Assignment 2 - Convolutional Networks

Administrative details

Deadline: 06.04.2023, 23:59

Scoring: 2 points. After the deadline, there is a 0.25 points penalty per day, for 4 days. If you delay your submission by more than 4 days, the maximum score for the assignment is 1 point. You will personally present the assignment to a teaching assistant in the assignments evaluation week (week 9).

Questions: Need help? Use the #*Assignments* Teams channel.

Submission: Upload link: `https://forms.gle/gwhSeVw6ZBY26Ehk6`. The assignment should be a python Notebook that contains the code and presents the results of the experiments. You can start from the following notebook: `https://drive.google.com/file/d/1Zv1v1Q--ADm0uA7xgGU-hezzJZ9R0RlB/view?usp=share_link`

Assignment

What will you learn? The goal of this assignment is to better understand how convolutional neural networks work. For this you will implement some basic building blocks, like fully connected layers and 2D convolutional layers.

Dataset Description We will use image datasets, like CIFAR 10 that contains small 32×32 images.

Tasks

We will guide you step by step through the necessary steps for solving each sub-task, providing you also details on how to verify that things work as expected.

Task 1. Classification task - 0.5 points

In this part of your assignment, you should train a neural network to classify the images into 10 classes.

Task 1.0. Make a new notebook in `https://colab.research.google.com`, mount your google drive and load the standard training and testing splits of CIFAR 10. You can use the dataloaders from torchvision [1].

Task 1.1. (0.1 points) Define a convolutional network Define a standard convolutional network using the standard PyTorch modules like: torch.nn.Conv2d, nn.MaxPool2d, nn.Linear. Your architecture should be the following: conv layer (8 kernels of size 5x5), ReLU, 2x2 maxpool, conv layer (16 kernels of size 5x5), ReLU, 2x2 maxpool, fully connected layer (128 output channels), ReLU, fully connected layer (64 output channels), ReLU, fully connected layer (10 output channels)

[1] `https://PyTorch.org/vision/stable/datasets`

Task 1.2. (0.4 points) Training and Evaluation. Train your model on CIFAR 10 using the appropiate loss function for classification. Make sure you use the GPU from the colab environment. Keep track of your performance for the training and validation set and plot them at the end of the training (both the loss value and the accuracy).

Task 2. Implement a fully connected layer - 0.3 points

A fully connected layer implements a linear operation, that processes a single input x of size c_{in}:

Parameters $\mathbf{b} \in \mathbb{R}^{c_{out} \times 1}, \mathbf{W} \in \mathbb{R}^{c_{out} \times c_{in}}$

$$\mathbf{y} = \mathbf{W}\,\mathbf{x} + \mathbf{b} \tag{1}$$

Input $\mathbf{x} \in \mathbb{R}^{c_{in} \times 1}$

But we usually want to process a batch of samples, meaning a set of B inputs. Thus we receive a batch of samples $\mathbf{x} \in \mathbb{R}^{B \times c_{in} \times 1}$ and we want to apply the same operation (use the same parameters) to each element s in the batch.

$$\mathbf{y}_s = \mathbf{W}\mathbf{x}_s + \mathbf{b} \tag{2}$$

$$\mathbf{x} \in \mathbb{R}^{B \times c_{in} \times 1}, \mathbf{y} \in \mathbb{R}^{B \times c_{out} \times 1}, \mathbf{b} \in \mathbb{R}^{c_{out} \times 1}, \mathbf{W} \in \mathbb{R}^{c_{out} \times c_{in}}$$

For understanding, it can be helpful to expand the previous equation for each element in the output (each neuron d, of each sample s in the batch).

$$\mathbf{y}_{s,d} = \sum_{c=0}^{c_{in}-1} \mathbf{W}_{d,c}\mathbf{x}_{s,c} + \mathbf{b}_d \tag{3}$$

Task 2.1. (0.3 points) Write a PyTorch module that implements a fully connected layer as defined by Equation 2, without using the standard PyTorch modules (e.g. without nn.Linear). Use the following snippet provided in the starting code.

```
class myLinear(nn.Module):
  def __init__(self, in_features, out_features, device=None):
    super().__init__()
    self.weight = torch.nn.parameter.Parameter...
    self.bias = torch.nn.parameter.Parameter...

  def forward(self, x):
    # we want to do matrix multiplication between the weights
    # and each element in the batch and further add the bias
    # as in Equation 2
    ...
```

Define two layers, one using nn.Linear and one using your implementation and run the same random input through both. Compute the mean absolute difference between the outputs. Copy the parameters from the default layer into your layer and recompute the difference.

```
# define two linear layers
linear      = nn.Linear(8,16)
my_linear   = myLinear(8,16)
# copy the parameters from linear into my_linear
my_linear.load_state_dict(linear.state_dict())
# TODO: compute the difference betwen the two layers
#  on random input
```

