Zhicheng DDNET+ my GAN= not working

August 16, 2018

1 Zhicheng Training DDNET

In [ ]:
```
from __future__ import print_function
import tensorflow as tf
import matplotlib.pyplot as plt
from scipy.misc import imresize
import numpy as np
from skimage.io import imread
import math
import csv
import random
import time
import scipy.io
from functools import reduce
#import function
import scipy.stats as st
# from skimage.measure import compare_ssim as psnr
# import gauss
from scipy import signal
from scipy import ndimage
##########################################################################################
var_dict= {}
# param = np.load('./test-save.npy', encoding='latin1').item()
##########################################################################################
def gkern(kernlen=21, nsig=3.):
    """Returns a 2D Gaussian kernel array."""

    interval = (2*nsig+1.)/(kernlen)
    x = np.linspace(-nsig-interval/2., nsig+interval/2., kernlen+1)
    kern1d = np.diff(st.norm.cdf(x))
    kernel_raw = np.sqrt(np.outer(kern1d, kern1d))
    kernel = kernel_raw/kernel_raw.sum()
    return kernel

def ssim(img1, img2, cs_map=False):
    """Return the Structural Similarity Map corresponding to input images img1
    and img2 (images are assumed to be uint8)
```

```
    This function attempts to mimic precisely the functionality of ssim.m a
    MATLAB provided by the author's of SSIM
    https://ece.uwaterloo.ca/~z70wang/research/ssim/ssim_index.m
    """
    img1 = img1.astype(np.float64)
    img2 = img2.astype(np.float64)
    size = 11
    sigma = 1.5
    window = gkern(size,nsig = sigma) #gauss.fspecial_gauss(size, sigma)
    K1 = 0.01
    K2 = 0.03
    L = 4000  # bitdepth of image
    C1 = (K1 * L) ** 2
    C2 = (K2 * L) ** 2
    mu1 = signal.fftconvolve(window, img1, mode='valid')
    mu2 = signal.fftconvolve(window, img2, mode='valid')
    mu1_sq = mu1 * mu1
    mu2_sq = mu2 * mu2
    mu1_mu2 = mu1 * mu2
    sigma1_sq = signal.fftconvolve(window, img1 * img1, mode='valid') - mu1_sq
    sigma2_sq = signal.fftconvolve(window, img2 * img2, mode='valid') - mu2_sq
    sigma12 = signal.fftconvolve(window, img1 * img2, mode='valid') - mu1_mu2
    if cs_map:
        return (((2 * mu1_mu2 + C1) * (2 * sigma12 + C2)) / ((mu1_sq + mu2_sq + C1) *
                                                              (sigma1_sq + sigma2_sq +
                (2.0 * sigma12 + C2) / (sigma1_sq + sigma2_sq + C2))
    else:
        return np.mean(((2 * mu1_mu2 + C1) * (2 * sigma12 + C2)) / ((mu1_sq + mu2_sq +
                                                                     (sigma1_sq + sigma2_sq + C

def Measure_Quality(res_prev, res, QualMeasOpts,N):
    if QualMeasOpts == 'RMSE':
        # N = len(res_prev.ravel())
        diff = res_prev - res
        return np.sqrt(sum(diff.ravel() ** 2) / N)

    if QualMeasOpts == 'CC':
        return np.corrcoef(res_prev.ravel(), res.ravel())

    if QualMeasOpts == 'MSSIM':
        # N = len(res_prev.ravel())
        # NOTE: May not be necessary in python

        res_prev = res_prev.ravel()
        res = res.ravel()

        # Compute the mean pixel values of the two images
```

```
        mean_res_p = res_prev.mean()
        mean_res = res.mean()
        if mean_res==0 and mean_res_p==0:
            raise ValueError('Initialising with 0 matrix not valid')
        # Luminance Comparison

        K1 = 0.01  # K1 is a small constant <<1
        d = max(res_prev) - min(res_prev)  # dynamic range of the pixel values
        l = ((2 * mean_res * mean_res_p) + (K1 * d) ** 2) / ((mean_res_p ** 2)
                                                             + (mean_res ** 2) + K1 * d

        # Contrast comparison

        K2 = 0.02
        sres_p = res_prev.std()
        sres = res.std()

        c = ((2 * sres_p * sres) + (K2 * d) ** 2) / ((sres_p ** 2) + (sres ** 2) + K2 *

        # Structure comparison
        diffres_p = res_prev - mean_res_p
        diffres = res - mean_res
        delta = (1 / (N - 1)) * sum(diffres_p * diffres)
        s = (delta + (((K2 * d) ** 2)) / 2) / ((sres_p * sres) + ((K2 * d ** 2) / 2))

        return (1 / N) * l * c * s
    if QualMeasOpts=='UQI':
        res = res.ravel()
        res_prev = res_prev.ravel()
        # N=len(res_prev)

        # Mean
        mean_res_p = np.mean(res_prev,dtype=np.float32)
        mean_res = np.mean(res, dtype=np.float32)

        # Variance
        varres_p = np.var(res_prev)
        varres = np.var(res)
        if mean_res==0 and mean_res_p==0:
            raise ValueError('Initialising with 0 matrix not valid')
        # Covariance
        cova = sum((res)-mean_res)*((res_prev) - mean_res_p)/(N-1)
        front = (2*cova)/(varres+varres_p)
        back = (2*mean_res*mean_res_p)/((mean_res**2)+(mean_res_p**2))

        return sum(front * back)
    # if QualMeasOpts=='SSD':
    #     return np.sum((res_prev[:,:,0:res_prev.shape[2]]-res[:,:,0:res.shape[2]])**2
```

```
def get_var_count():
    count = 0
    for v in list(var_dict.values()):
        count += reduce(lambda x, y: x * y, v.get_shape().as_list())
    return count

def save_npy(sess, npy_path="./DDNet_Param.npy"):
    assert isinstance(sess, tf.Session)
    data_dict = {}
    for (name, idx), var in list(var_dict.items()):
        var_out = sess.run(var)
        if name not in data_dict:
            data_dict[name] = {}
        data_dict[name][idx] = var_out

    np.save(npy_path, data_dict)
    print(("file saved", npy_path))
    return npy_path
def build_unpool(source, kernel_shape):
    input_shape = source.get_shape().as_list()
    return tf.image.resize_images(source, [input_shape[1]*kernel_shape[1], input_shape
def leaky_relu(x, alpha=0.0001):
    return tf.maximum(alpha * x, x)
def HU_Relu(x):
    x = tf.nn.relu(x)
    y = tf.ones_like(x)
    x = tf.minimum(y,x)
    # y = tf.ones_like(x)*(-1000)
    # x = tf.maximum(x,y)
    return x
def conv2d(img,w,b,k):
    conv = tf.nn.conv2d(img, w, strides=[1, k, k, 1], padding='SAME') + b
    return conv
def BN(img):
    batch_mean, batch_var = tf.nn.moments(img, [0, 1, 2], name='moments')
    img = tf.nn.batch_normalization(img, batch_mean, batch_var, 0, 1, 1e-3)
    return img
def conv_layer(img, shape,k, name, reuse=False):
    with tf.variable_scope(name, reuse=reuse):
        W_conv = tf.Variable(tf.truncated_normal(shape, stddev = 0.01, name=name + '_W
        var_dict[(name + '_W', 0)] = W_conv
        b_conv = tf.Variable(tf.constant(0.1, shape = [shape[3]],name=name+'_bias'))
        var_dict[(name + '_bias', 1)] = b_conv
        conv   = conv2d(img, W_conv, b_conv,k)
        return conv
```

```
def deconv_layer(img, Wshape,ouputshape,k, name, reuse=False):
    with tf.variable_scope(name, reuse=reuse):
        W_conv = tf.Variable(tf.truncated_normal(Wshape, stddev = 0.01, name=name + '_W
        var_dict[(name + '_W', 0)] = W_conv
        b_conv = tf.Variable(tf.constant(0.1, shape = [Wshape[2]],name=name+'_bias'))
        var_dict[(name + '_bias', 1)] = b_conv
        conv = tf.nn.conv2d_transpose(img, W_conv,ouputshape, strides=[1, k, k, 1],pado
        return conv

def DenseNet(input,name,growth_rate=16,nb_filter=16,filter_wh = 3,k=1, reuse=False):
    shape = input.get_shape().as_list()
    with tf.variable_scope(name, reuse=reuse):
        conv1_1 = HU_Relu(conv_layer(BN(input), [1, 1, shape[3], nb_filter*4],k, name+
        conv1 = HU_Relu(conv_layer(BN(conv1_1), [filter_wh, filter_wh, nb_filter*4, nb_

        h_concat1 = tf.concat([input,conv1],3)

        conv2_1 = HU_Relu(conv_layer(BN(h_concat1), [1, 1, shape[3] + nb_filter, nb_fil
        conv2 = HU_Relu(conv_layer(BN(conv2_1), [filter_wh, filter_wh, nb_filter*4, nb_

        h_concat2 = tf.concat([input, conv1, conv2], 3)

        conv3_1 = HU_Relu(conv_layer(BN(h_concat2), [1, 1, shape[3] + 2*nb_filter, nb_f
        conv3 = HU_Relu(conv_layer(BN(conv3_1), [filter_wh, filter_wh, nb_filter*4, nb_

        h_concat3 = tf.concat([input, conv1, conv2, conv3], 3)

        conv4_1 = HU_Relu(conv_layer(BN(h_concat3), [1, 1, shape[3] + 3*nb_filter, nb_f
        conv4 = HU_Relu(conv_layer(BN(conv4_1), [filter_wh, filter_wh, nb_filter*4, nb_

    return tf.concat([input, conv1, conv2, conv3, conv4], 3)




if __name__ == '__main__':
    row = 256
    column = 256
    size = row * column
    #
    train_num = 2780
    test_num = 500
