In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.utils import get_file
import time
import os
from scipy.io import loadmat
%load_ext tensorboard
```

Loading SVHN Data

In [2]:
```
data_file_train = get_file('train_32x32.mat', origin='http://ufldl.stanford.edu/housenumbers/train
_32x32.mat')
data_file_valid = get_file('test_32x32.mat', origin='http://ufldl.stanford.edu/housenumbers/test_3
2x32.mat')

train_dir = os.path.join(os.path.dirname(data_file_train), 'train_32x32.mat')
valid_dir = os.path.join(os.path.dirname(data_file_valid), 'test_32x32.mat')

train_data = loadmat(train_dir)
valid_data = loadmat(valid_dir)

X_train, y_train = train_data['X'], train_data['y']
X_valid, y_valid = valid_data['X'], valid_data['y']

print(X_train.shape) # shape is in (image_height, image_width, channels, records)
```

```
Downloading data from http://ufldl.stanford.edu/housenumbers/train_32x32.mat
182042624/182040794 [==============================] - 2s 0us/step
Downloading data from http://ufldl.stanford.edu/housenumbers/test_32x32.mat
64282624/64275384 [==============================] - 1s 0us/step
(32, 32, 3, 73257)
```

In [3]:
```
X_train = np.rollaxis(X_train, 3)
X_valid = np.rollaxis(X_valid, 3)

print(X_train.shape) # convert shape to (records, image_height, image_width, channels)
print(X_valid.shape)
```

```
(73257, 32, 32, 3)
(26032, 32, 32, 3)
```

In [4]:
```
# glimpse of the real images
sample_images = X_train[np.random.choice(len(X_train), size=64, replace=False)]

plt.figure(figsize=(8, 8))
for i in range(64):
    plt.subplot(8, 8, i+1)
    plt.imshow(sample_images[i])
    plt.xticks([])
    plt.yticks([])
plt.tight_layout()
plt.show()
```

Preprocessing the data

In [5]:
```
def preprocess(x):
    return (x-127.5)/127.5 # standardize to [-1, 1], so that tanh function is applicable

def decode(x):
    return np.uint8(127.5*x+127.5) # make sure to use uint8 type otherwise the image won't display
properly
```

In [6]:
```
X_train_real = preprocess(X_train)
X_valid_real  = preprocess(X_valid)
```

Generator

In [7]:
```
def build_generator(input_size, leaky_alpha=0.2):
    '''generates images in (32,32,3)
    with up-sampling technique'''

    model = tf.keras.Sequential([
        layers.Dense(4*4*512, input_shape=(input_size,)),
        layers.Reshape(target_shape=(4, 4, 512)),                                  # 4,4,512
        layers.BatchNormalization(),
        layers.LeakyReLU(alpha=leaky_alpha),
        layers.Dropout(0.2),
        layers.Conv2DTranspose(256, kernel_size=5, strides=2, padding='same', use_bias=False), #
 8,8,256
        layers.BatchNormalization(),
        layers.LeakyReLU(alpha=leaky_alpha),
        layers.Conv2DTranspose(128, kernel_size=5, strides=2, padding='same', use_bias=False), # 1
6,16,128
        layers.BatchNormalization(),
        layers.LeakyReLU(alpha=leaky_alpha),
        layers.Conv2DTranspose(3, kernel_size=5, strides=2, padding='same', use_bias=False),   # 3
2,32,3
        layers.Activation('tanh')
    ])

