KeypointDetection_1155135359

March 22, 2020

1 Keypoint Detection on Human Body Silhouette Images 1155135359

Huang Hejun (s1155135359)

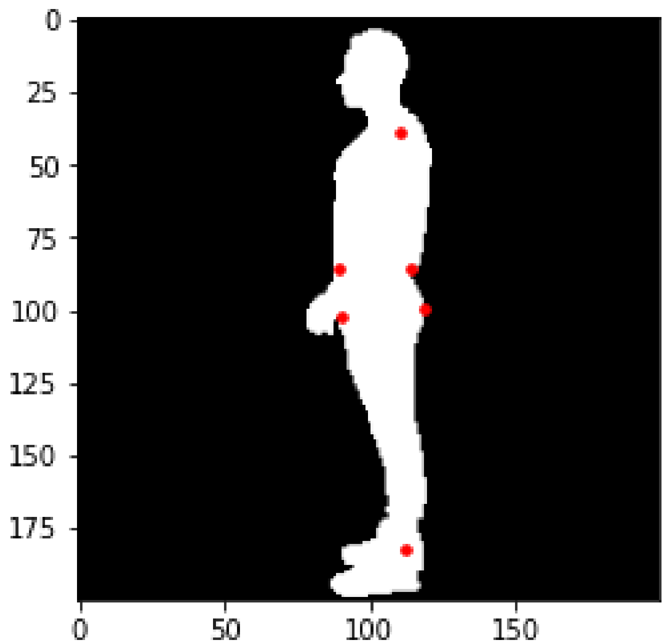
- Author: Zishun Liu liuzishun@gmail.com Jan. 2020
- Amended: Hejun Huang 1155135359@link.cuhk.edu.hk March 2020

1.1 Side view data process

[0]:
```
from google.colab import drive
drive.mount('/content/gdrive/')
ROOT_FOLDER = './gdrive/My Drive/Colab Notebooks/MAEG5735-2020-Assignment2/'

import glob
print('\nContents in the data folder:')
for x in glob.glob(ROOT_FOLDER+'data/*'):
  print(x)

import matplotlib.pyplot as plt
import numpy as np

def draw_points(image, kpts):
    plt.figure()
    plt.imshow(image, cmap='gray')
    keypoints = (kpts+0.5)*IMG_SIZE
    plt.scatter(keypoints[:, 0], keypoints[:, 1], s=50, marker='.', c='r')

# load SIDE view data
IMG_SIZE = 200
# The total amount of input data is 5000000, which is converted to binary by np.
↪shape function.
# That is, the form of m1 is 400000000.
IMG_S_TRAIN = np.load(ROOT_FOLDER+'data/train_img_side.npy')
print(np.shape(IMG_S_TRAIN))
IMG_S_TRAIN = np.unpackbits(IMG_S_TRAIN).reshape((-1,IMG_SIZE,IMG_SIZE))
IMG_S_TEST = np.load(ROOT_FOLDER+'data/test_img_side.npy')
IMG_S_TEST = np.unpackbits(IMG_S_TEST).reshape((-1,IMG_SIZE,IMG_SIZE))
```

```python
KPT_S_TRAIN = np.load(ROOT_FOLDER+'data/train_kpt_side.npy')/IMG_SIZE - 0.5
KPT_S_TEST = np.load(ROOT_FOLDER+'data/test_kpt_side.npy')/IMG_SIZE - 0.5
print(np.shape(IMG_S_TRAIN))
# show one
# idx = np.random.randint(0,1000)
idx=145
draw_points(IMG_S_TEST[idx,:,:], KPT_S_TEST[idx,:,:])
```

```
Drive already mounted at /content/gdrive/; to attempt to forcibly remount, call
drive.mount("/content/gdrive/", force_remount=True).

Contents in the data folder:
./gdrive/My Drive/Colab Notebooks/MAEG5735-2020-Assignment2/data/README.txt
./gdrive/My Drive/Colab
Notebooks/MAEG5735-2020-Assignment2/data/train_kpt_side.npy
./gdrive/My Drive/Colab
Notebooks/MAEG5735-2020-Assignment2/data/train_kpt_front.npy
./gdrive/My Drive/Colab
Notebooks/MAEG5735-2020-Assignment2/data/test_kpt_side.npy
./gdrive/My Drive/Colab
Notebooks/MAEG5735-2020-Assignment2/data/test_img_front.npy
./gdrive/My Drive/Colab
Notebooks/MAEG5735-2020-Assignment2/data/test_img_side.npy
./gdrive/My Drive/Colab
Notebooks/MAEG5735-2020-Assignment2/data/train_img_front.npy
./gdrive/My Drive/Colab
Notebooks/MAEG5735-2020-Assignment2/data/train_img_side.npy
(5000000,)
(1000, 200, 200)
```



1.1.1 Dataset loader

[0]:
```
import torch
from torch.utils.data import Dataset

class SideKeypointsDataset(Dataset):
    '''Side View Keypoints Dataset'''
    def __init__(self, img, kpt, train=True, transform=None):
        self.img = img
        self.kpt = kpt
        self.train = train
        self.transform = transform

    def __len__(self):
        return self.img.shape[0]

    def __getitem__(self, idx):
        image = self.img[idx,:,:].astype(np.float32)
        if self.train:
            keypoints = self.kpt[idx,:,:].ravel().astype(np.float32)
        else:
            keypoints = None
        sample = {'image': image, 'keypoints': keypoints}
        if self.transform:
            sample = self.transform(sample)
```

```
        return sample
```