    #
    batch = 20

    ##########################################################################################
    train_img120 = np.zeros((train_num, row * column), dtype=np.float32)
    with open('D://Program_zzc//Paper2-revise//Data//Results//Mix_train_FBP_sparse120_1
```

```
    reader = csv.reader(f)
    i = 0
    for ro in reader:
        train_img120[i] = ro
        i += 1

org_train_img = np.zeros((train_num, row * column), dtype=np.float32)
with open('D://Program_zzc//Paper2-revise//Data//Results//Mix_train_origin_2588.csv
    reader = csv.reader(f)
    i = 0
    for ro in reader:
        org_train_img[i] = ro
        i += 1
##############################################################################
test_img120 = np.zeros((test_num, row * column), dtype=np.float32)
with open('D://Program_zzc//Paper2-revise//Data//Results//test_sl_FBP_sparse120_20
    reader = csv.reader(f)
    i = 0
    for ro in reader:
        test_img120[i] = ro
        i += 1

org_img_test = np.zeros((test_num, row * column), dtype=np.float32)
with open('D://Program_zzc//Paper2-revise//Data//Results//test_sl_origin_20.csv') a
    reader = csv.reader(f)
    i = 0
    for ro in reader:
        org_img_test[i] = ro
        i += 1
##############################################################################

with tf.name_scope('inputs'):
    xs = tf.placeholder(tf.float32, [batch, size],name='xs')
    ys = tf.placeholder(tf.float32, [batch, size], name='ys')
    yy = tf.placeholder(tf.float32, [batch, size], name='yy')
    step = tf.placeholder(tf.float32, name='prob')
    keep_prob = tf.placeholder(tf.float32,name='prob')

input_image  = tf.reshape(xs, [batch, row, column, 1])
output_image = tf.reshape(ys, [batch, row, column, 1])
##############################################################################
nb_filter = 16
input_image = (input_image + 1000)/4000
c0 = HU_Relu(conv_layer(BN(input_image), [7, 7, 1, nb_filter],1, 'c0'))
p0 = tf.nn.max_pool(c0, ksize = [1, 3, 3, 1], strides = [1, 2, 2, 1], padding = 'SA

D1 = DenseNet(p0,'d1',growth_rate=16,nb_filter=nb_filter,filter_wh = 5) #128*128*(
```

```
    c1 = HU_Relu(conv_layer(BN(D1), [1, 1, nb_filter+nb_filter*4, nb_filter],1, 'c1'))
    p1 = tf.nn.max_pool(c1, ksize = [1, 3, 3, 1], strides = [1, 2, 2, 1], padding = 'SA
    ##
    D2 = DenseNet(p1, 'd2',growth_rate=16, nb_filter=nb_filter, filter_wh=5)  # 64*64*
    c2 = HU_Relu(conv_layer(BN(D2), [1, 1, nb_filter + nb_filter * 4, nb_filter],1, 'c2
    p2 = tf.nn.max_pool(c2, ksize = [1, 3, 3, 1], strides = [1, 2, 2, 1], padding = 'SA
    ##
    D3 = DenseNet(p2, 'd3',growth_rate=16, nb_filter=nb_filter, filter_wh=5)  # 32*32*
    c3 = HU_Relu(conv_layer(BN(D3), [1, 1, nb_filter + nb_filter * 4, nb_filter],1, 'c3
    p3 = tf.nn.max_pool(c3, ksize = [1, 3, 3, 1], strides = [1, 2, 2, 1], padding = 'SA
    ##
    D4 = DenseNet(p3, 'd4',growth_rate=16, nb_filter=nb_filter, filter_wh=5)  # 16*16*
    c4 = HU_Relu(conv_layer(BN(D4), [1, 1, nb_filter + nb_filter * 4, nb_filter], 1, 'c

    dc4 = HU_Relu(deconv_layer(BN(tf.concat([build_unpool(c4,[1,2,2,1]),c3],3)), [5, 5

    dc4_1 = HU_Relu(deconv_layer(BN(dc4), [1, 1, nb_filter,  (2*nb_filter)],[batch, 32

    dc5 = HU_Relu(deconv_layer(BN(tf.concat([build_unpool(dc4_1, [1, 2, 2, 1]), c2], 3)

    dc5_1 = HU_Relu(deconv_layer(BN(dc5), [1, 1, nb_filter, (2 * nb_filter)], [batch, 6

    dc6 = HU_Relu(deconv_layer(BN(tf.concat([build_unpool(dc5_1, [1, 2, 2, 1]), c1], 3)

    dc6_1 = HU_Relu(deconv_layer(BN(dc6), [1, 1, nb_filter, (2 * nb_filter)], [batch, 1

    dc7 = HU_Relu(deconv_layer(BN(tf.concat([build_unpool(dc6_1, [1, 2, 2, 1]), c0], 3)

    dc7_1 = HU_Relu(deconv_layer(BN(dc7), [1, 1, 1, (2 * nb_filter)], [batch, 256, 256

    # y = tf.ones_like(dc7_1)

    dc7_1 = dc7_1*4000 - 1000

    output_net_dense = tf.reshape(dc7_1, [batch, size])
    output2 = output_net_dense

##########################################################################################


    with tf.name_scope('Loss'):
        #tf.contrib.metrics.streaming_root_mean_squared_error
        # yy = (ys + 1000) / 4000
        # corr = tf.reduce_mean(tf.contrib.metrics.streaming_pearson_correlation(outpu
        tv = tf.reduce_sum(tf.image.total_variation(dc7_1))
        l2 = tf.div(tf.nn.l2_loss(ys -output2),batch) #   tf.sqrt(tf.reduce_mean(tf.sq
        tf_rmse = tf.reduce_mean(tf.metrics.root_mean_squared_error(ys,output2))
        loss = l2 + 0.2*tv # + 0.2*tf_ssim(dc7_1,output_image)#tf.div(tf.reduce_sum(tf
```

```
tf.summary.scalar("loss", loss)
with tf.name_scope('train'):
    train_1 = tf.train.AdamOptimizer(step).minimize(loss)
# #
saver = tf.train.Saver()
# init = tf.global_variables_initializer()
xxt_batch = np.zeros([batch, size])
yyt_batch = np.zeros([batch, size])
for i in range(batch):
    xxt_batch[i] = test_img120[i]
    yyt_batch[i] = org_img_test[i]

fig = plt.figure()

plt.ion()
plt.show()
plt.subplot(131)
data_org = (np.reshape(org_img_test[1], [row, column], order='F'))
plt.imshow(data_org, cmap="gray")
# scipy.io.savemat('orgin.mat', mdict={'data_org': data_org})
plt.title('orgin')
plt.subplot(132)
data_120  = (np.reshape(test_img120[1], [row, column], order='F'))
plt.imshow(data_120, cmap="gray")
# scipy.io.savemat('120_FBP.mat', mdict={'data_120': data_120})
plt.title('120-FBP')

xx_batch = np.zeros([batch,size])
yy_batch = np.zeros([batch,size])

plt.ion()
plt.subplot(133)
with tf.Session() as sess:
    sess.run(tf.group(tf.global_variables_initializer(), tf.local_variables_initial
    saver.restore(sess, "my_net/20170824_paper2.ckpt")
    merged = tf.summary.merge_all()
    writer = tf.summary.FileWriter("C:/logs/", sess.graph)
    epoch = 2000
    ss = 0.001
    bb = train_num//batch-100

    for ie in range(epoch):
        s2 = ((0.00001 - ss) / epoch) * (ie) + ss
        ind = np.int32(np.linspace(0, train_num - 1, train_num))
        random.shuffle(ind)
        for iv in range(bb):
            for i in range(batch):
                xx_batch[i] = train_img120[ind[i+iv*batch]]
```

```
                    yy_batch[i] = org_train_img[ind[i+iv*batch]]
                sess.run(train_1, feed_dict={xs: xx_batch, ys: yy_batch, step: s2})

            stage2 = sess.run(tf.transpose(dc7_1, perm=[0, 2, 1, 3]), feed_dict={xs: x

            rmse = Measure_Quality(np.reshape(stage2,[1,size]), np.reshape(data_org,[1
            # ssim = ssim(stage2, data_org)
            uqi = Measure_Quality(np.reshape(stage2,[1,size]), np.reshape(data_org,[1,

            # ssim[ie] = sess.run(tf_ssim(data_org, stage2), feed_dict={xs: xxt_batch}
            # var_dict[('ssim', 0)][ie] = ssim[ie]
            #
            # corr[ie] = sess.run(tf_corr(data_org, stage2, row, column), feed_dict={x
            # var_dict[('corr', 0)][ie] = corr[ie]
            #
            # rmse[ie] = sess.run(tf_rmse(stage2, data_org), feed_dict={xs: xxt_batch}
            # var_dict[('rmse', 0)][ie] = rmse[ie]

            if ie % 1 == 0:
                localtime = time.asctime(time.localtime(time.time()))
                pre = sess.run(loss, feed_dict={xs: xx_batch, ys: yy_batch})
                l2_loss = sess.run(l2, feed_dict={xs: xx_batch, ys: yy_batch})
                tv1 = sess.run(tv, feed_dict={xs: xx_batch, ys: yy_batch})
                tf_rmse1 = sess.run(tf_rmse, feed_dict={xs: xx_batch, ys: yy_batch})
                # co  = sess.run(corr, feed_dict={xs: xx_batch, ys: yy_batch})
                print(localtime, ':',
                    "Epoch {}/{}...".format(ie + 1, epoch),
                    "General Loss: {:.4f}(l2_loss: {:.4f} + tv: {:.4f})...".format(pre
                      "tf_rmse: {:.4f}...".format(tf_rmse1),
                    "sl_rmse: {:.4f}...".format(rmse))
                      # "cc: {:.4f}...".format(cc),
                      # "sl_ssim: {:.4f}...".format(ssim))
                      # "uqi: {:.4f}...".format(uqi))

                    # "The amount of Param:   ",get_var_count())

                plt.imshow(stage2, interpolation="nearest", cmap="gray") # clim=(-1000
                plt.title('result')
                plt.pause(.00001)
            if ie % 5 == 0:
                saver.save(sess, "my_net/20170824_paper2.ckpt")
                # save_npy(sess, "./20170824-DDNet_Param.npy")

    print('end')
    plt.pause(10000000000000000)
```

