Deep Learning Class MVA 2016
Final Assignment

Grade: Total 10 points + 1 bonus point

T.A.:
Riza Alp Guler – riza.guler@inria.fr
Alexis Conneau – alexis.conneau@gmail.com

Include: Report with answers + code (without including external libraries like MatConvNet, or pretrained models like VGG-ImageNet).

Part 1. Personal research questions (3 points)

Please research the following questions and write concise answers.

Question 1.1. Articles on big image category datasets such as ImageNet often include the "top-5" prediction error together with the "top-1" prediction error. Why is the "top-5" error often seen as a better measure of performance than the "top-1" error?

Question 1.2. The training of DNNs often includes procedures of data *corruption* and data *augmentation*. Give the intuition of why these steps can

- Improve robustness
- Prevent overfitting
- Enforce invariances in the model.

Question 1.3 Besides to adding *momentum*, adaptatively weighted descent methods are often used to improve stochastic gradient descent. A particularly popular one is *AdaGrad*. Research the principle of AdaGrad and explain intuitively how it can improve and fasten convergence to the minimum of an energy function.

For the next two questions read the paper [**yu2015sketch**], *Sketch-a-Net that Beats Humans.*, Qian Yu et al., `http://arxiv.org/abs/1501.07873`", on human sketch recognition with neural networks.

Question 1.4 A human sketches a drawing in a particular order. How do the authors incorportate this ordering in the model? Discuss the benefits of this way of incorporating the order, and its limitations.

Question 1.5 How do authors learn features for different drawing scales? How do they combine the classification results of these different scales together? What other feature fusion strategies could be used (some are suggested in the text)? In particular, think of a Directed Acyclic DNN that would learn this feature fusion in the training.

Figure 1: An image iteratively generated, or "dreamed", by the GoogleLeNet "Inception" trained on MIT Places-205 Dataset.

Part 2. Deep dreaming in MatConvNet (8 points)

Some of the recent research effort on Deep Learning has been devoted to visualizing the image features detected by each layers of a network. In convolutional networks, the first layers often specialize to detect simple features (edges, cominations of edges...) while the later layers detect higher-order combinations of these features (faces, whole objects or object parts...).

One of the ways to visualize the role played by a layer in the detection is by finding an image that activates many of the neurons of this layer. This leads to the notion of "Deep Dreaming", where we don't try to adjust the network's output on a particular image, but instead, we modify the input image to maximize the network output.

Although ideas around Deep Dreaming had appeard before as a visualization of DNN, e.g. in [**simonyan2013deep**], a blog post on Google Research Blog in June 2015 [**mordvintsev2015inceptionism**] widely advertised the idea, with crazy dreams produced by neural networks such as the example in Figure 1 appearing all over the web.

2.1. Deep dreaming: implementation and experiments (4 points)

The idea of deep dreaming is as follows: given a network of N layers, starting from an input image, we would like to iteratively modify the input image pixels to maximize the L^2 norm of a target layer $l \leq N$. In order to do so, we will use backward propagation. If z is an output function that we try to optimize, we have seen that backward propagation allows one to iteratively compute the gradients

$$\underbrace{\frac{\mathrm{d}z}{\mathrm{d}x^{(l)}}}_{\text{dzdx gradient}} \quad \text{and} \quad \underbrace{\frac{\mathrm{d}z}{\mathrm{d}w^{(l)}}}_{\text{dzdw gradient}} \tag{1}$$

where w^l are parameters of layer l, and x^l is the input to layer l. When tuning the parameters of the network, z is a usually loss function, and we are primarily interested in the *dzdw* gradients in order to minimize it.

In deep dreaming, we can set out ouput function z to be the the L^2 norm of layer l, and we can backpropagate the gradient from that layer to compute

$$\frac{\mathrm{d}z}{\mathrm{d}x^{(1)}},$$

i.e. the derivative of this objective function with respect to the input image. We can then do *gradient ascent* on the input image to maximize the objective z, i.e. modify input image with

$$x^{(1)} \leftarrow x^{(1)} + \lambda \frac{\mathrm{d}z}{\mathrm{d}x^{(1)}} \tag{2}$$

where λ is the step size.