[0]:
```
class FrontKeypointsDataset(Dataset):
    '''Front View Keypoints Dataset'''
    def __init__(self, img, kpt, train=True, transform=None):
        self.img = img
        self.kpt = kpt
        self.train = train
        self.transform = transform

    def __len__(self):
        return self.img.shape[0]

    def __getitem__(self, idx):
        image = self.img[idx,:,:].astype(np.float32)
        if self.train:
            keypoints = self.kpt[idx,:,:].ravel().astype(np.float32)
        else:
            keypoints = None
        sample = {'image': image, 'keypoints': keypoints}
        if self.transform:
            sample = self.transform(sample)
        return sample
```

[0]:
```
from torch.utils.data.sampler import SubsetRandomSampler
#
def prepare_train_valid_loaders(trainset, valid_size=0.3, batch_size=64):
    '''
    Split trainset data and prepare DataLoader for training and validation

    Args:
        trainset (Dataset): data
        valid_size (float): validation size, defalut=0.2
        batch_size (int) : batch size, default=128
    '''

    # obtain training indices that will be used for validation
    num_train = len(trainset)
    indices = list(range(num_train))
    np.random.shuffle(indices)
    split = int(np.floor(valid_size * num_train))
    train_idx, valid_idx = indices[split:], indices[:split]

    # define samplers for obtaining training and validation batches
    train_sampler = SubsetRandomSampler(train_idx)
    valid_sampler = SubsetRandomSampler(valid_idx)

    # prepare data loaders
```

```python
    train_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                sampler=train_sampler)
    valid_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                sampler=valid_sampler)
    return train_loader, valid_loader
```

[0]:
```python
from torchvision import transforms
import cv2

class Rescale(object):
    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        self.output_size = output_size

    def __call__(self, sample):
        image, key_pts = sample['image'], sample['keypoints']
        h, w = image.shape[:2]
        new_w = np.random.randint(w, self.output_size)
        new_h = new_w
        new_h, new_w = int(new_h), int(new_w)
        img = cv2.resize(image, (new_w, new_h))
        if key_pts is not None:
            return {'image': img, 'keypoints': key_pts}
        else:
            return {'image': img}

class RandomCrop(object):
    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        if isinstance(output_size, int):
            self.output_size = (output_size, output_size)
        else:
            assert len(output_size) == 2
            self.output_size = output_size

    def __call__(self, sample):
        image, key_pts = sample['image'], sample['keypoints']
        h, w = image.shape[:2]
        new_h, new_w = self.output_size
        if h == new_h:
            return sample
        top = np.random.randint(0, h - new_h)
        left = np.random.randint(0, w - new_w)
        #left = top # temp
        image = image[top: top + new_h,
                      left: left + new_w]
        if key_pts is not None:
```

```python
            #key_pts = key_pts - [left/output_size, top/output_size]
            key_pts[0::2] = ((key_pts[0::2]+0.5)*w-left)/new_w-0.5
            key_pts[1::2] = ((key_pts[1::2]+0.5)*h-top)/new_h-0.5
            return {'image': image, 'keypoints': key_pts}
        else:
            return {'image': image}

class ToTensor(object):
    '''Convert ndarrays in sample to Tensors.'''
    def __call__(self, sample):
        image, keypoints = sample['image'], sample['keypoints']
        # swap color axis because
        # numpy image: H x W x C
        # torch image: C X H X W
        image = image.reshape(1, IMG_SIZE, IMG_SIZE)
        image = torch.from_numpy(image)
        if keypoints is not None:
            keypoints = torch.from_numpy(keypoints)
            return {'image': image, 'keypoints': keypoints}
        else:
            return {'image': image}

batch_size = 32
valid_size = 0.2 # percentage of training set to use as validation

# Define a transform to normalize the data
tsfm_train = transforms.Compose([Rescale(205), RandomCrop(200), ToTensor()])
tsfm_test = transforms.Compose([ToTensor()])

# Load the training data and test data
trainset = SideKeypointsDataset(IMG_S_TRAIN, KPT_S_TRAIN, transform=tsfm_train)
testset = SideKeypointsDataset(IMG_S_TEST, None, train=False,␣
 ↪transform=tsfm_test)

# prepare data loaders
train_loader, valid_loader = prepare_train_valid_loaders(trainset,
                                                         valid_size,
                                                         batch_size)

test_loader = torch.utils.data.DataLoader(testset, batch_size=batch_size)
```

1.1.2 MLP

Neural network structure

[0]:
```python
from torch import nn, optim
import torch.nn.functional as F
```

```
class MLP(nn.Module):
    def __init__(self, input_size, output_size, hidden_layers, drop_p =0.5):
        '''
        Buid a forward network with arbitrary hidden layers.
        Arguments
            ---------
            input_size (integer): size of the input layer
            output_size (integer): size of the output layer
            hidden_layers (list of integers):, the sizes of each hidden layers
        '''
        super(MLP, self).__init__()
        # hidden layers
        layer_sizes = [(input_size, hidden_layers[0])] \
                      + list(zip(hidden_layers[:-1], hidden_layers[1:]))
        self.hidden_layers = nn.ModuleList([nn.Linear(h1, h2)
                                            for h1, h2 in layer_sizes])
        self.output = nn.Linear(hidden_layers[-1], output_size)
        self.dropout = nn.Dropout(drop_p)

    def forward(self, x):
        ''' Forward pass through the network, returns the output logits '''
        # flatten inputs
        x = x.view(x.shape[0], -1)
        for layer in self.hidden_layers:
            x = F.relu(layer(x))
            x = self.dropout(x)
        x = self.output(x)
        return x
```