In [16]:
```
from __future__ import print_function
import tensorflow as tf
```

```
import matplotlib.pyplot as plt
from scipy.misc import imresize
import numpy as np
from skimage.io import imread
import math
import csv
import random
import time
import scipy.io
from functools import reduce

def psnr(img1, img2):
    mse = np.mean( np.square(img1 - img2))
    if mse == 0:
        return 100
    PIXEL_MAX = np.max(img1)
    return 20 * np.log10(PIXEL_MAX / np.sqrt(mse))

def rmse(predictions, targets):
    return np.sqrt(np.mean(np.square(predictions - targets)))

def zzc_act_0_1(x):
    y = tf.zeros_like(x)
    x = tf.maximum(x, y)
    y = tf.ones_like(x)
    x = tf.minimum(x, y)
    return x

def zzc_act_bilateral_1_1(x):
    y = tf.ones_like(x)
    x = tf.maximum(x, -y)
    x = tf.minimum(x, y)
    return x

def zzc_norm(x):
    x = tf.div(x + 1000,4000)
    return x


def weight_variable(shape):
    with tf.name_scope('Weight'):
        initial = tf.truncated_normal(shape, stddev = 0.01, name='W')
    return tf.Variable(initial)

def bias_variable(shape):
    with tf.name_scope('Bias'):
        initial = tf.constant(0.1, shape = shape,name='b')
    return tf.Variable(initial)
```

```
        def conv2d(img,w,b):
            conv = tf.nn.conv2d(img, w, strides=[1, 1, 1, 1], padding='SAME') + b
            return conv

        def BN(img):
            batch_mean, batch_var = tf.nn.moments(img, [0, 1, 2], name='moments')
            img = tf.nn.batch_normalization(img, batch_mean, batch_var, 0, 1, 1e-3)
            return img

        def build_unpool(source, kernel_shape):
            input_shape = source.get_shape().as_list()
            return tf.image.resize_images(source, [input_shape[1]*kernel_shape[1], input_shape

        def DenseDeconvNet(input,name,img_H,growth_rate=16,nb_filter=16,filter_wh = 3,k=1):
            shape = input.get_shape().as_list()
            with tf.name_scope(name):
                deconv1_1 = zzc_relu(deconv_layer(input, [1, 1, nb_filter * 4, shape[2]], [ba
                deconv1   = zzc_relu(deconv_layer(deconv1_1, [filter_wh, filter_wh, nb_filter

                h_concat1 = tf.concat([input,deconv1],3)

                deconv2_1 = zzc_relu(deconv_layer(h_concat1, [1, 1, nb_filter * 4, shape[2] +
                deconv2 = zzc_relu(deconv_layer(deconv2_1, [filter_wh, filter_wh, nb_filter, n

                h_concat2 = tf.concat([input, deconv1, deconv2], 3)

                deconv3_1 = zzc_relu(deconv_layer(h_concat2, [1, 1, nb_filter * 4, shape[2] +
                deconv3 = zzc_relu(deconv_layer(deconv3_1, [filter_wh, filter_wh, nb_filter, n

                h_concat3 = tf.concat([input, deconv1, deconv2, deconv3], 3)

                deconv4_1 = zzc_relu(deconv_layer(h_concat3, [1, 1, nb_filter * 4, shape[2] +
                deconv4 = zzc_relu(deconv_layer(deconv4_1, [filter_wh, filter_wh, nb_filter, n

            return tf.concat([input, deconv1, deconv2, deconv3,deconv4], 3)
```

