Augmented Reality Deep Dream

May 15, 2016

0.1 Abstract

Onlookers to the Google Research blog post immediately reacted to how image captioning neural network-generated images resembled the same imagery seen under the influence of psychedelic drugs. This work uses the deep dream image making technique to explore the story of Frank Olson, a scientist and subject of an officially-acknowledged CIA drug experimentation program who died under mysterious circumstances. Despite a large financial settlement, his surviving son still pursues different conspiracies investigating his death by falling from a window of a hotel. This piece reenacts the experience of psychedelic drugs safely on a computer to answer one grim question posed by conspiracy theorists and distinguished journalists like Seymour Hersh: Was Frank Olson so high he accidentally walked out of a hotel window? Refining a technique used to generate this images to create a video, I argue that the experience of looking down from a tower while under the influence of LSD seems utterly terrifying: one becomes more afraid of, not attracted to, windows. This reenactment suggests Frank Olson's death was a homicide, killed in the hotel room and pushed out the window as a part of a coverup, as many suggest.

0.2 Prior Work

This notebook follows the outline of the Google Research blog post's notebook. The main modifications are some details in the optimization step, multi-processing for multiple frames, compatibility with Linux and Mac, compatibility with Fermi and Maxwell nVidia chips, and parameter testing.

The deepdream reddit collected some of the best visual work. It was very helpful to set a bar for what would actually be new. They also include the most up-to-date and vetted tutorials. Various tutorials online were helpful but generally were faulty in one or two details in the I.T. All the work presented below is original and wholly my own unless otherwise stated.

A great implementation for video was recently released and I plan to explore for later work. In general, videos were produced by running a process frame-by-frame, and this is what I did in this project.

In terms of controlling the output of a deep dream image process, I improved on techniques outlined largely in the original Google Research blog post. Most other attempts at control make mistakes, especially like the following exercerpted from the otherwise good Reddit tutorial:

> If you were to train your own neural network with lots of images of hands then you could generate your own deepdream images from this net and see everything be created from hands.

The opposite is actually true. The reason the image captioning neural networks do so well in image generation is that they have to perform well on a large variety of categories. The greater the diversity, the more likely the neural network would have to learn higher-level image features to distinguish between categories. In fact, training only on hands would basically result in noise, since the neural network's job becomes easier (e.g., a good classifier could be, just guess hands most of the time). This and other common errors were not tried in this work for the purposes of controlling output.

0.3 Outline of Procedure

Follow the complete Python notebook below.

1. Install a variety of dependencies.

2. Permute a variety of parameters of the image making process.
3. Choose good input and guide images. Choose layers close to the original image for textural features to reenact psychedelic changes in perception, or choose layers close to the output for high-level structural features to reenact psychedelic hallucinations.

0.4 Results

View the video here. View the still images here.

- **To control the output**: Use the layer and the guide image to control the intended deep dream output. For high order features, use layer Inception 4b of GoogLeNet. To make certain things hallucinated into the image, use a guide image that is a collage of training images of the desired category.
- **To reprodue the experience of psychedelics**: Use the textural features generated by layers 1 and 2, or use dark guide images like insects and furniture with layers 3, 4 and 5. The neural network process seems very effective at generating hallucinations for dark parts of an input image for a small number of iterations, which seems to match the experience of actual psychedelics.
- **Did Frank Olson fall out of that window because he was really high?**: Based on the video, probably not. He had been studying bacteria and insects. Hallucinating those things from the point of view of a window's ledge is terrifying. I felt like I would run away from that image, not walk towards it and fall off.

Can the deep dream process be controlled? Once we have control, can we reproduce the experience of looking out a window and looking down below?

In order to investigate the control of the images, this experiment permuted all the parameters of the imagemaking process to discover which ones had the biggest influence on the final output image. These parameters are the input image, the guide image, the number of octaves, the number of iterations, and the neural network layer optimized. The hypothesis was that the guide image followed by the neural network layer had the biggest impact on the final image, and this hypothesis turned out to be true.

