CSCE 5218 & 4930: Deep Learning:
Homework 2

Due: 05/03/2022, 11:59pm
Point: 100

For HW2, you will write your code and any requested responses or descriptions of your results, in a Colab Jupyter notebook.

1. Implement AlexNet (40 points)

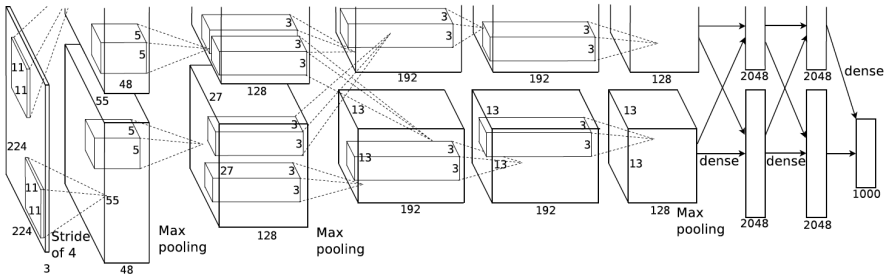

Figure 1. Architecture of AlexNet (Krizhevsky et al., NIPS, 2012).

AlexNet is a milestone in the resurgence of deep learning, and it astonished the computer vision community by winning the ILSVRC 2012 by a large margin. In this assignment, you need to implement the original AlexNet using PyTorch. The model architecture is shown in the Figure 1, which is from their original paper (Click the link to see the paper). You should have a good understanding about the paper because we discussed it in class and you reviewed it.

More specifically, your `AlexNet` should have the following architecture (e.g. for *domain* prediction task) in Figure 2:

Layer (type)	Kernel	Padding	Stride	Dilation	Output Shape	Param #
Conv2d-1	11 x 11		4		[-1, 96, 55, 55]	34,944
ReLU-2					[-1, 96, 55, 55]	0
MaxPool2d-3	3		2		[-1, 96, 27, 27]	0
Conv2d-4	5 x 5	2			[-1, 256, 27, 27]	614,656
ReLU-5					[-1, 256, 27, 27]	0
MaxPool2d-6	3		2		[-1, 256, 13, 13]	0
Conv2d-7	3 x 3	1			[-1, 384, 13, 13]	885,120
ReLU-8					[-1, 384, 13, 13]	0
Conv2d-9	3 x 3	1			[-1, 384, 13, 13]	1,327,488
ReLU-10					[-1, 384, 13, 13]	0
Conv2d-11	3 x 3	1			[-1, 256, 13, 13]	884,992
ReLU-12					[-1, 256, 13, 13]	0
MaxPool2d-13	3		2		[-1, 256, 6, 6]	0
Flatten-14					[-1, 9216]	0
Dropout-15					[-1, 9216]	0
Linear-16					[-1, 4096]	37,752,832
ReLU-17					[-1, 4096]	0
Dropout-18					[-1, 4096]	0
Linear-19					[-1, 4096]	16,781,312
ReLU-20					[-1, 4096]	0
Linear-21					[-1, 4]	16,388

Note: The `-1` in Output shape represents `batch_size`, which is flexible during program execution.

Figure 2. Detailed architecture of AlexNet.

Before you get started, we have several hints hopefully to make your life easier:

- In the figure the image size is $224 \times 224 \times 3$, but unfortunately that is a mistake. The real input should be RGB image of shape $227 \times 227 \times 3$.
- Thanks to the improved hardware (i.e. GPU with larger memory), you don't need to split the convolutions into two cards any more (as shown in the figure).
- There are many implementations online regarding the AlexNet, for example, torchvision library has an official implementation: here. We strongly recommend you to check it out, but please be aware that this implementation is incorrect as it is a modified version of AlexNet (not the one we described above). **Therefore, if you simply copy/paste this code, you won't get any points.**

  Instead, we expect you to read and understand this implementation, and make modifications based on it (which is much easier than you thought).

  Your model should use torch.nn.Conv2d, torch.nn.Linear, torch.nn.ReLU, torch.nn.Dropout etc. modules that are built-in to PyTorch. You don't need to (and are not supposed to) implement any of these modules on your own.
- **Lastly but probably most importantly**, for your convenience, the data loader and the training/evaluation scripts are provided in the starter code. Please download and read the code carefully.

  More specifically, there is a PACSDataset class for this assignment. You can download the images of the dataset from here.

  Briefly speaking, the dataset contains 9991 images (train + val), including images from 4 domains (art painting, cartoon, photo and sketch) and 7 different classes (dog, elephant, giraffe, guitar, horse, house and person) It is a good benchmark to study the domain shift problem, as you can tell an image of the same class can have very different appearances according to domain. More details about the dataset can be found here if you are interested.

  Therefore, each image has two different labels, one for class and one for domain.

  The provided dataset can handle both labels, and you could specify the label type by flag label_type. In next part you need to train models for domain prediction and class prediction.
- Please be aware that the signature of the function is fixed, so please DO NOT change the function signature.
- You need the following packages in your Python environment:

  `matplotlib seaborn torch absl-py scikit-image tensorboard torchvision tqdm`.

1.1. Some instructions

1. Download the starter code, which includes data, from Canvas, and put the starter code in a Colab notebook.
2. Complete the implementation of class AlexNet, and training the model for domain prediction task. A sample usage for the provided training/evaluation script (from the shell) is

   ```
   python cnn_trainer.py --task_type=training --label_type=domain --learning_rate=0.001
   --batch_size=128 --experiment_name=demo
   ```
3. You may need to tune the hyperparamters to achieve better performance.
4. Report the model architecture (i.e. call print(model) and copy/paste the output in your notebook), as well as the accuracy on the validation set.

2. Enhancing AlexNet (40 points)

In this part, you need to modify the AlexNet in previous part, and train different models with the following changes.

Just a friendly reminder, if you implement AlexNet following our recommendation, it should be very easy (e.g. just changing a few lines) to perform the following modifications.

2.1. Instructions

2.1.1 Larger kernel size

Initial AlexNet has 5 convolutional kernels as defined in the table in Section 1.

We observe that for the 1st, 2nd and 5th convolutional layers, a MaxPool2d layer is followed to downsample the inputs.

An alternative strategy is to use larger convolutional kernel (thus larger receptive field) and larger stride, which gives smaller output directly.

Please copy your AlexNet to a new class named AlexNetLargeKernel, and implement the model following the architectures given in Figure 3.

class AlexNetLargeKernel

Layer (type)	Kernel	Padding	Stride	Dilation	Output Shape	Param #
Conv2d-1	21 x 21	1	8		[-1, 96, 27, 27]	127,104
ReLU-2					[-1, 96, 27, 27]	0
Conv2d-3	7 x 7	2	2		[-1, 256, 13, 13]	1,204,480
ReLU-4					[-1, 256, 13, 13]	0
Conv2d-5	3 x 3	1			[-1, 384, 13, 13]	885,120
ReLU-6					[-1, 384, 13, 13]	0
Conv2d-7	3 x 3	1			[-1, 384, 13, 13]	1,327,488
ReLU-8					[-1, 384, 13, 13]	0
Conv2d-9	3 x 3		2		[-1, 256, 6, 6]	884,992
ReLU-10					[-1, 256, 6, 6]	0
Flatten-11					[-1, 9216]	0
Dropout-12					[-1, 9216]	0
Linear-13					[-1, 4096]	37,752,832
ReLU-14					[-1, 4096]	0
Dropout-15					[-1, 4096]	0
Linear-16					[-1, 4096]	16,781,312
ReLU-17					[-1, 4096]	0
Linear-18					[-1, 4]	16,388

Figure 3. AlexNet with larger kernel.

Note that, please use the same optimal hyperparameter with Section 1 to train the new model, and report architecture and accuracy in your report.

class AlexNetAvgPooling

Layer (type)	Kernel	Padding	Stride	Dilation	Output Shape	Param #
Conv2d-1	11 x 11		4		[-1, 96, 55, 55]	34,944
ReLU-2					[-1, 96, 55, 55]	0
AvgPool2d-3	3		2		[-1, 96, 27, 27]	0
Conv2d-4	5 x 5	2			[-1, 256, 27, 27]	614,656
ReLU-5					[-1, 256, 27, 27]	0
AvgPool2d-6	3		2		[-1, 256, 13, 13]	0
Conv2d-7	3 x 3	1			[-1, 384, 13, 13]	885,120
ReLU-8					[-1, 384, 13, 13]	0
Conv2d-9	3 x 3	1			[-1, 384, 13, 13]	1,327,488
ReLU-10					[-1, 384, 13, 13]	0
Conv2d-11	3 x 3	1			[-1, 256, 13, 13]	884,992
ReLU-12					[-1, 256, 13, 13]	0
AvgPool2d-13	3		2		[-1, 256, 6, 6]	0
Flatten-14					[-1, 9216]	0
Dropout-15					[-1, 9216]	0
Linear-16					[-1, 4096]	37,752,832
ReLU-17					[-1, 4096]	0
Dropout-18					[-1, 4096]	0
Linear-19					[-1, 4096]	16,781,312
ReLU-20					[-1, 4096]	0
Linear-21					[-1, 4]	16,388

Figure 4. AlexNet with different pooling.

2.1.2 Pooling strategies

Another tweak to the AlexNet is the pooling layer. Instead of MaxPool2d another common pooling strategy is AvgPool2d, i.e. to average all the neurons in the receptive field.

Please copy your AlexNet to a new class named AlexNetAvgPooling, and implement the model following the architectures given Figure 4.

Note that, please use the same optimal hyperparameter with Section 1 to train the new model, and report architecture and accuracy in your report.

3. Visualizing Learned Filter (20 points)

Different from hand-crafted features, the convolutional neural network extracted features from input images automatically thus are difficult for human to interpret. A useful strategy is to visualize the kernels learned from data. In this part, you are asked to check the kernels learned in AlexNet for two different tasks, i.e. classifying domains or classes.

You need to compare the kernels in different layers for two models (trained for two different tasks), and see if the kernels have different patterns.

For your convenience, we provided a visualization function in the starter code (`visualize_kernels`).

3.1. Instruction

- Train two AlexNet on the PACS dataset for two different tasks (predicting domain and predicting class, respectively) using the same, optimal hyperparameters from Section 1.
- Complete a function named analyze_model_kernels, which: (1) load the well-trained checkpoint, (2) get the model kernels from the checkpoint, (3) using the provided visualization helper function visualize_kernels to inspect the learned filters.
- Report the kernel visualization for two models, and 5 convolution kernels for each model, in your notebook. Compare the learned kernels and summarize your findings.

4. Submission

Please submit your Jupyter notebook with the classes and functions from all Python files in the startup zip file:

- classes and functions from `cnn_trainer.py`

Acknowledgement. The assignment is adapted from M. Zhang at University of Pittsburgh.