    return model
```

In [8]:
```
g = build_generator(100, 0.2)
g.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 8192)              827392
_________________________________________________________________
reshape (Reshape)            (None, 4, 4, 512)         0
_________________________________________________________________
batch_normalization (BatchNo (None, 4, 4, 512)         2048
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 4, 4, 512)         0
_________________________________________________________________
dropout (Dropout)            (None, 4, 4, 512)         0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 8, 8, 256)         3276800
_________________________________________________________________
batch_normalization_1 (Batch (None, 8, 8, 256)         1024
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 16, 16, 128)       819200
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 32, 32, 3)         9600
_________________________________________________________________
activation (Activation)      (None, 32, 32, 3)         0
=================================================================
Total params: 4,936,576
Trainable params: 4,934,784
Non-trainable params: 1,792
_________________________________________________________________
```

Discriminator

A classifier to distinguish between real and fake images.

Input: 32x32x3 (32x32 image with 3 channels), with values between [-1,1]

Output: probability of image being real

In [9]:
```
def build_discriminator(leaky_alpha=0.2):
  model = tf.keras.Sequential([
            layers.Conv2D(64, (5,5), strides=2, padding='same', input_shape=(32,32,3)), # 16,16,64
            layers.LeakyReLU(leaky_alpha),
            layers.Dropout(0.2),
            layers.Conv2D(128, (5,5), strides=2, padding='same'), # 8,8,128
            layers.BatchNormalization(),
            layers.LeakyReLU(leaky_alpha),
            layers.Dropout(0.2),
            layers.Conv2D(256, (5,5), strides=2, padding='same'), # 4,4,256
            layers.BatchNormalization(),
            layers.LeakyReLU(leaky_alpha),
            layers.Flatten(),
            layers.Dense(1, activation='sigmoid')
  ])
  return model
```

In [10]:
```
d = build_discriminator(0.2)
d.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 16, 16, 64)        4864
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 16, 16, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 8, 8, 128)         204928
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 8, 8, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 256)         819456
_________________________________________________________________
batch_normalization_4 (Batch (None, 4, 4, 256)         1024
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 4, 4, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 4097
=================================================================
Total params: 1,034,881
Trainable params: 1,034,113
Non-trainable params: 768
_________________________________________________________________
```

DCGAN

In [11]:
```
def build_gan(input_size, leaky_alpha,
              discriminator_lr, generator_lr,
              dis_beta_1, gen_beta_1):

  ''' input_size: the length of vector that would be put in the generator
      beta_1: the exponential decay rate for the 1st moment estimates in Adam optimizer
  '''

  # generator
  generator = build_generator(input_size, leaky_alpha)

  # discriminator
  discriminator = build_discriminator(leaky_alpha)
  discriminator.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=discriminator_lr, beta_1=
dis_beta_1),
                        loss='binary_crossentropy')

  # GAN
  gan = tf.keras.Sequential([generator, discriminator])
  gan.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=generator_lr, beta_1=gen_beta_1),
              loss='binary_crossentropy')

  return gan, generator, discriminator
```

Training GAN

In [12]:
```
# generate random vectors to be put into generator
def generate_random_vectors(n_samples, input_size):
  return np.random.normal(loc=0, scale=1, size=(n_samples, input_size))

# set discriminator to trainable or not
def set_to_trainable(model, trainable):
    for layer in model.layers:
        layer.trainable = trainable

# show 64 generated images in square
def show_images(generated_images):
    n_images = len(generated_images)
    cols = 8
    rows = n_images//cols