Of 2,674 still images produced, 105 (3.9%) of the images qualitatively reenacted the experience of psychedelics on a consensus of 3 individuals with expertise. These images can be viewed here.

The parameter that had the most influence was the "guide image," or the image used to activate the neural network just prior to "running it in reverse" and generating a new image. We hypothesized that if the guide image was an exact member of the training set for GoogLeNet, it would appear in the generated image; otherwise, it would not. Of the 34 guide images chosen, 33 produced the expected result.

After the guide image, the layer chosen had the most influence on the appearance of the image. Of the 121 neural network layers in GoogLeNet, 36 were unsuitable for image making, and all the remaining 85 produced consistently expected results once characterized. Broadly, the deeper the layer among the 5, the more complex the structure of the features generated by this process. Layers 1 and 2 produced textural effects our experts said were highly reminiscent of psychedelics, while Layers 3, 4 and 5 corresponded to some aspects (but not all) of the hallucinations.

To produce a video, this process used a sequence of stills from various extreme sports and daredevil first-person video. The best received was a cut of Russian parkour daredevils treated with spiders, one of the animals Frank Olson was studying. It used 4 octaves, 6 iterations and layer "Inception 4b" to produce the intended result.

0.5 Complete procedure

0.5.1 Dependencies

This notebook is designed to have as few dependencies as possible: * Standard Python scientific stack: NumPy, SciPy, PIL, IPython. Those libraries can also be installed as a part of one of the scientific packages for Python, such as Anaconda or Canopy. * Caffe deep learning framework (installation instructions). * Google protobuf library that is used for Caffe model manipulation.

```
In [ ]: from cStringIO import StringIO
        import numpy as np
```

```
import scipy.ndimage as nd
import PIL.Image
import sys
from IPython.display import clear_output, Image, display
from google.protobuf import text_format

import caffe

# If your GPU supports CUDA and Caffe was built with CUDA support,
# uncomment the following to run Caffe operations on the GPU.
caffe.set_mode_gpu()
# Selects the first GPU device if multiple devices exist
caffe.set_device(0)
```

We will also add a utility function for displaying images.

```
In [ ]: def showarray(a, fmt='jpeg'):
            a = np.uint8(np.clip(a, 0, 255))
            f = StringIO()
            PIL.Image.fromarray(a).save(f, fmt)
            display(Image(data=f.getvalue()))
```

0.5.2 Loading DNN model

In this notebook we are going to use a GoogLeNet model trained on ImageNet dataset.

```
In [ ]: import os
        WORKING_PATH = os.getcwd()

        model_path = WORKING_PATH + '/'
        net_fn = model_path + 'deploy.prototxt'
        caffe_model_name = 'bvlc_googlenet'
        # Place the downloaded caffemodel at this path
        param_fn = model_path + 'Caffe Models/' + caffe_model_name + '.caffemodel'

        # Patching model to be able to compute gradients.
        # Note that you can also manually add "force_backward: true" line to "deploy.prototxt".
        model = caffe.io.caffe_pb2.NetParameter()
        text_format.Merge(open(net_fn).read(), model)
        model.force_backward = True
        open('tmp.prototxt', 'w').write(str(model))

        net = caffe.Classifier('tmp.prototxt', param_fn,
                               mean = np.float32([104.0, 116.0, 122.0]), # ImageNet mean, training set
                               channel_swap = (2,1,0)) # the reference model has channels in BGR order

        # a couple of utility functions for converting to and from Caffe's input image layout
        def preprocess(net, img):
            return np.float32(np.rollaxis(img, 2)[::-1]) - net.transformer.mean['data']
        def deprocess(net, img):
            return np.dstack((img + net.transformer.mean['data'])[::-1])
```

0.5.3 Producing dreams

From the Google Research blog post:

Making the "dream" images is very simple. Essentially it is just a gradient ascent process that tries to maximize the L2 norm of activations of a particular DNN layer. Here are a few simple tricks that we found useful for getting good images: * offset image by a random jitter * normalize the magnitude of gradient ascent steps * apply ascent across multiple scales (octaves) First we implement a basic gradient ascent step function, applying the first two tricks:

```
In [ ]: def objective_L2(dst):
            dst.diff[:] = dst.data

        def make_step(net, step_size=1.5, end='inception_4c/output',
                      jitter=32, clip=True, objective=objective_L2):
            '''Basic gradient ascent step.'''

            src = net.blobs['data'] # input image is stored in Net's 'data' blob
            dst = net.blobs[end]

            ox, oy = np.random.randint(-jitter, jitter+1, 2)
            src.data[0] = np.roll(np.roll(src.data[0], ox, -1), oy, -2) # apply jitter shift

            net.forward(end=end)
            objective(dst)  # specify the optimization objective
            net.backward(start=end)
            g = src.diff[0]
            # apply normalized ascent step to the input image
            src.data[:] += step_size/np.abs(g).mean() * g

            src.data[0] = np.roll(np.roll(src.data[0], -ox, -1), -oy, -2) # unshift image

            if clip:
                bias = net.transformer.mean['data']
                src.data[:] = np.clip(src.data, -bias, 255-bias)
```

Next we implement an ascent through different scales. We call these scales "octaves".

```
In [ ]: def deepdream(net, base_img, iter_n=10, octave_n=4, octave_scale=1.4,
                      end='inception_4c/output', clip=True, **step_params):
            # prepare base images for all octaves
            octaves = [preprocess(net, base_img)]
            for i in xrange(octave_n-1):
                octaves.append(nd.zoom(octaves[-1], (1, 1.0/octave_scale,1.0/octave_scale), order=1))

            src = net.blobs['data']
            detail = np.zeros_like(octaves[-1]) # allocate image for network-produced details
            for octave, octave_base in enumerate(octaves[::-1]):
                h, w = octave_base.shape[-2:]
                if octave > 0:
                    # upscale details from the previous octave
                    h1, w1 = detail.shape[-2:]
                    detail = nd.zoom(detail, (1, 1.0*h/h1,1.0*w/w1), order=1)

                src.reshape(1,3,h,w) # resize the network's input image size
                src.data[0] = octave_base+detail
                for i in xrange(iter_n):
                    make_step(net, end=end, clip=clip, **step_params)
```

```
                # visualization
                # vis = deprocess(net, src.data[0])
                # if not clip: # adjust image contrast if clipping is disabled
                #     vis = vis*(255.0/np.percentile(vis, 99.98))
                # showarray(vis)
                print octave, i, end #, vis.shape
                clear_output(wait=True)

            # extract details produced on the current octave
            detail = src.data[0]-octave_base
        # returning the resulting image
        return deprocess(net, src.data[0])
```

Each neural network has different layers that can be "hallucinated" into. Let's see which ones are suitable for image making:

```
In [ ]: print 'All network layers:'
        print net.blobs.keys()
        print 'Number of layers unsuitable for image making:'
        print len([x for x in net.blobs.keys() if 'split' in x])
```

0.6 Controlling dreams

The image detail generation method described above tends to produce some patterns more often the others. Note that the neural network we use was trained on images downscaled to 224x224 size. So high resolution images might have to be downscaled, so that the network could pick up their features. The image we use here is already small enough.

