```
! shred -u setup_google_colab.py
! wget https://raw.githubusercontent.com/hse-aml/intro-to-dl/master/setup_google_co
import setup_google_colab
setup_google_colab.setup_week4()

# !pip uninstall keras-nightly
# !pip uninstall -y tensorflow
# !pip install tensorflow==1.15.0
# !pip install keras==2.1.6
# !pip install install h5py==2.10.0




# set tf 1.x for colab
%tensorflow_version 1.x
```

```
shred: setup_google_colab.py: failed to open for writing: No such file or dire
--2022-04-27 05:02:32--  https://raw.githubusercontent.com/hse-aml/intro-to-dl
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.10
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.10
HTTP request sent, awaiting response... 200 OK
Length: 3636 (3.6K) [text/plain]
Saving to: ‘setup_google_colab.py’

setup_google_colab. 100%[===================>]   3.55K  --.-KB/s    in 0s

2022-04-27 05:02:32 (29.9 MB/s) - ‘setup_google_colab.py’ saved [3636/3636]

**************************************************
lfw-deepfunneled.tgz
**************************************************
lfw.tgz
**************************************************
lfw_attributes.txt
TensorFlow 1.x selected.
```

Generating human faces with Adversarial Networks


© research.nvidia.com

This time we'll train a neural net to generate plausible human faces in all their subtlty: appearance, expression, accessories, etc. 'Cuz when us machines gonna take over Earth, there won't be any more faces left. We want to preserve this data for future iterations. Yikes...

Based on https://github.com/Lasagne/Recipes/pull/94 .

```
import sys
sys.path.append("..")
import grading
import download_utils
import tqdm_utils
```

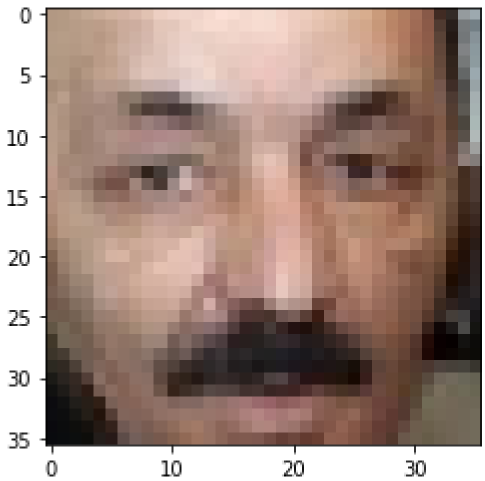
```
download_utils.link_week_4_resources()
```

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
plt.rcParams.update({'axes.titlesize': 'small'})

from sklearn.datasets import load_digits
#The following line fetches you two datasets: images, usable for autoencoder traini
#Those attributes will be required for the final part of the assignment (applying s
from lfw_dataset import load_lfw_dataset
data,attrs = load_lfw_dataset(dimx=36,dimy=36)

#preprocess faces
data = np.float32(data)/255.

IMG_SHAPE = data.shape[1:]
```

```
**************************************************
```

```
#print random image
plt.imshow(data[np.random.randint(data.shape[0])], cmap="gray", interpolation="none
```

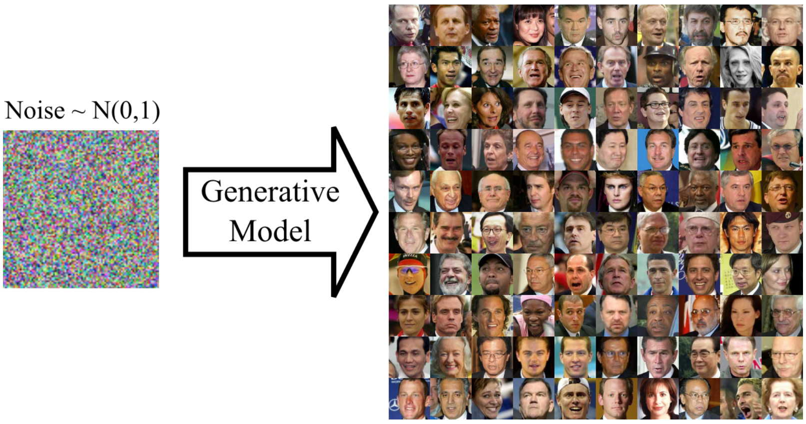
```
<matplotlib.image.AxesImage at 0x7fcc9304db10>
```




