In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
from tensorflow.keras import layers
import time
import tensorflow as tf
from IPython import display
import datetime
import pickle as pkl
from scipy.io import loadmat
from tensorflow.keras.utils import get_file
```

In [ ]:
```
%load_ext tensorboard
!rm -rf ./logs/wgan_gradient_tape/
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

In [ ]:
```
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_32x32.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 [==============================] - 6s 0us/step
Downloading data from http://ufldl.stanford.edu/housenumbers/test_32x32.mat
64282624/64275384 [==============================] - 3s 0us/step
(32, 32, 3, 73257)
```

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

In [ ]:
```
# glimpse of the real images
sample_images = X_train[np.random.choice(len(X_train_real), 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()
```

In [ ]:
```
batch_size = 64
n_critic = 1
#n_critic = 1
num_batches = len(X_train_real)//batch_size
```

In [ ]:
```
def build_generator(input_size, leaky_alpha=0.2):
  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_bi
as=False), # 8,8,256
      layers.BatchNormalization(),
      layers.LeakyReLU(alpha=leaky_alpha),
      layers.Conv2DTranspose(128, kernel_size=5, strides=2, padding='same', use_bi
as=False), # 16,16,128
      layers.BatchNormalization(),
      layers.LeakyReLU(alpha=leaky_alpha),
      layers.Conv2DTranspose(3, kernel_size=5, strides=2, padding='same', use_bias
=False),   # 32,32,3
      layers.Activation('tanh')
  ])

  return model
```

In [ ]:
```
generator = build_generator(100, 0.2)
generator.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
_________________________________________________________________
```

In [ ]:
```
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)
  ])
  return model
```

In [ ]:
```
discriminator = build_discriminator(0.2)
discriminator.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
_________________________________________________________________
```

In [ ]:
```
generator_optimizer = tf.keras.optimizers.RMSprop(learning_rate=5e-5)
discriminator_optimizer = tf.keras.optimizers.RMSprop(learning_rate=5e-5)
```

In [ ]:
```
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()
```

In [ ]:
```
EPOCHS = 200
np.random.seed(2020)
seed = np.random.normal(loc=0, scale=1, size=(64, 100))

current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
gen_log_dir = 'logs/wgan_gradient_tape/' + current_time + '/gen'
disc_log_dir = 'logs/wgan_gradient_tape/' + current_time + '/disc'
gen_summary_writer = tf.summary.create_file_writer(gen_log_dir)
disc_summary_writer = tf.summary.create_file_writer(disc_log_dir)
```

