CS540 Fall 2023 Homework 7

Assignment Goals

- Implement and train a convolutional neural network (CNN), specifically LeNet
- Understand and count the number of trainable parameters in CNN
- Explore different training configurations such as batch size, learning rate and training epochs.
- Design and customize your own deep network for scene recognition

Summary

Your implementation in this assignment might take one or two hours to run. We highly recommend starting working on this assignment early! In this homework, we will explore building deep neural networks, particularly Convolutional Neural Networks (CNNs), using PyTorch. Helper code is provided in this assignment.

In this HW, you are still expected to use the Pytorch library and virtual environment for programming. You can find the relevant tutorials in HW6 Part I.

Design a CNN model for MiniPlaces Dataset

In this part, you will design a simple CNN model along with your own convolutional network for a more realistic dataset – MiniPlaces, again using PyTorch.

Dataset

MiniPlaces is a scene recognition dataset developed by MIT. This dataset has 120K images from 100 scene categories. The categories are mutually exclusive. The dataset is split into 100K images for training, 10K images for validation, and 10K for testing.

Figure 1: Examples of images in miniplaces

Our data loader will try to download the full dataset the first time you run **train_miniplaces.py**. If the miniplaces website is down, follow these instructions to set up miniplaces manually:

1. Create an empty data folder in the same directory as **train_miniplaces.py** and download the backup miniplaces data manually into the new data folder. We can scp the downloaded tarball for use in CSL:

   scp data.tar.gz <userid>@best-linux.cs.wisc.edu:~

2. Unpack the miniplaces tarball (data.tar.gz) by running the command below. It may fail because of expired SSL certificates, but the data will be unpacked in the correct format. **Note:** You may need to implement the functions in **student_code.py** before this step.

   python3 train_miniplaces.py

3. If step 2 fails from expired SSL certificates, move **train.txt** and **val.txt** into the data/miniplaces folder so it should contain test, train, and val folders and train and val txt files. Miniplaces should now be ready to use

Helper Code

We provide helper functions in t**rain_miniplaces.py** and **dataloader.py,** and skeleton code in **student_code.py**. See the comments in these files for more implementation details.

The original image resolution for images in MiniPlaces is 128x128. To make the training feasible, our data loader reduces the image resolution to 32x32. You can always assume this input resolution.

Before the training procedure, we define the dataloader, model, optimizer, image transform and criterion. We execute the training and testing in the functions train_model and test_model, which is similar to what we have for HW6.

Part I Creating LeNet-5

Background: LeNet was one of the first CNNs and its success was foundational for further research into deep learning. We are implementing an existing architecture in the assignment, but it might be helpful to think about why early researchers chose certain kernel sizes, padding, and strides which we learned about conceptually in class.

In this part, you have to implement a classic CNN model, called LeNet-5 in Pytorch for the MiniPlaces dataset. We use the following layers in this order:

1. One convolutional layer with the number of output channels to be 6, kernel size to be 5, stride to be 1, followed by a relu activation layer and then a 2D max pooling layer (kernel size to be 2 and stride to be 2).
2. One convolutional layer with the number of output channels to be 16, kernel size to be 5, stride to be 1, followed by a relu activation layer and then a 2D max pooling layer (kernel size to be 2 and stride to be 2).
3. A Flatten layer to convert the 3D tensor to a 1D tensor.
4. A Linear layer with output dimension to be 256, followed by a relu activation function.
5. A Linear layer with output dimension to be 128, followed by a relu activation function.
6. A Linear layer with output dimension to be the number of classes (in our case, 100).

You have to fill in the LeNet class in **student_code.py**. You are expected to create the model following this tutorial, which is different from using nn.Sequential() in the last HW.

In addition, given a batch of inputs with shape [N, C, W, H], where N is the batch size, C is the input channel and W, H are the width and height of the image (both 32 in our case), **you are expected to return both the output of the model along with the shape of the intermediate outputs for the above 6 stages**. The shape should be a dictionary with the keys to be 1,2,3,4,5,6 (integers) denoting each stage, and the corresponding value to be a list that denotes the shape of the intermediate outputs.

