



CS 537

Computer Vision

Recitation:

Conda, JupyterLab, and PyTorch

Jan- 17th - 2019

Agenda today:

- Pelican server
- Install Conda on server
- Coding with Jupyter
- PyTorch Introduction

Pelican server

Step 1: Login:

```
ssh your_osu_name@pelican01.eecs.oregonstate.edu
```

Step 2: Create your folder under /scratch

```
bash
```

```
cd /scratch
```

```
mkdir yourfolder
```

Step 3: link your folder

```
cd
```

```
ln -s /scratch/yourfolder ~/yourfolder
```

Install Conda on Server

Why do we need conda?

- To manage the python packages;

How to install?

- Tutorials: <https://conda.io/docs/user-guide/install/index.html>

Other Options?

- Pipenv: <https://pipenv.readthedocs.io/en/latest/>
- Virtualenv: <https://virtualenv.pypa.io/en/latest/>

Install Conda on Server

We will use miniconda to manage packages.

Step 1: download binary file from <https://conda.io/miniconda.html>

```
cd yourfolder
```

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

```
bash Miniconda3-latest-Linux-x86_64.sh
```

(DO NOT INSTALL IN YOUR ROOT DIRECTORY, INSTALL IT UNDER ~/yourfolder)

```
source ~/.bashrc
```

Step 2: create your virtual environment, and activate it

```
conda create -n myenv python=3.6
```

```
conda activate myenv
```

Install JupyterLab

Step 1: install JupyterLab to myenv

```
conda install -n myenv -c conda-forge jupyterlab
```

Step 2: set config, create password, and launch it

```
jupyter notebook --generate-config
```

```
jupyter notebook password
```

Install JupyterLab

Step 3: launch your JupyterLab (it is better to open a new screen here)

```
screen -S newscreen
```

```
conda activate myenv
```

```
jupyter lab --no-browser --port=8889 --ip=0.0.0.0
```

Step 4: link server to your localhost on your laptop. In your local terminal:

```
ssh -NfL localhost:8888:localhost:8889 yourname@pelican02.eecs.oregonstate.edu
```

Step 5: open your local web browser, go to: <http://localhost:8888> ; input your saved password, done!

Install PyTorch, Opencv ...

Step 1: Install pytorch with cuda9.0 to myenv. Go back to your main screen (ctrl + A + D)

```
conda install -n myenv pytorch torchvision -c pytorch
```

Step 2: Install cv2

```
pip install opencv-contrib-python==3.3.0.10
```

Step 3: Install pandas, matplotlib

```
conda install -n myenv pandas matplotlib
```

Step 4: Install tqdm

```
conda install -n myenv -c conda-forge tqdm
```


Other packages:

Step 1: Install pytorch with cuda9.0 to myenv. Go back to your main screen (ctrl + A + D)

```
pip install tensorboard_logger
```

Keypoint Descriptor

Copy framework to your folder:

```
cp -rf /scratch/CS537_2019_Winter/keypoint_descriptor ~/yourfolder/
```

Create symbolic link of the Datasets in your folder:

```
cd ~/yourfolder/keypoint_descriptor
```

```
ln -s /scratch/CS537_2019_Winter/data data
```

Launch your JupyterLab, and open the notebook:

[keypoint_description.ipynb](#)

Keypoint Descriptor

File Browser

> keypoint_descriptor

Name	Last Modified
models	25 minutes ago
logs	25 minutes ago
images	5 minutes ago
data	26 minutes ago
keypoint_description...	4 minutes ago
W1BS.py	26 minutes ago
Utils.py	26 minutes ago
Losses.py	26 minutes ago
Loggers.py	26 minutes ago
descriptor.py	26 minutes ago
config_profile.py	26 minutes ago

Launcher

keypoint_description.ipynb

Markdown

Python 3

Keypoint(Patch) Description

This project will be all about defining and training a convolutional neural network to perform keypoint description. The first step is to load and visualize the data you'll be working with.