In [15]:
```
        import skimage
        import skimage.io
        import skimage.transform
        import numpy as np


        # synset = [l.strip() for l in open('synset.txt').readlines()]


        # returns image of shape [224, 224, 3]
```

```
# [height, width, depth]
def load_image(path):
    # load image
    img = skimage.io.imread(path)
    img = img / 255.0
    assert (0 <= img).all() and (img <= 1.0).all()
    # print "Original Image Shape: ", img.shape
    # we crop image from center
    short_edge = min(img.shape[:2])
    yy = int((img.shape[0] - short_edge) / 2)
    xx = int((img.shape[1] - short_edge) / 2)
    crop_img = img[yy: yy + short_edge, xx: xx + short_edge]
    # resize to 224, 224
    resized_img = skimage.transform.resize(crop_img, (224, 224))
    return resized_img


# returns the top1 string
def print_prob(prob, file_path):
    synset = [l.strip() for l in open(file_path).readlines()]

    # print prob
    pred = np.argsort(prob)[::-1]

    # Get top1 label
    top1 = synset[pred[0]]
    print(("Top1: ", top1, prob[pred[0]]))
    # Get top5 label
    top5 = [(synset[pred[i]], prob[pred[i]]) for i in range(5)]
    print(("Top5: ", top5))
    return top1


def load_image2(path, height=None, width=None):
    # load image
    img = skimage.io.imread(path)
    img = img / 255.0
    if height is not None and width is not None:
        ny = height
        nx = width
    elif height is not None:
        ny = height
        nx = img.shape[1] * ny / img.shape[0]
    elif width is not None:
        nx = width
        ny = img.shape[0] * nx / img.shape[1]
    else:
        ny = img.shape[0]
```

```
            nx = img.shape[1]
        return skimage.transform.resize(img, (ny, nx))


    def test():
        img = skimage.io.imread("lena.png")
        ny = 300
        nx = img.shape[1] * ny / img.shape[0]
        img = skimage.transform.resize(img, (ny, nx))
        skimage.io.imsave("lena.png", img)


    if __name__ == "__main__":
        test()
```

```
C:\Users\svekhand\AppData\Local\Continuum\anaconda3\lib\site-packages\skimage\transform\_warps
  warn("The default mode, 'constant', will be changed to 'reflect' in "
C:\Users\svekhand\AppData\Local\Continuum\anaconda3\lib\site-packages\skimage\util\dtype.py:122
  .format(dtypeobj_in, dtypeobj_out))
```

```
In [50]: import tensorflow as tf

    import numpy as np
    import matplotlib.pyplot as plt
    import matplotlib.gridspec as gridspec
    import os
    import csv


    def xavier_init(size):
        in_dim = size[0]
        xavier_stddev = 1. / tf.sqrt(in_dim / 2.)
        return tf.random_normal(shape=size, stddev=xavier_stddev)


    X = tf.placeholder(tf.float32, shape=[None, 65536])

    D_W1 = tf.Variable(xavier_init([65536, 128]))
    D_b1 = tf.Variable(tf.zeros(shape=[128]))

    D_W2 = tf.Variable(xavier_init([128, 1]))
    D_b2 = tf.Variable(tf.zeros(shape=[1]))

    theta_D = [D_W1, D_W2, D_b1, D_b2]


    Z = tf.placeholder(tf.float32, shape=[None, 100])
```

```
G_W1 = tf.Variable(xavier_init([100, 128]))
G_b1 = tf.Variable(tf.zeros(shape=[128]))

G_W2 = tf.Variable(xavier_init([128, 65536]))
G_b2 = tf.Variable(tf.zeros(shape=[65536]))

theta_G = [G_W1, G_W2, G_b1, G_b2]


DC_D_W1 = tf.Variable(xavier_init([5, 5, 1, 16]))
DC_D_b1 = tf.Variable(tf.zeros(shape=[16]))

DC_D_W2 = tf.Variable(xavier_init([3, 3, 16, 32]))
DC_D_b2 = tf.Variable(tf.zeros(shape=[32]))

DC_D_W3 = tf.Variable(xavier_init([7 * 7 * 32, 128]))
DC_D_b3 = tf.Variable(tf.zeros(shape=[128]))

DC_D_W4 = tf.Variable(xavier_init([128, 1]))
DC_D_b4 = tf.Variable(tf.zeros(shape=[1]))

theta_DC_D = [DC_D_W1, DC_D_b1, DC_D_W2, DC_D_b2, DC_D_W3, DC_D_b3, DC_D_W4, DC_D_b4]


def sample_Z(m, n):
    return np.random.uniform(-1., 1., size=[m, n])


def generator(z):
    G_h1 = tf.nn.relu(tf.matmul(z, G_W1) + G_b1)
    G_log_prob = tf.matmul(G_h1, G_W2) + G_b2
    G_prob = tf.nn.sigmoid(G_log_prob)

    return G_prob


def discriminator(x):
    print(x.shape)
    D_h1 = tf.nn.relu(tf.matmul(x, D_W1) + D_b1)
    D_logit = tf.matmul(D_h1, D_W2) + D_b2
    D_prob = tf.nn.sigmoid(D_logit)

    return D_prob, D_logit


def plot(samples):
    fig = plt.figure(figsize=(4, 4))
```