Task 3. Implement a convolutional layer - 1.2 points

As explained in more detail in Lecture 5, a 2D convolutional layer takes as input a sample with c_{in} input channels and spatial size of $h \times w$ and processes it with a series of c_{out} learnable filters, each of size $c_{in} \times k \times k$. For each of the c_{out} output channels dimension, we also have a learnable bias parameter. The entire operation can be described by the following equation:

$$\mathbf{y}_{s,d,i,j} = \sum_{c=0}^{c_{in}-1} \sum_{n=0}^{K-1} \sum_{m=0}^{K-1} \mathbf{x}_{s,c,i+n,j+m} \cdot \mathbf{W}_{d,c,n,m} + \mathbf{b_d} \tag{4}$$

$$\mathbf{x} \in \mathbb{R}^{B \times c_{in} \times h \times w}, \mathbf{b} \in \mathbb{R}^{c_{out} \times 1}, \mathbf{W} \in \mathbb{R}^{c_{out} \times c_{in} \times k \times k}$$

Task 3.1. (0.6 points) Write a PyTorch module that implements a 2D convolutional layer as defined by Equation 4, without using the standard PyTorch modules (e.g. without nn.Conv2d).

As before compute the mean average difference between a default nn.conv2d and your implementation using random images from the dataset.

Task 3.2. (0.3 points) Use the same architecture as in Task 1 to define a new network using the modules that you implemented in Tasks 3 and 4. Take the parameters of the network obtained in Task 2 and use them as parameters for the newly defined network. As in Task 3 you can use load_state_dict and net.state_dict() to copy the weights of an existing model.

Compute the accuracy of both networks on a small number of testing samples (e.g. 20 samples). Compute the average time for inference of both networks and compute the ratio between their duration (own / default). The default implementation should be significantly better that the naive implementation and should always be preferred from now on.

Task 3.3. (0.3 points) Plot the activation of intermediate feature maps. By processing an image by a convolutional network we obtain different intermediate features maps, i.e. the activations after each layer in the network. We can gain insights into the features learned by the network if we visualize these features maps and see where do they strongly activate (what are the important zones in the image that produce strong response by the network).

During passing through the network, we obtain features maps after each convolutional and each pooling layers. Visualize the first 5 channels of the feature maps corresponding to all convolutional and pooling layers in the network for 3 images. Normalise each image to be in interval $[0, 1]$.

An example of visualisation can be seen in Figure 1. We can observe some correlations: the second filter seems to be activating on animals eyes. Although it is tempting to generalise easily the observations of the feature maps, in order to make objective statements about the learned features we need more rigorous testing.

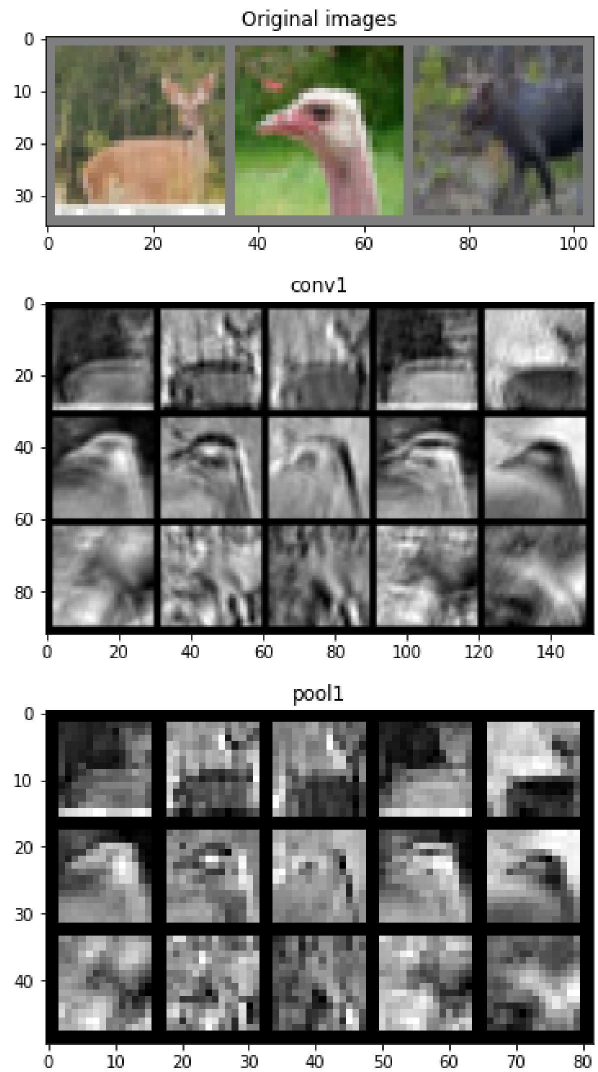

Figure 1: Visualisation of the first 5 channels of the feature maps corresponding to the first convolutional and first pooling layer.