CSCI 677: Advanced Computer Vision - Fall 2024

Instructor: Prof. Nevatia

Assignment 4

Due on October 31, 2024 before 15:00 PDT

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 two small networks.

Architecture

You are asked to construct and experiment with two, relatively small, CNNs. First is a LeNet-5 network; the second is ResNet-9. Details of both are given below. The figures include filter sizes and number of channels for both. Please define the network architecture like:

```
class LeNet5(nn.Module):
    def __init__(self, num_classes=10):
        super(LeNet5, self).__init__()
        # TODO
    def forward(self, x):
        # TODO
```

LeNet-5

A sketch of LeNet-5 is shown in Figure 1; the network is a bit different than described in the original, 1998, paper. All convolutions are across all channels of the layer a filter is applied to. All activatiion units should be ReLU. Note that the architecture does not include a softmax layer but you may add so if you prefer.

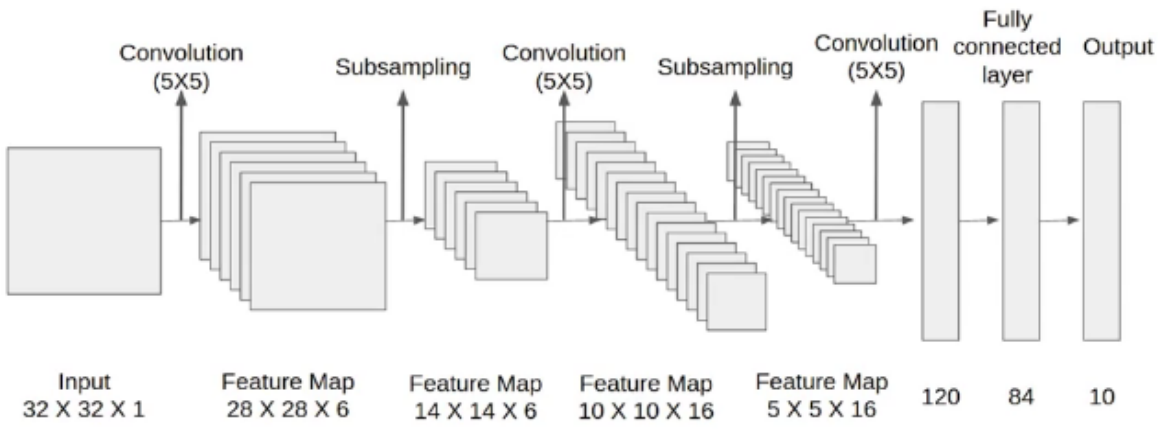

Figure 1: LeNet-5 Architecture

ResNet-9

A sketch of ResNet-9 is shown in Figure 2.

