deep-dream

September 7, 2024

[24]:
```
import torch.nn.functional as F
from torchvision import transforms
from torch.utils.data import DataLoader
import numpy as np
import torch
from torch.optim import Adam
from torch import nn
from datasets import load_dataset
from torchvision.transforms import Compose, ToTensor, Lambda, ToPILImage,␣
 ↪CenterCrop, Resize
import matplotlib.pyplot as plt
import math
from tqdm import tqdm
```

[255]:
```
import os

os.environ['CUDA_LAUNCH_BLOCKING'] = '1'

gpu = 1

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Test to see if cpu or gpu is being used
torch.cuda.set_device(device)

# Setting to make GPU computations more efficient
torch.backends.cuda.matmul.allow_tf32 = False
torch.backends.cuda.matmul.allow_bf16_reduced_precision_reduction = False
torch.backends.cudnn.enabled = True
torch.backends.cudnn.benchmark = True
torch.autograd.set_detect_anomaly(False)
torch.autograd.profiler.profile(use_cuda=False)
```

[255]:
```
<unfinished torch.autograd.profile>
```

0.0.1 On DeepDream

In this task we will implement a DeepDream model in order to generate 'dreams': images enhanced by the neural network.

Here I will work with a pre-trained model (Inception-V3). The idea is as follows: Load an image. Load the Inception-V3 model. Pass it in an input and obtain the activation outputs from a certain layer. I 'asked' the neural network to maximise the values of the activation, i.e. enhance the image with what those layer activations is 'liked' to see. For this implement a gradient ascent method in order to maximise a norm of the activations over that input. This will change the input into a 'dream'.

The steps below are summarised are follows

1. A piece of code is written that will load the Inception-V3 model and allow you to get access to the intermediate activations.
2. Implement the deepdream optimisation loop. Use gradient ascent to optimise for the norm of the activation. Implement a function that performs this gradient ascent. This function will take as input the number of iterations to perform, the learning rate, and the start image (i.e. the image which the model will enhance). Define in the function a for loop over the number of iterations, obtain the model output, get the output of the hook, compute the loss which will be L2 norm of this output (i.e. the L2 norm of the activations from a chosen layer), compute the gradients of this L2 norm loss, scale the gradients by their absolute average and define a gradient ascent step over the image. Zero out gradients where needed.
3. Implement code to load an image and display the output from the optimisation loop. The generated images are presented for different choices of layer activations and different number of optimisation steps.
4. Method is changed such that it will allow to compute the loss (L2 norm) of multiple layers' outputs. Presented are three different setups using different numbers & combinations of layers and discuss your results in max. 3 lines.

[1]:
```
import torchvision.models as models

# Import Pre-trained inception
model = models.inception_v3(pretrained=True)

# Put into evaluation mode
model.eval()

# Change in zero_gradient to avoid changes
model.zero_grad()
```

```
C:\Users\brian\abcdefg\Lib\site-packages\torchvision\models\_utils.py:208:
UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be
removed in the future, please use 'weights' instead.
  warnings.warn(
C:\Users\brian\abcdefg\Lib\site-packages\torchvision\models\_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are
```

```
deprecated since 0.13 and may be removed in the future. The current behavior is
equivalent to passing `weights=Inception_V3_Weights.IMAGENET1K_V1`. You can also
use `weights=Inception_V3_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

[13]:
```python
# Keep track of Chosen layer outputs
def output_tracker(outputs):
    def hook(module, input, output):
        outputs.append(output)
    return hook
```

1 Part 2

[14]:
```python
def deepdream_optimization(dog_image, chosen_layer, num_epochs, lr):
    model.eval()

    # Keep track of outputs for chosen layer
    outputs = []
    chosen_layer.register_forward_hook(output_tracker(outputs))

    # Creating new training image
    dog_image_train = dog_image.clone().detach().requires_grad_(True)


    for i in range(num_epochs):
        model.zero_grad()

        # Run model on training image
        model(dog_image_train)

        # Compute L2 norm loss
        loss = torch.norm(outputs[-1])

        loss.backward()