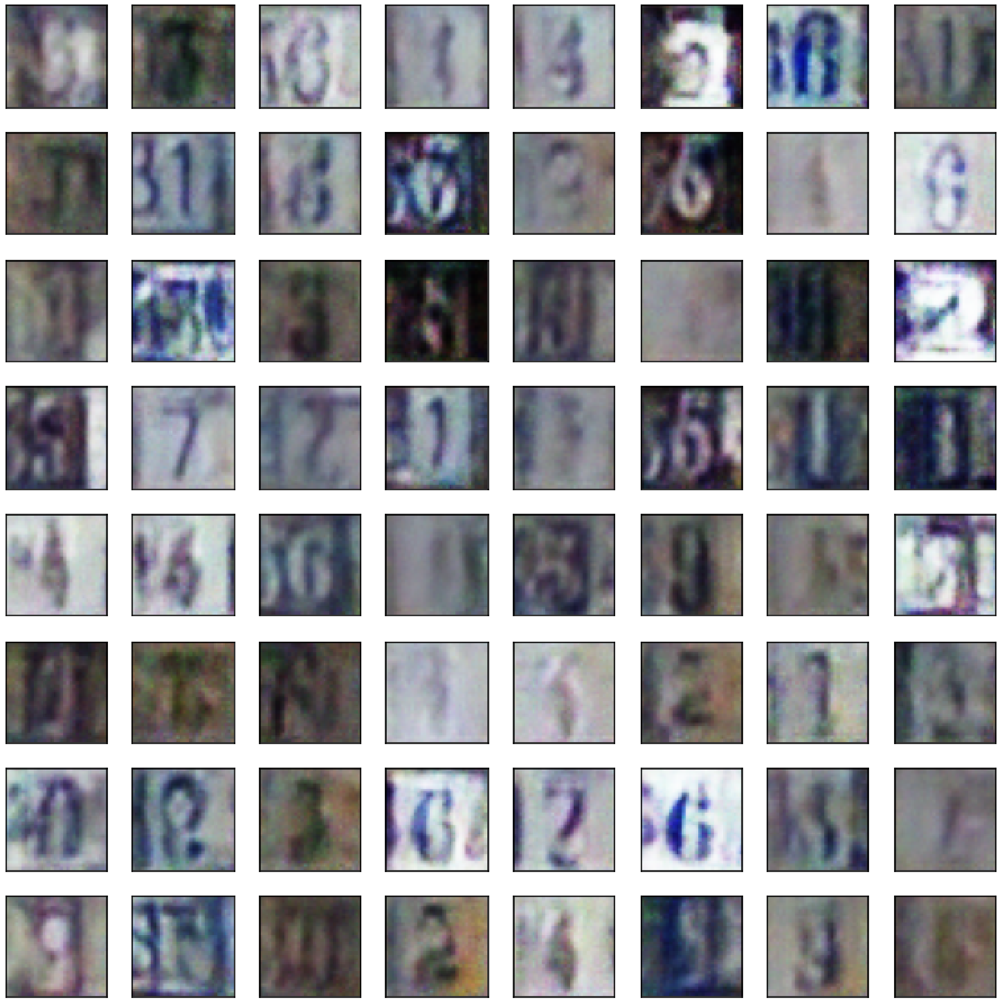
In [ ]:
```
@tf.function
def generator_gradient():
    # step 9
    noise = tf.random.normal([batch_size, 100])
    with tf.GradientTape() as gen_tape:
        # step 10
        generated_images = generator(noise, training=True)
        gen_loss_val = -tf.reduce_mean(discriminator(generated_images, training=Tr
ue))
    # step 10
    gradients_of_generator = gen_tape.gradient(gen_loss_val, generator.trainable_v
ariables)
    # step 11
    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trai
nable_variables))
    return gen_loss_val
```

In [ ]:
```
@tf.function
def discriminator_gradient():
    # step 3
    noise = tf.random.normal([batch_size, 100])
    # step 4
    images = X_train_real[np.random.choice(len(X_train_real), size=batch_size, rep
lace=False)]

    with tf.GradientTape() as disc_tape:
        # step 5
        generated_images = generator(noise, training=True)
        dis_loss_val = -tf.reduce_mean(discriminator(images, training=True)) + tf.
reduce_mean(discriminator(generated_images, training=True))
    # step 5
    gradients_of_discriminator = disc_tape.gradient(dis_loss_val, discriminator.tr
ainable_variables)
    # step 6
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discri
minator.trainable_variables))

    # step 7, clip
    for p in discriminator.trainable_variables:
        p.assign(tf.clip_by_value(p, -0.01, 0.01))

    return dis_loss_val
```

In [ ]:
```
# the number of step corresponds to Algorithm 1 in "Wasserstein GAN"
def train(epochs):

    start = time.time()
    # step 1
    for e in range(epochs):
        start2 = time.time()
        for i in range(num_batches):
            # step 2
            for _ in range(n_critic):
              dis_loss_val = discriminator_gradient()
            # step 8
            gen_loss_val = generator_gradient()
            #if (i+1) % 50 == 0:
                #print('This is the {}/{} of epoch {}.'.format(i+1, num_batches, e
+1))

        with gen_summary_writer.as_default():
            tf.summary.scalar('loss', gen_loss_val, step=e)
        with disc_summary_writer.as_default():
            tf.summary.scalar('loss', dis_loss_val, step=e)

        print("Epoch: {}/{} Discriminator Loss: {:.4f}  Generator Loss: {:.4f}".fo
rmat(e+1, epochs, dis_loss_val, gen_loss_val))
        if e==0 or e==4 or e==9 or e==49 or e==99 or e==149 or e==199:
            imgs = generator.predict_on_batch(seed)
            show_images(imgs)
        print ('Time for epoch {} is {} sec'.format(e + 1, time.time()-start2))

    # step 12
    end = time.time()
    print(end-start)
```

In [ ]:
```
train(200)
```

Epoch: 1/200 Discriminator Loss: -0.0149 Generator Loss: -0.0062



Time for epoch 1 is 61.050485134124756 sec
Epoch: 2/200 Discriminator Loss: -0.0136 Generator Loss: -0.0073
Time for epoch 2 is 56.70659685134888 sec
Epoch: 3/200 Discriminator Loss: -0.0137 Generator Loss: -0.0053
Time for epoch 3 is 57.16050457954407 sec
Epoch: 4/200 Discriminator Loss: -0.0162 Generator Loss: -0.0072
Time for epoch 4 is 57.761735677719116 sec
Epoch: 5/200 Discriminator Loss: -0.0131 Generator Loss: -0.0059

```
Time for epoch 5 is 59.584129095077515 sec
Epoch: 6/200 Discriminator Loss: -0.0137  Generator Loss: -0.0020
Time for epoch 6 is 57.57205772399902 sec
Epoch: 7/200 Discriminator Loss: -0.0112  Generator Loss: -0.0067
Time for epoch 7 is 57.804524183273315 sec
Epoch: 8/200 Discriminator Loss: -0.0135  Generator Loss: -0.0077
Time for epoch 8 is 57.805200815200806 sec
Epoch: 9/200 Discriminator Loss: -0.0126  Generator Loss: -0.0069
Time for epoch 9 is 57.811567068099976 sec
Epoch: 10/200 Discriminator Loss: -0.0120  Generator Loss: -0.0083
```

```
Epoch: 42/200 Discriminator Loss: -0.0096  Generator Loss: -0.0099
Time for epoch 42 is 57.61907720565796 sec
Epoch: 43/200 Discriminator Loss: -0.0096  Generator Loss: -0.0091
Time for epoch 43 is 57.65308117866516 sec
Epoch: 44/200 Discriminator Loss: -0.0115  Generator Loss: -0.0115
Time for epoch 44 is 57.54136562347412 sec
Epoch: 45/200 Discriminator Loss: -0.0093  Generator Loss: -0.0074
Time for epoch 45 is 57.62323474884033 sec
Epoch: 46/200 Discriminator Loss: -0.0096  Generator Loss: -0.0088
Time for epoch 46 is 57.357157945632935 sec
Epoch: 47/200 Discriminator Loss: -0.0099  Generator Loss: -0.0100
Time for epoch 47 is 57.439390659332275 sec
Epoch: 48/200 Discriminator Loss: -0.0100  Generator Loss: -0.0094
Time for epoch 48 is 57.452401876449585 sec
Epoch: 49/200 Discriminator Loss: -0.0104  Generator Loss: -0.0098
Time for epoch 49 is 57.338640213012695 sec
Epoch: 50/200 Discriminator Loss: -0.0083  Generator Loss: -0.0078
```

```
Time for epoch 50 is 59.48170065879822 sec
Epoch: 51/200 Discriminator Loss: -0.0074  Generator Loss: -0.0101
Time for epoch 51 is 57.458508014678955 sec
Epoch: 52/200 Discriminator Loss: -0.0102  Generator Loss: -0.0075
Time for epoch 52 is 57.83629894256592 sec
Epoch: 54/200 Discriminator Loss: -0.0105  Generator Loss: -0.0114
Time for epoch 54 is 57.316943407058716 sec
```

In [ ]:

```
%tensorboard --logdir logs/wgan_gradient_tape
```

loss

loss
tag: loss