Keypoint(Patch) Description

This project will be all about defining and training a convolutional neural network to perform keypoint description. PyTorch tutorials are available at here: pytorch tutorials

Today we will go through:

1. Load and visualize the data

2. Build an example deep network

3. Train the deep network

4. Generate deep features

We will use below dataset in this project:

The Photo Tourism dataset

It is also available in PyTorch torchvision datasets: pytorch version

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)

Import packages

In [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
from Utils import L2Norm, cv2_scale, np_reshape
# import torchvision
```

```
/nfs/hpc/share/heli/miniconda3/envs/myenv/lib/python3.6/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Check GPU availability, using nvidia-smi

In [2]:

```
# Since there are two GPUs on each pelican server, you can either select it as 0 or 1
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
```

In [3]:

```
print(f"pytorch version = {torch.__version__}")
! nvidia-smi
```

```
pytorch version = 1.9.0
Mon Apr 10 11:16:12 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  Off  | 00000000:84:00.0 Off |                  N/A |
| 30%   35C    P8    18W / 250W |     12MiB / 11264MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA GeForce ...  Off  | 00000000:85:00.0 Off |                  N/A |
| 29%   30C    P8     1W / 250W |     12MiB / 11264MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A     12866      G   /usr/bin/Xorg                      10MiB |
|    1   N/A  N/A     12918      G   /usr/bin/Xorg                      10MiB |
+-----------------------------------------------------------------------------+
```

Load and visualize the data

In this section, we will

1. Define a PyTorch dataset
2. Define a PyTorch dataloader
3. Load data
4. Visualizaiton of the Training and Testing Data

BackToTop

Define PyTorch dataset

BackToSection

In [4]:

```
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
        self.triplets = self.generate_triplets(self.labels, self.n_triplets)

    @staticmethod
    def generate_triplets(labels, num_triplets):
        def create_indices(_labels):
            inds = dict()
            for idx, ind in enumerate(_labels):
                if ind not in inds:
                    inds[ind] = []
                inds[ind].append(idx)
            return inds

        triplets = []
        indices = create_indices(labels.numpy())
        unique_labels = np.unique(labels.numpy())
```

```
        n_classes = unique_labels.shape[0]
        # add only unique indices in batch
        already_idxs = set()

        for x in tqdm(range(num_triplets)):
            if len(already_idxs) >= args.batch_size:
                already_idxs = set()
            c1 = np.random.randint(0, n_classes)
            while c1 in already_idxs:
                c1 = np.random.randint(0, n_classes)
            already_idxs.add(c1)
            c2 = np.random.randint(0, n_classes)
            while c1 == c2:
                c2 = np.random.randint(0, n_classes)
            if len(indices[c1]) == 2:  # hack to speed up process
                n1, n2 = 0, 1
            else:
                n1 = np.random.randint(0, len(indices[c1]))
                n2 = np.random.randint(0, len(indices[c1]))
                while n1 == n2:
                    n2 = np.random.randint(0, len(indices[c1]))
            n3 = np.random.randint(0, len(indices[c2]))
            triplets.append([indices[c1][n1], indices[c1][n2], indices[c2][n3]])
        return torch.LongTensor(np.array(triplets))

    def __getitem__(self, index):
        def transform_img(img):
            if self.transform is not None:
                img = self.transform(img.numpy())
            return img

        t = self.triplets[index]
        a, p, n = self.data[t[0]], self.data[t[1]], self.data[t[2]]

        img_a = transform_img(a)
        img_p = transform_img(p)
        img_n = None
        if self.out_triplets:
            img_n = transform_img(n)
        # transform images if required
        if args.fliprot:
            do_flip = random.random() > 0.5
            do_rot = random.random() > 0.5
            if do_rot:
                img_a = img_a.permute(0,2,1)
                img_p = img_p.permute(0,2,1)
                if self.out_triplets:
                    img_n = img_n.permute(0,2,1)
            if do_flip:
                img_a = torch.from_numpy(deepcopy(img_a.numpy()[:,:,::-1]))
                img_p = torch.from_numpy(deepcopy(img_p.numpy()[:,:,::-1]))
                if self.out_triplets:
                    img_n = torch.from_numpy(deepcopy(img_n.numpy()[:,:,::-1]))
        return (img_a, img_p, img_n)


    def __len__(self):
        return self.triplets.size(0)
```