        # Compute scaled gradients
        gradients = dog_image_train.grad / torch.abs(dog_image_train.grad).
 ↪mean()

        # Update image without gradient
        with torch.no_grad():
            dog_image_train += lr * gradients

        # Put the training image into 0, 1
        dog_image_train.data.clamp_(0, 1)
```

```
        # Set gradient to zero
        dog_image_train.grad.zero_()

    dog_image_train = dog_image_train.detach().numpy()

    return dog_image_train
```

2 Part 3

[15]:
```
from PIL import Image
import requests
import io

def load_url(input_image_link):
    # Get permission to use an image
    permission = requests.get(input_image_link)

    dog_image = Image.open(io.BytesIO(permission.content))

    dog_image = torch.tensor(np.array(dog_image)).permute(2, 0, 1) / 255

    return dog_image.unsqueeze(0)
input_image_link = "https://raw.githubusercontent.com/pytorch/vision/main/
 ↪gallery/assets/dog1.jpg"
dog_image = load_url(input_image_link)
```

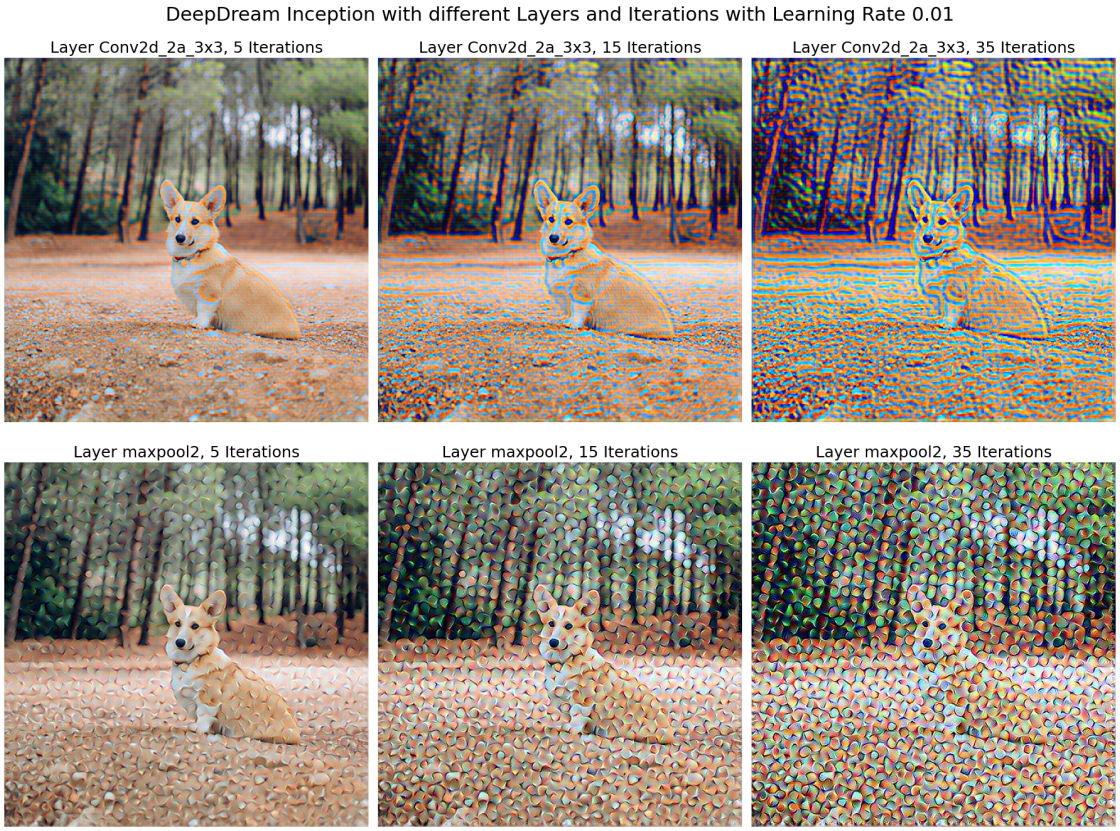
[8]:
```
plt.imshow(np.transpose(dog_image.reshape(3,500,500), (1, 2, 0)))
plt.title('Original Image')
plt.axis('off')
plt.show()
```

[16]:
```
def plot_deepdream_optimization(num_epochs_list, learning_rate, chosen_layers):

    fig, axes = plt.subplots(len(chosen_layers)
                             , len(num_epochs_list)
                             , figsize=(6*len(num_epochs_list),␣
↪7*len(chosen_layers))
                             ,constrained_layout=True)
    fig.suptitle(f'DeepDream Inception with different Layers and Iterations␣
↪with Learning Rate {learning_rate}'
                 ,fontsize = 22)

    for i, (name, chosen_layer) in enumerate(chosen_layers.items()):
        for j, epochs in enumerate(num_epochs_list):
            plot_image = deepdream_optimization(dog_image
                                                , chosen_layer
                                                , epochs
                                                , learning_rate)
            ax = axes[i, j]
            ax.imshow(np.array(plot_image.reshape(3,500,500)).transpose(1,2,0))
            ax.set_title(f"Layer {name}, {epochs} Iterations",
                         fontsize=18)
            ax.axis('off')
```

