CS5542 Big Data Apps and Analytics

In Class Programming –8 Report
(Jongkook Son)

Project Overview:

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.

Requirements/Task(s):

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)

What I learned in ICP:

I could have learned the basic structure of the encoder and decoder and variational auto encoders by doing this ICP. Autoencoders are a specific type of feedforward neural networks where the input is the same as the output. So in our icp I tried to make encoder and decoder with asymmetric structure. Like adding Conv2dTranspose to decoder if I add a Conv2d to encoder. The whole training process is like Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data. In other words, they are self-supervised because they generate their own labels from the training data. In this ICP I used a picture datas and by using autoencoders could genereate own labels of data and could have generated new pictures from it. Finally it is unavoidable that there could be some loss in auto encoders so the generated picture is not 100 percent same with the original picture.

ICP description what was the task you were performing and Screen shots that shows the successful execution of each required step of your code

Configuration

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.

```
[ ]  #use the whole data
     input_train=input_train_1
     target_train=target_train_1
     input_test=input_test_1
     target_test=target_test_1
```

```
[7]  # Data & model configuration
     img_width, img_height = input_train.shape[1], input_train.shape[2]
     batch_size = 128
     no_epochs = 80
     validation_split = 0.2
     verbosity = 1
     latent_dim = 2
     num_channels = 1
```

Next, we reshape the data so that it takes the shape (X, 28, 28, 1), where X is the number of samples in either the training or testing dataset. We also set (28, 28, 1) as input_shape.

Next, we parse the numbers as floats, which presumably speeds up the training process, and normalize it, which the neural network appreciates

```
[8]  # 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
```

⇨ **In the source code, training set and test is reduced due to memory issue. However in my ICP I tried to use the whole data since my local environment can handle the datas. Also I increased the number of epoch 80 to get more accurate result.**

ADD 4 layers in encoder path and 4layeres in decoder path

```python
# Encoder Definition
i       = Input(shape=input_shape, name='encoder_input')

# Conv2D with 8 filters
cx      = Conv2D(filters=8, kernel_size=3, strides=1, padding='same', activation='relu')(i)
cx      = BatchNormalization()(cx)

# Conv2D with 16 filters
cx      = Conv2D(filters=16, kernel_size=3, strides=1, padding='same', activation='relu')(cx)
cx      = BatchNormalization()(cx)

# Conv2D with 32 filters
cx      = Conv2D(filters=32, kernel_size=3, strides=1, padding='same', activation='relu')(cx)
cx      = BatchNormalization()(cx)

# Conv2D with 64 filters
cx      = Conv2D(filters=64, kernel_size=3, strides=1, 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)
```

```python
# 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, 8)    80          encoder_input[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 28, 28, 8)    32          conv2d[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 28, 28, 16)   1168        batch_normalization[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 28, 28, 16)   64          conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 28, 28, 32)   4640        batch_normalization_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 28, 28, 32)   128         conv2d_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 28, 28, 64)   18496       batch_normalization_2[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 28, 28, 64)   256         conv2d_3[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 50176)        0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 20)           1003540     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: 1,028,568
Trainable params: 1,028,288
Non-trainable params: 280
__________________________________________________________________________________________________
```

⇨ **I add 4 layers in my ICP. Which are Conv2d layers with 32 filters 64 filters. The reason I add these layers is that by adding increased filter I can train more complex and deeper pattern of the picture. Also I added batch normalization layer after each Conv2D layers. This layer ensures that the outputs of the Conv2D layer that are input to the next Conv2D layer have a steady mean and rate and makes overall training faster and steady.**

```python
# 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)

#Conv2DTranspose with 64 filters
cx    = Conv2DTranspose(filters=64, kernel_size=3, strides=1, padding='same',  activation='relu')(cx)
cx    = BatchNormalization()(cx)

#Conv2DTranspose with 32 filters
cx    = Conv2DTranspose(filters=32, kernel_size=3, strides=1, padding='same',  activation='relu')(cx)
cx    = BatchNormalization()(cx)

#Conv2DTranspose with 16 filters
cx    = Conv2DTranspose(filters=16, kernel_size=3, strides=1, padding='same', activation='relu')(x)
cx    = BatchNormalization()(cx)

#Conv2DTranspose with 8 filters
cx    = Conv2DTranspose(filters=8, kernel_size=3, strides=1, padding='same',  activation='relu')(cx)
cx    = BatchNormalization()(cx)

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