Define the dataloader

BackToSection

In [5]:

```
def create_loaders(dataset_names, load_random_triplets = False, verbose=False):
    """
    For training, we use dataset 'liberty';
    For testing, we use dataset 'notredame' and 'yosemite'

    """
    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,))])
    if not args.augmentation:
        transform_train = transform
        transform_test = transform
    train_loader = torch.utils.data.DataLoader(
            TripletPhotoTour(train=True,
                             load_random_triplets = load_random_triplets,
                             batch_size=args.batch_size,
                             root=args.dataroot,
                             name=args.training_set,
                             download=True,
                             transform=transform_train),
                             batch_size=args.batch_size,
                             shuffle=False, **kwargs)

    test_loaders = [{'name': name,
                     'dataloader': torch.utils.data.DataLoader(
             TripletPhotoTour(train=False,
                     batch_size=args.test_batch_size,
                     load_random_triplets = load_random_triplets,
                     root=args.dataroot,
                     name=name,
                     download=True,
                     transform=transform_test),
                        batch_size=args.test_batch_size,
                        shuffle=False, **kwargs)}
                    for name in test_dataset_names]

    return train_loader, test_loaders[0]
```

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 setings are specified in config_profile.py.

BackToSection

In [6]:
```
dataset_names = ['liberty', 'notredame']

args.n_triplets = 5000     # for illustration, here we only use 5000 triples; in your experiment, set it as 100000
args.epochs = 5              # in your experiment, set it as 60; For CNN1, it will take ~ 1hr20mins if n_triplets = 100000

train_loader, validation_loader = create_loaders(dataset_names, load_random_triplets = args.load_random_triplets)
```

```
/nfs/hpc/share/heli/miniconda3/envs/myenv/lib/python3.6/site-packages/torchvision/transforms/transforms.py:1238: UserWarn
ing: Argument interpolation should be of type InterpolationMode instead of int. Please, use InterpolationMode enum.
  "Argument interpolation should be of type InterpolationMode instead of int. "
# Found cached data data/sets/liberty.pt
100%|██████████| 5000/5000 [00:00<00:00, 46093.89it/s]
# Found cached data data/sets/notredame.pt
100%|██████████| 5000/5000 [00:00<00:00, 44802.71it/s]
```

Visualizaiton of the Training and Testing Data

Below are some examples of patches in this dataset.