[0]:
```
def train(train_loader, valid_loader, model, criterion, optimizer,
          n_epochs=50, saved_model='model.pt'):
    '''
    Train the model

    Args:
        train_loader (DataLoader): DataLoader for train Dataset
        valid_loader (DataLoader): DataLoader for valid Dataset
        model (nn.Module): model to be trained on
        criterion (torch.nn): loss funtion
        optimizer (torch.optim): optimization algorithms
        n_epochs (int): number of epochs to train the model
        saved_model (str): file path for saving model

    Return:
        tuple of train_losses, valid_losses
    '''
```

```
    # initialize tracker for minimum validation loss
    valid_loss_min = np.Inf # set initial "min" to infinity

    train_losses = []
    valid_losses = []

    for epoch in range(n_epochs):
        # monitor training loss
        train_loss = 0.0
        valid_loss = 0.0

        ###################
        # train the model #
        ###################
        model.train() # prep model for training
        for batch in train_loader:
            # clear the gradients of all optimized variables
            optimizer.zero_grad()
            # forward pass: compute predicted outputs by passing inputs to the␣
↪model
            output = model(batch['image'].to(device))
            # calculate the loss
            loss = criterion(output, batch['keypoints'].to(device))
            # backward pass: compute gradient of the loss with respect to model␣
↪parameters
            loss.backward()
            # perform a single optimization step (parameter update)
            optimizer.step()
            # update running training loss
            train_loss += loss.item()*batch['image'].size(0)

        ######################
        # validate the model #
        ######################
        model.eval() # prep model for evaluation
        for batch in valid_loader:
            # forward pass: compute predicted outputs by passing inputs to the␣
↪model
            output = model(batch['image'].to(device))
            # calculate the loss
            loss = criterion(output, batch['keypoints'].to(device))
            # update running validation loss
            valid_loss += loss.item()*batch['image'].size(0)

        # print training/validation statistics
        # calculate average Root Mean Square loss over an epoch
        train_loss = np.sqrt(train_loss/len(train_loader.sampler.indices))
```

```
        valid_loss = np.sqrt(valid_loss/len(valid_loader.sampler.indices))

        train_losses.append(train_loss)
        valid_losses.append(valid_loss)

        print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'
              .format(epoch+1, train_loss, valid_loss))

        # save model if validation loss has decreased
        if valid_loss <= valid_loss_min:
            print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model␣
↪...'
                  .format(valid_loss_min, valid_loss))
            torch.save(model.state_dict(), saved_model)
            valid_loss_min = valid_loss

    return train_losses, valid_losses
```

Criterion and optimizer for MLP

[0]:
```
from torch import optim
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# output_size ?
# Side view keypoints: 6x2 = 12
model = MLP(input_size=IMG_SIZE*IMG_SIZE, output_size=12,
            hidden_layers=[128, 64], drop_p=0.2)
model = model.to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.005)

train_losses, valid_losses = train(train_loader, valid_loader,
                                   model, criterion, optimizer,
                                   n_epochs=30,
                                   saved_model=ROOT_FOLDER+'model.pt')
```

```
Epoch: 1        Training Loss: 2.004709         Validation Loss: 0.125718
Validation loss decreased (inf --> 0.125718).  Saving model ...
Epoch: 2        Training Loss: 0.169977         Validation Loss: 0.093189
Validation loss decreased (0.125718 --> 0.093189).  Saving model ...
Epoch: 3        Training Loss: 0.136124         Validation Loss: 0.055776
Validation loss decreased (0.093189 --> 0.055776).  Saving model ...
Epoch: 4        Training Loss: 0.100797         Validation Loss: 0.053726
Validation loss decreased (0.055776 --> 0.053726).  Saving model ...
Epoch: 5        Training Loss: 0.062363         Validation Loss: 0.035204
Validation loss decreased (0.053726 --> 0.035204).  Saving model ...
Epoch: 6        Training Loss: 0.057763         Validation Loss: 0.033247
Validation loss decreased (0.035204 --> 0.033247).  Saving model ...
```

```
Epoch: 7        Training Loss: 0.055643         Validation Loss: 0.037422
Epoch: 8        Training Loss: 0.054215         Validation Loss: 0.037629
Epoch: 9        Training Loss: 0.052614         Validation Loss: 0.032642
Validation loss decreased (0.033247 --> 0.032642).  Saving model ...
Epoch: 10       Training Loss: 0.068150         Validation Loss: 0.035586
Epoch: 11       Training Loss: 0.050291         Validation Loss: 0.031937
Validation loss decreased (0.032642 --> 0.031937).  Saving model ...
Epoch: 12       Training Loss: 0.049558         Validation Loss: 0.034148
Epoch: 13       Training Loss: 0.049052         Validation Loss: 0.031652
Validation loss decreased (0.031937 --> 0.031652).  Saving model ...
Epoch: 14       Training Loss: 0.050222         Validation Loss: 0.033491
Epoch: 15       Training Loss: 0.048847         Validation Loss: 0.032277
Epoch: 16       Training Loss: 0.046724         Validation Loss: 0.031059
Validation loss decreased (0.031652 --> 0.031059).  Saving model ...
Epoch: 17       Training Loss: 0.047209         Validation Loss: 0.031657
Epoch: 18       Training Loss: 0.048156         Validation Loss: 0.033286
Epoch: 19       Training Loss: 0.046277         Validation Loss: 0.032889
Epoch: 20       Training Loss: 0.046005         Validation Loss: 0.030772
Validation loss decreased (0.031059 --> 0.030772).  Saving model ...
Epoch: 21       Training Loss: 0.044641         Validation Loss: 0.031325
Epoch: 22       Training Loss: 0.046882         Validation Loss: 0.031747
Epoch: 23       Training Loss: 0.045153         Validation Loss: 0.031164
Epoch: 24       Training Loss: 0.043555         Validation Loss: 0.030453
Validation loss decreased (0.030772 --> 0.030453).  Saving model ...
Epoch: 25       Training Loss: 0.043125         Validation Loss: 0.031944
Epoch: 26       Training Loss: 0.043456         Validation Loss: 0.031309
Epoch: 27       Training Loss: 0.043521         Validation Loss: 0.032236
Epoch: 28       Training Loss: 0.043599         Validation Loss: 0.032507
Epoch: 29       Training Loss: 0.046110         Validation Loss: 0.031234
Epoch: 30       Training Loss: 0.044258         Validation Loss: 0.031661
```