```python
# 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, 50176)             150528
_________________________________________________________________
batch_normalization_5 (Batch (None, 50176)             200704
_________________________________________________________________
reshape (Reshape)            (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 28, 28, 16)        9232
_________________________________________________________________
batch_normalization_8 (Batch (None, 28, 28, 16)        64
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 28, 28, 8)         1160
_________________________________________________________________
batch_normalization_9 (Batch (None, 28, 28, 8)         32
_________________________________________________________________
decoder_output (Conv2DTransp (None, 28, 28, 1)         73
=================================================================
Total params: 361,793
Trainable params: 261,393
Non-trainable params: 100,400
_________________________________________________________________
```

⇨ **Since we added Conv2d layers with 32 filters and 64 filters. We need to add Conv2DTranspose and BatchNormalization in the exact opposite order as with our encoder.**

Result of the ICP

<The training result of the Source Code>

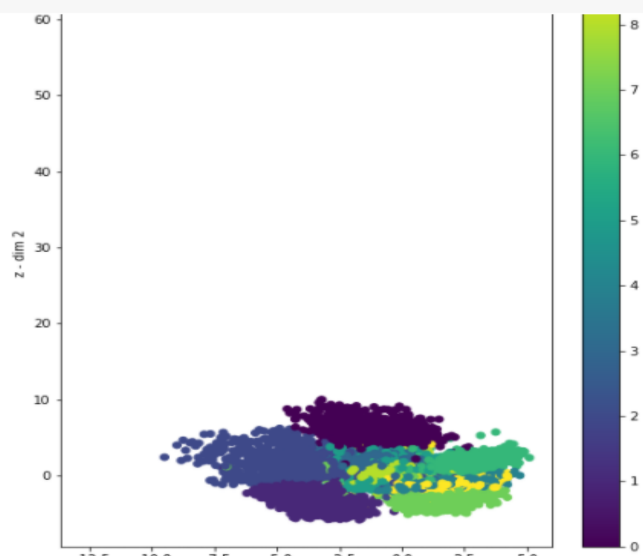
```
63/63 [==============================] - 2s 32ms/step - loss: 0.1881 - val_loss: 0.1954
Epoch 45/50
63/63 [==============================] - 2s 34ms/step - loss: 0.1880 - val_loss: 0.1941
Epoch 46/50
63/63 [==============================] - 2s 31ms/step - loss: 0.1882 - val_loss: 0.1951
Epoch 47/50
63/63 [==============================] - 2s 32ms/step - loss: 0.1875 - val_loss: 0.1950
Epoch 48/50
63/63 [==============================] - 2s 32ms/step - loss: 0.1874 - val_loss: 0.1951
Epoch 49/50
63/63 [==============================] - 2s 32ms/step - loss: 0.1871 - val_loss: 0.1959
Epoch 50/50
63/63 [==============================] - 2s 34ms/step - loss: 0.1879 - val_loss: 0.1951
<tensorflow.python.keras.callbacks.History at 0x7f38d01fe668>
```

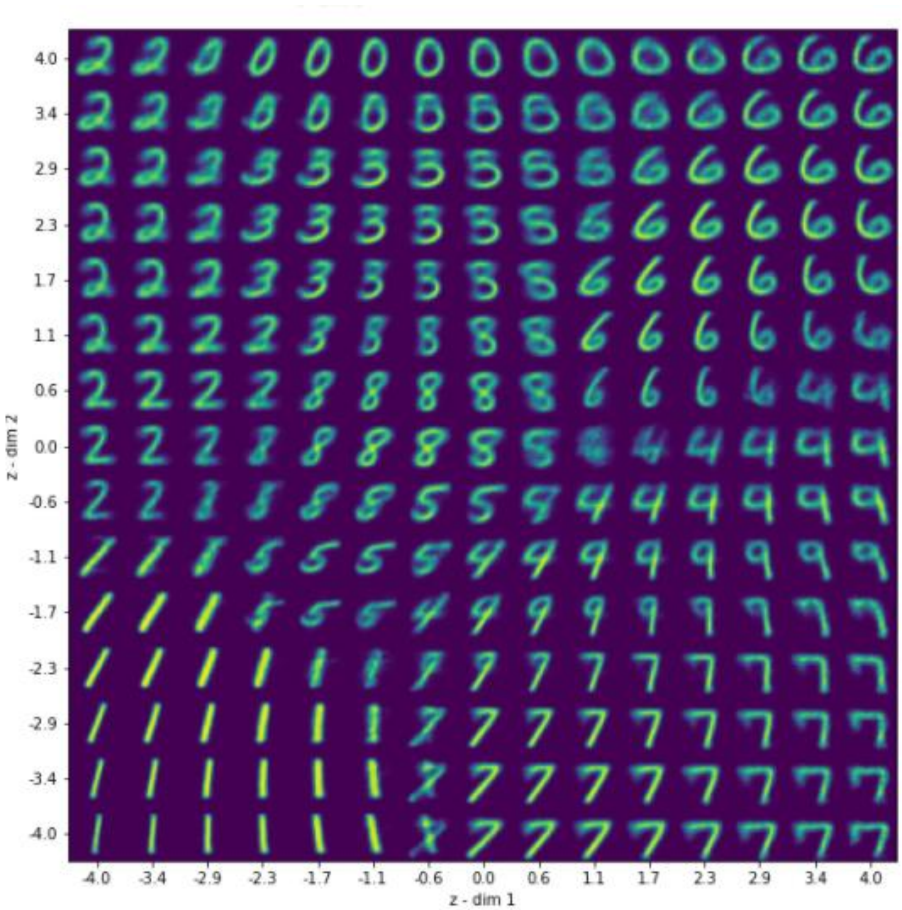
<The training result of the ICP>