We see that we only need to do forward/backward propagation in the network up to some layer l. In the code provided with this assignment, we have split MatConvNet function `vl_simple_nn.m` into three functions `init_res`, `forwardto`, `backwardfrom`, that allow you to initialize the network response, do forward propagation up to some layer, and backward propagation from some layer to the source. See `demo.m` for a sample use of these functions.

We will first experiment with Deep Dreaming with a MatConvNet implementation of VGG "Very Deep" Networks [**simonyan2014very**], contained in `imagenet-vgg-verydeep-16.mat`, that you should get by running `demo.m`. You may use `imresize` to match the expected input size to the network, stored in the variable `net.normalization.imageSize`.

You may have a look at the ipython notebook published by Google on `https://github.com/google/deepdream/blob/master/dream.ipynb` which implements Deep Dreams with the Caffe library with Google Network GoogLeNet for a source of inspiration to your own implementation. *Because of the different network architecture, the dreams you obtain from VGG may be very different from the dreams obtained with GoogleLeNet.*

Question 2.1 If our objective z is the L^2 norm of the response of layer l, contained in $x^{(l+1)}$ (or `res(l+1).x` in Matconvnet), show that the jacobian equals

$$\frac{\mathrm{d}z}{\mathrm{d}x^{(l+1)}} = 2 * x^{(l+1)}$$

Question 2.2 Implement one step of gradient ascent: given an input image, do forward propagation up to layer l with `forwardto`, then backpropagate the gradient of the L^2 norm of layer l to the input image, and modify the input image by following the direction of the gradient, by a factor `step_size` (eq. (2)).

Question 2.3 Peek into VGG: use MatConvNet function `vl_simplenn_display` to have a peek into the architecture of VGG. Why are the last 5 layers different? What is the function of the last layer?

Question 2.4 First experiments: use 3 images of your choice as input images, and normalize them to the right input size for the network. Iterate a few steps (10-50) of gradient ascent on these images. Do the following experiments:

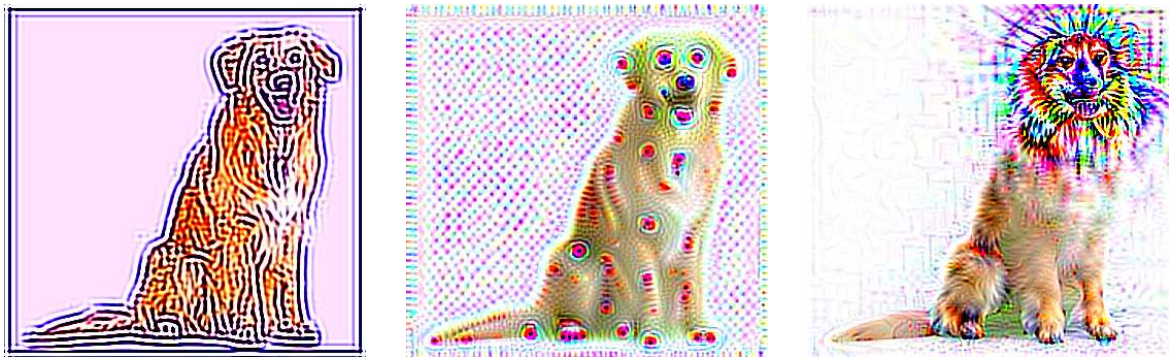

Figure 2: Images obtained by deep dreaming on layers (3), (15) and (27) of VGG-ImageNet respectively, with a methodology similar to *Question 2.4*.

- Monitor the L^2 norm: plot the evolution of the L^2 norm of the target layer with the iteration of gradient ascent to show its optimization. Does it seem to converge to a stable point? You may try to run the optimization for more steps.
- Experiment with the target layers: show that when picking one of the early layers, local features (edges, corners) tend to be enhanced. On the opposite, show that picking one of the later layers draws complex high-level patterns on the image.
- Experiment with step size, and show the influence of this parameter

In these first experiments, you may have difficulty interpreting the patterns produced by the network. In order to improve this, you may introduce some regularization by applying a gaussian blur to the image between each iteration of gradient ascent. In Matlab, this can be achieved with

$$\texttt{h} = \texttt{fspecial}(\texttt{'gaussian'}, \texttt{size}(\texttt{A}), \texttt{1.0});$$
$$\texttt{g} = \texttt{imfilter}(\texttt{A}, \texttt{h});$$

Question 2.5 Bigger input sizes: if you feed the network bigger input sizes that the ones it has been trained on, do the convolutional layers still work the same? Do the completely connected layers work the same?