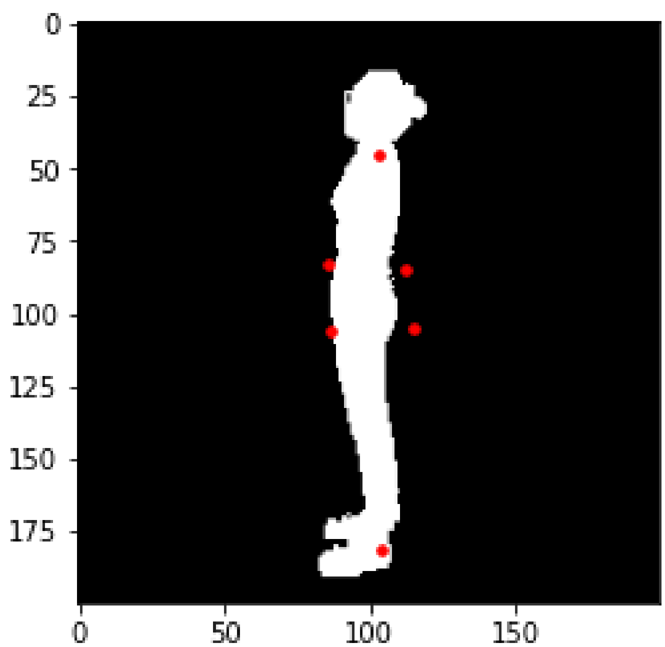
Evaluation for MLP

[0]:
```python
def predict(data_loader, model):
    '''
    Predict keypoints
    Args:
        data_loader (DataLoader): DataLoader for Dataset
        model (nn.Module): trained model for prediction.
    Return:
        predictions (array-like): keypoints in float (no. of images x␣
↪keypoints).
    '''

    model.eval() # prep model for evaluation

    with torch.no_grad():
        for i, batch in enumerate(data_loader):
```

```
            # forward pass: compute predicted outputs by passing inputs to the␣
 ↪model
            output = model(batch['image'].to(device)).cpu().numpy()
            if i == 0:
                predictions = output
            else:
                predictions = np.vstack((predictions, output))

    return predictions
```

[0]:
```
model.load_state_dict(torch.load(ROOT_FOLDER+'model.pt'))
predictions = predict(test_loader, model)

print('Error: ', np.linalg.norm(predictions-KPT_S_TEST.reshape((predictions.
 ↪shape[0],-1)), axis=1).mean())

idx = np.random.randint(0,277)
draw_points(IMG_S_TEST[idx,:,:], predictions[idx,:].reshape((-1,2)))
```

```
Error:  0.0933923989487642
```



1.1.3 CNN

Neural network structure

[0]:
```python
class CNN(nn.Module):
    def __init__(self, output_size):
      super(CNN, self).__init__()
      # 200 x 20
      self.conv1 = nn.Conv2d(1, 32, 5, padding=2)
      # (w-f)/s+1 = 196
      self.pool1 = nn.MaxPool2d(4, 4)
      # 98
      self.conv2 = nn.Conv2d(32, 64, 3, padding=2)
      # (98-3)/1 + 1 = 96
      self.pool2 = nn.MaxPool2d(2, 2)
      # 48
      self.conv3 = nn.Conv2d(64, 128, 3)
      # (48-3)/1 + 1 = 46
      self.pool3 = nn.MaxPool2d(2, 2)
      # 23
      self.conv4 = nn.Conv2d(128, 256, 3, stride=2)
      # (23-3)/2 + 1 = 11
      self.conv5 = nn.Conv2d(256, 512, 1)
      # (11-1)/2+1 = 6

      # Linear Layer
      self.fc1 = nn.Linear(12800, 1024)
      self.fc2 = nn.Linear(1024, output_size)
      self.drop1 = nn.Dropout(p=0.1)
      self.drop2 = nn.Dropout(p=0.25)
      self.drop3 = nn.Dropout(p=0.25)
      self.drop4 = nn.Dropout(p=0.25)
      self.drop5 = nn.Dropout(p=0.35)
      self.drop6 = nn.Dropout(p=0.4)

    def forward(self, x):
      x = self.pool1(F.relu(self.conv1(x)))
      x = self.drop1(x)
      x = self.pool2(F.relu(self.conv2(x)))
      x = self.drop2(x)
      x = self.pool3(F.relu(self.conv3(x)))
      x = self.drop3(x)
      x = F.relu(self.conv4(x))
      x = self.drop4(x)
      x = F.relu(self.conv5(x))
      x = self.drop5(x)
      x = x.view(x.size(0), -1)
      x = F.relu(self.fc1(x))
      x == self.drop6(x)
      x = self.fc2(x)
      return x
```

