ECE 157B/272B Homework 2: DeepDream

Due date: Friday, January 28th, 2022, 11:59PM

Introduction

In the last several lectures, we have seen how convolutional neural networks (CNN) can be applied to various image-based machine learning tasks. However, as with most neural networks, CNN's can act like a black box. We know WHAT we train it on (dataset, X, y) , we know HOW it trains (i.e. gradient descent), and we can observe the performance (cross validation). What is difficult to grasp though, is what the (C)NN actually learned.

For this homework we will implement "DeepDream", described in this blog post, both as a method to visualize the patterns learned by a network and to create interesting images. Here is the corresponding Tensorflow tutorial. You will also utilize free GPU/TPU resources available on Google Cloud to speed up computations.

Google CoLab

Google Colab is a Python development environment that runs in the browser using Google Cloud. It is similar Jupyter Notebook but with the latest Tensorflow package pre-installed. You can learn about CoLab's basic features here. We will be mainly touching three features of Colab in this homework:

- Online python editing environment based on Jupyter.
- Virtual machine with private file system that's associated with your Google account.
- Free GPU and TPU resources.

Data Set

The dataset can be any images you want! The provided `.ipynb` contains URLs as starting images, which you may use. Here's the code snippet for loading an image from a URL and clipping it to an smaller size:

```
# Download an image and read it into a NumPy array.
def download(url, max_dim=None):
    name = url.split('/')[-1]
    image_path = tf.keras.utils.get_file(name, origin=url)
    img = PIL.Image.open(image_path)
    # if a max dimension is given, resize the image to smaller size.
    if max_dim:
        img.thumbnail((max_dim, max_dim))
    return np.array(img)
```

The data you work with will also include a purely noise image, which you will have to generate. Remember, RGB has 3 layers and takes *integer* values from 0 to 255.

```
# Return a noisy RGB image
def noisy_image(shape=(500,500,3)):
    # TODO: noise = np.random. ...
    # TODO: scaling, datatype, etc
    return noise
```

Figure 1: Example set of original Images

Workflow

You must perform the following guiding tasks and write down answers to the following questions:

All students (157B/272B)

1. (5 pts) Prepare data and model
   - (1 pt) Select at least 2 images, and download them via URL. Cut down to a maximum size and display them.
   - (1 pt) Generate a RGB image of random noise, of comparable size.
   - (1 pt) Download InceptionV3 model for ImageNet.
   - (2 pts) Write a deprocess function to convert images from the format of InceptionV3 (float -1.0-1.0) back to RGB image format (int 0-255).

2. (5 pts) Select "dream" layers
   - (2 pts) Extract 1-4 layers of interest from InceptionV3, and create a Tensorflow model that takes image input and outputs the activations of these layers.
   - (1 pts) Record which layers you (initially) picked to "dream" from.
   - (2 pts) Explain what you expect to happen to the output image as you pick layers that are deeper (layer 0 is the shallowest, layer 11 is the deepest). (hint: think back to CNN's in HW 1 - what layer usually follows a convolution, and what does it do?)

3. (8 pts) Develop the loss function
   - (1 pts) What is the loss function in DeepDream?
   - (2 pts) What normalizing step(s) is taken in the loss function? Why? (hint: activations are the output of layers - what can differ if you randomly pick two layers to compare?)
   - (2 pts) Write a function to calculate the loss from passing an image through the model.
   - (3 pts) Are we trying to maximize or minimize this loss? Why does your answer make sense in terms of visualizing what the network has learned? (hint: read the blog post)

4. (6 pts) Complete the DeepDream algorithm
   - (4 pts) In homework 1, our loss relates to the error: $L \propto E = prediction - true$. As a result the gradient was computed as $\frac{dL}{dw}$, relating the error (E) and the network weights (w). By adjusting the weights to *descend* this gradient, we hoped for the loss and therefore the error to decrease.
     - (2 pts) What gradient are we computing in the DeepDream algorithm? (i.e. is it still gradient of error with respect to weights, or something else)?
     - (2 pts) Are we still using gradient descent? Explain how the dream algorithm works. (hint: the gradient relates two values - we are trying to maximize/minimize one of those values, so how do we use the gradient?).

- (2 pts) Within the DeepDream(tf.Module) in the notebook, complete the algorithm by appropriately using the calculating the gradient and using as you answered above.

5. (4 pts) Write `run_deep_dream_simple` as a "main" function that takes ONE image: preprocess as necessary, and run the DeepDream algorithm on it. Allow for a variable number of iterations. Have the function output the images and losses at regular intervals.

6. (4 pts) Run your algorithm on each image. (Hint: make sure **runtime −> change runtime type −> GPU** to speed up the processing. CPU only will take about 1 minute per 100 steps per image). Display the evolution of your images - do at least 150 steps and display the progress at every 50. Caption them with #steps and loss. The `show_side_by_side` function you (optionally) write at the top can take a list of images and may be helpful.