```
    plt.tight_layout()
    plt.show()
```

[10]:
```
num_epochs_list = [5, 15, 35]
learning_rate = 0.01

chosen_layers = {'Conv2d_2a_3x3': model.Conv2d_2a_3x3,
                 'maxpool2' : model.maxpool2}
plot_deepdream_optimization(num_epochs_list, learning_rate, chosen_layers)
```

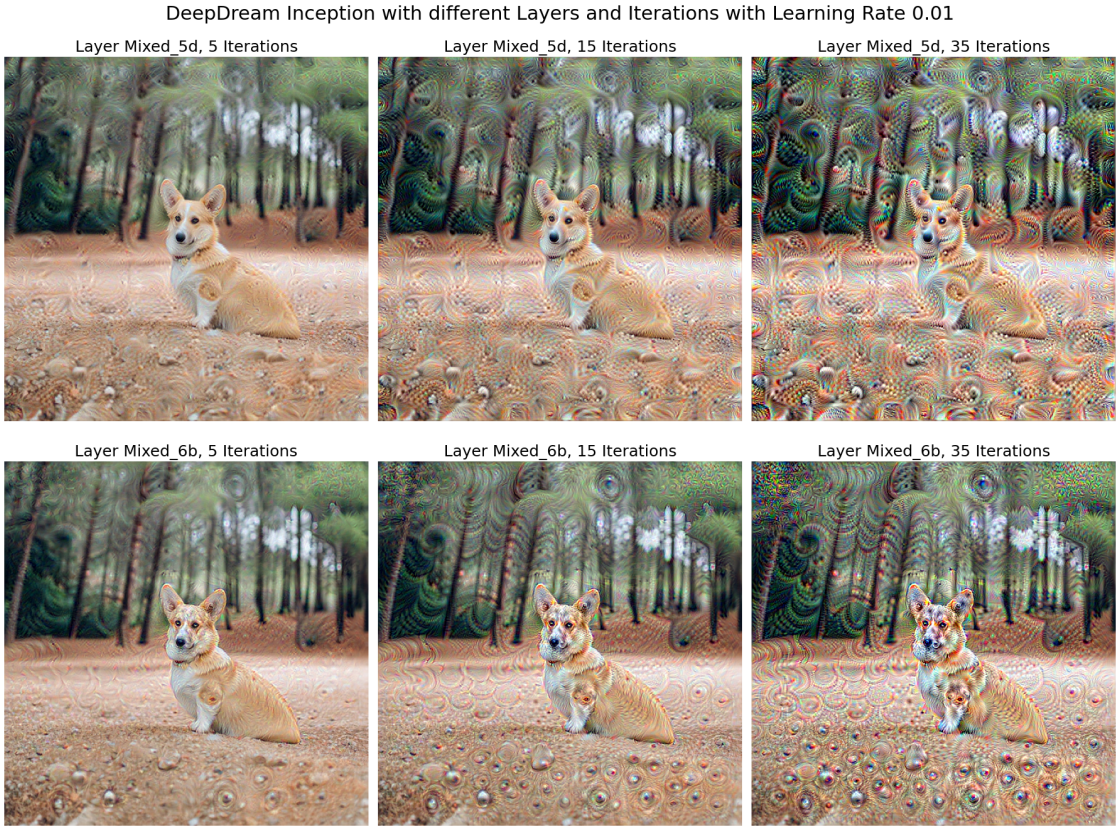
```
C:\Users\brian\AppData\Local\Temp\ipykernel_19108\1916864636.py:21: UserWarning:
The figure layout has changed to tight
  plt.tight_layout()
```



[11]:
```
num_epochs_list = [5, 15, 35]
learning_rate = 0.01


