is output by the output layer o.

```
[15] # Instantiate decoder
     decoder = Model(d_i, o, name='decoder')
     decoder.summary()
```

```
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
decoder_input (InputLayer)   [(None, 2)]               0
_________________________________________________________________
dense_1 (Dense)              (None, 1568)              4704
_________________________________________________________________
batch_normalization_5 (Batch (None, 1568)              6272
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 7, 7, 32)          9248
_________________________________________________________________
batch_normalization_6 (Batch (None, 7, 7, 32)          128
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 14, 14, 32)        0
_________________________________________________________________
batch_normalization_7 (Batch (None, 14, 14, 32)        128
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 14, 14, 32)        9248
_________________________________________________________________
batch_normalization_8 (Batch (None, 14, 14, 32)        128
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 28, 28, 32)        0
_________________________________________________________________
batch_normalization_9 (Batch (None, 28, 28, 32)        128
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 28, 28, 32)        9248
_________________________________________________________________
batch_normalization_10 (Batc (None, 28, 28, 32)        128
_________________________________________________________________
decoder_output (Conv2DTransp (None, 28, 28, 1)         289
=================================================================
Total params: 39,649
Trainable params: 36,193
Non-trainable params: 3,456
_________________________________________________________________
```

Creating the whole VAE:

ICP_8.ipynb - Colaboratory

colab.research.google.com/drive/1GwN3t5JAZmS1R_N3D6j2CAdQ-uylTKNI#scrollTo=BLsGI5u858G-

ICP_8.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Creating the whole VAE

Now that the encoder and decoder are complete, we can create the VAE as a whole.

If you think about it, the outputs of the entire VAE are the original inputs, encoded by the encoder, and decoded by the decoder.

That's how we arrive at vae_outputs = decoder(encoder(i)[2]): inputs i are encoded by the encoder into [mu, sigma, z] (the individual means and standard deviations with the sampled z as well). We then take the sampled z values (hence the [2]) and feed it to the decoder, which ensures that we arrive at correct VAE output.

We then instantiate the model: i are our inputs indeed, and vae_outputs are the outputs. We call the model vae, because it simply is Variational Auto Encoder.

```
[16] # Instantiate VAE
     vae_outputs = decoder(encoder(i)[2])
     vae         = Model(i, vae_outputs, name='vae')
     vae.summary()
```

```
Model: "vae"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_input (InputLayer)   [(None, 28, 28, 1)]       0
_________________________________________________________________
encoder (Functional)         [(None, 2), (None, 2), (N 41624
_________________________________________________________________
decoder (Functional)         (None, 28, 28, 1)         39649
=================================================================
Total params: 81,273
Trainable params: 77,521
Non-trainable params: 3,752
_________________________________________________________________
```

Compilation and Training:

ICP_8.ipynb - Colaboratory

colab.research.google.com/drive/1GwN3t5JAZmS1R_N3D6j2CAdQ-uyITKNl#scrollTo=AW4iPhE-58J2

ICP_8.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Compilation & training:

we can compile our model. We do so using the Adam optimizer and binary crossentropy loss function.

