CNN Visualization and Adversarial Patterns

Deep Learning (SS2020)
5. computer lab (10p)

Introduction

In this lab we will consider a CNN classifier and visualize *activations* and *attention maps* for hidden layers, look for input patterns that *maximize activations* of specific neurons and see to how craft *adversarial attacks* fooling the network. All of these tasks share very similar techniques.
For this lab, we recommend you to use jupyter notebooks, as computations are relatively light and we need lots of visualization. We provide a *template notebook* as a starter. It loads the vgg11 network, its respective classes and a test image. It also provides visualization examples and some useful functions. The network model has the following `features` part:

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (11): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): ReLU(inplace=True)
    (13): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (14): ReLU(inplace=True)
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (16): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace=True)
    (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (19): ReLU(inplace=True)
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
...
```

We will visualize outputs of all of these 21 *feature maps*.

Assignments

Assignment 1 (Network Attention)**. (2p)**
The template shows how to compute the l_2 norms over the activation channels for each of the 21 *feature maps* and to display them. You task is to visualize the "network attention" by computing the gradient w.r.t. this intermediate outputs.

1. For a target layer $l = 0 \dots 20$ compute the feature map at that layer (as shown in the template). Compute the gradient of the network classification score for the predicted class w.r.t. this feature map. You'll need to detach the feature map from the graph and set `requires_grad=True`. Then you need to forward propagate the network to its final output (score). Finally, you may use either `.backward()` or `torch.autograd.grad` to compute the gradient in the feature map of interest.

2. Display a panel of images as in "Visualization 1" showing the gradient strength (e.g. sum of absolute values over all channels of a feature map) w.r.t. each intermediate feature.

3. You should observe stronger gradients in locations that were important for the current prediction. This holds also for lower layers, contrasting with their activation maps, which are strong on all edges. Try to see this difference on some other image of your choice from the internet.

Assignment 2 (Network Filters)**. (4p)**
The linear activations of the first layer are maximized when the input matches the filter (scalar product $\langle w, w \rangle$ is the strongest). Let's find out about deep features!

1. Extract the convolutional weight tensor from the first convolutional layer of the network and display the weights as a panel of RGB images.

2. Complete the function `show_activation_max` that will display the patterns that maximize activations of deeper layers. Use the following approach. We know how to compute the activation map y for a given layer l. To find the input that maximises the activation of an unit in that layer, we will numerically optimize over a patch of the input image.

   - Choose the layer, e.g. $l = 8$.
   - Start from an image x of small size, initialized with zeros. The size should be equal to the receptive field of the unit in the target layer (see function `receptive_field`).
   - Forward propagate x through the network to compute the feature map y of the target layer.
   - Select the centrally located pixel and the target channel in the feature map y .
   - Use the Adam optimizer to maximize the selected feature (i.e. forward-backward loop with optimizer steps). Make sure you optimize only in the input image and not in the network parameters.
   - Find such an activating image x for each channel of the target layer and display them in a panel. If you like to speed-up the optimization, it can be done in parallel over a batch of activation images, one per target channel.

   The obtained optimal patterns are not resembling patches of natural images. We will therefore add a regularizer that enforces more realistic patterns.

3. Add a regularization that enforces smooth patterns and solve the problem:

$$\min_x \left[-y_{c_0,i_0,j_0}(x) + \lambda \, \text{reg}(x) \right], \tag{1}$$

   where c_0 is the target channel and (i_0, j_0) is the central pixel in the feature map y. The regularization function reg(x) is provided in the code template and penalizes spatial gradients of the input image.

$$\text{reg}(x) = \Big(\sum_{c,i,j} |x_{c,i+1,j} - x_{c,i,j}|^q \Big)^{\frac{1}{q}} + \Big(\sum_{c,i,j} |x_{c,i,j+1} - x_{c,i,j}|^q \Big)^{\frac{1}{q}}, \tag{2}$$

   where the norm parameter q can be set close to 1 to promote sparsity of gradients.
   In the implementation, you just need to add reg(x) to your objective.

4. Tune the regularization strength so that the optimal activation patterns resemble natural image patches. Of course, if we could constrain the optimization to the manifold of natural images, the results would be even better.

Assignment 3 (Adversarial Pattern)**. (4p)**
The template code shows how to craft an untargeted single step adversarial attack. You will implement a targeted iterative adversarial attack.

1. Choose an input image and verify that it is correctly classified by the net.

2. Choose a target class different from the true class and fix an $\varepsilon > 0$. Implement a projected gradient ascent that aims to maximize the softmax output of the target class w.r.t. the input image, but constrains the search to the ε-ball around the clean image.

- Start the optimization from the clean image.
- You may use the Adam optimizer for computing the gradient and performing the gradient step. For this you have to require gradients for the input image and to disable gradients for the network parameters.
- To enforce the constraint, you may e.g. use the following code after each gradient step

```
dx = (x.detach() - x0)
dx = dx / dx.flatten().norm() * eps
x.data = x0 + dx
```

  where x_0 is the clean image (tensor) and x is the current image (tensor).

3. Run the projected gradient ascent for a fixed number of steps.
4. Report the ε which admits a succesful attack, show the obtained adversarial example along with the clean image and report the prediction probabilities for them.