BackToSection

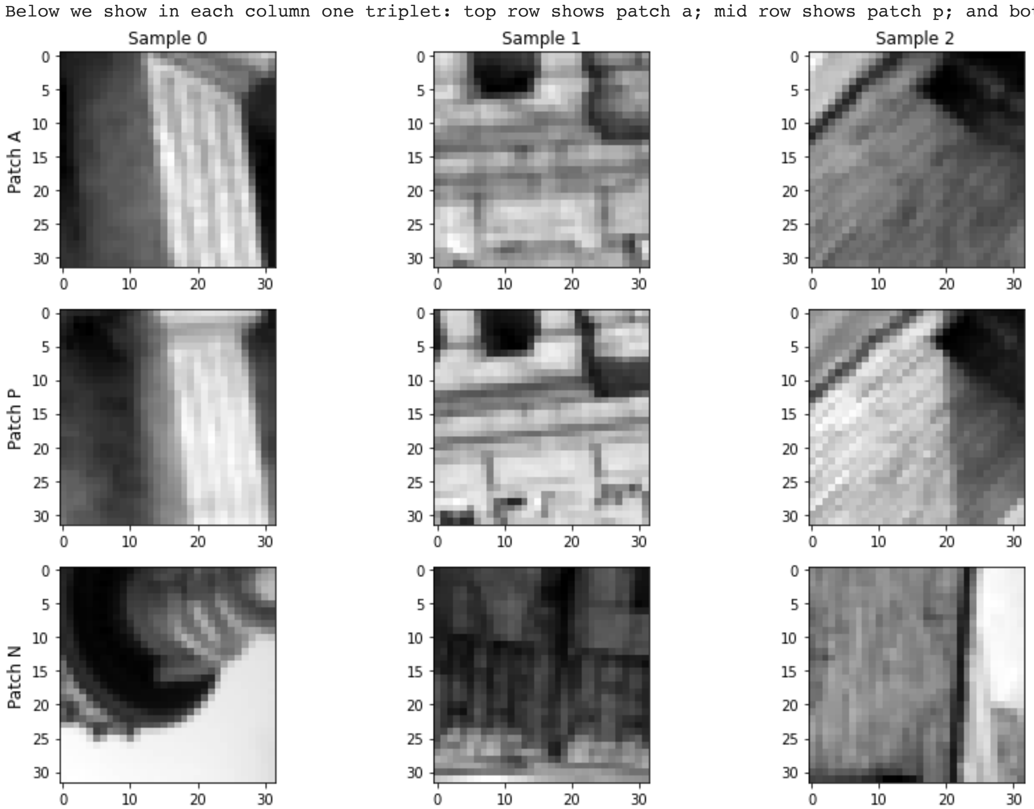
Data in Training

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.

In [7]:
```
def plot_examples(sample_batched, n_samples=3, labels=['A', 'P', 'N']):

    cols = ['Sample {}'.format(col) for col in range(0, n_samples)]
    rows = ['Patch {}'.format(row) for row in labels]
```

```
    nrow = len(rows)

    fig, axes = plt.subplots(nrows=len(rows), ncols=n_samples, figsize=(12, 8))
    for ax, col in zip(axes[0], cols):
        ax.set_title(col)

    for ax, row in zip(axes[:,0], rows):
        ax.set_ylabel(row, rotation=90, size='large')

#     for idx, img_tensor in enumerate(sample_batched):
    for idx in range(nrow):
        img_tensor = sample_batched[idx]
        for jdx in range(n_samples):
            img = img_tensor[jdx, 0]
            axes[idx][jdx].imshow(img, cmap='gray')

    fig.tight_layout()
    plt.show()

for i_batch, sample_batched in enumerate(train_loader):
    print("In training and validation, each data entry generates {} elements: anchor, positive, and negative.".format(len
    print("Each of them have the size of: {}".format(sample_batched[0].shape))
    print("Below we show in each column one triplet: top row shows patch a; mid row shows patch p; and bot row shows patc

    if i_batch == 0:
        plot_examples(sample_batched, 3)
        break
```

```
In training and validation, each data entry generates 3 elements: anchor, positive, and negative.
Each of them have the size of: torch.Size([1024, 1, 32, 32])
Below we show in each column one triplet: top row shows patch a; mid row shows patch p; and bot row shows patch n.
```



Build an exmaple deep network

In this section, we will:

1. Build the deep network: DesNet

2. Setup optimization

BackToTop

Build the deep network: DesNet

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

BackToSection

In [8]:
```
# load network from the python file. You need to submit these .py files to TA
from CNN1 import DesNet       # uncomment this line if you are using DesNet from CNN1.py
# from CNN2 import DesNet      # uncomment this line if you are using DesNet from CNN2.py
# from CNN3 import DesNet      # uncomment this line if you are using DesNet from CNN3.py

model = DesNet()
# check model architecture

print(model)

if args.cuda:
    model.cuda()
```

```
DesNet(
  (features): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(32, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
    (5): ReLU()
    (6): Dropout(p=0.3, inplace=False)
    (7): Conv2d(128, 128, kernel_size=(8, 8), stride=(1, 1), bias=False)
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
  )
)
```

Define optimization

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

BackToSection

In [9]:
```
# 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)
    else:
        raise Exception('Not supported optimizer: {0}'.format(args.optimizer))
    return optimizer
optimizer1 = create_optimizer(model.features, args.lr)
```

---

train the deep network

In this section, we will:

1. Define a training module
2. Define a testing module
3. Train and test on the validation data

BackToTop

Define a training module

BackToSection

In [10]:
```
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:
        data_a, data_p, data_n = data


        if args.cuda:
```

```python
            data_a, data_p, data_n  = data_a.cuda(), data_p.cuda(), data_n.cuda()
            out_a = model(data_a)
            out_p = model(data_p)
            out_n = model(data_n)

        loss = loss_DesNet(out_a, out_p, out_n, anchor_swap = False, margin = 1.0, loss_type = "triplet_margin")

        if args.decor:
            loss += CorrelationPenaltyLoss()(out_a)

        if args.gor:
            loss += args.alpha*global_orthogonal_regularization(out_a, out_n)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        adjust_learning_rate(optimizer)
        if batch_idx % args.log_interval == 0:
            pbar.set_description(
                'Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                    epoch, batch_idx * len(data_a), len(train_loader.dataset),
                           100. * batch_idx / len(train_loader),
                    loss.item()))

    if (args.enable_logging):
        logger.log_value('loss', loss.item()).step()

    try:
        os.stat('{}{}'.format(args.model_dir,suffix))
    except:
        os.makedirs('{}{}'.format(args.model_dir,suffix))

    torch.save({'epoch': epoch + 1, 'state_dict': model.state_dict()},
               '{}{}/checkpoint_{}.pth'.format(args.model_dir,suffix,epoch))


def adjust_learning_rate(optimizer):
    """Updates the learning rate given the learning rate decay.
    The routine has been implemented according to the original Lua SGD optimizer
    """
    for group in optimizer.param_groups:
        if 'step' not in group:
            group['step'] = 0.
        else:
            group['step'] += 1.
        group['lr'] = args.lr * (
        1.0 - float(group['step']) * float(args.batch_size) / (args.n_triplets * float(args.epochs)))
    return
```

Define a testing module

BackToSection

In [11]:

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

    losses = 0

    pbar = tqdm(enumerate(test_loader))

    for batch_idx, data in pbar:
        data_a, data_p, data_n = data


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

        loss = loss_DesNet(out_a, out_p, out_n, anchor_swap = False, margin = 1.0, loss_type = "triplet_margin")
        losses = losses + loss.cpu().numpy()
    ave_loss = losses/len(test_loader)
    print('\33[91mLoss on validation: {:.8f}\n\33[0m'.format(ave_loss))

    if (args.enable_logging):
        logger.log_value(logger_test_name+' vloss', ave_loss)
    return


def ErrorRateAt95Recall(labels, scores):
    distances = 1.0 / (scores + 1e-8)
```

```python
    recall_point = 0.95
    labels = labels[np.argsort(distances)]
    # Sliding threshold: get first index where recall >= recall_point.
    # This is the index where the number of elements with label==1 below the threshold reaches a fraction of
    # 'recall_point' of the total number of elements with label==1.
    # (np.argmax returns the first occurrence of a '1' in a bool array).
    threshold_index = np.argmax(np.cumsum(labels) >= recall_point * np.sum(labels))

    FP = np.sum(labels[:threshold_index] == 0) # Below threshold (i.e., labelled positive), but should be negative
    TN = np.sum(labels[threshold_index:] == 0) # Above threshold (i.e., labelled negative), and should be negative
    return float(FP) / float(FP + TN)
```

Training

BackToSection

In [12]:

```python
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 = '{}_{}_{}_as_fliprot'.format(args.experiment_name, args.training_set, args.batch_reduce)

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)
    with torch.no_grad():
        test(validation_loader['dataloader'], model, epoch, logger, validation_loader['name'])