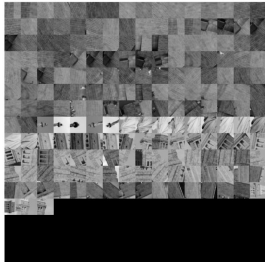
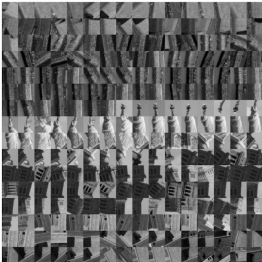
We will use below dataset in this project:

The Photo Tourism dataset

<http://phototour.cs.washington.edu/patches/default.htm>

It is also available in PyTorch torchvision datasets:
https://pytorch.org/docs/stable/_modules/torchvision/datasets/phototour.html#PhotoTour

This dataset consists of 1024 x 1024 bitmap (.bmp) images, each containing a 16 x 16 array of image patches. Here are some examples:



For details of how the scale and orientation is established, please see the paper:

S. Winder and M. Brown. **Learning Local Image Descriptors**. To appear *International Conference on Computer Vision and Pattern Recognition (CVPR2007)* ([pdf 300Kb](#))

Keypoint Descriptor

Import packages

```
[1]: from __future__ import division, print_function
import glob
import os
import cv2
import PIL
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import torch
import torch.nn.init
import torch.nn as nn
import torch.optim as optim
import torch.backends.cudnn as cudnn
import torch.nn.functional as F
import torchvision.datasets as dset
import torchvision.transforms as transforms
from tqdm import tqdm
from torch.autograd import Variable
from copy import deepcopy, copy
from config_profile import args
from Utils import cv2_scale36, cv2_scale, np_reshape, np_reshape64
```

Check GPU availability, using nvidia-smi

```
[2]: os.environ["CUDA_VISIBLE_DEVICES"] = "1"
```

Keypoint Descriptor

Define PyTorch dataset

```
[3]: class TripletPhotoTour(dset.PhotoTour):
    """
    From the PhotoTour Dataset it generates triplet samples
    note: a triplet is composed by a pair of matching images and one of
    different class.
    """
    def __init__(self, train=True, transform=None, batch_size = None, load_random_triplets = False, *arg, **kw):
        super(TripletPhotoTour, self).__init__(*arg, **kw)
        self.transform = transform
        self.out_triplets = load_random_triplets
        self.train = train
        self.n_triplets = args.n_triplets
        self.batch_size = batch_size

    if self.train:
        print('Generating {} triplets'.format(self.n_triplets))
        self.triplets = self.generate_triplets(self.labels, self.n_triplets)
```

Keypoint Descriptor

Define the dataloader

```
def create_loaders(load_random_triplets = False):
    test_dataset_names = copy(dataset_names)
    test_dataset_names.remove(args.training_set)

    kwargs = {'num_workers': args.num_workers, 'pin_memory': args.pin_memory} if args.cuda else {}

    np_reshape64 = lambda x: np.reshape(x, (64, 64, 1))
    transform_test = transforms.Compose([
        transforms.Lambda(np_reshape64),
        transforms.ToPILImage(),
        transforms.Resize(32),
        transforms.ToTensor()])
    transform_train = transforms.Compose([
        transforms.Lambda(np_reshape64),
        transforms.ToPILImage(),
        transforms.RandomRotation(5, PIL.Image.BILINEAR),
        transforms.RandomResizedCrop(32, scale = (0.9, 1.0), ratio = (0.9, 1.1)),
        transforms.Resize(32),
        transforms.ToTensor()])
    transform = transforms.Compose([
        transforms.Lambda(cv2_scale),
        transforms.Lambda(np_reshape),
        transforms.ToTensor(),
        transforms.Normalize((args.mean_image,), (args.std_image,))])
```

Keypoint Descriptor

Load Data

Load the Photo Tourism dataset by PyTorch. Below line (function 'create_loader') will help you to download the dataset to your directory. The data dir and other configuration settings are specified in config_profile.py.

```
[5]: dataset_names = ['liberty', 'notredame', 'yosemite']  
train_loader, test_loaders = create_loaders(dataset_names, load_random_triplets = args.load_random_triplets)  
  
# Found cached data data/sets/liberty.pt  
Generating 5000 triplets  
100%|██████████| 5000/5000 [00:00<00:00, 27351.15it/s]  
# Found cached data data/sets/notredame.pt  
# Found cached data data/sets/yosemite.pt
```

Keypoint Descriptor

Visualizaiton of the Training and Testing Data

Below are some examples of patches in this dataset.