    plt.figure(figsize=(8, 8))
    for i in range(n_images):
        img = decode(generated_images[i])
        ax = plt.subplot(rows, cols, i+1)
        plt.imshow(img)
        plt.xticks([])
        plt.yticks([])
    plt.tight_layout()
    plt.show()
```

To train a GAN:

- train discriminator with real images with the label smoothing (labels=1-smooth) (**smooth: trick**)
- train discriminator with fake images generated by the generator (labels=0)
- set the discriminator non-trainable
- train the GAN with generated images (labels=1) (i.e. train the generator with discriminator fixed)

In [13]:
```
# Transform train_on_batch return value to dict
def named_logs(model, logs):
  result = {}
  for l in zip(['training_loss','validation_loss'], logs):
    result[l[0]] = l[1]
  return result
```

In [14]:
```
# the seed would be repeatedly used for image output at different epoch
np.random.seed(2020)
seed = np.random.normal(loc=0, scale=1, size=(64, 100))
```

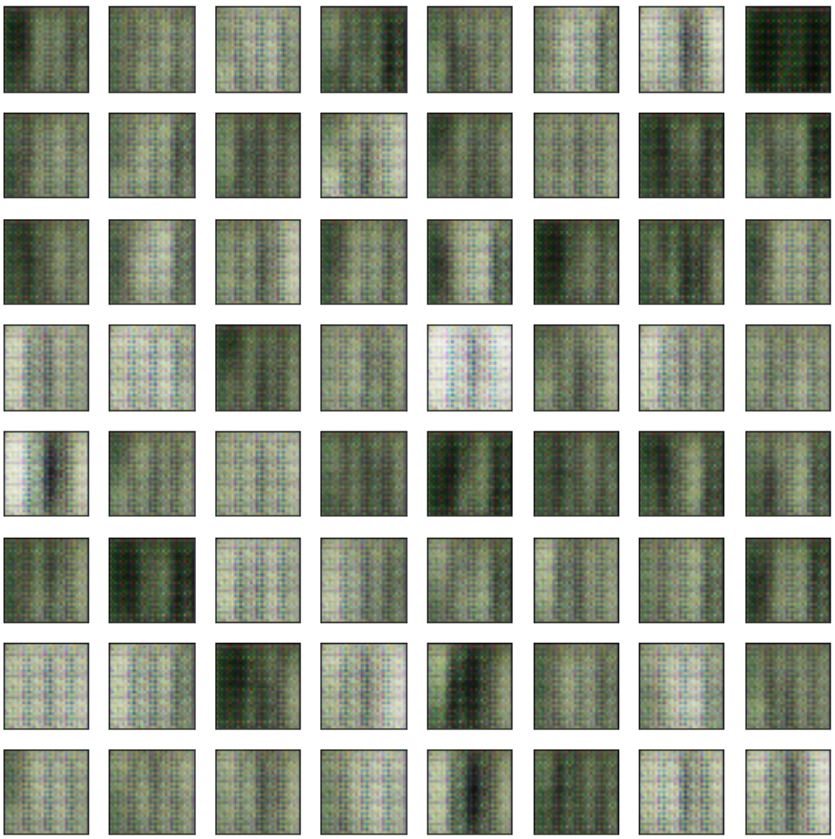
In [15]:
```
!rm -rf ./logs_SVHN/
```

In [16]:
```
def train(generator_lr,
          discriminator_lr,
          gen_beta_1,
          dis_beta_1,
          leaky_alpha,
          smooth=0.1,
          input_size=100,
          epochs=50,
          batch_size=128,
          val_size=36):

    start = time.time()

    # labels for the batch size and the validation size
    y_train_real, y_train_fake = np.ones([batch_size, 1]), np.zeros([batch_size, 1])
    y_val_real,  y_val_fake  = np.ones([val_size, 1]), np.zeros([val_size, 1])

    # create a GAN, a generator and a discriminator
    gan, generator, discriminator = build_gan(input_size, leaky_alpha, discriminator_lr, generator
_lr, dis_beta_1, gen_beta_1)

    # prepare for tensorboard
    tb_dis = tf.keras.callbacks.TensorBoard(
        log_dir='./logs_SVHN/discriminator',
        histogram_freq=1,
        write_graph=True)
    tb_dis.set_model(discriminator)

    tb_gan = tf.keras.callbacks.TensorBoard(
        log_dir='./logs_SVHN/gan',
        histogram_freq=1,
        write_graph=True)
    tb_gan.set_model(gan)

    # iterate epochs to train the GAN
    num_batches = len(X_train_real)//batch_size
    for e in range(epochs):
      for i in range(num_batches):
        # real images in a batch
        X_batch_real = X_train_real[i*batch_size:(i+1)*batch_size]

        # generate a batch of fake images using random vectors
        random_vectors = generate_random_vectors(batch_size, input_size)
        X_batch_fake = generator.predict_on_batch(random_vectors)

        # train the discriminator
        # try: soft labels - random number between 0 and 0.1 to represent 0 labels (real images)
        # random number between 0.9 and 1.0 to represent 1 labels
        # smooth = np.random.uniform(low=0,high=0.1,size=1)[0]
        set_to_trainable(discriminator, True)
        dis_loss_train = discriminator.train_on_batch(X_batch_real, y_train_real-0.1)
        dis_loss_train += discriminator.train_on_batch(X_batch_fake, y_train_fake)