In the following, you may use bigger input sizes if needed, as long as you optimize objectives related to the convolutional layers.

Question 2.6 Multi-scale dreaming: instead of optimizing the whole image during gradient ascent, select a zoomed-in portion of the input image at each iteration, and do a gradient step on the subimage, before pasting it back to the original image. You may choose the strategy for picking the scale and offset of the image portion (at random, or in a cycle). Show some figures obtained with this multi-scale dreaming procedure.

Question 2.7 Dreaming from noise: instead of slighly modifying input images, try to input noise, and let the gradient ascent procedure invent everything in the image. Show some images obtained.

2.2. Personal experiments (4 points)

The following is a list of experiments that give you some room for personal experimentation with Deep Dreaming. You may get up to 4 points if you do two (or more) of the

following experiments. You may also include other personal experiments if you are able to shed light on interesting properties with Deep Dreaming.

Experiment 1 Modify the network: Sketch-a-Net authors [**yu2015sketch**] made their matconvnet trained networks available on `http://www.eecs.qmul.ac.uk/~tmh/downloads.html`. Experiment with deep dreaming with a Sketch-a-net network. For simplicity, use one of the networks that do not include drawing order information, such as `dataset_without_order_info_256.mat`. Some suggestions of questions:

- See if feeding a natural image to a Sketch-a-net network tends to enhance "drawing-like" features in the image
- Inspect whether the representations learned for sketch classification are very different than the features learned by vgg

Experiment 2 Modifying the objective: the L2 norm of some layer is the objective you have been experimenting with. Experiment with other kind of objectives, show some images obtained and explain whether they meet your intuitions:

- The sum of the norms of several layers at the same time
- The activation norm of a particular ouput neuron (interesting on one of the two last layers – before or after the softmax – if you try to make the image look like a particular image category)
- The dot product between the activations of a layer, and the activations of that layer for another *reference* image (see " Controlling dreams" section on Google's `dream.ipynb` notebook).

Experiment 3 Modify the training dataset: on `http://places.csail.mit.edu/` you might download an implementation of "VGG" trained on MIT Places database. If you import this network to MatConvNet using the script provided my MatConvNet `import-caffe.py`, you should be able to visualize whether you are able to draw more "place-like" features, like in Figure 1. You might also try out the `Places205-GoogLeNet` network.

Further experiments Here are some suggestions for further experiments:

- Gradient ascent often produces images that are difficult to interpret. Gaussian bluring is a simple form of regularization, that enforces correlation of nearby pixels in the input. Come up with other forms of regularization that enforces that the optimization produces *natural images*.
- *Adversarial examples* are images of class a A that are optimized to be classified by the network as another class B. Try to modify the optimization to produce adversarial examples.
- Saliency maps: introduced in [**simonyan2013deep**], *saliency maps* are interesting representations of which pixels a network mainly bases its classification on. The saliency of one pixel p for class C is the magnitude of the derivative of the ouput score for class C with respect to the value of the pixel p. You may visualize some of these saliency maps with little more work than in *Experiment 2*, since you computed this derivative if you tried to maximize the score of an output neuron.

For the interested readers, deep dreaming and related methods give good insight into the internals of neural networks and their failure modes. See e.g. [**simonyan2013deep**, **yosinski2015understanding**] for works on deep visualization methods. See [**szegedy2013intriguing** **nguyen2014deep**, **goodfellow2014explaining**] for works on *adversarial examples*, where

neural nets are tricked into misclassifying images. Some of the ideas of the article can be used in the *further experiments*.

Other approaches have been applied with success to visualize the internals of Neural Networks, including DeConvNets [**zeiler2014visualizing**] and feature inversion [**mahendran2014und**