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

```
Train Epoch: 0 [0/5000 (0%)]    Loss: 0.836733: : 5it [00:03,  1.26it/s]
5it [00:01,  2.74it/s]
/nfs/hpc/share/heli/miniconda3/envs/myenv/lib/python3.6/site-packages/torchvision/transforms/transforms.py:1238: UserWarn
ing: Argument interpolation should be of type InterpolationMode instead of int. Please, use InterpolationMode enum.
  "Argument interpolation should be of type InterpolationMode instead of int. "
Loss on validation: 0.52489202

# Found cached data data/sets/liberty.pt
100%|██████████| 5000/5000 [00:00<00:00, 59278.92it/s]
# Found cached data data/sets/notredame.pt
100%|██████████| 5000/5000 [00:00<00:00, 57484.10it/s]
Train Epoch: 1 [0/5000 (0%)]    Loss: 0.453763: : 5it [00:01,  2.53it/s]
5it [00:01,  2.67it/s]
Loss on validation: 0.44628187

# Found cached data data/sets/liberty.pt
100%|██████████| 5000/5000 [00:00<00:00, 58723.85it/s]
# Found cached data data/sets/notredame.pt
100%|██████████| 5000/5000 [00:00<00:00, 55281.90it/s]
Train Epoch: 2 [0/5000 (0%)]    Loss: 0.441408: : 5it [00:02,  2.09it/s]
5it [00:01,  2.76it/s]
Loss on validation: 0.40323426

# Found cached data data/sets/liberty.pt
100%|██████████| 5000/5000 [00:00<00:00, 60146.50it/s]
# Found cached data data/sets/notredame.pt
100%|██████████| 5000/5000 [00:00<00:00, 58688.02it/s]
Train Epoch: 3 [0/5000 (0%)]    Loss: 0.411440: : 5it [00:01,  2.51it/s]
5it [00:01,  2.75it/s]
Loss on validation: 0.38375980

# Found cached data data/sets/liberty.pt
100%|██████████| 5000/5000 [00:00<00:00, 52400.77it/s]
# Found cached data data/sets/notredame.pt
100%|██████████| 5000/5000 [00:00<00:00, 56621.78it/s]
Train Epoch: 4 [0/5000 (0%)]    Loss: 0.444416: : 5it [00:02,  2.42it/s]
5it [00:01,  2.73it/s]
Loss on validation: 0.37915759

# Found cached data data/sets/liberty.pt
```

```
100%|██████████| 5000/5000 [00:00<00:00, 31127.95it/s]
# Found cached data data/sets/notredame.pt
100%|██████████| 5000/5000 [00:00<00:00, 53796.30it/s]
```

Select the best model, and save it as CNN.*pth;* can be 1, 2, or 3.

Generate deep features

In this section, we will use your trained network to generate deep features for each patch:

1. load weights
2. load patches
3. get deep features

BackToTop

Load network weights

BackToSection

In [15]:
```
trained_weight_path = "models/liberty_train/_liberty_min_as_fliprot/checkpoint_4.pth" # suppose you select  checkpoint_4.
test_model = DesNet()
if args.cuda:
    test_model.cuda()
trained_weight = torch.load(trained_weight_path)['state_dict']
test_model.load_state_dict(trained_weight)
test_model.eval()
```

Out[15]:
```
DesNet(
  (features): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(32, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
    (5): ReLU()
    (6): Dropout(p=0.3, inplace=False)
    (7): Conv2d(128, 128, kernel_size=(8, 8), stride=(1, 1), bias=False)
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=False, track_running_stats=True)
  )
)
```

Load raw patch files

Assume that the raw patch file is stored as patches.pth

BackToSection

In [17]:
```
patches_dir = "patches.pth"            # these patches are from keypoint detection results
patches = torch.load(patches_dir)
print(patches.shape)                  # in your case, the shape should be [10, 200, 1, 32, 32]
num_imgs, num_pts, _, _, _ = patches.shape
patches = patches[0].view(-1, 1, 32, 32).cuda()
print(patches.shape)
```

```
torch.Size([1, 10, 1, 32, 32])
torch.Size([10, 1, 32, 32])
```

Get deep features

BackToSection

In [18]:
```
features = test_model(patches)
print(features.shape)
features = features.view(num_imgs, num_pts, 128).cpu().data
print(features.shape)                  # in your case, the shape should be [10, 200, 128]
```

```
torch.Size([10, 128])
torch.Size([1, 10, 128])
```

In [19]:

```python
# save to file, with the name of *_features_CNN*.pth
features_dir = "features_CNN1.pth"
torch.save(features, features_dir)
```

In [ ]:

In [ ]:

In [ ]: