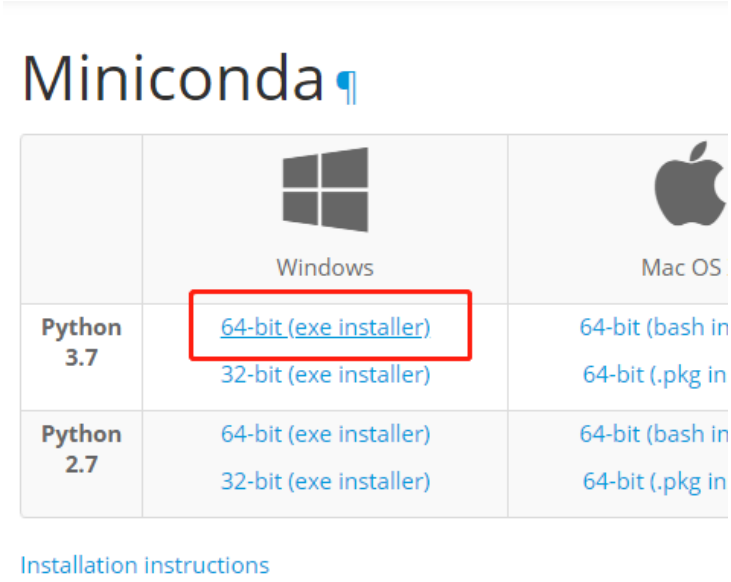
Deep Learning and Computer Vision

Task 1 Prepare the Environment

1. Download the Miniconda installer: https://conda.io/miniconda.html
2. Select **Python 3.7 64-bit (exe installer)** to download.



3. Click the exe and you will the following installation page:

4. Click yes or choose the default recommendations.
5. During the installation, download the workspace file: https://github.com/yikang-li/Tutorial-HandWriting-Cls/archive/master.zip
6. Unzip the file to the directory you can find: C:\Users\t-yikl\workspace\
7. After the installation, press "windows" button, search "Anaconda Prompt" and open it.

8. Install the PyTorch and Torchvision Package:

```
conda install -c pytorch pytorch torchvision
```

9. Jump to the project directory:

```
cd C:\Users\t-yikl\workspace\Tutorial-HandWriting-Cls
```

10. Install the other dependencies and packages:

```
pip install -r requirements.txt
```

Task 2 Prepare the dataset

We have collected some hand-written Chinese character images for you: “香” “港” “中” “文” “大” “学” “电” “子” “工” “程” “系”.

- Download the data folder from link:

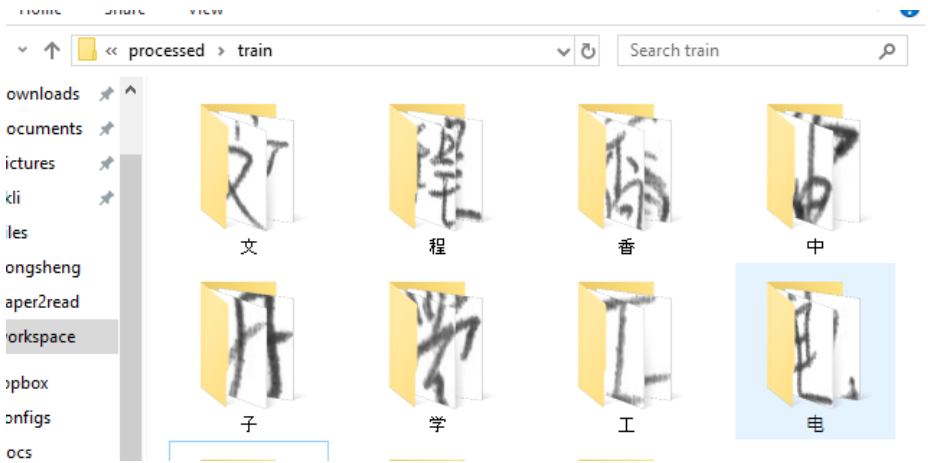
  https://tinyurl.com/y7xeufeh

- Unzip the file and place it under your workspace. Images are like this:

Figure 1: an example of cropped character. Black borderline denotes the boundary of the image.