```
375/375 [==============================] - 27s 72ms/step - loss: 0.1707 - val_loss: 0.1752
Epoch 72/80
375/375 [==============================] - 26s 71ms/step - loss: 0.1705 - val_loss: 0.1753
Epoch 73/80
375/375 [==============================] - 27s 71ms/step - loss: 0.1704 - val_loss: 0.1769
Epoch 74/80
375/375 [==============================] - 27s 71ms/step - loss: 0.1710 - val_loss: 0.2173
Epoch 75/80
375/375 [==============================] - 27s 71ms/step - loss: 0.1703 - val_loss: 0.1754
Epoch 76/80
375/375 [==============================] - 27s 72ms/step - loss: 0.1699 - val_loss: 0.1743
Epoch 77/80
375/375 [==============================] - 27s 71ms/step - loss: 0.1702 - val_loss: 0.1761
Epoch 78/80
375/375 [==============================] - 27s 71ms/step - loss: 0.1699 - val_loss: 0.1764
Epoch 79/80
375/375 [==============================] - 27s 71ms/step - loss: 0.1690 - val_loss: 0.1772
Epoch 80/80
375/375 [==============================] - 27s 72ms/step - loss: 0.1700 - val_loss: 0.1751
<tensorflow.python.keras.callbacks.History at 0x7f9d70352[illegible]>
```

<Visualization Result>

```
# Plot results
data = (input_test, target_test)
viz_latent_space(encoder, data)
viz_decoded(encoder, decoder, data)
```





⇨ **The loss and validate loss number for the training is getting better in this ICP. Even though there are some loss in the training process the overall visualization result is quite good.**

Conclusion:

By adding a Conv2d layers with 32 filters and 64 filters and batch normalization. More complex and deeper patterns of the pictures in the dataset is trained. So the loss and val_loss number is decreased and overall training result is getting better. And the visualization result is getting better compared to Source Code.

Challenges that I faced:

The most difficult challenge that I faced was that it was hard to grasp the structure of the variational autoencoders. And especially hard to understand how encoder and decoder is connected in this VAE. But I figured out there is an process named reparameritization in this process we can reparameterize the sample fed to the function into the shape $\mu+\sigma2\times\epsilon$, it now becomes possible to use gradient descent for estimating the gradients accurately the sampled z values and feed it to the decoder, which ensures that we arrive at correct VAE output. And this makes encoder and decoder connected and makes gradient decent work more efficiently.

Video link

https://www.youtube.com/watch?v=rXc9OlKBnw8