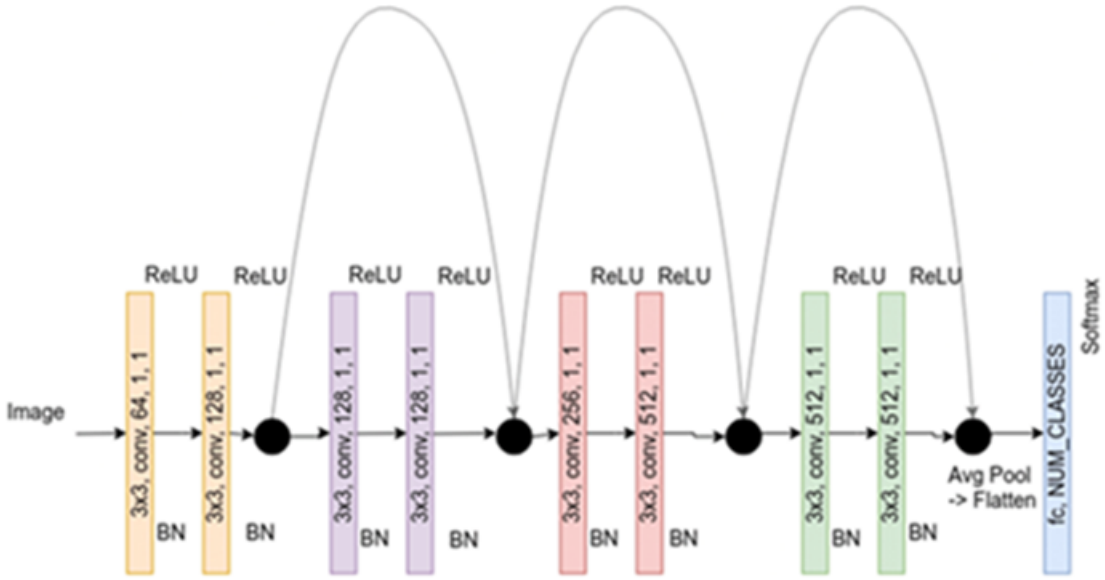

Figure 2: ResNet-9 Architecture

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. You are free to use PyTorch document or other sources to help but do not import pre-defined LeNet or ResNet definitiions.

Dataset

We will use the CINIC-10 dataset. It consists of 10 mutually exclusive classes with a total of 270,000 images equally split amonst three subsets: train, validate, and test. Each image is a 32×32 RGB image. You can download the dataset from https://drive.google.com/file/d/1ZEWAU7k0lTEmabjEmtrd9SGsmNGHEVIC/view?usp=sharing. Afte extracting the downloaed file, it should contains 3 data folders: *test*, *train*, and *valid*, each contains 10 subfolders indicating 10 classes. Each subfolder consists of 9,000 images. To load CINIC data, you may refer to the demo code provided below:

```
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

class CINIC10Dataset:
    def __init__(self, batch_size=64, root='.'):
        self.transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(( , , .), ( , , .))
        ])
        self.batch_size = batch_size

        self.train_dataset = datasets.ImageFolder(root=f'{root}/train', \
        transform=self.transform)
        self.train_dataoader = DataLoader(self.train_dataset, batch_size=batch_size, \
        shuffle=True, num_workers=2)

```

```
        self.valid_dataset = datasets.ImageFolder(root=f'{root}/valid', \
        transform=self.transform)
        self.valid_dataoader = DataLoader(self.valid_dataset, batch_size=batch_size, \
        shuffle=False, num_workers=2)

        self.test_dataset = datasets.ImageFolder(root=f'{root}/test', \
        transform=self.transform)
        self.test_dataoader = DataLoader(self.test_dataset, batch_size=batch_size, \
        shuffle=False, num_workers=2)
```

Note: You should normalize the images to zero mean and unit variance for pre-processing. First normalize your image pixel values to (0,1) range, calculate the dataset mean/std values, and then normalize the images to be zero mean and unit variance.

Training

Train the networks using the given training data for 20 epochs. For the main experiment setting, we suggest starting with a mini-batch size of 128, ADAM optimizer with learning rate $\alpha = 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. 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: To plot the loss function or accuracy, you can use pylab, matplotlib or tensorboard to show the curve.

Test Results

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). Also compare the LeNet-5 and ResNet results. We have also not asked for use of augmentations; you are encouraged to experiment with them but this is not required as our dataset is relatively simple.

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 accuracy with multiple steps (training log + curves) for the two networks
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 two networks and any variations you may have tried.

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.

Hints

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. A demo code snippet for evaluation is provided below for reference.

```python
import torch

def compute_accuracy(model, dataloader, device='cuda'):
    model.eval()  # Set the model to evaluation mode
    correct_predictions = 0
    total_predictions = 0

    with torch.no_grad():
        for images, labels in dataloader:
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)

            _, predicted_labels = torch.max(outputs, 1)

            correct_predictions += (predicted_labels == labels).sum().item()
            total_predictions += labels.size(0)

    # Calculate accuracy
    accuracy = correct_predictions / total_predictions * 100
    return 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.