chosen_layers = {'Mixed_5d': model.Mixed_5d,
                 'Mixed_6b' : model.Mixed_6b}
```

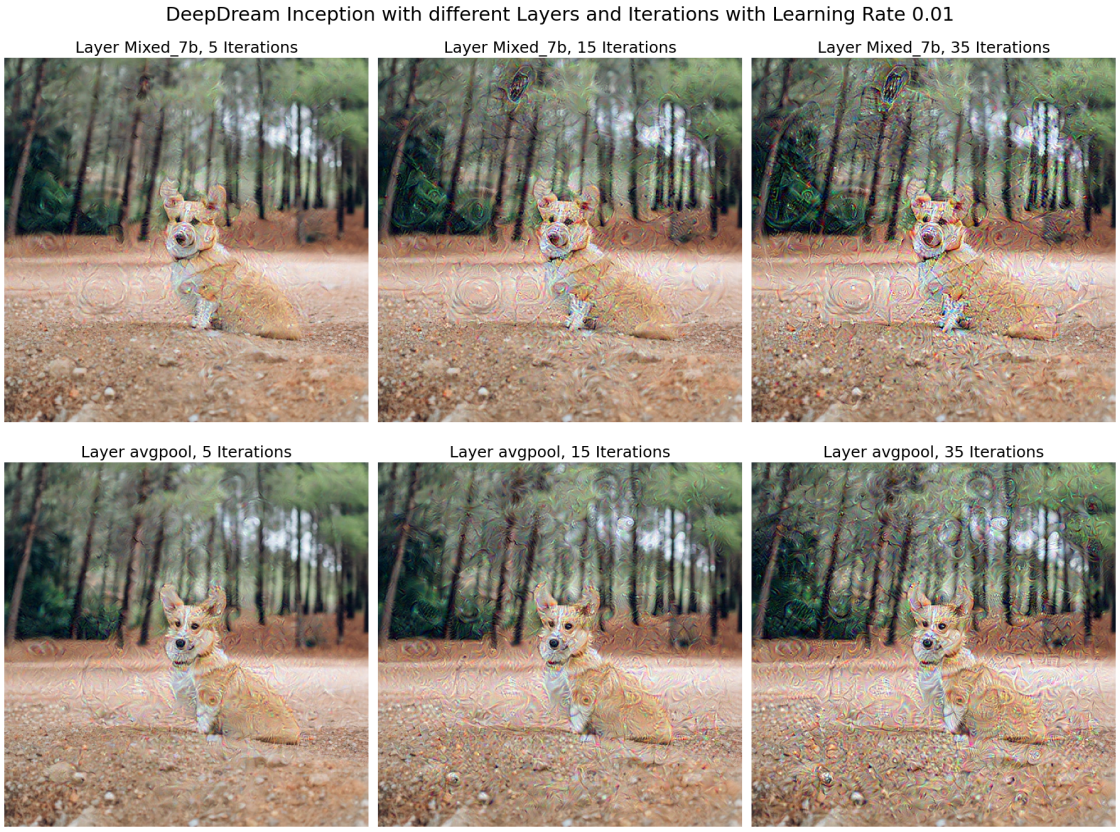
```
plot_deepdream_optimization(num_epochs_list, learning_rate, chosen_layers)
```

```
C:\Users\brian\AppData\Local\Temp\ipykernel_19108\1916864636.py:21: UserWarning:
The figure layout has changed to tight
  plt.tight_layout()
```



[18]:
```
num_epochs_list = [5, 15, 35]
learning_rate = 0.01

chosen_layers = {'Mixed_7b': model.Mixed_7b,
                 'avgpool' : model.avgpool}
plot_deepdream_optimization(num_epochs_list, learning_rate, chosen_layers)
```

```
C:\Users\brian\AppData\Local\Temp\ipykernel_30548\1916864636.py:21: UserWarning:
The figure layout has changed to tight
  plt.tight_layout()
```



To examine the effect of different Inception-V3 model layers on the final image, different pre-trained Inception-V3 layers were tested at different iterations. The layers were choosen such that there is a fair repressentation between layers at different depths in the model.

As seen in the graph above the shallowist layers produced more dotty distortions meaning the red, green and blue parts of the output image are more distinict. The only distinct discriminate changes that the Conv2d_2a_3x3 layer does is when there is large change in colour, for example between the dog's back and the ground or between the trees trucks and the indistinct far background. The distortions in the maxpool2 layer seem to have nothing to do with the underlying image.

As the layers get deeper it is harder to distinguish between the different red, blue and green parts and the distortions inside stop being dotty and become more waivy as in the Mixed_5d and Mixed_6b layers or more scribbly as in the Mixed_7b and avgpool layers.

Another broad observation in the plots is that the ability to see the original image under the distortion decreases as the layers get deeper. The distortions seem to become far more consintrated around the dog in the image and less consintrated around the corners as the target layers become deeper. This shows the deep dream neural network's ability to interpret more complicated structures increases with deeper layers.

As the layers get deeper the image is distorted at more complicated places. This can be seen in the fact that deeper layers tend exacerbate the light in the right background of the image. Particularly in the Mixed_7b and avgpool layers which are the deepest layers plotted above. The level of

distortion around the face of the dog tends to increase dramatically. This is in contrast to the middle layers which tend to expand general features like the the rocks on the foreground and the shapes in the trees along with the dog.

3 Part 4

[8]:
```
def output_tracker2(outputs, name):
    def hook(module, input, output):
        outputs[name].append(output)
    return hook
```

[9]:
```
def deepdream_optimization2(dog_image, chosen_layers, epoch, learning_rate):