```
IMG_SHAPE
```

```
(36, 36, 3)
```

Generative adversarial nets 101



© torch.github.io

Deep learning is simple, isn't it?

- build some network that generates the face (small image)
- make up a **measure** of **how good that face is**
- optimize with gradient descent :)

The only problem is: how can we engineers tell well-generated faces from bad? And i bet you we won't ask a designer for help.

If we can't tell good faces from bad, we delegate it to yet another neural network!

That makes the two of them:

- **G**enerator - takes random noize for inspiration and tries to generate a face sample.
  - Let's call him **G**(z), where z is a gaussian noize.
- **D**iscriminator - takes a face sample and tries to tell if it's great or fake.
  - Predicts the probability of input image being a **real face**
  - Let's call him **D**(x), x being an image.
  - **D(x)** is a predition for real image and **D(G(z))** is prediction for the face made by generator.

Before we dive into training them, let's construct the two networks.

```
import tensorflow as tf
from keras_utils import reset_tf_session
s = reset_tf_session()

import keras
from keras.models import Sequential
from keras import layers as L
```

```
WARNING:tensorflow:From /content/keras_utils.py:68: The name tf.get_default_se

WARNING:tensorflow:From /content/keras_utils.py:75: The name tf.ConfigProto is
```

```
WARNING:tensorflow:From /content/keras_utils.py:77: The name tf.InteractiveSes

Using TensorFlow backend.
```

```
from keras.layers import Convolution2D as Conv2D
from keras.layers.convolutional import Deconv2D as Conv2DTranspose

CODE_SIZE = 256

generator = Sequential()
generator.add(L.InputLayer([CODE_SIZE],name='noise'))
generator.add(L.Dense(10*8*8, activation='elu'))

generator.add(L.Reshape((8,8,10)))
generator.add(L.Conv2DTranspose(64,kernel_size=(5,5),activation='elu'))
generator.add(L.Conv2DTranspose(64,kernel_size=(5,5),activation='elu'))
generator.add(L.UpSampling2D(size=(2,2)))
generator.add(L.Conv2DTranspose(32,kernel_size=3,activation='elu'))
generator.add(L.Conv2DTranspose(32,kernel_size=3,activation='elu'))
generator.add(L.Conv2DTranspose(32,kernel_size=3,activation='elu'))

generator.add(L.Conv2D(3,kernel_size=3,activation=None))
```

```
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/python/op
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

```
assert generator.output_shape[1:] == IMG_SHAPE, "generator must output an image of
```

Discriminator

- Discriminator is your usual convolutional network with interlooping convolution and pooling layers
- The network does not include dropout/batchnorm to avoid learning complications.
- We also regularize the pre-output layer to prevent discriminator from being too certain.

```
discriminator = Sequential()

discriminator.add(L.InputLayer(IMG_SHAPE))

discriminator.add(L.Conv2D(8, (3, 3)))
discriminator.add(L.LeakyReLU(0.1))
discriminator.add(L.Conv2D(16, (3, 3)))
discriminator.add(L.LeakyReLU(0.1))
discriminator.add(L.MaxPool2D())
discriminator.add(L.Conv2D(32, (3, 3)))
discriminator.add(L.LeakyReLU(0.1))
discriminator.add(L.Conv2D(64, (3, 3)))
discriminator.add(L.LeakyReLU(0.1))
discriminator.add(L.MaxPool2D())   #YOUR OWN CODE
```

```
discriminator.add(L.Flatten())
discriminator.add(L.Dense(256,activation='tanh'))
discriminator.add(L.Dense(2,activation=tf.nn.log_softmax))
```

```
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/keras/backend/tensorflow_
```

```
generator.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 640)               164480    
_________________________________________________________________
reshape_1 (Reshape)          (None, 8, 8, 10)          0         
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 12, 12, 64)        16064     
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 16, 16, 64)        102464    
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 32, 32, 64)        0         
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 34, 34, 32)        18464     
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 36, 36, 32)        9248      
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 38, 38, 32)        9248      
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 36, 36, 3)         867       
=================================================================
Total params: 320,835
Trainable params: 320,835
Non-trainable params: 0
_________________________________________________________________
```

```
discriminator.summary()
```

