Computer Vision & Geometry Group
Songyou Peng

Computer Vision

Exercise 3 – Classification Task with PyTorch

3.1 Environment Setup

For setting up the Python environment, we will be using `miniconda`. For the best and easiest experience, we recommend using a Linux distribution (e.g., Ubuntu). However, `miniconda` is also compatible with Windows and Mac. Please install the latest `miniconda` version for your OS from the following link https://docs.conda.io/en/latest/miniconda.html. Once the `miniconda` is installed, you can run the following commands from the root of the code directory to install and activate the environment:

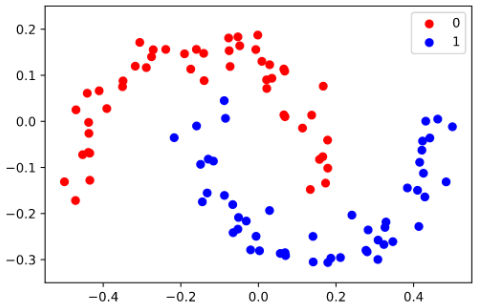
```
conda env create -f env.yml
conda activate dl4cv
```

Next, change the directory to where the jupyter notebooks are located, and run:

```
jupyter notebook
```

NOTE: **Do not modify the provided complete code. Simply add in your codes in parts marked with TODO. No need to write a report for this exercise.** Remember to keep all outputs in every cell of your notebook, do not clean them in your submission!

3.2 Binary Classification on 2D Point Clouds (20 pts.)



Now, in the jupyter notebook `pcl_binary_classification.ipynb`, you can find the code for loading a 2D point cloud dataset. Here, the input `X` contain the 100 2D points, and y are their corresponding labels (0 or 1). The goal is to train a model that can classify every point to its correct label.

The notebook has self-contained descriptions, so the points below should serve as checkpoints for you.

a) Build a PyTorch dataloader (5 pts.)

b) Implement the linear classifier (5 pts.)

c) Implement the training loop (5 pts.)

d) Implement the MLP class (5 pts.)

3.3 Image Classification on MNIST (50 pts.)

Now we move to a slightly more difficult task: classification on hand-written image of 10 digits. For this coding exercises we will use `image_classification.ipynb`. The notebook has self-contained descriptions, so the points below should serve as checkpoints for you.

a) Calculate the normalization constants for MNIST & build a PyTorch dataloader (5 pts.)

b) Implement the MLP class (5 pts.)

c) Implement the training loop. (5 pts.)

d) Reach a test accuracy of > 94 % on MNIST with an MLP (5 pts.)

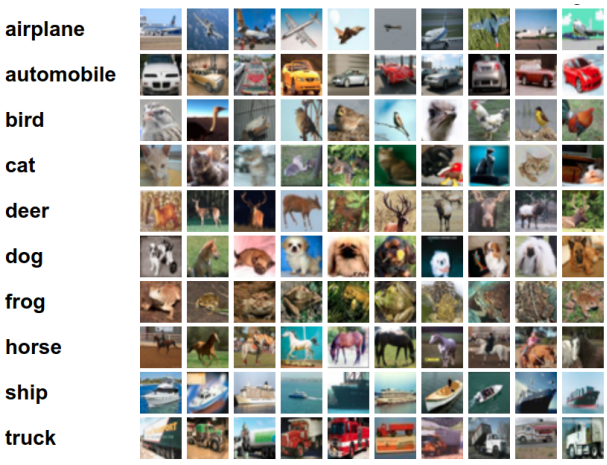
e) Tune the hyperparameter to reach a test accuracy of > 97 % with an MLP (5 pts.)

f) Implement the CNN class (20 pts.)

g) Reach a test accuracy of > 99 % on MNIST with a CNN (5 pts.)

3.4 Scaling to CIFAR10 (30 pts.)

We now hit a point where we will leave MNIST behind us; it served us well. If everything worked out in 3.3, you should see test errors below 1%. Let's scale to something more complex. The CIFAR-10 dataset consists of 60000 32×32 color images in 10 classes, with 6k images per class. There are 50k training images and 10k test images. You can see the classes and example data points above.



a) Calculate the normalization constants for CIFAR (5 pts.)

b) Build train and test dataloaders with different transformations (10 pts.)

c) Adapt the model definition to work on CIFAR (10 pts.)

d) Train a model to reach a performance of > 40 % (5 pts.)

Bonus: Have fun with GPU. If you have fun until this point and want more fun, we provide some more for you. They won't be graded.

Now we want to use GPUs to speed up training massively and also boost the performance significantly with a bigger model. For this purpose, we are going to use Google Colab. Go to Colab and select the "Upload" tab. Upload the lesson notebook. Execute all cells so we can use all functions (comment out all the previous cells for training, this will save you time). Complete the following tasks.

a) Move the model to GPU, adapt Resnet18 to work on CIFAR

b) Reach a test accuracy > 75 %

c) Use different torchvision architectures, data augmentation techniques, and hyperparameter search to achieve a test accuracy of > 90 %