        # train the GAN (i.e. train the generator while keeping discriminator fixed)
        set_to_trainable(discriminator, False)
        # set the target of fake images to 1 because we want the loss between fake images and 1 (l
abel for real image) to be minimized
        # i.e. we want the fake images to be real
        gen_loss_train = gan.train_on_batch(random_vectors, y_train_real)

        # keep noticed by the training process
        if (i+1) % 50 == 0:
          print('This is the {}/{} of epoch {}.'.format(i+1, num_batches, e+1))

      # compute loss on validation set
      X_val_real = X_valid_real[np.random.choice(len(X_valid_real), size=val_size, replace=False)]

      random_vectors = generate_random_vectors(val_size, input_size)
      X_val_fake = generator.predict_on_batch(random_vectors)

      dis_loss_val = discriminator.test_on_batch(X_val_real, y_val_real)
      dis_loss_val += discriminator.test_on_batch(X_val_fake, y_val_fake)
      gen_loss_val = gan.test_on_batch(random_vectors, y_val_real)

      print("Epoch: {}/{} Discriminator Loss: {:.4f}  Generator Loss: {:.4f}".format(e+1, epochs,
dis_loss_val, gen_loss_val))

      tb_dis.on_epoch_end(e, named_logs(discriminator, [dis_loss_train, dis_loss_val]))
      tb_gan.on_epoch_end(e, named_logs(gan, [gen_loss_train, gen_loss_val]))

      if e==0 or (e+1) % 5 == 0:
```

```
        imgs = generator.predict_on_batch(seed)
        show_images(imgs)

    end = time.time()
    print(end-start)
    return generator
```

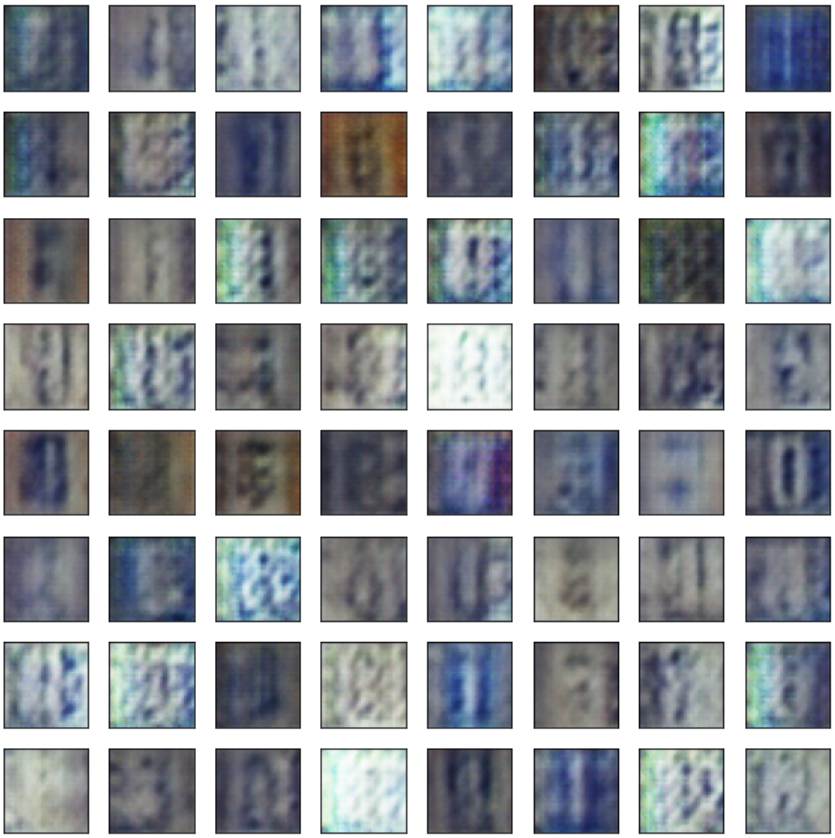
In [17]:
```
train(generator_lr=0.0001,
      discriminator_lr=0.001,
      gen_beta_1=0.5,
      dis_beta_1=0.5,
      leaky_alpha=0.2,
      smooth=0.1,
      input_size=100,
      epochs=200,
      batch_size=128,
      val_size=36)
```