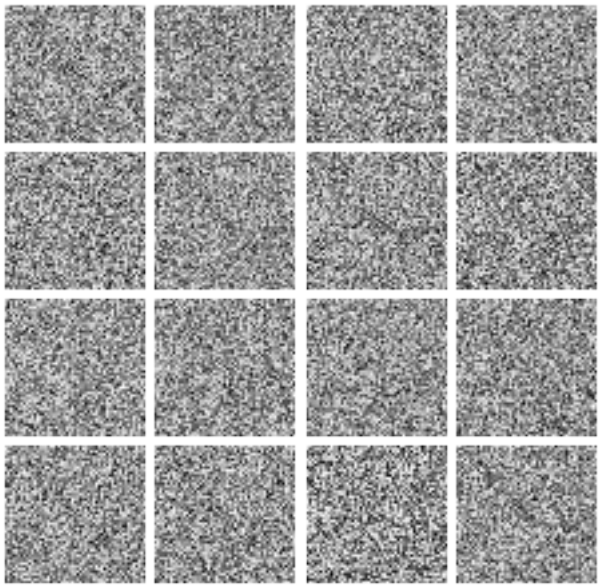
```
    gs = gridspec.GridSpec(4, 4)
    gs.update(wspace=0.05, hspace=0.05)
    plt.ion()

    for i, sample in enumerate(samples):
        ax = plt.subplot(gs[i])
        plt.axis('off')
        ax.set_xticklabels([])
        ax.set_yticklabels([])
        ax.set_aspect('equal')
        plt.imshow(sample.reshape(256, 256), cmap='Greys_r')
        plt.ion()
    return fig


G_sample = generator(Z)
D_real, D_logit_real = discriminator(X)
D_fake, D_logit_fake = discriminator(G_sample)

# D_loss = -tf.reduce_mean(tf.log(D_real) + tf.log(1. - D_fake))
# G_loss = -tf.reduce_mean(tf.log(D_fake))

# Alternative losses:
# -------------------
D_loss_real = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=D_logit_re
D_loss_fake = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=D_logit_fa
D_loss = D_loss_real + D_loss_fake
G_loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=D_logit_fake, l

D_solver = tf.train.AdamOptimizer().minimize(D_loss, var_list=theta_D)
G_solver = tf.train.AdamOptimizer().minimize(G_loss, var_list=theta_G)

mb_size = 128
Z_dim = 100


if __name__ == '__main__':
    row = 256
    column = 256
    size = row * column
    #
    train_num = 2780
    test_num = 500
    #
    batch = 20

    ##########################################################################
    train_img120 = np.zeros((train_num, row * column), dtype=np.float32)
```

```
    with open('D://Program_zzc//Paper2-revise//Data//Results//Mix_train_FBP_sparse120_
        reader = csv.reader(f)
        i = 0
        for ro in reader:
            train_img120[i] = ro
            i += 1

    org_train_img = np.zeros((train_num, row * column), dtype=np.float32)
    with open('D://Program_zzc//Paper2-revise//Data//Results//Mix_train_origin_2588.cs
        reader = csv.reader(f)
        i = 0
        for ro in reader:
            org_train_img[i] = ro
            i += 1
    ##########################################################################################
    test_img120 = np.zeros((test_num, row * column), dtype=np.float32)
    with open('D://Program_zzc//Paper2-revise//Data//Results//New folder//test_sl_FBP_
        reader = csv.reader(f)
        i = 0
        for ro in reader:
            test_img120[i] = ro
            i += 1

    org_img_test = np.zeros((test_num, row * column), dtype=np.float32)
    with open('D://Program_zzc//Paper2-revise//Data//Results//New folder//test_sl_orig
        reader = csv.reader(f)
        i = 0
        for ro in reader:
            org_img_test[i] = ro
            i += 1
    ##########################################################################################

    with tf.name_scope('inputs'):
        xs = tf.placeholder(tf.float32, [batch, size],name='xs')
        ys = tf.placeholder(tf.float32, [batch, size], name='ys')
        yy = tf.placeholder(tf.float32, [batch, size], name='yy')
        step = tf.placeholder(tf.float32, name='prob')
        keep_prob = tf.placeholder(tf.float32,name='prob')

    input_image  = tf.reshape(xs, [batch, row, column, 1])
    output_image = tf.reshape(ys, [batch, row, column, 1])
    ##########################################################################################


sess = tf.Session()
sess.run(tf.global_variables_initializer())
#writer = tf.summary.FileWriter(<some-directory>, sess.graph)
```