tf.config.run_functions_eagerly(True) is added to overcome the vae.fit error with colab, you may not need it if you are running the code in pycharm or jupyter notebooks.

```
[17] tf.config.run_functions_eagerly(True)
     # Compile VAE
     vae.compile(optimizer='adam', loss='binary_crossentropy')

     # Train autoencoder
     vae.fit(input_train, input_train, epochs = no_epochs, batch_size = batch_size, validation_split = validation_split)
```

```
Epoch 17/50
125/125 [==============================] - 6s 46ms/step - loss: 0.1898 - val_loss: 0.1932
Epoch 18/50
125/125 [==============================] - 6s 46ms/step - loss: 0.1888 - val_loss: 0.1882
Epoch 19/50
125/125 [==============================] - 6s 46ms/step - loss: 0.1874 - val_loss: 0.1873
Epoch 20/50
125/125 [==============================] - 6s 46ms/step - loss: 0.1878 - val_loss: 0.1894
Epoch 21/50
125/125 [==============================] - 6s 45ms/step - loss: 0.1866 - val_loss: 0.1890
Epoch 22/50
125/125 [==============================] - 6s 46ms/step - loss: 0.1859 - val_loss: 0.1906
Epoch 23/50
125/125 [==============================] - 6s 46ms/step - loss: 0.1856 - val_loss: 0.1852
Epoch 24/50
125/125 [==============================] - 6s 45ms/step - loss: 0.1853 - val_loss: 0.1877
Epoch 25/50