```
This is the 50/572 of epoch 1.
This is the 100/572 of epoch 1.
This is the 150/572 of epoch 1.
This is the 200/572 of epoch 1.
This is the 250/572 of epoch 1.
This is the 300/572 of epoch 1.
This is the 350/572 of epoch 1.
This is the 400/572 of epoch 1.
This is the 450/572 of epoch 1.
This is the 500/572 of epoch 1.
This is the 550/572 of epoch 1.
Epoch: 1/200 Discriminator Loss: 1.1818  Generator Loss: 2.2662
```



```
This is the 50/572 of epoch 2.
This is the 100/572 of epoch 2.
This is the 150/572 of epoch 2.
This is the 200/572 of epoch 2.
This is the 250/572 of epoch 2.
This is the 300/572 of epoch 2.
This is the 350/572 of epoch 2.
This is the 400/572 of epoch 2.
This is the 450/572 of epoch 2.
This is the 500/572 of epoch 2.
This is the 550/572 of epoch 2.
Epoch: 2/200 Discriminator Loss: 1.0918  Generator Loss: 0.9048
This is the 50/572 of epoch 3.
This is the 100/572 of epoch 3.
This is the 150/572 of epoch 3.
This is the 200/572 of epoch 3.
This is the 250/572 of epoch 3.
This is the 300/572 of epoch 3.
This is the 350/572 of epoch 3.
This is the 400/572 of epoch 3.
This is the 450/572 of epoch 3.
This is the 500/572 of epoch 3.
This is the 550/572 of epoch 3.
Epoch: 3/200 Discriminator Loss: 1.0724  Generator Loss: 0.9790
This is the 50/572 of epoch 4.
This is the 100/572 of epoch 4.
This is the 150/572 of epoch 4.
This is the 200/572 of epoch 4.
This is the 250/572 of epoch 4.
This is the 300/572 of epoch 4.
This is the 350/572 of epoch 4.
This is the 400/572 of epoch 4.
This is the 450/572 of epoch 4.
This is the 500/572 of epoch 4.
This is the 550/572 of epoch 4.
Epoch: 4/200 Discriminator Loss: 1.6541  Generator Loss: 0.3343
This is the 50/572 of epoch 5.
This is the 100/572 of epoch 5.
This is the 150/572 of epoch 5.
This is the 200/572 of epoch 5.
This is the 250/572 of epoch 5.
This is the 300/572 of epoch 5.
This is the 350/572 of epoch 5.
This is the 400/572 of epoch 5.
This is the 450/572 of epoch 5.
This is the 500/572 of epoch 5.
This is the 550/572 of epoch 5.
Epoch: 5/200 Discriminator Loss: 1.4448  Generator Loss: 0.6570
```

```
This is the 50/572 of epoch 6.
This is the 100/572 of epoch 6.
This is the 150/572 of epoch 6.
This is the 200/572 of epoch 6.
This is the 250/572 of epoch 6.
This is the 300/572 of epoch 6.
This is the 350/572 of epoch 6.
This is the 400/572 of epoch 6.
This is the 450/572 of epoch 6.
This is the 500/572 of epoch 6.
This is the 550/572 of epoch 6.
Epoch: 6/200 Discriminator Loss: 1.3173  Generator Loss: 1.4898
This is the 50/572 of epoch 7.
This is the 100/572 of epoch 7.
This is the 150/572 of epoch 7.
This is the 200/572 of epoch 7.
This is the 250/572 of epoch 7.
This is the 300/572 of epoch 7.
This is the 350/572 of epoch 7.
This is the 400/572 of epoch 7.
This is the 450/572 of epoch 7.
This is the 500/572 of epoch 7.
This is the 550/572 of epoch 7.
Epoch: 7/200 Discriminator Loss: 1.2405  Generator Loss: 1.3723
This is the 50/572 of epoch 8.
This is the 100/572 of epoch 8.
This is the 150/572 of epoch 8.
This is the 200/572 of epoch 8.
This is the 250/572 of epoch 8.
This is the 300/572 of epoch 8.
This is the 350/572 of epoch 8.
This is the 400/572 of epoch 8.
This is the 450/572 of epoch 8.
This is the 500/572 of epoch 8.
This is the 550/572 of epoch 8.
Epoch: 8/200 Discriminator Loss: 1.1848  Generator Loss: 0.8504
This is the 50/572 of epoch 9.
This is the 100/572 of epoch 9.
This is the 150/572 of epoch 9.
This is the 200/572 of epoch 9.
This is the 250/572 of epoch 9.
This is the 300/572 of epoch 9.
This is the 350/572 of epoch 9.
This is the 400/572 of epoch 9.
This is the 450/572 of epoch 9.
This is the 500/572 of epoch 9.
This is the 550/572 of epoch 9.
Epoch: 9/200 Discriminator Loss: 1.4371  Generator Loss: 1.2343
This is the 50/572 of epoch 10.
This is the 100/572 of epoch 10.
This is the 150/572 of epoch 10.
This is the 200/572 of epoch 10.
This is the 250/572 of epoch 10.
This is the 300/572 of epoch 10.
This is the 350/572 of epoch 10.
This is the 400/572 of epoch 10.
This is the 450/572 of epoch 10.
This is the 500/572 of epoch 10.
This is the 550/572 of epoch 10.
Epoch: 10/200 Discriminator Loss: 1.5515  Generator Loss: 0.5687
```



```
This is the 50/572 of epoch 46.
This is the 100/572 of epoch 46.
This is the 150/572 of epoch 46.
This is the 200/572 of epoch 46.
This is the 250/572 of epoch 46.
This is the 300/572 of epoch 46.
This is the 350/572 of epoch 46.
This is the 400/572 of epoch 46.
This is the 450/572 of epoch 46.
This is the 500/572 of epoch 46.
This is the 550/572 of epoch 46.
Epoch: 46/200 Discriminator Loss: 1.5772  Generator Loss: 2.1724
This is the 50/572 of epoch 47.
This is the 100/572 of epoch 47.
This is the 150/572 of epoch 47.
This is the 200/572 of epoch 47.
This is the 250/572 of epoch 47.
This is the 300/572 of epoch 47.
This is the 350/572 of epoch 47.
This is the 400/572 of epoch 47.
This is the 450/572 of epoch 47.
This is the 500/572 of epoch 47.
This is the 550/572 of epoch 47.
Epoch: 47/200 Discriminator Loss: 1.3849  Generator Loss: 2.0000
This is the 50/572 of epoch 48.
This is the 100/572 of epoch 48.
This is the 150/572 of epoch 48.
This is the 200/572 of epoch 48.
This is the 250/572 of epoch 48.
This is the 300/572 of epoch 48.
This is the 350/572 of epoch 48.
This is the 400/572 of epoch 48.
This is the 450/572 of epoch 48.
This is the 500/572 of epoch 48.
This is the 550/572 of epoch 48.
Epoch: 48/200 Discriminator Loss: 1.4218  Generator Loss: 2.3933
This is the 50/572 of epoch 49.
This is the 100/572 of epoch 49.
This is the 150/572 of epoch 49.
This is the 200/572 of epoch 49.
This is the 250/572 of epoch 49.
This is the 300/572 of epoch 49.
This is the 350/572 of epoch 49.
This is the 400/572 of epoch 49.
This is the 450/572 of epoch 49.
This is the 500/572 of epoch 49.
This is the 550/572 of epoch 49.
Epoch: 49/200 Discriminator Loss: 1.4778  Generator Loss: 1.6668
This is the 50/572 of epoch 50.
This is the 100/572 of epoch 50.
This is the 150/572 of epoch 50.
This is the 200/572 of epoch 50.
This is the 250/572 of epoch 50.
This is the 300/572 of epoch 50.
This is the 350/572 of epoch 50.
This is the 400/572 of epoch 50.
This is the 450/572 of epoch 50.
This is the 500/572 of epoch 50.
This is the 550/572 of epoch 50.
Epoch: 50/200 Discriminator Loss: 1.4739  Generator Loss: 2.1755
```