To help visualization, recall the LeNet architecture from lecture:

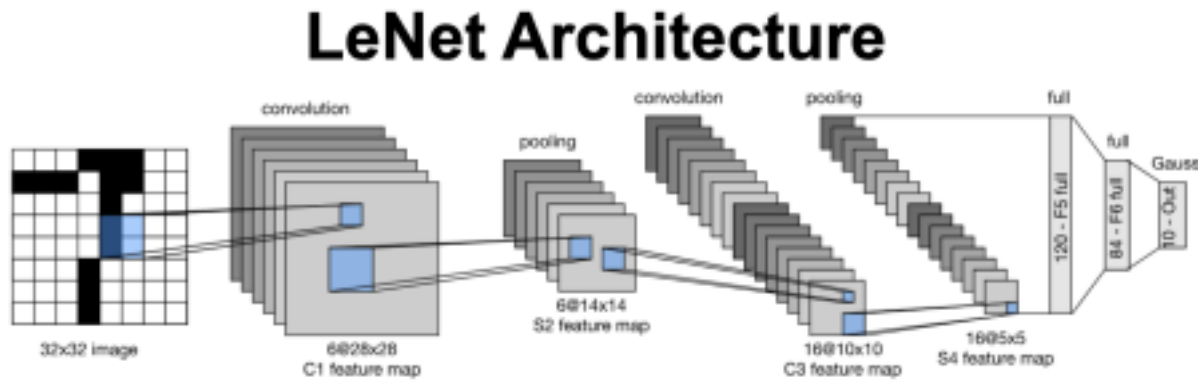

Figure 2: LeNet Architecture

Hints

The expected model has the following form:

```
class LeNet(nn.Module):
    def __init__(self, input_shape=(32, 32), num_classes=100): super(LeNet,
        self).__init__()
        # certain definitions
    def forward(self, x):
        shape_dict = {}
        # certain operations
        return out, shape_dict
```

Shape_dict should has the following form:

{1: [a,b,c,d], 2:[e,f,g,h], .., 6: [x,y]}

The linear layer and the convolutional layer have bias terms.

You should need to use the conv2d function to create a convolutional layer. The parameters allow us to specify the details of the layer eg. the input/output dimensions, padding, stride, and kernel size. More information can be found in the documentation.

Part II Count the number of trainable parameters of LeNet-5

Background: As discussed in lecture, fully connected models (like what we made in the previous homework) are dense with many parameters we need to train. After finishing this part, it might be helpful to think about the number of parameters in this model compared to the number of parameters in a fully connected model of similar depth (similar number of layers). Especially, how does the difference in size impact efficiency and accuracy?

In this part, you are expected to return the number of trainable parameters of your created LeNet model in the previous part. You have to fill in the function count_model_params in **student_code.py**.

The function output is in the unit of Million(1e6) ie. how many millions of trainable parameters there are in your implementation of LeNet. Please do not use any external library which directly calculates the number of parameters (other libraries, such as NumPy can be used as helpers)!

Hint: You can use the model.named_parameters() to gain the name and the corresponding parameters of a model. Please do not do any rounding to the result.

Part III Training LeNet-5 under different configurations

Background: A large part of creating neural networks is designing the architecture (part 1). However, there are other ways of tuning the neural net to change its performance. In this section, we can see how batch size, learning rate, and number of epochs impact how well the model learns. As you get your results, it might be helpful to think about how and why the changes have impacted the training.

Based on the LeNet-5 model created in the previous parts, in this section, you are expected to train the LeNet-5 model under different configurations.

You will use similar implementations of train_model and test_model as you did for HW6 (which we provide in **student_code.py**). When you run You may need to implement the functions in **train_miniplaces.py**, the python script will save two files in the ”outputs” folder.

- checkpoint.pth.tar is the model checkpoint at the latest epoch.
- model_best.pth.tar is the model weights that has highest accuracy on the validation set.