Criterion and optimizer for CNN

[0]:
```
model = CNN(output_size=12)
model = model.to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.0005)
# optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.8)

train_losses, valid_losses = train(train_loader, valid_loader,
                                   model, criterion, optimizer,
                                   n_epochs=100,
                                   saved_model=ROOT_FOLDER+'model_side.pt')
```

```
Epoch: 1        Training Loss: 0.070959         Validation Loss: 0.034161
Validation loss decreased (inf --> 0.034161).  Saving model ...
Epoch: 2        Training Loss: 0.037768         Validation Loss: 0.032527
Validation loss decreased (0.034161 --> 0.032527).  Saving model ...
Epoch: 3        Training Loss: 0.034801         Validation Loss: 0.030405
Validation loss decreased (0.032527 --> 0.030405).  Saving model ...
Epoch: 4        Training Loss: 0.032094         Validation Loss: 0.028902
Validation loss decreased (0.030405 --> 0.028902).  Saving model ...
Epoch: 5        Training Loss: 0.030734         Validation Loss: 0.028253
Validation loss decreased (0.028902 --> 0.028253).  Saving model ...
Epoch: 6        Training Loss: 0.030012         Validation Loss: 0.027474
Validation loss decreased (0.028253 --> 0.027474).  Saving model ...
Epoch: 7        Training Loss: 0.029681         Validation Loss: 0.030193
Epoch: 8        Training Loss: 0.029580         Validation Loss: 0.027311
Validation loss decreased (0.027474 --> 0.027311).  Saving model ...
Epoch: 9        Training Loss: 0.029519         Validation Loss: 0.027906
Epoch: 10       Training Loss: 0.029388         Validation Loss: 0.027840
Epoch: 11       Training Loss: 0.028449         Validation Loss: 0.029245
Epoch: 12       Training Loss: 0.027831         Validation Loss: 0.026901
Validation loss decreased (0.027311 --> 0.026901).  Saving model ...
Epoch: 13       Training Loss: 0.027325         Validation Loss: 0.026284
Validation loss decreased (0.026901 --> 0.026284).  Saving model ...
Epoch: 14       Training Loss: 0.026317         Validation Loss: 0.027530
Epoch: 15       Training Loss: 0.026103         Validation Loss: 0.024540
Validation loss decreased (0.026284 --> 0.024540).  Saving model ...
Epoch: 16       Training Loss: 0.026273         Validation Loss: 0.027680
Epoch: 17       Training Loss: 0.025759         Validation Loss: 0.023895
Validation loss decreased (0.024540 --> 0.023895).  Saving model ...
Epoch: 18       Training Loss: 0.025542         Validation Loss: 0.023561
Validation loss decreased (0.023895 --> 0.023561).  Saving model ...
Epoch: 19       Training Loss: 0.025917         Validation Loss: 0.024897
Epoch: 20       Training Loss: 0.025117         Validation Loss: 0.027275
Epoch: 21       Training Loss: 0.025714         Validation Loss: 0.022826
Validation loss decreased (0.023561 --> 0.022826).  Saving model ...
Epoch: 22       Training Loss: 0.024695         Validation Loss: 0.022825
```

```
Validation loss decreased (0.022826 --> 0.022825).  Saving model ...
Epoch: 23       Training Loss: 0.024477         Validation Loss: 0.023706
Epoch: 24       Training Loss: 0.025359         Validation Loss: 0.026153
Epoch: 25       Training Loss: 0.024398         Validation Loss: 0.022644
Validation loss decreased (0.022825 --> 0.022644).  Saving model ...
Epoch: 26       Training Loss: 0.024109         Validation Loss: 0.022576
Validation loss decreased (0.022644 --> 0.022576).  Saving model ...
Epoch: 27       Training Loss: 0.024262         Validation Loss: 0.022185
Validation loss decreased (0.022576 --> 0.022185).  Saving model ...
Epoch: 28       Training Loss: 0.024154         Validation Loss: 0.022536
Epoch: 29       Training Loss: 0.023276         Validation Loss: 0.023277
Epoch: 30       Training Loss: 0.023247         Validation Loss: 0.022264
Epoch: 31       Training Loss: 0.022927         Validation Loss: 0.022266
Epoch: 32       Training Loss: 0.023148         Validation Loss: 0.024242
Epoch: 33       Training Loss: 0.023262         Validation Loss: 0.022577
Epoch: 34       Training Loss: 0.022745         Validation Loss: 0.021667
Validation loss decreased (0.022185 --> 0.021667).  Saving model ...
Epoch: 35       Training Loss: 0.022854         Validation Loss: 0.022155
Epoch: 36       Training Loss: 0.022615         Validation Loss: 0.022492
Epoch: 37       Training Loss: 0.022021         Validation Loss: 0.023456
Epoch: 38       Training Loss: 0.022051         Validation Loss: 0.023327
Epoch: 39       Training Loss: 0.022270         Validation Loss: 0.023638
Epoch: 40       Training Loss: 0.022584         Validation Loss: 0.022035