```
This is the 50/572 of epoch 96.
This is the 100/572 of epoch 96.
This is the 150/572 of epoch 96.
This is the 200/572 of epoch 96.
This is the 250/572 of epoch 96.
This is the 300/572 of epoch 96.
This is the 350/572 of epoch 96.
This is the 400/572 of epoch 96.
This is the 450/572 of epoch 96.
This is the 500/572 of epoch 96.
This is the 550/572 of epoch 96.
Epoch: 96/200 Discriminator Loss: 1.5604  Generator Loss: 2.1650
This is the 50/572 of epoch 97.
This is the 100/572 of epoch 97.
This is the 150/572 of epoch 97.
This is the 200/572 of epoch 97.
This is the 250/572 of epoch 97.
This is the 300/572 of epoch 97.
This is the 350/572 of epoch 97.
This is the 400/572 of epoch 97.
This is the 450/572 of epoch 97.
This is the 500/572 of epoch 97.
This is the 550/572 of epoch 97.
Epoch: 97/200 Discriminator Loss: 1.9035  Generator Loss: 2.6662
This is the 50/572 of epoch 98.
This is the 100/572 of epoch 98.
This is the 150/572 of epoch 98.
This is the 200/572 of epoch 98.
This is the 250/572 of epoch 98.
This is the 300/572 of epoch 98.
This is the 350/572 of epoch 98.
This is the 400/572 of epoch 98.
This is the 450/572 of epoch 98.
This is the 500/572 of epoch 98.
This is the 550/572 of epoch 98.
Epoch: 98/200 Discriminator Loss: 1.4342  Generator Loss: 1.9297
This is the 50/572 of epoch 99.
This is the 100/572 of epoch 99.
This is the 150/572 of epoch 99.
This is the 200/572 of epoch 99.
This is the 250/572 of epoch 99.
This is the 300/572 of epoch 99.
This is the 350/572 of epoch 99.
This is the 400/572 of epoch 99.
This is the 450/572 of epoch 99.
This is the 500/572 of epoch 99.
This is the 550/572 of epoch 99.
Epoch: 99/200 Discriminator Loss: 1.1888  Generator Loss: 1.0061
This is the 50/572 of epoch 100.
This is the 100/572 of epoch 100.
This is the 150/572 of epoch 100.
This is the 200/572 of epoch 100.
This is the 250/572 of epoch 100.
This is the 300/572 of epoch 100.
This is the 350/572 of epoch 100.
This is the 400/572 of epoch 100.
This is the 450/572 of epoch 100.
This is the 500/572 of epoch 100.
This is the 550/572 of epoch 100.
Epoch: 100/200 Discriminator Loss: 1.6090  Generator Loss: 2.0289
```

```
This is the 50/572 of epoch 146.
This is the 100/572 of epoch 146.
This is the 150/572 of epoch 146.
This is the 200/572 of epoch 146.
This is the 250/572 of epoch 146.
This is the 300/572 of epoch 146.
This is the 350/572 of epoch 146.
This is the 400/572 of epoch 146.
This is the 450/572 of epoch 146.
This is the 500/572 of epoch 146.
This is the 550/572 of epoch 146.
Epoch: 146/200 Discriminator Loss: 2.4261  Generator Loss: 2.7198
This is the 50/572 of epoch 147.
This is the 100/572 of epoch 147.
This is the 150/572 of epoch 147.
This is the 200/572 of epoch 147.
This is the 250/572 of epoch 147.
This is the 300/572 of epoch 147.
This is the 350/572 of epoch 147.
This is the 400/572 of epoch 147.
This is the 450/572 of epoch 147.
This is the 500/572 of epoch 147.
This is the 550/572 of epoch 147.
Epoch: 147/200 Discriminator Loss: 1.6256  Generator Loss: 2.2680
This is the 50/572 of epoch 148.
This is the 100/572 of epoch 148.
This is the 150/572 of epoch 148.
This is the 200/572 of epoch 148.
This is the 250/572 of epoch 148.
This is the 300/572 of epoch 148.
This is the 350/572 of epoch 148.
This is the 400/572 of epoch 148.
This is the 450/572 of epoch 148.
This is the 500/572 of epoch 148.
This is the 550/572 of epoch 148.
Epoch: 148/200 Discriminator Loss: 1.4876  Generator Loss: 1.7390
This is the 50/572 of epoch 149.
This is the 100/572 of epoch 149.
This is the 150/572 of epoch 149.
This is the 200/572 of epoch 149.
This is the 250/572 of epoch 149.
This is the 300/572 of epoch 149.
This is the 350/572 of epoch 149.
This is the 400/572 of epoch 149.
This is the 450/572 of epoch 149.
This is the 500/572 of epoch 149.
This is the 550/572 of epoch 149.
Epoch: 149/200 Discriminator Loss: 1.6378  Generator Loss: 1.9026
This is the 50/572 of epoch 150.
This is the 100/572 of epoch 150.
This is the 150/572 of epoch 150.
This is the 200/572 of epoch 150.
This is the 250/572 of epoch 150.
This is the 300/572 of epoch 150.
This is the 350/572 of epoch 150.
This is the 400/572 of epoch 150.
This is the 450/572 of epoch 150.
This is the 500/572 of epoch 150.
This is the 550/572 of epoch 150.
Epoch: 150/200 Discriminator Loss: 1.5081  Generator Loss: 1.7136
```

```
This is the 50/572 of epoch 196.
This is the 100/572 of epoch 196.
This is the 150/572 of epoch 196.
This is the 200/572 of epoch 196.
This is the 250/572 of epoch 196.
This is the 300/572 of epoch 196.
This is the 350/572 of epoch 196.
This is the 400/572 of epoch 196.
This is the 450/572 of epoch 196.
This is the 500/572 of epoch 196.
This is the 550/572 of epoch 196.
Epoch: 196/200 Discriminator Loss: 1.7758  Generator Loss: 2.0924
This is the 50/572 of epoch 197.
This is the 100/572 of epoch 197.
This is the 150/572 of epoch 197.
This is the 200/572 of epoch 197.
This is the 250/572 of epoch 197.
This is the 300/572 of epoch 197.
This is the 350/572 of epoch 197.
This is the 400/572 of epoch 197.
This is the 450/572 of epoch 197.
This is the 500/572 of epoch 197.
This is the 550/572 of epoch 197.
Epoch: 197/200 Discriminator Loss: 1.8074  Generator Loss: 2.2938
This is the 50/572 of epoch 198.
This is the 100/572 of epoch 198.
This is the 150/572 of epoch 198.
This is the 200/572 of epoch 198.
This is the 250/572 of epoch 198.
This is the 300/572 of epoch 198.
This is the 350/572 of epoch 198.
This is the 400/572 of epoch 198.
This is the 450/572 of epoch 198.
This is the 500/572 of epoch 198.
This is the 550/572 of epoch 198.
Epoch: 198/200 Discriminator Loss: 1.8307  Generator Loss: 1.8116
This is the 50/572 of epoch 199.
This is the 100/572 of epoch 199.
This is the 150/572 of epoch 199.
This is the 200/572 of epoch 199.
This is the 250/572 of epoch 199.
This is the 300/572 of epoch 199.
This is the 350/572 of epoch 199.
This is the 400/572 of epoch 199.
This is the 450/572 of epoch 199.
This is the 500/572 of epoch 199.
This is the 550/572 of epoch 199.
Epoch: 199/200 Discriminator Loss: 2.2312  Generator Loss: 2.7710
This is the 50/572 of epoch 200.
This is the 100/572 of epoch 200.
This is the 150/572 of epoch 200.
This is the 200/572 of epoch 200.
This is the 250/572 of epoch 200.
This is the 300/572 of epoch 200.
This is the 350/572 of epoch 200.
This is the 400/572 of epoch 200.
This is the 450/572 of epoch 200.
This is the 500/572 of epoch 200.
This is the 550/572 of epoch 200.
Epoch: 200/200 Discriminator Loss: 1.4912  Generator Loss: 1.7491
```

```
12572.459484577179
```

Out[17]: `<tensorflow.python.keras.engine.sequential.Sequential at 0x7ff5f0185240>`

In [18]:
```
%tensorboard --logdir=./logs_SVHN
```

epoch_training_loss

epoch_validation_loss