125/125 [==============================] - 6s 45ms/step - loss: 0.1844 - val_loss: 0.1854
Epoch 26/50
125/125 [==============================] - 6s 45ms/step - loss: 0.1854 - val_loss: 0.1892
Epoch 27/50
125/125 [==============================] - 6s 46ms/step - loss: 0.1849 - val_loss: 0.1860
Epoch 28/50
125/125 [==============================] - 6s 46ms/step - loss: 0.1840 - val_loss: 0.1845
Epoch 29/50
125/125 [==============================] - 6s 46ms/step - loss: 0.1836 - val_loss: 0.1853
Epoch 30/50
125/125 [==============================] - 6s 46ms/step - loss: 0.1829 - val_loss: 0.1829
Epoch 31/50
125/125 [==============================] - 6s 46ms/step - loss: 0.1826 - val_loss: 0.1827
Epoch 32/50
125/125 [==============================] - 6s 46ms/step - loss: 0.1820 - val_loss: 0.1825
Epoch 33/50
125/125 [==============================] - 6s 46ms/step - loss: 0.1821 - val_loss: 0.1867
Epoch 34/50
125/125 [==============================] - 6s 46ms/step - loss: 0.1819 - val_loss: 0.1842
Epoch 35/50
```

Result Visualization:

ICP_8.ipynb - Colaboratory

colab.research.google.com/drive/1GwN3t5JAZmS1R_N3D6j2CAdQ-uylTKNI#scrollTo=bLyv256e8kXM

ICP_8.ipynb

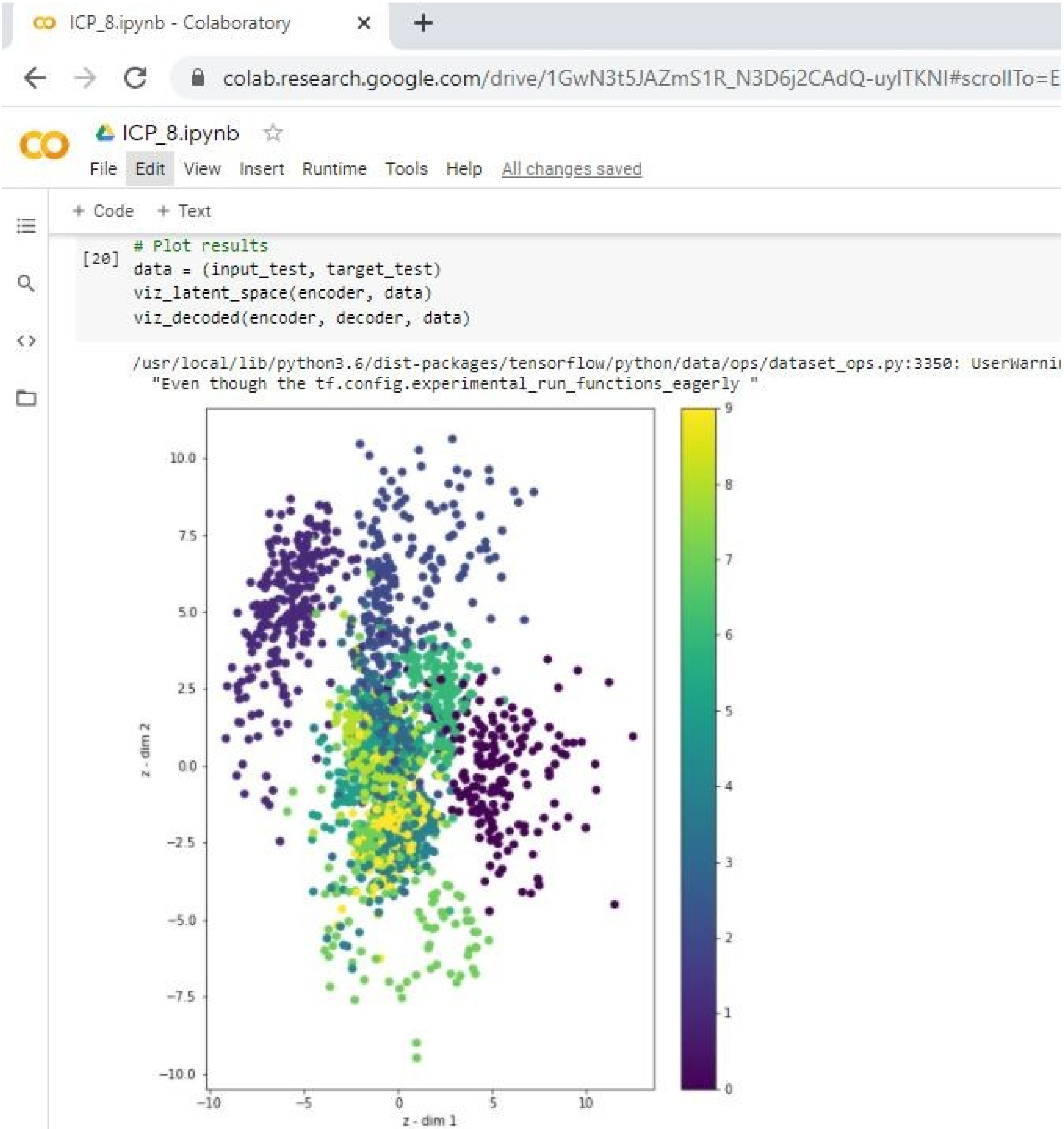
File Edit View Insert Runtime Tools Help All changes saved

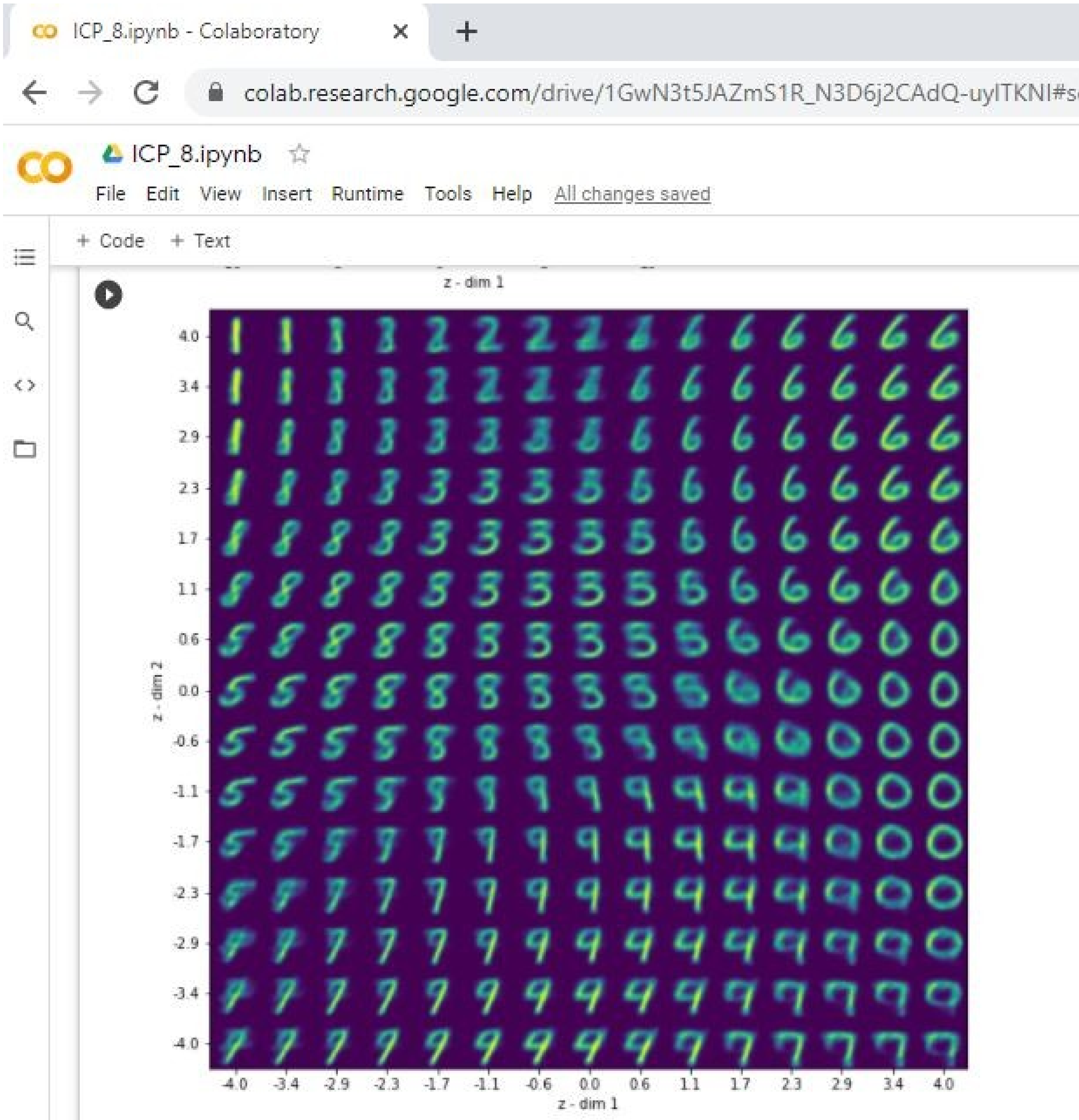
\+ Code + Text

```
[18] # Results visualization
     # Credits for original visualization code: https://keras.io/examples/variational_autoencoder_deconv/


     def viz_latent_space(encoder, data):
       input_data, target_data = data
       mu, _, _ = encoder.predict(input_data)
       plt.figure(figsize=(8, 10))
       plt.scatter(mu[:, 0], mu[:, 1], c=target_data)
       plt.xlabel('z - dim 1')
       plt.ylabel('z - dim 2')
       plt.colorbar()
       plt.show()
```

```
[19] def viz_decoded(encoder, decoder, data):
       num_samples = 15
       figure = np.zeros((img_width * num_samples, img_height * num_samples, num_channels))
       grid_x = np.linspace(-4, 4, num_samples)
       grid_y = np.linspace(-4, 4, num_samples)[::-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(img_width, img_height, num_channels)
               figure[i * img_width: (i + 1) * img_width,
                       j * img_height: (j + 1) * img_height] = digit
       plt.figure(figsize=(10, 10))
       start_range = img_width // 2
       end_range = num_samples * img_width + start_range + 1
       pixel_range = np.arange(start_range, end_range, img_width)
       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 - dim 1')
       plt.ylabel('z - dim 2')
       # matplotlib.pyplot.imshow() needs a 2D array, or a 3D array with the third dimension being of shape 3 or 4!
       # So reshape if necessary
       fig_shape = np.shape(figure)
       if fig_shape[2] == 1:
         figure = figure.reshape((fig_shape[0], fig_shape[1]))
       # Show image
       plt.imshow(figure)
       plt.show()
```

Plot Results:




ICP_8.ipynb - Colaboratory
colab.research.google.com/drive/1GwN3t5JAZmS1R_N3D6j2CAdQ-uylTKNI#sc
ICP_8.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
z - dim 1
4.0 3.4 2.9 2.3 1.7 1.1 0.6 0.0 -0.6 -1.1 -1.7 -2.3 -2.9 -3.4 -4.0
z - dim 2
-4.0 -3.4 -2.9 -2.3 -1.7 -1.1 -0.6 0.0 0.6 1.1 1.7 2.3 2.9 3.4 4.0
z - dim 1

[Video Link]

Any in site about the data or the ICP in general

The larger the image size, the more layers that can be added with minimal effert.