7. (3 pts) Read the blog post, if you haven't yet. Now that you've seen your results for an image of all noise, describe the differences with the noisy image dream in the blog post. Can you explain why these differences are so distinct? (hint - remember the gradients. What do our gradients represent? ImageNet is a dataset for image classification, so what gradients might the blog post be using instead to show such results?)

8. (3 pts) Rerun your algorithm on **at least one image** with different parameters (layer(s), step_size, skip gradient normalization, etc...) to try generate something noticeably different. Display results and make any comments. If you cannot find significantly different examples then "Surprisingly, nothing changed" is perfectly valid - BUT include thoughts on **why**.

   BONUS: If you can generate especially interesting images from your noise, you will receive some bonus points. However, this will require a lot of work and should be pursued only if the topic is very interesting to you. Start with the *octaves* step detailed below. Examples of images generated purely from noise from the blog post are shown in the following figure, as a (exemplary) benchmark for "interesting".

Figure 2: Images dreamed from noise. Source

Grad (272B)

1. (6pts) Add "octaves" to the Algorithm

   - (3 pts) Read the section from the Tensorflow tutorial on octaves. Why does resizing the image as we go "allow patterns generated at smaller scales to be incorporated into patterns at higher scales and filled in with additional detail.". (hint: we are still looking at the same layers as in the basic algorithm, so depth is not a factor.)
   - (3 pts) Write `run_deep_dream_octaves` as a "main" function that takes ONE image and performs the DeepDream with "octaves" algorithm. Allow for a variable number of octaves and variable steps per octave. Have the function output images and losses for each octave. (You should be able to easily incorporate `run_deep_dream_simple` within this function).

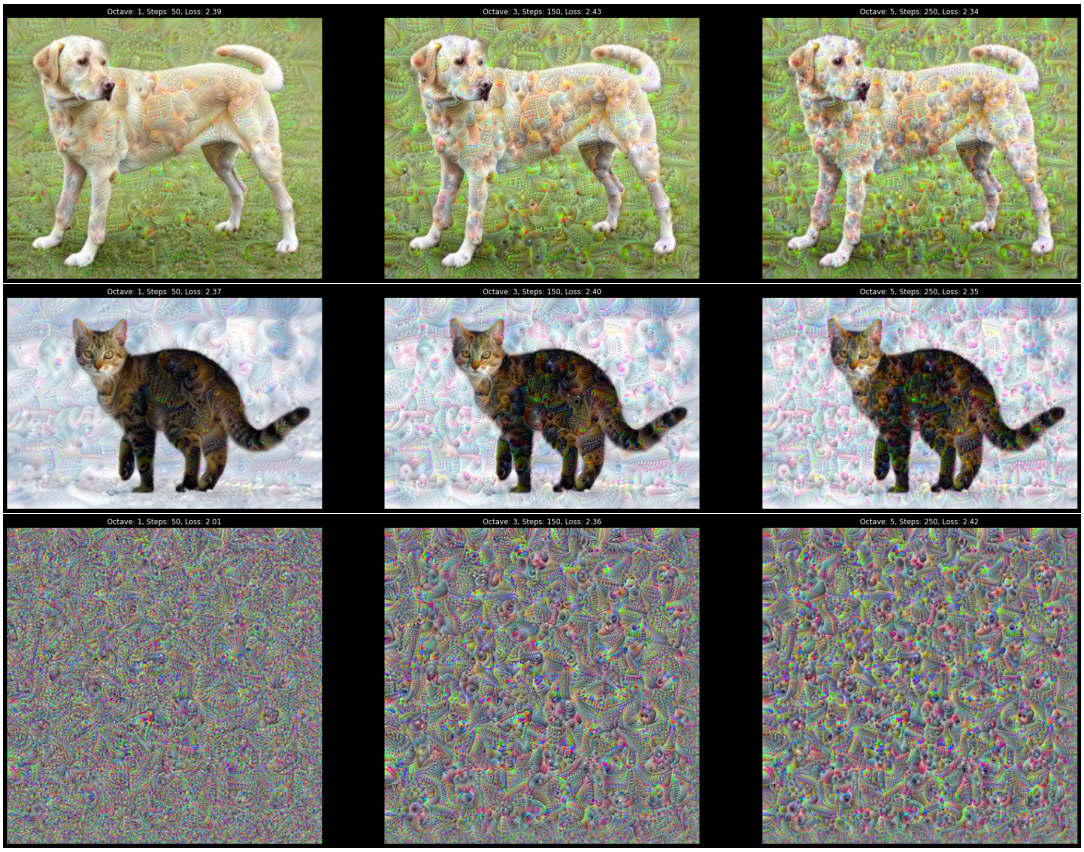

Figure 3: Dream output (including octaves)

2. (4 pts) Display the evolution of your images across the octaves. This means you will have to make sure the images are restored to their proper size after each octave. Do at least 3 octaves, caption them with octave # and loss.