Epoch: 41       Training Loss: 0.021512         Validation Loss: 0.023524
Epoch: 42       Training Loss: 0.021835         Validation Loss: 0.021806
Epoch: 43       Training Loss: 0.021204         Validation Loss: 0.021704
Epoch: 44       Training Loss: 0.021223         Validation Loss: 0.023859
Epoch: 45       Training Loss: 0.021964         Validation Loss: 0.021722
Epoch: 46       Training Loss: 0.021560         Validation Loss: 0.023248
Epoch: 47       Training Loss: 0.020815         Validation Loss: 0.021723
Epoch: 48       Training Loss: 0.021543         Validation Loss: 0.022690
Epoch: 49       Training Loss: 0.020144         Validation Loss: 0.021218
Validation loss decreased (0.021667 --> 0.021218).  Saving model ...
Epoch: 50       Training Loss: 0.020690         Validation Loss: 0.021948
Epoch: 51       Training Loss: 0.021787         Validation Loss: 0.026554
Epoch: 52       Training Loss: 0.020816         Validation Loss: 0.024335
Epoch: 53       Training Loss: 0.021175         Validation Loss: 0.022091
Epoch: 54       Training Loss: 0.019682         Validation Loss: 0.021354
Epoch: 55       Training Loss: 0.019578         Validation Loss: 0.022953
Epoch: 56       Training Loss: 0.019267         Validation Loss: 0.022012
Epoch: 57       Training Loss: 0.019422         Validation Loss: 0.022628
Epoch: 58       Training Loss: 0.019354         Validation Loss: 0.022823
Epoch: 59       Training Loss: 0.019798         Validation Loss: 0.021598
Epoch: 60       Training Loss: 0.018995         Validation Loss: 0.022030
Epoch: 61       Training Loss: 0.018204         Validation Loss: 0.021426
Epoch: 62       Training Loss: 0.018427         Validation Loss: 0.022207
Epoch: 63       Training Loss: 0.018690         Validation Loss: 0.021247
Epoch: 64       Training Loss: 0.017793         Validation Loss: 0.021832
```

```
Epoch: 65       Training Loss: 0.018444         Validation Loss: 0.022716
Epoch: 66       Training Loss: 0.018074         Validation Loss: 0.022572
Epoch: 67       Training Loss: 0.017796         Validation Loss: 0.021299
Epoch: 68       Training Loss: 0.017797         Validation Loss: 0.021425
Epoch: 69       Training Loss: 0.016961         Validation Loss: 0.021517
Epoch: 70       Training Loss: 0.017863         Validation Loss: 0.025106
Epoch: 71       Training Loss: 0.017924         Validation Loss: 0.023268
Epoch: 72       Training Loss: 0.018440         Validation Loss: 0.024036
Epoch: 73       Training Loss: 0.017924         Validation Loss: 0.021009
Validation loss decreased (0.021218 --> 0.021009).  Saving model ...
Epoch: 74       Training Loss: 0.016753         Validation Loss: 0.022472
Epoch: 75       Training Loss: 0.016699         Validation Loss: 0.021053
Epoch: 76       Training Loss: 0.016161         Validation Loss: 0.021365
Epoch: 77       Training Loss: 0.016894         Validation Loss: 0.022131
Epoch: 78       Training Loss: 0.015962         Validation Loss: 0.022139
Epoch: 79       Training Loss: 0.016928         Validation Loss: 0.020731
Validation loss decreased (0.021009 --> 0.020731).  Saving model ...
Epoch: 80       Training Loss: 0.015862         Validation Loss: 0.021165
Epoch: 81       Training Loss: 0.015823         Validation Loss: 0.021108
Epoch: 82       Training Loss: 0.015992         Validation Loss: 0.020970
Epoch: 83       Training Loss: 0.017576         Validation Loss: 0.021218
Epoch: 84       Training Loss: 0.016458         Validation Loss: 0.021659
Epoch: 85       Training Loss: 0.015451         Validation Loss: 0.021128
Epoch: 86       Training Loss: 0.015944         Validation Loss: 0.022625
Epoch: 87       Training Loss: 0.016677         Validation Loss: 0.022783
Epoch: 88       Training Loss: 0.016052         Validation Loss: 0.021274
Epoch: 89       Training Loss: 0.015560         Validation Loss: 0.022509
Epoch: 90       Training Loss: 0.015629         Validation Loss: 0.021778
Epoch: 91       Training Loss: 0.015623         Validation Loss: 0.020894
Epoch: 92       Training Loss: 0.016011         Validation Loss: 0.020865
Epoch: 93       Training Loss: 0.015728         Validation Loss: 0.021837
Epoch: 94       Training Loss: 0.015158         Validation Loss: 0.021416
Epoch: 95       Training Loss: 0.017878         Validation Loss: 0.021943
Epoch: 96       Training Loss: 0.017016         Validation Loss: 0.022247
Epoch: 97       Training Loss: 0.016179         Validation Loss: 0.021336
Epoch: 98       Training Loss: 0.015786         Validation Loss: 0.021115
Epoch: 99       Training Loss: 0.016679         Validation Loss: 0.022837
Epoch: 100      Training Loss: 0.016907         Validation Loss: 0.022491
```