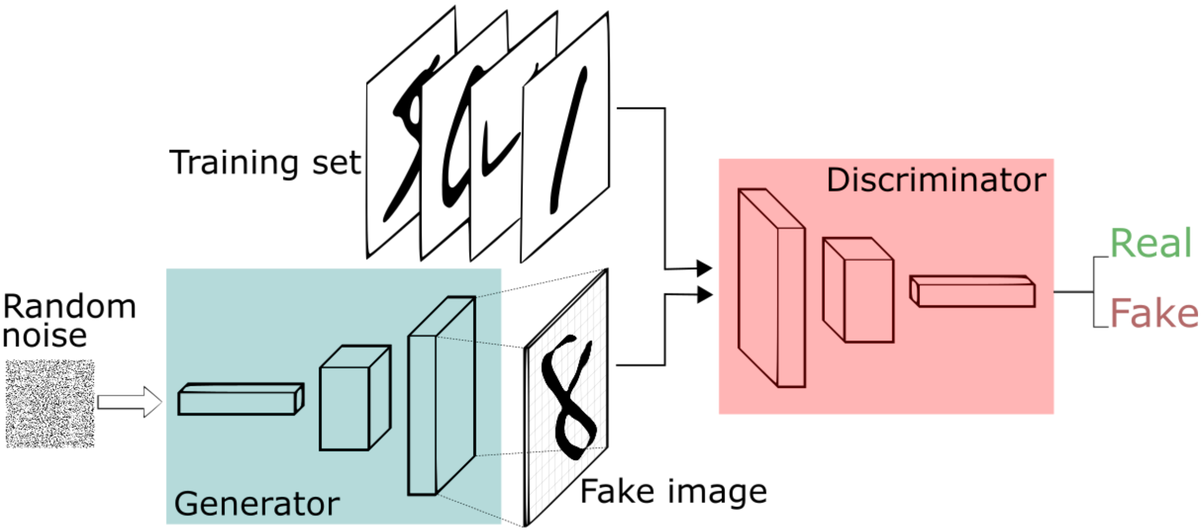
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_2 (Conv2D)            (None, 34, 34, 8)         224       
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 34, 34, 8)         0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 16)        1168      
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 32, 32, 16)        0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 16)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 32)        4640      
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 14, 14, 32)        0         
```

```
conv2d_5 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 12, 12, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               590080
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 514
=================================================================
Total params: 615,122
Trainable params: 615,122
Non-trainable params: 0
_________________________________________________________________
```

Training

We train the two networks concurrently:

- Train **discriminator** to better distinguish real data from **current** generator
- Train **generator** to make discriminator think generator is real
- Since discriminator is a differentiable neural network, we train both with gradient descent.



© deeplearning4j.org

Training is done iteratively until discriminator is no longer able to find the difference (or until you run out of patience).

Tricks:

- Regularize discriminator output weights to prevent explosion
- Train generator with **adam** to speed up training. Discriminator trains with SGD to avoid problems with momentum.
- More: https://github.com/soumith/ganhacks

```
noise = tf.placeholder('float32',[None,CODE_SIZE])
```

```
real_data = tf.placeholder('float32',[None,]+list(IMG_SHAPE))

logp_real = discriminator(real_data)

#YOUR OWN CODE
generated_data = generator(noise)

logp_gen = discriminator(generated_data)
```

```
########################
#discriminator training#
########################

d_loss = -tf.reduce_mean(logp_real[:,1] + logp_gen[:,0])

#regularize
d_loss += tf.reduce_mean(discriminator.layers[-1].kernel**2)

#optimize
disc_optimizer =  tf.train.GradientDescentOptimizer(1e-3).minimize(d_loss,var_list=
```

```
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/python/op
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

```
########################
###generator training###
########################

g_loss = -tf.reduce_mean(logp_gen[:,1])
gen_optimizer = tf.train.AdamOptimizer(1e-4).minimize(g_loss,var_list=generator.tra
```

```
s.run(tf.global_variables_initializer())
```

Auxiliary functions

Here we define a few helper functions that draw current data distributions and sample training batches.

