Due Nov 16, 2021 by 10:59am Points 100 **Submitting** a file upload **Available** Nov 4, 2021 at 11am - Nov 16, 2021 at 11am

This assignment was locked Nov 16, 2021 at 11am.

Update

November 13 11:00AM: Relaxing the range in part 3 from 1% to 1.5%. The sample testing script should change its range to 12.35 to 15.35.

November 8 9:30PM: a testing script is now available here: sample testing.py. Run the script using "python3 sample_testing.py".

November 8 2PM: there are issues with downloading the train/validation splits from the miniplaces github. After unpacking the backup tarball from @477, you will likely see an SSL Certificate error. Download the train.txt and val.txt files into the data/miniplaces folder. The dataloader should work now.

November 4, 2:00 PM: due date extended to November 16 at 10:59am

Assignment Goals

- Compare creating a CNN to a basic neural network as in HW6
- Design your own deep network for scene recognition

Summary Your implementation in this assignment might take one or two hours to run. We highly recommend to start working on this assignment early! In this homework, we will

explore building deep neural networks, including Convolutional Neural Networks (CNNs), using PyTorch. Helper code is provided in this assignment and can be downloaded from this zip file code hw7-2.zip \downarrow . In this HW, you are still expected to use the Pytorch backend and virtual environment for programming where you can find the tutorials in HW6.

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.

class.

Helper Code We provide helper functions in train_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

Before the training procedure, we define the dataloader, model, optimizer, image transform and criterion. We execute the training and testing in function train_model and

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 just implementing an existing architecture in

the assignment, but it might be interesting to think about why early researchers chose certain kernel sizes, padding, and strides which we learned about conceptually in

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:

assume this input resolution. Our data loader will also download the full dataset the first time you run train_miniplaces.py

- 1. One convolutional layer with the number of output channels to be 6, kernel size to be 5, stride to be 1, padding to be 0 followed by a relu activation layer and then a 2D max pooling layer (kernel size to be 2, padding to be 0 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, padding to be 0 followed by a relu activation layer and then a 2D max pooling layer (kernel size to be 2, padding to be 0 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.

test_model, which is similar to what we have for HW6.

- 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. **Hint:** The expected model has the following form:

def __init__(self, input_shape=(32, 32), num_classes=100):

class LeNet(nn.Module):

super(LeNet, self).__init__() # certain definitions

def forward(self, x):

shape_dict = {} # certain operations

return out, shape_dict

Shape_dict should have the following form: {1: [a,b,c,d], 2:[e,f,g,h], .., 6: [x,y]}

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

The linear layer and the convolutional layer have bias terms.

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 performance?

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). Please do not use any external library which directly calculates the number of parameters (other libraries, such as NumPy

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

can be used)! 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: Part of creating neural networks is designing the architecture (part 1). However, there are still ways of tweaking 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 the

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.

checkpoint.pth.tar is the model checkpoint at the latest epoch.

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

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 train_miniplaces.py, the python script will save two files in the "outputs" folder.

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

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

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 ...

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

=> loaded checkpoint './outputs/model_best.pth.tar' (epoch x) Evaluting the model ...

changes have impacted the training (hint: think back to gradient descent from the neural net lectures).

model_best.pth.tar is the model weights that has highest accuracy on the validation set.

[Test set] Epoch: xx, Accuracy: xx.xx% Evaluation took 2.26 sec

=> loading checkpoint './outputs/model_best.pth.tar'

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:

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 (revert learning rate to default of 0.001).

7. Set the epochs to 5, the remaining configurations are kept the same.

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.

After training, you are expected to get the validation accuracy for each configuration 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:

1. The default configuration provided in the code, which means you do not have to make modifications.

4. Set the learning rate to 0.05, the remaining configurations are kept the same (revert batch size to default of 32).

17.39

The exact accuracy may not align well with your results. They are just for illustration purposes.

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 expected is a bit slower than expected.

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

You have to submit the **results.txt** file together with your **student_code.py**.

18.00

12.31

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 the efficiency and the accuracy.

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.

Deliverable

Scoring Rubric

your tmux sessions after this assignment.

See here for a <u>brief tutorial</u> ⇒ on the powerful tool of tmux.

Optional: Training on CSL

Optional: Profiling Your Model

• If you want to detach a tmux session without closing it, press "ctrl + b" then "d" (detach) within a tmux session. This will exist 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

We recommend training the model on your local machine or CSL 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

- 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**

the validation accuracy file with the proper name as mentioned above. Do not submit a Jupyter notebook .ipynb file.

Be sure to remove all debugging output before submission. Failure to remove debugging output will be penalized (10pts). If a regrading request isn't justifiable (the initial grade is correct and clear, subject to the instructors' judgment), the request for regrading will be penalized (10 pts).

This assignment is due on Tuesday November 16, 10:59 AM. We highly recommend to start 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.

You will need to submit a single zip file named hw7_<netid>.zip, where you replace <netid> with your netID (your wisc.edu login). This file will include student_code.py and

Criteria	Ratings		Pts
Correctly create the LeNet model with the right output shape.	30 pts Full Marks	0 pts No Marks	30 pts
Correctly calculate the number of parameters of LeNet.	20 pts Full Marks	0 pts No Marks	20 pts
Correctly train the model under different configurations and report the right accuracy.	50 pts Full Marks	0 pts No Marks	50 pts