Lab Project Part 2
CNNs for Image Classification

Wei Wang, Duy-Kien Nguyen, and Yingjun Du
University of Amsterdam

Due 11:59pm, October 24, 2021 (Amsterdam time)

General Guideline

- **Aim:**
  - Able to understand the Image Classification/Recognition pipeline using a data-driven approach (train/predict stages).
  - Able to implement and test a simple CNN image classify.
- **Prerequisite:**
  - Familiar with Python and relevant packages.
  - Known the basic knowledge of Convolutional Neural Networks.
- **Guidelines:** Students should work on the assignments in a group of **three person** for two weeks. Some minor additions and changes might be done during these three weeks. Students will be informed for these changes via Canvas. Any questions regarding the assignment content can be discussed on Canvas Discussions. Students are expected to do this assignment in Python and Pytorch, however students are free to choose other tools (like Tensorflow). Your source code and report must be handed in together in a zip file (`ID1_ID2_ID3.zip`) before the deadline. Make sure your report follows these guidelines:
  - The maximum number of pages is 10 (single-column, including tables and figures). Please express your thoughts concisely.
  - Follow the given script and answer all given questions (in green boxes). Briefly describe what you implemented. Blue boxes are there to give you hints to answer questions.
  - Analyze your results and discuss them, e.g. why algorithm A works better than algorithm B in a certain problem.
  - Tables and figures must be accompanied by a brief description. Do not forget to add a number, a title, and if applicable name and unit of variables in a table, name and unit of axes and legends in a figure.

  Late submissions are not allowed. Assignments that are submitted after the strict deadline will not be graded. In case of submission conflicts, TAs' system clock is taken as reference. We strongly recommend submitting well in advance, to avoid last minute system failure issues.

  Plagiarism note: Keep in mind that plagiarism (submitted materials which are not your work) is a serious crime and any misconduct shall be punished with the university regulations.

PyTorch Tutorial

This tutorial aims to make you familiar with the programming environment that will be used throughout the course. If you have experience with PyTorch or other frameworks (TensorFlow, MXNet *etc.*), you can skip the tutorial exercises; otherwise, we suggest that you complete them all, as they are helpful for getting hands-on experience.

Anaconda Environment We recommend installing *anaconda* for configuring *python* package dependencies, whereas it's also fine to use other environment managers as you like. The installation of anaconda can be found in `https://docs.anaconda.com/anaconda/install/`.

Installation The installation of PyTorch is available at `https://pytorch.org/get-started/locally/` depending on your device and system.

Getting start The 60-minute blitz can be found at `https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`, and and examples are at `https://pytorch.org/tutorials/beginner/pytorch_with_examples.html`

Documents There might be potential unknown functions or classes, you shall look through the official documents website (`https://pytorch.org/docs/stable/index.html`) and figure them out by yourself. (***Think*:** What's the difference between $torch.nn.Conv2d$ and $torch.nn.functional.conv2d$?)

1 Introduction

This part of the assignment makes use of Convolutional Neural Networks (CNN). The previous part makes use of hand-crafted features like SIFT to represent images, then trains a classifier on top of them. In this way, learning is a two-step procedure with image representation and learning. The method used here instead *learns* the features jointly with the classification. Training CNNs roughly consists of three parts: (i) Creating the network architecture, (ii) Reprocessing the data, (iii) Feeding the data to the network, and updating the parameters. Please follow the instruction and finish the below tasks. (**Note:** You do not need to strictly follow the structure/functions of the provided script.)

2 Session 1: Image Classification on CIFAR-10

2.1 Installation

First of all, you need to install PyTorch and relevant packages. In this session, we will use CIFAR-10 as the training and testing dataset.

CIFAR-10 (3-*pts*)

The relevant script is provided in *Lab_project_part2.pynb*. You need to run and modify the given code and **show** the example images of CIFAR-10, **describe** the classes and images of CIFAR-10. (Please visualize at least one picture for each class.)

2.2 Architecture understanding

In this section, we provide two wrapped classes of architectures defined by *nn.Module*. One is an ordinary two-layer network (*TwolayerNet*) with fully connected layers and ReLu, and the other is a Convolutional Network (*ConvNet*) utilizing the structure of LeNet-5[2].

Architectures (5-*pts*)

1. Complement the architecture of *TwolayerNet* class, and complement the architecture of *ConvNet* class using the structure of LeNet-5[2]. (3-*pts*)
2. Since you need to feed color images into these two networks, what's the kernel size of the first convolutional layer in *ConvNet*? and how many trainable parameters are there in "F6" layer (given the calculation process)? (2-*pts*)

2.3 Preparation of training

In above section, we use the *CIFAR10* dataset class from *torchvision.utils* provided by PyTorch. Whereas in most cases, you need to prepare the dataset yourself. One of the ways is to create a *dataset* class yourself and then use the *DataLoader* to make it iterable. After preparing the training and testing data, you also need to define the transform function for data augmentation and optimizer for parameter updating.

Preparation of training (7-*pts*)

1. Complement the *CIFAR10_loader*(2-*pts*)
2. Complement *transform* function and Optimizer (3-*pts*)
3. Train the *ConvNet* with *CIFAR10_loader*, *transform* and *optimizer* you implemented and show the learning curve. (2-*pts*)

2.4 Setting up the hyperparameters

Some parameters must be set properly before the training of CNNs. These parameters shape the training procedure. They determine how many images are to be processed at each step, how much the weights of the network will be updated, how many iterations will the network run until convergence. These parameters are called hyperparameters in the machine learning literature.

Hyperparameter Optimization and Evaluation (10-*pts*)

1. Play with ConvNet and TwolayerNet yourself, set up the hyperparameters, and reach the accuracy as high as you can. You can modify the *train*, *Dataloader*, *transform* and *Optimizer* function as you like.
2. You can also modify the architectures of these two Nets.
   - Let's add 2 more layers in "TwolayerNet" and ConvNet, and show the results. (You can decide the size of these layers and where to add them.) Will you get higher performances? explain why.
3. Show the final results and described what you've done to improve the results. Describe and explain the influence of hyperparameters among *TwolayerNet* and *ConvNet*.
4. Compare and explain the differences of these two networks regarding the architecture, performances, and learning rates.

Hint

You can adjust the following parameters and other parameters not listed as you like: *Learning rate, Batch size, Number of epochs, optimizer, transform function, Weight decay etc.* You can also change the structure a bit, for instance, adding Batch Normalization[4] layers. Please do not use external well-defined networks and please do not add more than 3 additional (beyond the original network) convolutional layers.

3 Session 2: Fine-tuning the ConvNet

In the previous session, the above-implemented network (ConvNet) is trained on a dataset named CIFAR-10, which contains the images of 10 different object categories. The size of each image is $32 \times 32 \times 3$. In this session, we will use a subset of STL-10 with **larger sizes** and **different object classes**. Consequently, there is a discrepancy between the dataset we used to train (CIFAR-10) and the new dataset (STL-10). One of the solutions is to train the whole network from scratch. However, the number of parameters is too large to be trained properly with such few numbers of images provided from STL-10. Another solution is to shift the learned weights in a way to perform well on the test set, while preserving as much information as necessary from the training class. This procedure is called transfer learning and has been widely used in the literature. Fine-tuning is often used in such circumstances, where the weights of the pre-trained network change gradually. One of the ways of fine-tuning is to use the same architectures in all layers except the output layer, as the number of output classes changes (**from 10 to 5**).

3.1 STL-10 Dataset

STL-10 Dataset (5-*pts*)

Download STL-10 from provided link. In this session, we only need 5 classes from STL-10. The labels of images can be defined as $\{1 : airplanes, 2 : birds, 3 : ships, 4 : cats, 5 : dogs\}$

- Extract mentioned 5 classes of images from STL-10. Complement *stl10_data* class and match each class with the label accordingly.

Hint

You can use the *functions* from *stl10_input.py* to help to complement *stl10_data* class.

3.2 Fine-tuning ConvNet

In this case, you need to modify the output layer of pre-trained ConvNet module from 10 to 5. In this way, you can either load the pre-trained parameters and then modify the output layer or change the output layer firstly and then load the matched pre-trained parameters. You can find the examples from `https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html` and `https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html`.

Finetuning ConvNet (10-*pts*)

1. Complement the architecture by fine-tuning from ConvNet and load the pre-trained parameters (pretrained on CIFAR-10). (5-*pts*)
2. Train the model and show the results (settings of hyperparameters, accuracy, learning curve). (5-*pts*)

Hint (Not Necessary)

Once the network is trained, it is a good practice to understand the feature space by visualization techniques. There are several techniques to visualize the feature space. **t-sne**(`https://lvdmaaten.github.io/tsne/`) is a dimensionality reduction method which can help you better understand the feature learning process.

3.3 Bonus (optional)

Bonus (5-*pts*)

- Play with the code and try to get a higher accuracy on the test dataset (5 class from STL-10) as high as you can. The only data you can use is from CIFAR-10 and SLT-10. The higher accuracy among all teams can get extra points. Specifically, 1st: 5-*pts*, 2nd and 3rd: 4-*pts*, 4th and 5th : 3-*pts*, 6th and 7th: 2-*pts*. 8th-10th: 1-*pts*. You can adjust the hyperparameters and changing structures. Your strategies should be described and explained in your report. **Please do not use external well-defined networks and please do not add more than 3 additional (beyond the original network) convolutional layers.**

Hint

You can try the following methods but are not restricted to these methods:

- **Data augmentation.**
- **Grid Search.**
- **Freezing early layers.**
- **Modifying Architecture.**
- **Modifying hyperparameters**
- **Other advice**: `https://cs231n.github.io/transfer-learning/` and Razavian *et. al.* 2014 [3]

References

[1] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

[2] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

[3] Sharif Razavian, Ali, et al. "CNN features off-the-shelf: an astounding baseline for recognition." Proceedings of the IEEE conference on computer vision and pattern recognition workshops. 2014.

[4] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International conference on machine learning. PMLR, 2015.