Our code supports resuming from a previous checkpoint, such that you can pause the training and resume later. This can be achieved by running

```
python train_miniplaces.py --resume ./outputs/checkpoint.pth.tar
```

After training, we provide eval_miniplaces.py to help you evaluate the model on the validation set and also help you in timing your model. This script will grab a pre-trained model and evaluate it on the validation set of 10K images. For example, you can run

```
python eval_miniplaces.py --load ./outputs/model_best.pth.tar
```

The output shows the validation accuracy and also the model evaluation time in seconds (see an example below).

```
=> Loading from cached file ./data/miniplaces/cached_val.pkl …
=> loading checkpoint ’./outputs/model_best.pth.tar’
=> loaded checkpoint ’./outputs/model_best.pth.tar’ (epoch x)
Evaluting the model ...
 [Test set] Epoch: xx, Accuracy: xx.xx%
Evaluation took 2.26 sec
```

You can run this script a few times to see the average runtime of your model.
Please train the model under the following different configurations:

1. The default configuration provided in the code, which means you do not have to make modifications.
2. Set the batch size to 8, the remaining configurations are kept the same.
3. Set the batch size to 16, the remaining configurations are kept the same.
4. Set the learning rate to 0.05, the remaining configurations are kept the same.
5. Set the learning rate to 0.01, the remaining configurations are kept the same.
6. Set the epochs to 20, the remaining configurations are kept the same.
7. Set the epochs to 5, the remaining configurations are kept the same.

After training, you are expected to get the validation accuracy as stated above using the best model (model_best.pth.tar), then save these accuracies into a **results.txt** file, where the accuracy of each configuration is placed in one line in order. Your .txt file will end up looking like this:

```
11.11
22.22
33.33
...
```

These exact accuracy may not align well with your results. They are just for illustration purposes. You have to submit the **results.tx**t file together with your **student_code.py.**

Optional: Profiling Your Model

You might find that the training or evaluation of your model is a bit slower than expected. Fortunately, PyTorch has its own profiling tool. Here is a quick tutorial of using PyTorch profiler. You can easily inject the profiler into **train_miniplaces.py** to inspect the runtime and memory consumption of different parts of your model. A general principle is that a deep (many layers) and wide (many feature channels) network will train much slower. It is your design choice to balance between efficiency and accuracy.

Optional: Training on CSL

We recommend training the model on your local machine and time your model (using **eval_miniplaces.py**) on CSL machines. If you decide to train the model on a CSL machine, you will need to find a way to allow your remote session to remain active when you are disconnected from CSL. In this case, we recommend using tmux, a terminal multiplexer for Unix-like systems. tmux is already installed on CSL. To use tmux, simply type tmux in the terminal. Now you can run your code in a tmux session. And the session will remain active even if you are disconnected.

- If you want to detach a tmux session without closing it, press ”ctrl + b” then ”d” (detach) within a tmux session. This will exit to the terminal while keeping the session active. Later you can re-attach the session.
- If you want to enter an active tmux session, type ”tmux a” to attach to the last session in the terminal (outside of tmux).
- If you want to close a tmux session, press ”ctrl + b” then ”x” (exit) within a tmux session. You won’t be able to enter this session again. Please make sure that you close your tmux sessions after this assignment.
- See here for a brief tutorial on the powerful tool of tmux.

Deliverable

You will need to submit **student_code.py** together with your **results.txt** for Part III. The validation accuracy for different configurations must be in.txt format and named as **results.txt**

Submission Notes

You will need to submit **student_code.py** and the **results.txt** validation accuracy file. Do not submit a Jupyter notebook .ipynb file. Be sure to remove all debugging output before submission. Failure to remove debugging output may be penalized.
This assignment is due on November 16, 9:30AM. We highly recommend starting early. It is preferable to first submit a version well before the deadline (at least one hour before) and check the content/format of the submission to make sure it’s the right version. Then, later update the submission until the deadline if needed.