Evaluation for CNN

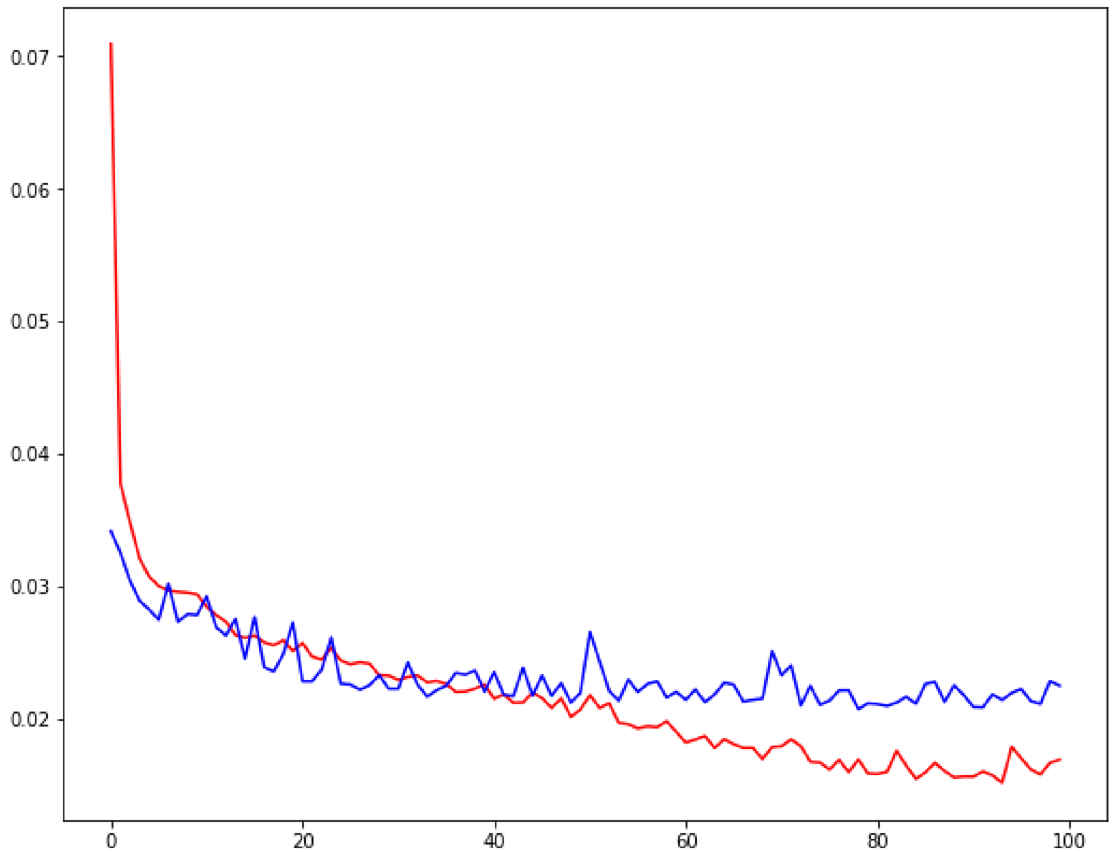
[0]:
```
# Evaluate this one
model.load_state_dict(torch.load(ROOT_FOLDER+'model_side.pt'))
predictions = predict(test_loader, model)

# baseline: 0.069
print('Error: ', np.linalg.norm(predictions-KPT_S_TEST.reshape((predictions.
 ↪shape[0],-1)), axis=1).mean())
```

```
Error:  0.05954668872645633
```

[0]:
```python
# Draw the changing curve
n_epochs=100
x=range(0,n_epochs)
plt.figure(figsize=(10,8))
y1=train_losses
y2=valid_losses
print(np.shape(x))
print(np.shape(y1))
print(np.shape(y2))
plt.plot(x,y1,color="red")
plt.plot(x,y2,color="blue")
print(y1)
```

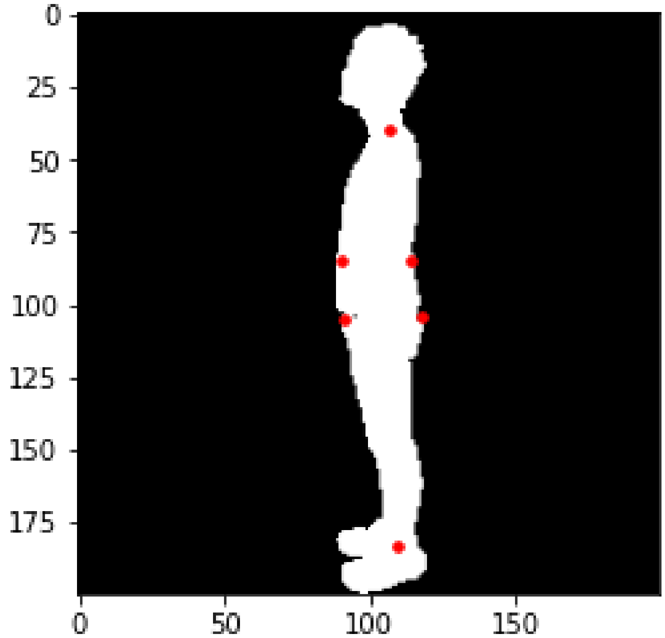
```
(100,)
(100,)
(100,)
[0.07095920494445143, 0.03776760966272079, 0.03480145217005998,
0.032093732756957816, 0.030733572367433296, 0.03001192055365873,
0.029680886049488205, 0.029579894671999975, 0.029519113823409146,
0.029388204338040676, 0.028449407639119316, 0.02783068925566834,
0.027325099945515435, 0.02631672392312688, 0.02610266567791651,
0.02627257751330082, 0.0257587499494943, 0.025542100086833826,
0.025916661418730308, 0.025117216299316365, 0.025713575200473095,
0.024694610831431957, 0.024477311840406708, 0.025358991885295407,
0.02439758505291692, 0.02410925287861962, 0.02426243980455123,
0.024154079837468655, 0.02327592731381889, 0.023246847389809318,
0.022927408964212085, 0.02314808871999895, 0.02326156255427784,
0.02274518354877993, 0.02285425962931189, 0.022614935973802486,
0.02202126570019976, 0.02205086614606782, 0.02227038462800707,
0.0225841735332775, 0.0215116071153951, 0.021834674374934835,
0.021204437888784625, 0.021222809901156093, 0.021964232816981896,
0.021560358124886537, 0.020815137573195135, 0.021543069204397122,
0.02014387875024094, 0.02069010581051444, 0.021786661838105334,
0.020816375772777264, 0.02117548211868437, 0.01968245931721512,
0.019578228168066356, 0.0192670868356842, 0.01942182833506954,
0.019353649697968787, 0.019798275383451015, 0.01899476399661342,
0.018203612404701492, 0.018426913174635295, 0.01869039189031255,
0.017792786556445877, 0.01844428321681181, 0.01807419539735657,
0.01779584446527657, 0.017797209128936967, 0.016960903979533655,
0.017863039933993168, 0.017923525547290077, 0.018440277484683246,
0.017924221501362452, 0.01675303269568373, 0.01669865789921662,
0.016160854045226592, 0.01689394639083321, 0.01596160929806977,
0.016928351905708065, 0.01586248834269167, 0.015823277387684045,
0.015992279999793444, 0.017575592514238726, 0.01645798503614645,
0.015451163800651785, 0.015943749534972778, 0.016676689309537213,
```