Furthermore, you can also collect your own Chinese (Traditional) character dataset: xxxxxxx.

- Each student will write 10 times for each character.
- Crop the characters to make bounding box tightly contain the character (like Figure 1)
- put the images under **the corresponding data folder.**
- The folder should look like this and keep the path in mind:



Task 3 Finish your DataLoader

How to load the data in the expected way is very important in implementing your own deep learning algorithm. In this section, you should try to write your own dataloader:

1) read the image from the folder;
2) get the number of categories of characters from the folder;
3) read the image as a 96 x 96 (Width x Height) image;
4) stack the images along Batch axis
5) (optional) shuffle the order of the mini-batches

Therefore, the dataloader should load the data in minibatch of the size: B x 3 x 96 x 96, where B is the batch size, and 3 means it is a three-channel image, (e.g. RGB images).

```
## Fill the data directory: [train] and [test] should be at this path:
data_dir = '/home/denglong/workspace/processed/processed'
# write your own dataloader to read images and targets from [data_dir]
# Then initialize your own [train_loader], [val_loader] and [num_classes]
# you can check torch.utils.data.DataLoader for help
############################
num_classes   # number of categories
train_loader  # training set loader
test_loader   # testing set loader      You, a few seconds ago • Uncommitte
############################
# get the model definition
model = Net(num_classes=num_classes)
if args.resume:
```

Hint: document is a very useful tool during coding. How to use document is a necessary skill for implementing your Deep Learning ideas. So feel free to seek for help from: https://pytorch.org/docs/stable/index.html

Task 4 Finish one training iteration

We have provided you almost the entire project for you, while leaving the forward iteration part to you to complete.

- Open "train.py" and find the function one_iteration

```python
import glog as log
# pytorch related packages
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as T
import torchvision.datasets as datasets
import argparse
# Model Definition
from model import Net

# Training Part
# Please fill the training part based on the given model/dataloader/optimizer/criterion
def train(args, model, train_loader, optimizer, criterion, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(
                pyprind.prog_bar(train_loader,
                title="[Epoch {}: Training]".format(epoch),
                width=40,
                )):
                def one_iteration(model, data, target, criterion):
                    '''
                     Please fill the training iteration with given components:

                      Model: our provided convolutional neural network
                      data: Chinese Character Images
                      target: category of the images
                      criterion: the loss function
                    '''
                    ############################
                    print("Please fill the forward iteration.")
                    ############################

                optimizer.zero_grad()
                one_iteration(model, data, target, criterion)
                optimizer.step()

# Testing Part
def test(args, model, test_loader, epoch):
    model.eval()
    correct = 0
    with torch.no_grad():
        for (data, target) in test_loader:
```

- You don't need to worry about anything. Just use model/data/target/criterion we provide.
- model: our provided convolutional neural network
    - Input: images
    - Output: the probability across different categories.
    - Usage: `output = model(input)`
- data: Chinese Character Images
- target: ground-truth category of the images
- criterion: the loss function to measure quality of the prediction
    - input: prediction & target
    - output: a scalar to measure how good is the prediction matching the target
    - usage: loss = criterion(prediction, target)
- After getting the loss, you can use `loss.backward()` to get the gradients with respective to every layer of CNN.
- We have done all the remaining for you.

Task 5 CNN Model Training

Training and validation part have been provided.

- Please fill the data_dir, which is the path to the dataset. It should contain [train] and [test] subfolders.

```
    parser.add_argument('--seed', type=int, default=1, metavar='S',
                        help='random seed (default: 1)')
    parser.add_argument('--log-interval', type=int, default=1000, metavar='N',
                        help='how many batches to wait before logging training status'
    parser.add_argument('--resume', type=str, default=None, help="Model Path.")
    args = parser.parse_args()
    torch.manual_seed(args.seed)

    ## Fill the data directory: [train] and [test] should be at this path:
    data_dir = 'path/to/your'
    # We randomly sample the [image, target] pairs,
    # Then use the pairs to train the model
    trainset = datasets.ImageFolder(
        osp.join(data_dir, 'train'),
        transform=T.Compose([
            # padding the input image
```