Training

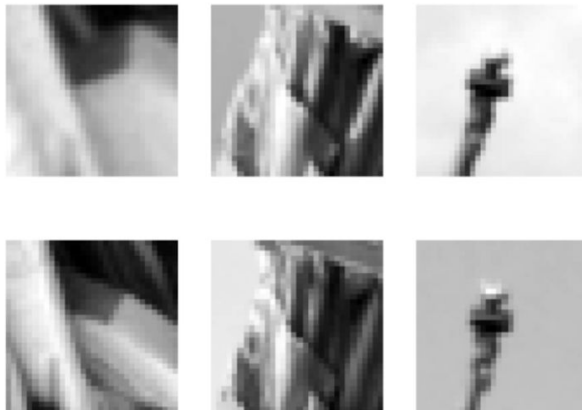
In the training phase, the input data is a batch of patch pairs: $X = \{(patch_a, patch_p)\}$, which represents the anchor patch and the positive patch, respectively.

```
nrow = 3
def plot_examples(img_tensor, nrow):
    fig, axs = plt.subplots(1, nrow)
    for i, ax in enumerate(axs):
        img = img_tensor[i, 0]
        ax.imshow(img, cmap='gray')
        ax.axis('off')

for i_batch, sample_batched in enumerate(train_loader):
    print("IN TRAINing, each data entry has {} elements, each with size of: {}".format(len(sample_batched)))
    print(sample_batched[0].shape)
    print("Below two rows images are {} examples for patch_a and patch_p".format(nrow))
    if i_batch == 0:
        plot_examples(sample_batched[0], nrow)
        plot_examples(sample_batched[1], nrow)
        plt.show()
    break
```


Keypoint Descriptor

IN TRAINing, each data entry has 2 elements, each with size of:
`torch.Size([1024, 1, 32, 32])`
Below two rows images are 3 examples for patch_a and patch_p



Keypoint Descriptor

Testing

In the testing phase, the input data is a batch of patch pairs, and a label that indicates the matching result of this pair (1 means match and 0 means not match)

```
for i_batch, sample_batched in enumerate(test_loaders[0]['dataloader']):
    print("IN TESTING, each data entry has {} elements, with size of: {}, {}, and {}".format(len(sample_batched),
                                                                                               sample_batched[0].shape,
                                                                                               sample_batched[1].shape,
                                                                                               sample_batched[2].shape))

    print("\nBelow two rows images are {} examples for for patch_a and patch_p.".format(nrow))
    if i_batch == 0:
        plot_examples(sample_batched[0], nrow)
        plot_examples(sample_batched[1], nrow)
        print("labels are :", sample_batched[2][:nrow])
        plt.show()
        break
```

IN TESTING, each data entry has 3 elements, with size of: torch.Size([1024, 1, 32, 32]), torch.Size([1024, 1, 32, 32]), and torch.Size([1024])

Below two rows images are 3 examples for for patch_a and patch_p.
labels are : tensor([0, 0, 1])



Keypoint Descriptor

Build Network Model

The DesNet is a simple CNN network, which only contains two CNN blocks.

```
# load network
from descriptor import DesNet
model = DesNet()
if args.cuda:
    model.cuda()
```

Define optimize

We will use SGD, but you can change it to ADAM by modifying arg.lr in config_profile.py

```
# define optimizer
def create_optimizer(model, new_lr):
    # setup optimizer
    if args.optimizer == 'sgd':
        optimizer = optim.SGD(model.parameters(), lr=new_lr,
                               momentum=0.9, dampening=0.9,
                               weight_decay=args.wd)
    elif args.optimizer == 'adam':
        optimizer = optim.Adam(model.parameters(), lr=new_lr,
                               weight_decay=args.wd)
    else:
        raise Exception('Not supported optimizer: {}'.format(args.optimizer))
    return optimizer
optimizer1 = create_optimizer(model.features, args.lr)
```

