CSCI 677: Advanced Computer Vision - Fall 2021

Instructor: Prof. Nevatia

Assignment 4

Due on November 4, 2021 before 17:00 PDT

Instructions

This is a programming assignment to create, train and test a CNN for the task of image classification. To keep the task manageable, we will use a small dataset and a small network.

Architecture

Construct a LeNet-5 style CNN network, using PyTorch functions. LeNet-5 is shown in Figure 1. Note that the network is not exactly the same as described in the original, 1998, paper.

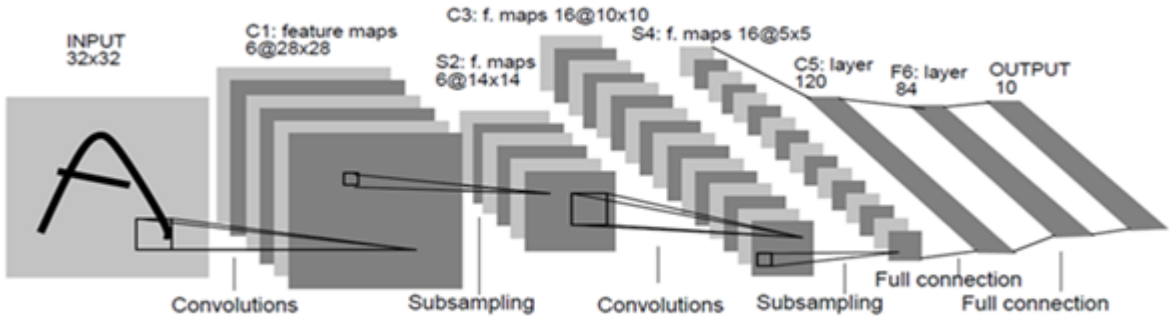

Figure 1: LeNet-5 Architecture

We ask you to experiment with varying the parameters of the network but use the following to start with (which is referred as 'main experiment' in the following sections):

1. The first layer has six 5×5 convolution filters, stride as 1, each followed by a max-pooling layer of 2×2 with stride as 2.
2. Second convolution layer has sixteen, 5×5 convolution filters, stride as 1, each followed by 2×2 max pooling with stride as 2.
3. Next is a fully connected layer of dimensions 120 followed by another fully connected layer of dimensions 84.
4. Next is a fully connected layer of dimensions 10 that gives unnormalized scores of the 10 classes.
5. All activation units should be ReLU.

Note: The above design does not include a softmax layer as, in PyTorch implementation, nn.CrossEntropyLoss() includes the LogSoftmax function; at inference (test) time, you will have only unnormalized scores available but these are sufficient for classification. You may add an explicit softmax layer if you prefer.

Framework

PyTorch is the required framework to use for this assignment. You are asked to define your model layer-by-layer by using available functions from PyTorch. Please do not import available definitions from the Internet though you are free to use them as a guide.

Dataset

We will use the STL-10 dataset. It consists of 10 mutually exclusive classes with 5,000 training images and 8,000 test images, evenly distributed across the 10 classes. Each image is a 96×96 RGB image. You can download the Python version of the dataset from https://cs.stanford.edu/~acoates/stl10/. You can refer to this code https://github.com/mttk/STL10 for downloading, extracting and parsing the data. For 800 test images in each category in the STL-10, please use 300 of them to construct the validation set and remaining 500 as test set for your experiment. You can use the code attached to this document for downloading and spliting the dataset into train, validation and test splits. Images and labels used for each split is stored in the folder `splits` after you run the code.

Training

Train the network using the given training data. For the main experiment setting, we suggest starting with a mini-batch size of 128, ADAM optimizer with initial learning rate of 1e-3 for no more than 100 epochs and decaying the learning rate by 50% after every 20 epochs. You are free to experiment with other learning rates and other optimizers. Please use cross entropy loss for your main experiment. Record the error after each step (i.e. after each batch) so you can monitor it and plot it to show results. During training, you should test on the validation set at some regular intervals; say every 5 epochs, to check whether the model is overfitting.
Note: Shuffle the training data after every training epoch for a better fit. To plot the loss function or accuracy, you can use pylab, matplotlib or tensorboard to show the curve.