Now we pick some target layer and extract guide image features.

```
In [ ]: '''
        Get features from the guide image. Use 'inception_3b/output' by default.
        '''
        def get_guide_features(net, guide, end='inception_3b/output'):
            h, w = guide.shape[:2]
            src, dst = net.blobs['data'], net.blobs[end]
            src.reshape(1,3,h,w)
            src.data[0] = preprocess(net, guide)
            net.forward(end=end)
            return dst.data[0].copy()

In [ ]: def save_array(arr, path):
            PIL.Image.fromarray(np.uint8(arr)).save(path)

In [ ]: def get_objective_guide(_guide_features):
            def objective_guide(dst):
                x = dst.data[0].copy()
                y = _guide_features
                ch = x.shape[0]
                x = x.reshape(ch,-1)
                y = y.reshape(ch,-1)
                A = x.T.dot(y) # compute the matrix of dot-products with guide features
                dst.diff[0].reshape(ch,-1)[:] = y[:,A.argmax(1)] # select ones that match best
            return objective_guide
```

The original experiment use 5 different octave choises, 5 different iteration choices, **all** layers and **all** guide images for the purposes of permutations. Below we'll just use a few.

```
In [ ]: iters = [(octaves, iterations, layers) for octaves in [4,8] \
                for iterations in [4,8,16] \
                for layers in ['inception_3b/output',
                               'inception_4c/output',
                               'inception_4e/output',
                               'inception_5a/output',
                               'inception_5b/output']]

In [ ]: from os import walk, path, remove, makedirs
        import sys

        # Choose the directories that contain the permutable guide and input images.
        guide_images_folder = 'Guide Images'
        input_images_folder = 'Input Images'

        root = WORKING_PATH + '/'
        # On Mac OS X nVidia platforms, we run out of memory above 8 octaves.
        # No Mac today ships with an nVidia card with more than 2GB of VRAM.
        if sys.platform == 'darwin':
            max_octaves = 8
        else:
            max_octaves = 16

        # make folder for the caffe model
        caffe_model_folder_name = root + '/' + caffe_model_name
        if not path.exists(caffe_model_folder_name):
            makedirs(caffe_model_folder_name)

        for (dirpath, dirnames, filenames) in walk(root + guide_images_folder):
            for (guide_path, guide_name) in [(path.join(dirpath, i), i) for i in filenames]:
                # make folder for the guides inside the caffe model folder
                guide_folder_name = caffe_model_folder_name + '/' + \
                    guide_name.replace('.','').replace('_','')
                if not path.exists(guide_folder_name):
                    makedirs(guide_folder_name)

                for (imgdirpath, imgdirnames, imgnames) in walk(root + input_images_folder):
                    for (image_path, image_name) in \
                        [(path.join(imgdirpath, i), i) for i in imgnames]:
                        pilimage = PIL.Image.open(image_path)
                        img = np.float32(pilimage)
                        pilguide = PIL.Image.open(guide_path)
                        guide = np.float32(pilguide)

                        # Make folder for the inputs inside guide folder
                        input_folder_name = guide_folder_name + '/' + \
                            image_name.replace('.','').replace('_','')
                        if not path.exists(input_folder_name):
                            makedirs(input_folder_name)

                        for (octaves, iterations, end) in iters:
                            # Skip layers unsuitable for image making.
                            if 'split' in end:
                                continue
```

```
    if octaves > max_octaves:
        continue
    try:
        path_name = path.join(input_folder_name, '%s_%d_%d_%s.png' % (
                    image_name.replace('.','').replace('_',''),
                    octaves,
                    iterations,
                    end.replace('/','')
                ))

        # Multiprocessing on NFS
        job_name = path_name + '.job'

        if path.exists(path_name) or path.exists(job_name):
            continue

        open(job_name, 'a').close()

        # Clear every round
        objective = get_objective_guide(
            get_guide_features(net=net, guide=guide, end=end))
        image = deepdream(net, img,
                          end=end,
                          objective=objective,
                          iter_n=iterations,
                          octave_n=octaves)
        # showarray(image)
        save_array(image, path_name)
        remove(job_name)
    except (KeyboardInterrupt, SystemExit):
        remove(job_name)
        raise
    except:
        try:
            remove(job_name.replace('//','/'))
        except:
            print 'Multiprocessing error.'
try:
    image = None
    img = None
    guide = None
    pilimage.close()
    pilguide.close()
except:
    print 'Memory freeing produced an exception.'
```