Keypoint Descriptor

Define a training module ¶

```
: def train(train_loader, model, optimizer, epoch, logger, load_triplets = False):
    # switch to train mode
    model.train()
    pbar = tqdm(enumerate(train_loader))
    for batch_idx, data in pbar:
        if load_triplets:
            data_a, data_p, data_n = data
        else:
            data_a, data_p = data

        if args.cuda:
            data_a, data_p = data_a.cuda(), data_p.cuda()
            data_a, data_p = Variable(data_a), Variable(data_p)
            out_a = model(data_a)
            out_p = model(data_p)

        if load_triplets:
            data_n = data_n.cuda()
            data_n = Variable(data_n)
            out_n = model(data_n)

        if args.batch_reduce == 'L2Net':
            loss = loss_L2Net(out_a, out_p, anchor_swap = args.anchorswap,
                              margin = args.margin, loss_type = args.loss)
        elif args.batch_reduce == 'random_global':
            loss = loss_random_sampling(out_a, out_p, out_n,
                                         margin=args.margin,
                                         anchor_swap=args.anchorswap,
                                         loss_type = args.loss)
        else:
            loss = loss_DesNet(out_a, out_p,
                               margin=args.margin,
                               anchor_swap=args.anchorswap,
                               anchor_ave=args.anchorave,
                               batch_reduce = args.batch_reduce,
                               loss_type = args.loss)
```

Keypoint Descriptor

Define a test module

```
def test(test_loader, model, epoch, logger, logger_test_name):
    # switch to evaluate mode
    model.eval()

    labels, distances = [], []

    pbar = tqdm(enumerate(test_loader))
    for batch_idx, (data_a, data_p, label) in pbar:
        # data_a.shape= torch.Size([1024, 1, 32, 32])
        # data_p.shape =torch.Size([1024, 1, 32, 32])
        # label.shape = torch.Size([1024])
        if args.cuda:
            data_a, data_p = data_a.cuda(), data_p.cuda()

        data_a, data_p, label = Variable(data_a, volatile=True), \
                                Variable(data_p, volatile=True), Variable(label)

        out_a = model(data_a)
        out_p = model(data_p)
        dists = torch.sqrt(torch.sum((out_a - out_p) ** 2, 1)) # euclidean distance
        distances.append(dists.data.cpu().numpy().reshape(-1,1))
        ll = label.data.cpu().numpy().reshape(-1, 1)
        labels.append(ll)

    if batch_idx % args.log_interval == 0:
        pbar.set_description(logger_test_name+' Test Epoch: {} [{} / {}] {:.0f}%')'.format(
            epoch, batch_idx * len(data_a), len(test_loader.dataset),
            100. * batch_idx / len(test_loader)))

    num_tests = test_loader.dataset.matches.size(0)
    labels = np.vstack(labels).reshape(num_tests)
    distances = np.vstack (distances).reshape(num_tests)

    fpr95 = ErrorRateAt95Recall(labels, 1.0 / (distances + 1e-8))
    print('\33[91mTest set: Accuracy(FPR95): {:.8f}\n\33[0m'.format(fpr95))

    if (args.enable_logging):
        logger.log_value(logger_test_name+' fpr95', fpr95)
    return
```

Keypoint Descriptor

Training ¶

```
: start = args.start_epoch
end = start + args.epochs
logger, file_logger = None, None
triplet_flag = args.load_random_triplets
from Losses import loss_DesNet
TEST_ON_W1BS = True
LOG_DIR = args.log_dir
if(args.enable_logging):
    from Loggers import Logger, FileLogger
    logger = Logger(LOG_DIR)

suffix = '{}_{}_{}'.format(args.experiment_name, args.training_set, args.batch_reduce)
if args.gor:
    suffix = suffix+'_gor_alpha{:1.1f}'.format(args.alpha)
if args.anchorswap:
    suffix = suffix + '_as'
if args.anchorave:
    suffix = suffix + '_av'
if args.fliprot:
    suffix = suffix + '_fliprot'

res_fpr_liberty = torch.zeros(end-start,1)
res_fpr_notredame = torch.zeros(end-start, 1)
res_fpr_yosemite = torch.zeros(end-start, 1)

for epoch in range(start, end):

    # iterate over test loaders and test results
    train(train_loader, model, optimizer1, epoch, logger, triplet_flag)
    for test_loader in test_loaders:
        test(test_loader['dataloader'], model, epoch, logger, test_loader['name'])

    #randomize train loader batches
    train_loader, test_loaders2 = create_loaders(load_random_triplets=triplet_flag)
```