Preprocessing

The size of STL-10 images is 96×96 while LeNet takes images of size 32×32. For your main experiment, you are encouraged to resize the input images to 32x32 with https://pytorch.org/vision/stable/transforms.html#torchvision.transforms.Resize. You should normalize the images to zero mean and unit variance for pre-processing. First normalize your images to (0,1) range, calculate the dataset mean/std values, and then normalize the images to be zero mean and unit variance. You can check https://pytorch.org/vision/stable/transforms.html#torchvision.transforms.Normalize to apply the normalization in your code.

Test Result

Test the trained network on the test data to obtain classification results and show the results in the form of confusion matrix and classification accuracies for each class. For the confusion matrix you could either write the code on your own, or use scikit-learn to to compute the confusion matrix. (See: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html for more details).

Variations

The above defined LeNet-5 network does not have normalization or regularization implemented. Similar to the main experiment, conduct additional experiments using batch normalization and L-2 regularization of the trainable weights (independently).

Hint: For Adam optimizer in PyTorch, there are two different implementations, Adam and AdamW. You can compare their implementations, https://pytorch.org/docs/stable/generated/torch.optim.Adam.html#torch.optim.Adam for Adam and https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html#torch.optim.AdamW for AdamW, to see if you can find any information useful for this task.

SUBMISSION

For your submission, include 1) your source code and 2) a report. Please follow the following instructions for preparing your submission.

1. For your main experiment setting, show the evolution of training losses and validation losses with multiple steps.
2. Show the confusion matrix and per-class classification accuracy for this setting.
3. Show some examples of failed cases, with some analysis if feasible.
4. Compare your results for the variations with the main experiment setting.

For the source code, we encourage you to submit the code of the main experiment setting with the variation of the settings mentioned above. For your report, you should include the results of both main experiment settings and those with different experiment variations.

Hint

Following are some general hints on structuring your code, it is not required to follow this template.

1. You need to create
   - *Dataset and dataloader* Dataset class is to do preprocessing for the raw data and return the specific example with the given ID. It can coorperate with Dataloader for selecting the examples in the batches. Please check https://pytorch.org/docs/stable/data.html# for details.
   - *Loss Function* calculates the loss, given the outputs of the model and the ground-truth labels of the data.
   - *Model* This is the main model. Data points would pass through the model in the forward pass. In the backward pass, using the backpropagation algorithm, gradients are stored. Please write your own LeNet-5 model instead using the pre-built one in torchvision.
   - *Optimizer* These are several optimization schemes: Standard ones are available in Pytorch; we suggest use of ADAM, though you could also try using the plain SGD. You would be calling these to update the model (after the gradients have been stored).
   - *Evaluation* Compute the predictions of the model and compare with the ground-truth labels of your data. In this homework, we are interested in the top-1 prediction accuracy.
   - *Training Loops* This is the main function that will be called after the rest are initialized.
2. There is an official PyTorch tutorial online, please refer to this tutorial for constructing the above parts: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
3. Apart from the above, it is suggested to log your results. You could use TensorBoard (compatible with both PyTorch/TensorFlow) or simply use 'logger' module to keep track of loss values, and metrics.
4. Note on creating batches: It is highly recommended to use the DataLoader class of PyTorch which uses multiprocessing to speed up the mini-batch creation.

Notes on using PyTorch Dataset Class

PyTorch supports two different types of datasets: map-style datasets and iterable-style datasets. A example of map-style dataset is shown as below:

```
from torch.utils.data import Dataset
class CIFAR10(Dataset):
    def __init__(self, *args):
        # Initialize some paths
    def __len__(self):
        # return the length of the full dataset.
        # You might want to compute this in
        # __init__ function and store it in a variable
        # and just return the variable, to avoid recomputation
        return len
    def __getitem__(self):
        # Somehow get the image and the label
        img_file, label = get_img_label()
        # read the image file
        img_pil = PIL.Image.open(img_file)
        img_transformed = transforms(img_pil)
```