"""

Some codes from https://github.com/Newmu/dcgan\_code

"""

from \_\_future\_\_ import division

import math

import json

import random

import pprint

import scipy.misc

import numpy as np

from time import gmtime, strftime

from six.moves import xrange

import tensorflow as tf

import tensorflow.contrib.slim as slim

pp = pprint.PrettyPrinter()

get\_stddev = lambda x, k\_h, k\_w: 1/math.sqrt(k\_w\*k\_h\*x.get\_shape()[-1])

def show\_all\_variables():

model\_vars = tf.trainable\_variables()

slim.model\_analyzer.analyze\_vars(model\_vars, print\_info=True)

def get\_image(image\_path, input\_height, input\_width,

resize\_height=64, resize\_width=64,

crop=True, grayscale=False):

image = imread(image\_path, grayscale)

return transform(image, input\_height, input\_width,

resize\_height, resize\_width, crop)

def save\_images(images, size, image\_path):

return imsave(inverse\_transform(images), size, image\_path)

def imread(path, grayscale = False):

if (grayscale):

return scipy.misc.imread(path, flatten = True).astype(np.float)

else:

return scipy.misc.imread(path).astype(np.float)

def merge\_images(images, size):

return inverse\_transform(images)

def merge(images, size):

h, w = images.shape[1], images.shape[2]

if (images.shape[3] in (3,4)):

c = images.shape[3]

img = np.zeros((h \* size[0], w \* size[1], c))

for idx, image in enumerate(images):

i = idx % size[1]

j = idx // size[1]

img[j \* h:j \* h + h, i \* w:i \* w + w, :] = image

return img

elif images.shape[3]==1:

img = np.zeros((h \* size[0], w \* size[1]))

for idx, image in enumerate(images):

i = idx % size[1]

j = idx // size[1]

img[j \* h:j \* h + h, i \* w:i \* w + w] = image[:,:,0]

return img

else:

raise ValueError('in merge(images,size) images parameter '

'must have dimensions: HxW or HxWx3 or HxWx4')

def imsave(images, size, path):

image = np.squeeze(merge(images, size))

return scipy.misc.imsave(path, image)

def center\_crop(x, crop\_h, crop\_w,

resize\_h=64, resize\_w=64):

if crop\_w is None:

crop\_w = crop\_h

h, w = x.shape[:2]

j = int(round((h - crop\_h)/2.))

i = int(round((w - crop\_w)/2.))

return scipy.misc.imresize(

x[j:j+crop\_h, i:i+crop\_w], [resize\_h, resize\_w])

def transform(image, input\_height, input\_width,

resize\_height=64, resize\_width=64, crop=True):

if crop:

cropped\_image = center\_crop(

image, input\_height, input\_width,

resize\_height, resize\_width)

else:

cropped\_image = scipy.misc.imresize(image, [resize\_height, resize\_width])

return np.array(cropped\_image)/127.5 - 1.

def inverse\_transform(images):

return (images+1.)/2.

def to\_json(output\_path, \*layers):

with open(output\_path, "w") as layer\_f:

lines = ""

for w, b, bn in layers:

layer\_idx = w.name.split('/')[0].split('h')[1]

B = b.eval()

if "lin/" in w.name:

W = w.eval()

depth = W.shape[1]

else:

W = np.rollaxis(w.eval(), 2, 0)

depth = W.shape[0]

biases = {"sy": 1, "sx": 1, "depth": depth, "w": ['%.2f' % elem for elem in list(B)]}

if bn != None:

gamma = bn.gamma.eval()

beta = bn.beta.eval()

gamma = {"sy": 1, "sx": 1, "depth": depth, "w": ['%.2f' % elem for elem in list(gamma)]}

beta = {"sy": 1, "sx": 1, "depth": depth, "w": ['%.2f' % elem for elem in list(beta)]}

else:

gamma = {"sy": 1, "sx": 1, "depth": 0, "w": []}

beta = {"sy": 1, "sx": 1, "depth": 0, "w": []}

if "lin/" in w.name:

fs = []

for w in W.T:

fs.append({"sy": 1, "sx": 1, "depth": W.shape[0], "w": ['%.2f' % elem for elem in list(w)]})

lines += """

var layer\_%s = {

"layer\_type": "fc",

"sy": 1, "sx": 1,

"out\_sx": 1, "out\_sy": 1,

"stride": 1, "pad": 0,

"out\_depth": %s, "in\_depth": %s,

"biases": %s,

"gamma": %s,

"beta": %s,

"filters": %s

};""" % (layer\_idx.split('\_')[0], W.shape[1], W.shape[0], biases, gamma, beta, fs)

else:

fs = []

for w\_ in W:

fs.append({"sy": 5, "sx": 5, "depth": W.shape[3], "w": ['%.2f' % elem for elem in list(w\_.flatten())]})

lines += """

var layer\_%s = {

"layer\_type": "deconv",

"sy": 5, "sx": 5,

"out\_sx": %s, "out\_sy": %s,

"stride": 2, "pad": 1,

"out\_depth": %s, "in\_depth": %s,

"biases": %s,

"gamma": %s,

"beta": %s,

"filters": %s

};""" % (layer\_idx, 2\*\*(int(layer\_idx)+2), 2\*\*(int(layer\_idx)+2),

W.shape[0], W.shape[3], biases, gamma, beta, fs)

layer\_f.write(" ".join(lines.replace("'","").split()))

def make\_gif(images, fname, duration=2, true\_image=False):

import moviepy.editor as mpy

def make\_frame(t):

try:

x = images[int(len(images)/duration\*t)]

except:

x = images[-1]

if true\_image:

return x.astype(np.uint8)

else:

return ((x+1)/2\*255).astype(np.uint8)

clip = mpy.VideoClip(make\_frame, duration=duration)

clip.write\_gif(fname, fps = len(images) / duration)

def visualize(sess, dcgan, config, option):

image\_frame\_dim = int(math.ceil(config.batch\_size\*\*.5))

if option == 0:

z\_sample = np.random.uniform(-0.5, 0.5, size=(config.batch\_size, dcgan.z\_dim))

samples = sess.run(dcgan.sampler, feed\_dict={dcgan.z: z\_sample})

save\_images(samples, [image\_frame\_dim, image\_frame\_dim], './samples/test\_%s.png' % strftime("%Y%m%d%H%M%S", gmtime()))

elif option == 1:

values = np.arange(0, 1, 1./config.batch\_size)

for idx in xrange(100):

print(" [\*] %d" % idx)

z\_sample = np.zeros([config.batch\_size, dcgan.z\_dim])

for kdx, z in enumerate(z\_sample):

z[idx] = values[kdx]

if config.dataset == "mnist":

y = np.random.choice(10, config.batch\_size)

y\_one\_hot = np.zeros((config.batch\_size, 10))

y\_one\_hot[np.arange(config.batch\_size), y] = 1

samples = sess.run(dcgan.sampler, feed\_dict={dcgan.z: z\_sample, dcgan.y: y\_one\_hot})

else:

samples = sess.run(dcgan.sampler, feed\_dict={dcgan.z: z\_sample})

save\_images(samples, [image\_frame\_dim, image\_frame\_dim], './samples/test\_arange\_%s.png' % (idx))

elif option == 2:

values = np.arange(0, 1, 1./config.batch\_size)

for idx in [random.randint(0, 99) for \_ in xrange(100)]:

print(" [\*] %d" % idx)

z = np.random.uniform(-0.2, 0.2, size=(dcgan.z\_dim))

z\_sample = np.tile(z, (config.batch\_size, 1))

#z\_sample = np.zeros([config.batch\_size, dcgan.z\_dim])

for kdx, z in enumerate(z\_sample):

z[idx] = values[kdx]

if config.dataset == "mnist":

y = np.random.choice(10, config.batch\_size)

y\_one\_hot = np.zeros((config.batch\_size, 10))

y\_one\_hot[np.arange(config.batch\_size), y] = 1

samples = sess.run(dcgan.sampler, feed\_dict={dcgan.z: z\_sample, dcgan.y: y\_one\_hot})

else:

samples = sess.run(dcgan.sampler, feed\_dict={dcgan.z: z\_sample})

try:

make\_gif(samples, './samples/test\_gif\_%s.gif' % (idx))

except:

save\_images(samples, [image\_frame\_dim, image\_frame\_dim], './samples/test\_%s.png' % strftime("%Y%m%d%H%M%S", gmtime()))

elif option == 3:

values = np.arange(0, 1, 1./config.batch\_size)

for idx in xrange(100):

print(" [\*] %d" % idx)

z\_sample = np.zeros([config.batch\_size, dcgan.z\_dim])

for kdx, z in enumerate(z\_sample):

z[idx] = values[kdx]

samples = sess.run(dcgan.sampler, feed\_dict={dcgan.z: z\_sample})

make\_gif(samples, './samples/test\_gif\_%s.gif' % (idx))

elif option == 4:

image\_set = []

values = np.arange(0, 1, 1./config.batch\_size)

for idx in xrange(100):

print(" [\*] %d" % idx)

z\_sample = np.zeros([config.batch\_size, dcgan.z\_dim])

for kdx, z in enumerate(z\_sample): z[idx] = values[kdx]

image\_set.append(sess.run(dcgan.sampler, feed\_dict={dcgan.z: z\_sample}))

make\_gif(image\_set[-1], './samples/test\_gif\_%s.gif' % (idx))

new\_image\_set = [merge(np.array([images[idx] for images in image\_set]), [10, 10]) \

for idx in range(64) + range(63, -1, -1)]

make\_gif(new\_image\_set, './samples/test\_gif\_merged.gif', duration=8)