- Run python train.py –help to check the running arguments.
- Run python train.py to train the model and record the validation accuracy and the model name (default: char_cnn.pt).
- Change the arguments (training epochs / batch size / learning rate) to check whether it influence the validation accuracy.

```
(base) C:\Users\t-yikl\workspace\Tutorial-HandWriting-Cls>python train.py
[Epoch 1: Training]
0% [####################] 100% | ETA: 00:00:00
Total time elapsed: 00:00:38
I0104 17:05:31.446521 26132 train.py:57] Test set: Accuracy: 60/657 (9%)

[Epoch 2: Training]
0% [####################] 100% | ETA: 00:00:00
Total time elapsed: 00:00:37
I0104 17:06:19.097092 26132 train.py:57] Test set: Accuracy: 261/657 (40%)

[Epoch 3: Training]
0% [####################] 100% | ETA: 00:00:00
Total time elapsed: 00:00:37
I0104 17:07:07.204657 26132 train.py:57] Test set: Accuracy: 563/657 (86%)

[Epoch 4: Training]
0% [####################] 100% | ETA: 00:00:00
Total time elapsed: 00:00:37
I0104 17:07:55.621870 26132 train.py:57] Test set: Accuracy: 586/657 (89%)

[Epoch 5: Training]
0% [####################] 100% | ETA: 00:00:00
Total time elapsed: 00:00:37
I0104 17:08:44.402374 26132 train.py:57] Test set: Accuracy: 607/657 (92%)
```

Task 6 Recognize the character with Trained Model

We have prepared the inference tools inference.py, which uses the model trained in Task 4 to recognize the new hand-written Chinese character images.

- Start the inference main function with setting the –resume

  python inference.py –resume char_cnn.pt

- Then you can type the path to the new image (or press "**q**" to exit).

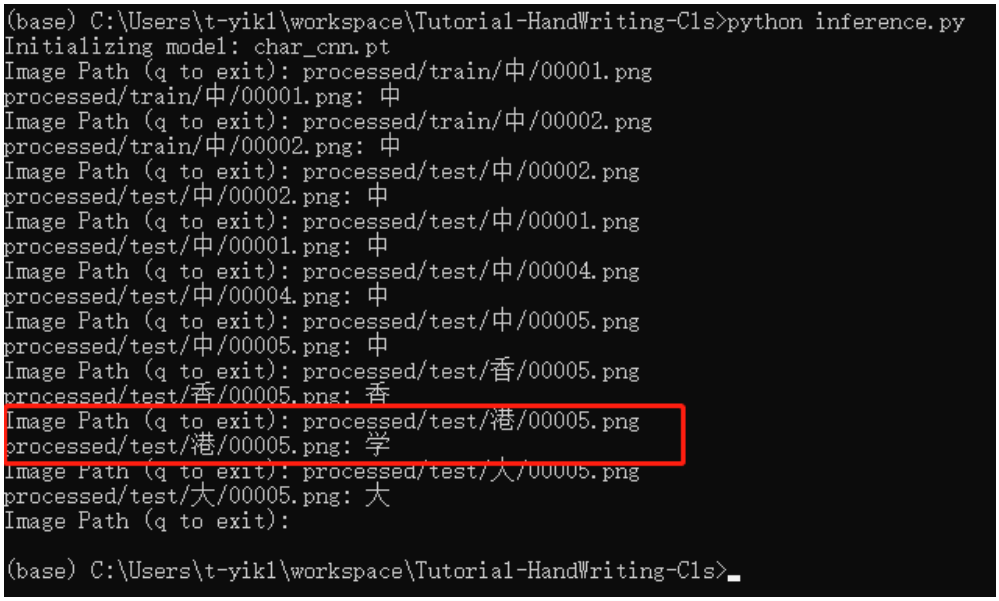


Figure 2 an example of inference results

- You can write some characters to check whether your model can recognize what you write.
  - the character should belong to the categories of the training set.
  - Don't crop the character before doing the inference.
  - You can also check whether the model can recognized the character without cropping and think why it can/cannot recognize the character.
- If the prompt cannot display the Chinese correctly. It is caused by loading the wrong code page. Change the coding scheme to GBK (code: 936) by typing the command at Anaconda Prompt:

```
chcp 936
```