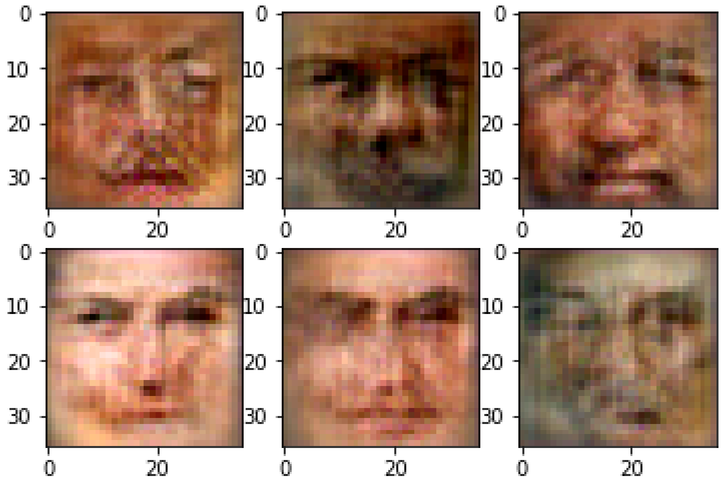
```
def sample_noise_batch(bsize):
    return np.random.normal(size=(bsize, CODE_SIZE)).astype('float32')

def sample_data_batch(bsize):
    idxs = np.random.choice(np.arange(data.shape[0]), size=bsize)
    return data[idxs]

def sample_images(nrow,ncol, sharp=False):
    images = generator.predict(sample_noise_batch(bsize=nrow*ncol))
```

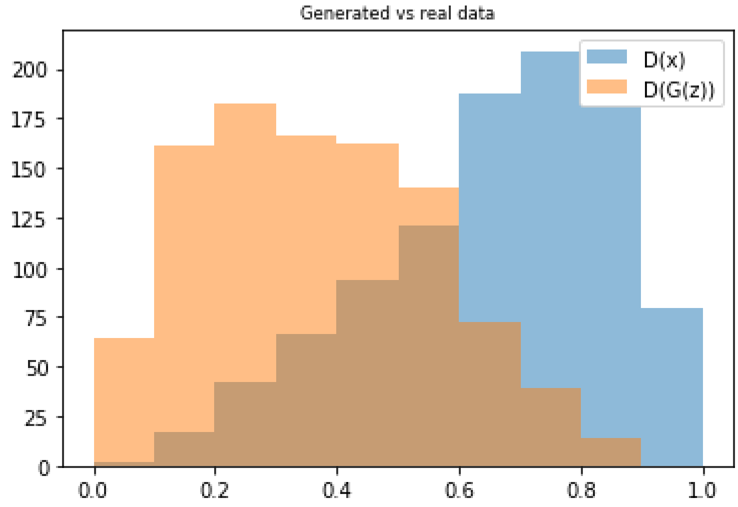
```
    if np.var(images)!=0:
        images = images.clip(np.min(data),np.max(data))
    for i in range(nrow*ncol):
        plt.subplot(nrow,ncol,i+1)
        if sharp:
            plt.imshow(images[i].reshape(IMG_SHAPE),cmap="gray", interpolation="non
        else:
            plt.imshow(images[i].reshape(IMG_SHAPE),cmap="gray")
    plt.show()

def sample_probas(bsize):
    plt.title('Generated vs real data')
    plt.hist(np.exp(discriminator.predict(sample_data_batch(bsize)))[:,1],
             label='D(x)', alpha=0.5,range=[0,1])
    plt.hist(np.exp(discriminator.predict(generator.predict(sample_noise_batch(bsiz
             label='D(G(z))',alpha=0.5,range=[0,1])
    plt.legend(loc='best')
    plt.show()
```

Training

Main loop. We just train generator and discriminator in a loop and plot results once every N iterations.

```
from IPython import display

for epoch in tqdm_utils.tqdm_notebook_failsafe(range(50000)):

    feed_dict = {
        real_data:sample_data_batch(100),
        noise:sample_noise_batch(100)
    }

    for i in range(5):
        s.run(disc_optimizer,feed_dict)

    s.run(gen_optimizer,feed_dict)

    if epoch %100==0:
        display.clear_output(wait=True)
        sample_images(2,3,True)
        sample_probas(1000)
```