    # List of outputs and output layers
    outputs = {name: [] for name in chosen_layers.keys()}

    # Register each layer
    for name, layer in chosen_layers.items():
        layer.register_forward_hook(output_tracker2(outputs, name))

    dog_image_train = dog_image.clone().detach().requires_grad_(True)


    for i in range(epoch):
        # Set zero gradient
        model.zero_grad()

        # Run image to get outputs
        model(dog_image_train)

        loss = sum(torch.norm(outputs[name][-1]) for name in outputs)

        loss.backward()

        gradients = dog_image_train.grad / torch.abs(dog_image_train.grad).
↪mean()

        with torch.no_grad():
            dog_image_train += learning_rate * gradients

        dog_image_train.data.clamp_(0, 1)

        # Set gradient to zero
        dog_image_train.grad.zero_()


    dog_image_train = dog_image_train.detach().cpu().numpy()
```

```
    return dog_image_train
```

[10]:
```
def plot_deepdream_optimization2(num_epochs_list, learning_rate,␣
↪chosen_layers_list, titles):

    fig, axes = plt.subplots(len(chosen_layers_list)
                             , len(num_epochs_list)
                             , figsize=(6*len(num_epochs_list),␣
↪7*len(chosen_layers_list))
                             ,constrained_layout=True)

    fig.suptitle(f'DeepDream Inception with different Layer combinations and␣
↪Iterations with learning rate {learning_rate}'
             ,fontsize = 22)

    for i, chosen_layers in enumerate(chosen_layers_list):
        for j, epoch in enumerate(num_epochs_list):
            final_image = deepdream_optimization2(dog_image
                                                  , chosen_layers
                                                  , epoch
                                                  , learning_rate)
            ax = axes[i, j]
            ax.imshow(np.transpose(final_image.squeeze(), (1, 2, 0)))
            ax.set_title(f"Layers: {titles[i]} Iterations: {epoch}",
                         fontsize=18)
            ax.axis('off')
    plt.tight_layout()
    plt.show()
```

[11]:
```
num_epochs_list = [5, 15, 35]
learning_rate = 0.01

chosen_layers1 = {
    'Conv2d_2a_3x3': model.Conv2d_2a_3x3,
    'maxpool2': model.maxpool2,
}

chosen_layers2 = {
    'Mixed_5d': model.Mixed_5d,
    'Mixed_6b': model.Mixed_6b,
    'Mixed_6d': model.Mixed_6d,
}

chosen_layers3 = {
    'Conv2d_1a_3x3': model.Conv2d_1a_3x3,
    'Mixed_6b': model.Mixed_6b,
```

```
    'Mixed_7b': model.Mixed_7b,
}


titles = ['Shallow Layers', 'Middle Layers', 'Mixed Layers']

chosen_layers_list = [chosen_layers1, chosen_layers2, chosen_layers3]

plot_deepdream_optimization2(num_epochs_list,learning_rate,chosen_layers_list,␣
  ↪titles)
```

```
C:\Users\brian\AppData\Local\Temp\ipykernel_30548\1374640519.py:22: UserWarning:
The figure layout has changed to tight
  plt.tight_layout()
```

To put it simply the combination of layers above seem to be the same as the addition of the distortion from each layer individually.

The shallow layers seems to have minimal relation to the underlying image. The distortion from the middles layers seems to be related to the curves in the dog, the rocks and the tree trucks. In the mixed layers it notably shows both the scribbly lines and focus in the Mixed_7b layer and also the curves from the Mixed_6b layer.