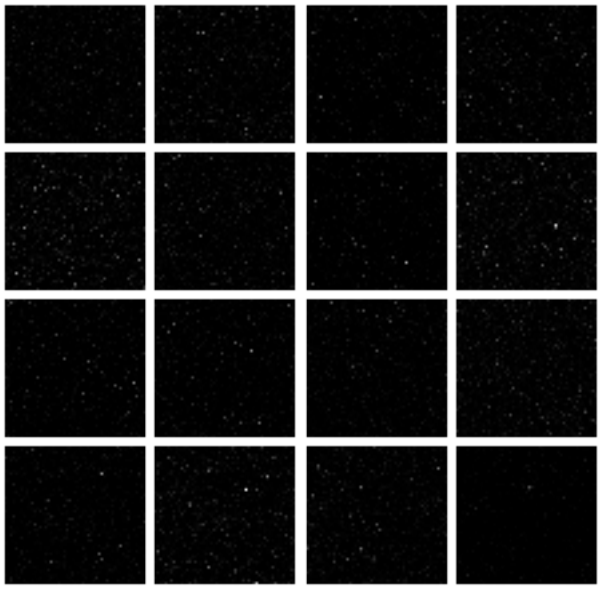
```
if not os.path.exists('out/'):
    os.makedirs('out/')

i = 0
batch = 20

#tv = tf.reduce_sum(tf.image.total_variation(D_real))
#l2 = tf.div(tf.nn.l2_loss(ys -D_fake),batch) #   tf.sqrt(tf.reduce_mean(tf.square(ys
#f_rmse = tf.reduce_mean(tf.metrics.root_mean_squared_error(ys,D_fake))
#loss = l2 + 0.2*tv # + 0.2*tf_ssim(dc7_1,output_image
train_1 = tf.train.AdamOptimizer(step).minimize(G_loss)

#import tf.saver as saver
#with tf.Session() as sess:
sess.run(tf.group(tf.global_variables_initializer(), tf.local_variables_initializer())
        #tf.summary.saver.restore(sess, "my_net/20170824_paper2.ckpt")
merged = tf.summary.merge_all()
writer = tf.summary.FileWriter("C:/logs/", sess.graph)
epoch = 2000
ss = 0.001
bb = train_num//batch-100

xx_batch = np.zeros([batch,size])
yy_batch = np.zeros([batch,size])
#
#for ie in range(epoch):
#    s2 = ((0.00001 - ss) / epoch) * (ie) + ss
#    ind = np.int32(np.linspace(0, train_num - 1, train_num))
#    random.shuffle(ind)
#for iv in range(bb):
#    for i in range(batch):
#        xx_batch[i] = train_img120[ind[i+iv*batch]]
#        yy_batch[i] = org_train_img[ind[i+iv*batch]]
#        sess.run(train_1, feed_dict={xs: xx_batch, ys: yy_batch, step: s2})


for it in range(1000000):
    if it % 1000 == 0:
        samples = sess.run(G_sample, feed_dict={Z: sample_Z(16, Z_dim)})

        fig = plot(samples)
        plt.savefig('out/{}.png'.format(str(i).zfill(3)), bbox_inches='tight')
        i += 1
        plt.show()
        plt.ion()
        plt.close(fig)

    X_mb=test_img120
```

```
            D_loss_curr_list = np.array([])
            G_loss_curr_list=np.array([])
            _, D_loss_curr = sess.run([D_solver, D_loss], feed_dict={X: X_mb, Z: sample_Z(mb_s
            _, G_loss_curr = sess.run([G_solver, G_loss], feed_dict={Z: sample_Z(mb_size, Z_di

            if it % 1000 == 0:
                print('Iter: {}'.format(it))
                print(D_loss_curr)
                print('D loss: {:.4}'. format(D_loss_curr))
                print('G_loss: {:.4}'.format(G_loss_curr))
                D_loss_curr_list=np.append(D_loss_curr_list,D_loss_curr)
                G_loss_curr_list=np.append(G_loss_curr_list,G_loss_curr)
                print()
```

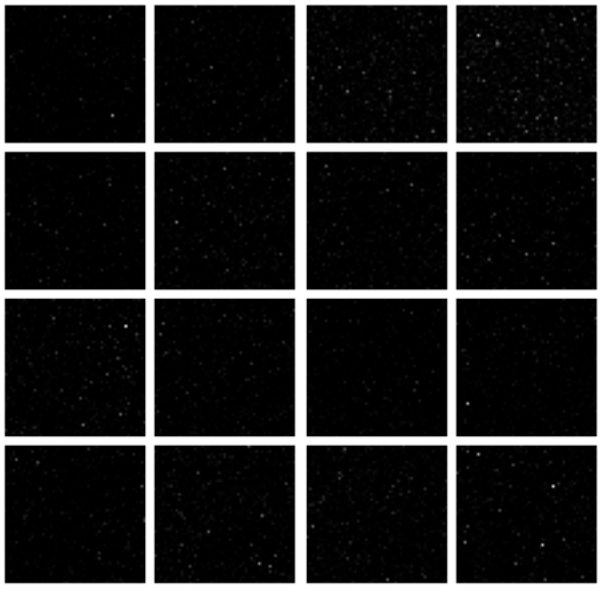
```
(?, 65536)
(?, 65536)
```



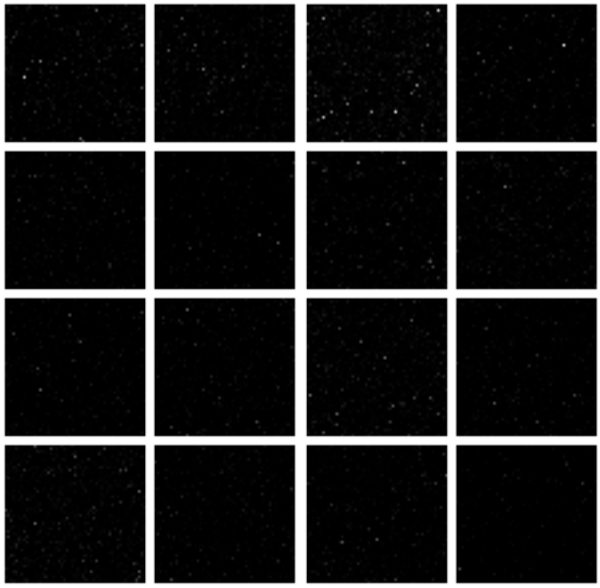
```
Iter: 0
1.0360416
D loss: 1.036
G_loss: 184.9
```



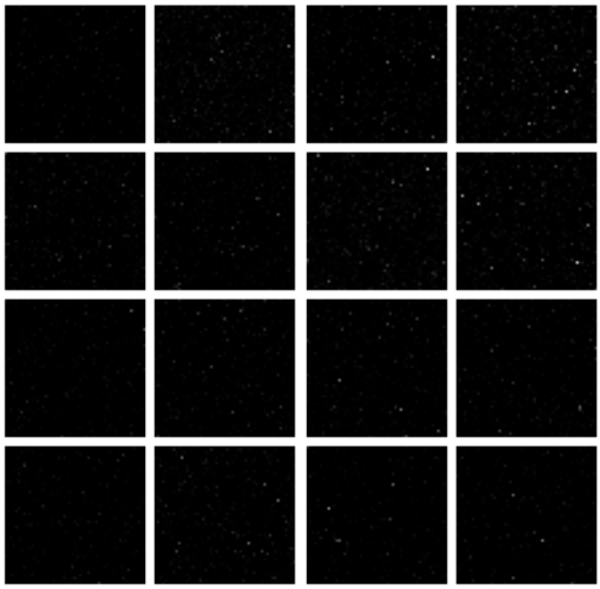
```
Iter: 1000
1.0397465
D loss: 1.04
G_loss: 0.9547
```



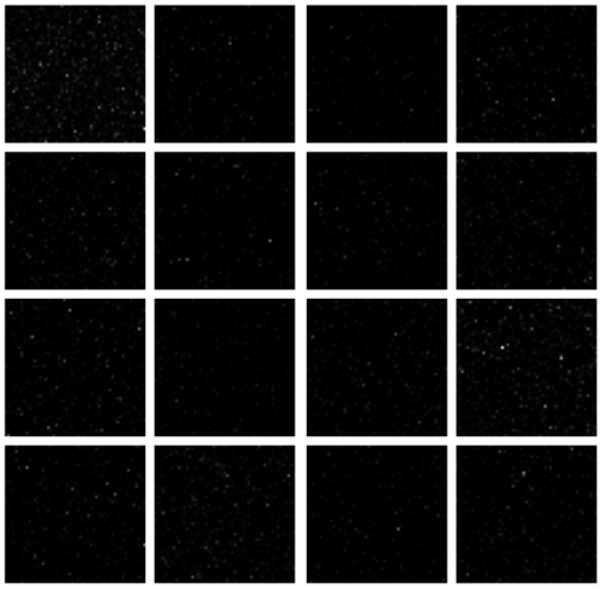
```
Iter: 2000
1.2623241
D loss: 1.262
G_loss: 0.7796
```



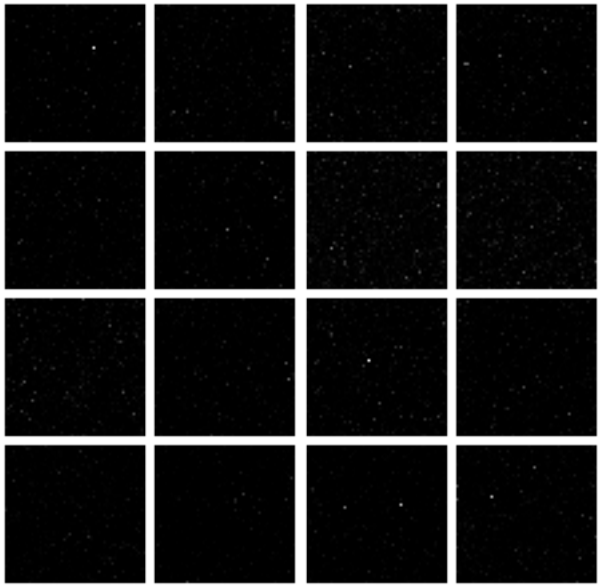
```
Iter: 3000
1.3108022
D loss: 1.311
G_loss: 0.7337
```



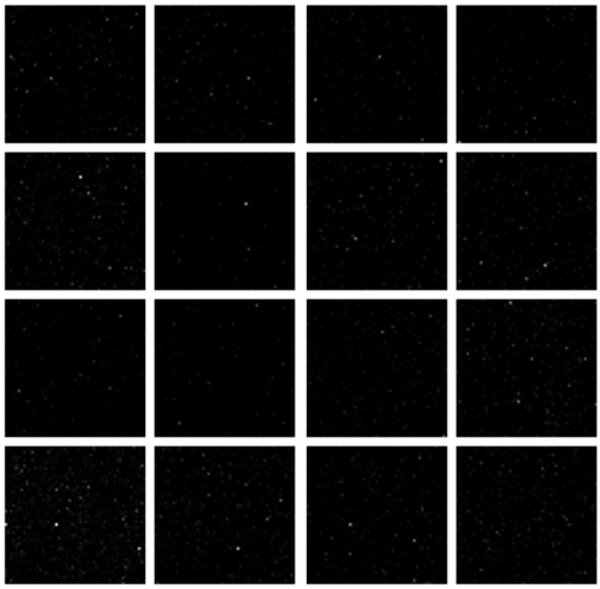
```
Iter: 4000
1.3411863
D loss: 1.341
G_loss: 0.7247
```



```
Iter: 5000
1.3484726
D loss: 1.348
G_loss: 0.7192
```



```
Iter: 6000
1.3526983
D loss: 1.353
G_loss: 0.714
```



```
Iter: 7000
1.3554997
D loss: 1.355
G_loss: 0.7148
```

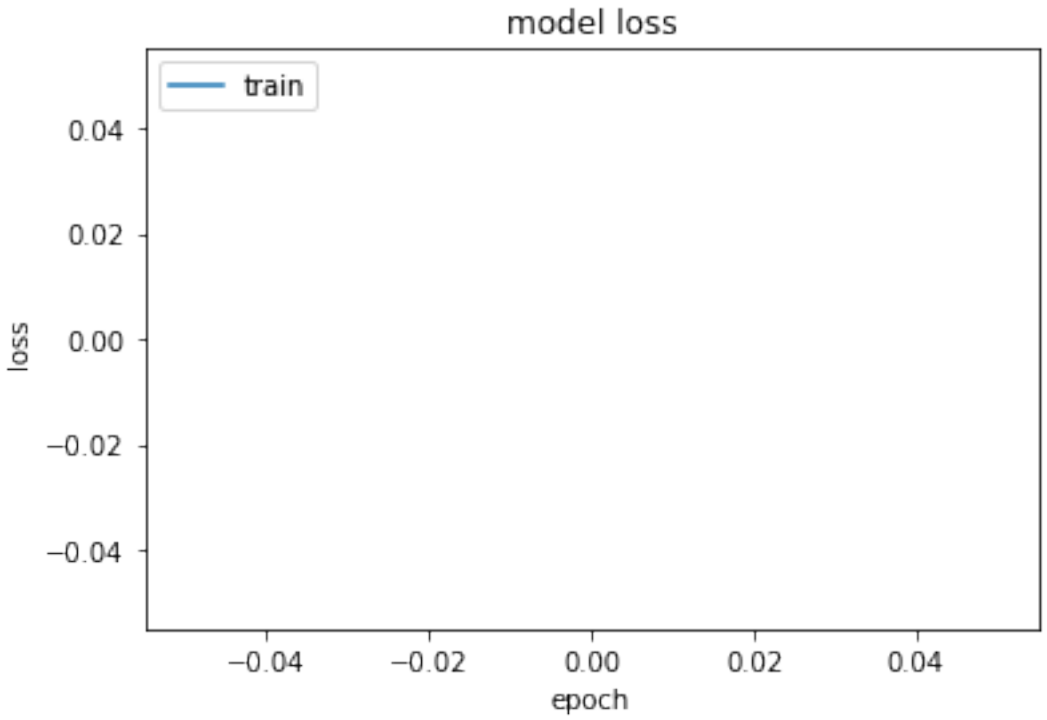
```
---------------------------------------------------------------------------

KeyboardInterrupt                         Traceback (most recent call last)

<ipython-input-50-189e5ecedd63> in <module>()
    220     D_loss_curr_list = np.array([])
    221     G_loss_curr_list=np.array([])
--> 222     _, D_loss_curr = sess.run([D_solver, D_loss], feed_dict={X: X_mb, Z: sample_Z(
    223     _, G_loss_curr = sess.run([G_solver, G_loss], feed_dict={Z: sample_Z(mb_size, 
    224 


C:\Users\svekhand\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python
    898     try:
    899       result = self._run(None, fetches, feed_dict, options_ptr,
--> 900                          run_metadata_ptr)
    901       if run_metadata:
    902         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)
```

```
C:\Users\svekhand\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python
   1133     if final_fetches or final_targets or (handle and feed_dict_tensor):
   1134       results = self._do_run(handle, final_targets, final_fetches,
-> 1135                              feed_dict_tensor, options, run_metadata)
   1136     else:
   1137       results = []


C:\Users\svekhand\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python
   1314     if handle is None:
   1315       return self._do_call(_run_fn, feeds, fetches, targets, options,
-> 1316                            run_metadata)
   1317     else:
   1318       return self._do_call(_prun_fn, handle, feeds, fetches)


C:\Users\svekhand\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python
   1320   def _do_call(self, fn, *args):
   1321     try:
-> 1322       return fn(*args)
   1323     except errors.OpError as e:
   1324       message = compat.as_text(e.message)


C:\Users\svekhand\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python
   1305       self._extend_graph()
   1306       return self._call_tf_sessionrun(
-> 1307           options, feed_dict, fetch_list, target_list, run_metadata)
   1308
   1309     def _prun_fn(handle, feed_dict, fetch_list):


C:\Users\svekhand\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python
   1407       return tf_session.TF_SessionRun_wrapper(
   1408           self._session, options, feed_dict, fetch_list, target_list,
-> 1409           run_metadata)
   1410     else:
   1411       with errors.raise_exception_on_not_ok_status() as status:


KeyboardInterrupt:
```

In [25]:
```
plt.plot(D_loss_curr_list,G_loss_curr_list )
print(G_loss_curr_list)
print(len(G_loss_curr_list))
plt.title('model loss')
```

```
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='upper left')
        plt.show()
        plt.ion()
```

```
[]
0
```




```
In [19]: D_loss_curr_list=np.append(D_loss_curr_list,6)
```

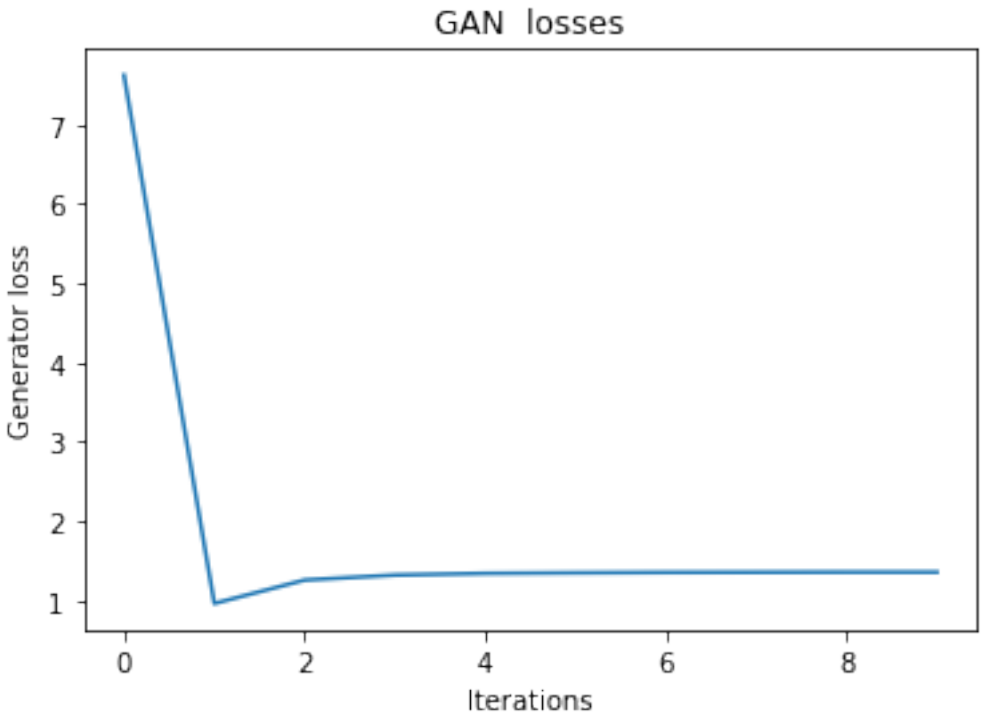
```
In [37]: print((G_loss_curr_list))
```

```
[  7.622    0.9615   1.257    1.32     1.341    1.348    1.353    1.355
   1.357    1.357  181.5      1.051    0.7672   0.7414   0.7278   0.7182
   0.7146   0.7159   0.7154   0.7135 181.5      1.051    0.7672   0.7414
   0.7278   0.7182   0.7146   0.7159   0.7154   0.7135 181.5      1.051
   0.7672   0.7414   0.7278   0.7182   0.7146   0.7159   0.7154   0.7135]
```

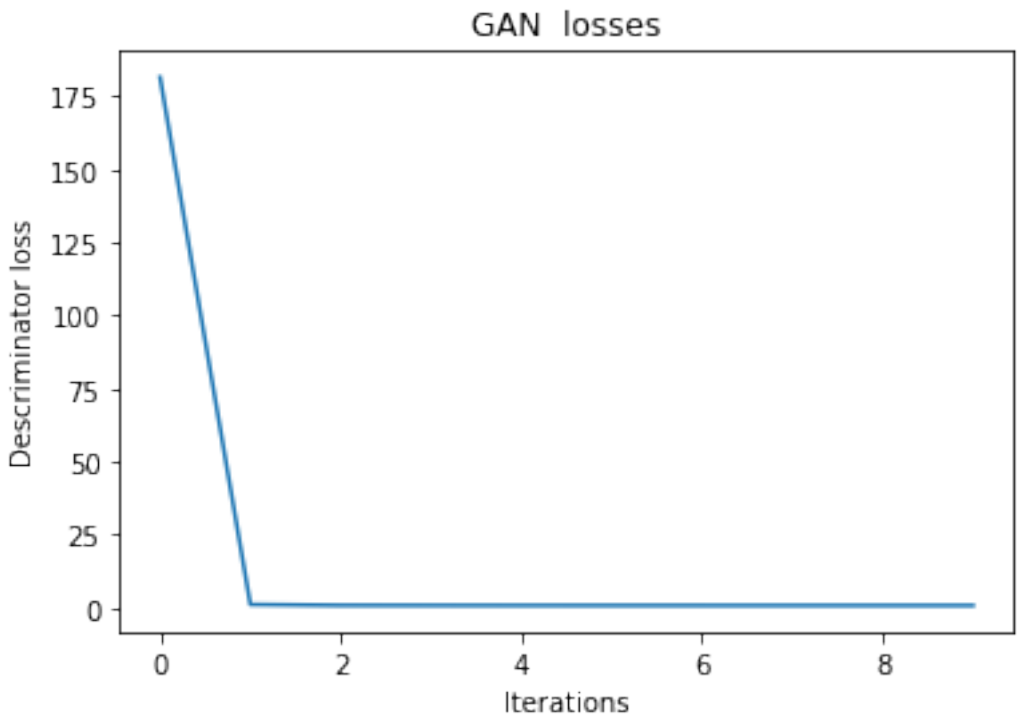
```
In [39]: G_loss_curr_list=[]
         G_loss_curr_list=np.append(G_loss_curr_list,[181.5
         ,1.051
```

```
,0.7672
,0.7414
,0.7278
,0.7182
,0.7146
, 0.7159
,0.7154
,0.7135
])
```

In [41]:
```
plt.plot(D_loss_curr_list )
plt.title('GAN  losses')
plt.ylabel('Generator loss')
plt.xlabel('Iterations in thousands')
#plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.ion()
```



In [42]:
```
plt.plot(G_loss_curr_list )
plt.title('GAN  losses')
plt.ylabel('Descriminator loss')
plt.xlabel('Iterations in thousands')
#plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.ion()
```




```
In [44]: import os
         os.environ["CUDA_VISIBLE_DEVICES"]="1"
```

```
In [53]: plt.imshow(train_img120)
         plt.show()
```