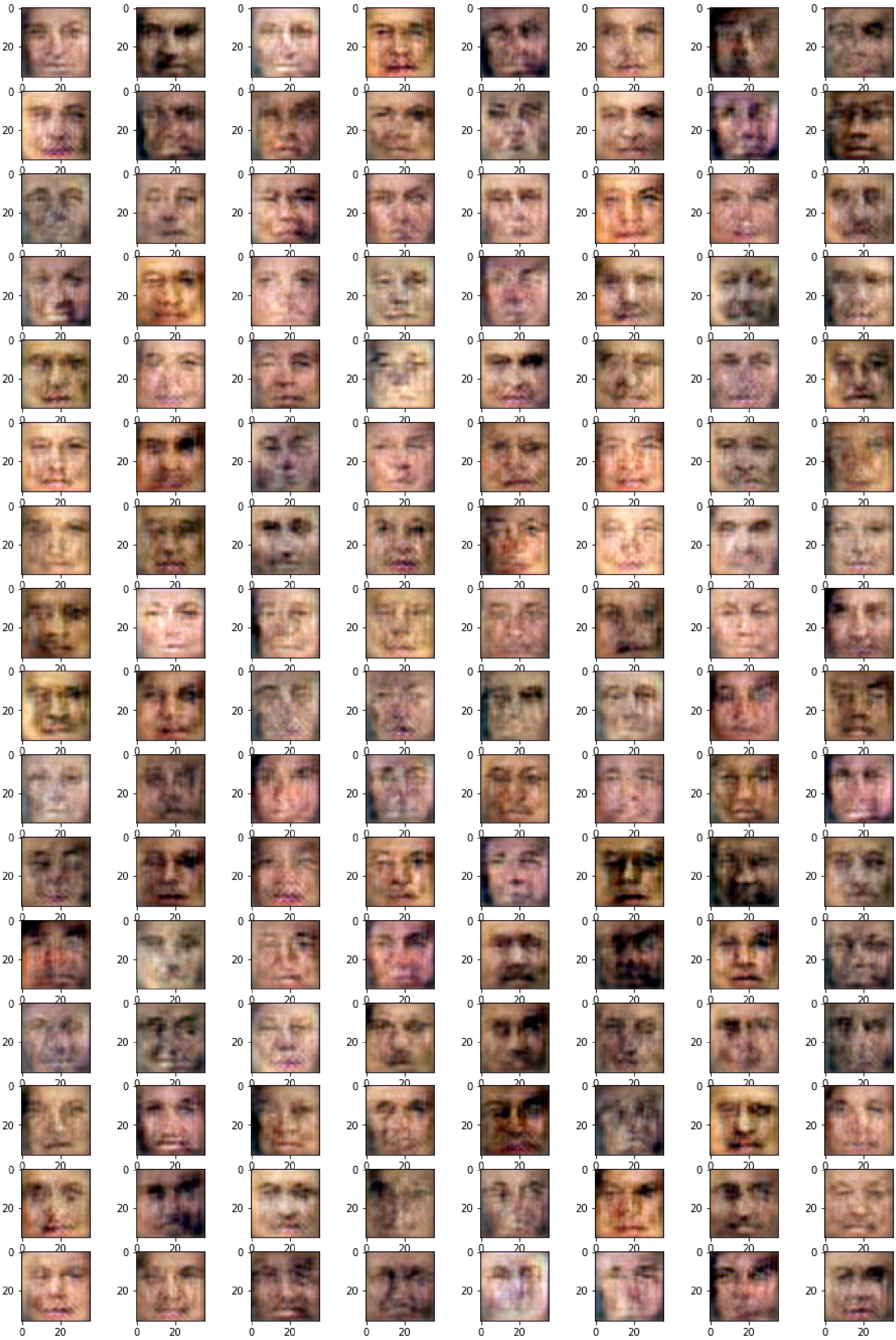

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-18-b6788efadda2> in <module>()
      9 
     10     for i in range(5):
---> 11         s.run(disc_optimizer,feed_dict)
     12 
     13     s.run(gen_optimizer,feed_dict)
```

5 frames

```
/tensorflow-1.15.2/python3.7/tensorflow_core/python/client/session.py in _call
run_metadata)
   1441     return tf_session.TF_SessionRun_wrapper(self._session, options, fe
```

```
from submit_honor import submit_honor
submit_honor((generator, discriminator),
             'e0321294@u.nus.edu',
             'bWRvZqVSHxtIK8yh')
```

```
#The network was trained for about 15k iterations.
#Training for longer yields MUCH better results
plt.figure(figsize=[16,24])
sample_images(16,8)
```



34s completed at 13:28