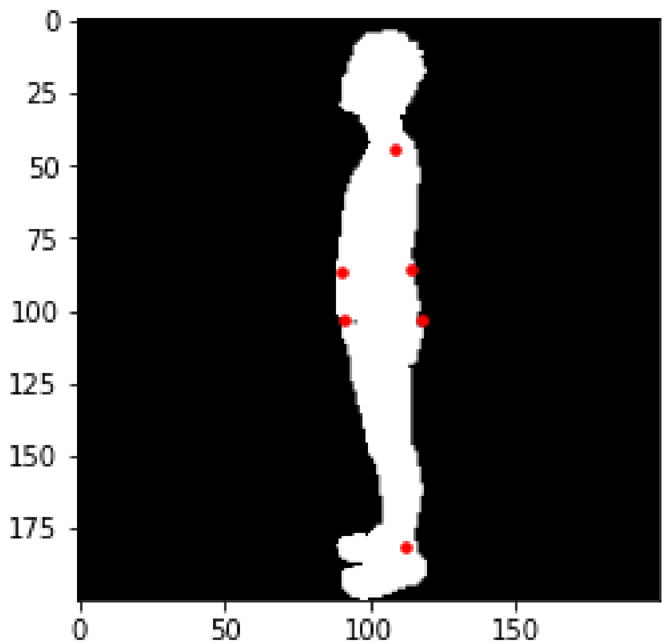
```
0.016051519798122027, 0.015560141617421223, 0.015629360309562055,
0.01562284646616277, 0.016010558821549246, 0.01572806941905491,
0.015158374181424199, 0.017877890171963375, 0.017016387360700998,
0.016179021924585677, 0.015786154006149692, 0.016679397975951627,
0.016907316793604105]
```



[0]:
```python
idx = np.random.randint(0,277)
print(idx)
draw_points(IMG_S_TEST[idx,:,:], predictions[idx,:].reshape((-1,2)))
draw_points(IMG_S_TEST[idx,:,:], KPT_S_TEST[idx,:,:])
```

```
7
```


0
25
50
75
100
125
150
175
0
50
100
150


0
25
50
75
100
125
150
175
0
50
100
150

[0]:

1.2 Front view data process

1.2.1 Notice:

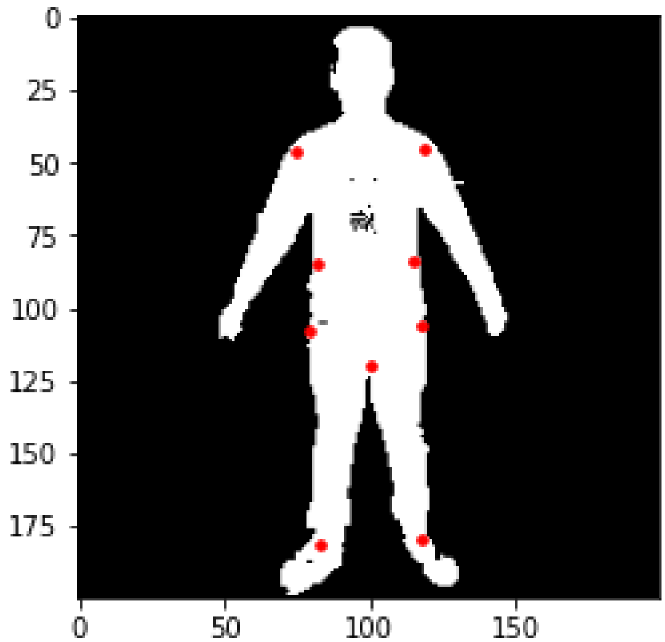
1. **According to the model results between MLP and CNN, I choose to use the CNN rather than the MLP in the front_view dataset.**
2. **The next step is training the front view based on the CNN structure.**

1.2.2 Dataset loader

[0]:

```
# load Front view data
# The total amount of input data is 5000000, which is converted to binary by np.
↪shape function.
# That is, the form of m1 is 400000000.
IMG_S_TRAIN = np.load(ROOT_FOLDER+'data/train_img_front.npy')
print(np.shape(IMG_S_TRAIN))
IMG_F_TRAIN = np.unpackbits(IMG_S_TRAIN).reshape((-1,IMG_SIZE,IMG_SIZE))
print(np.shape(IMG_S_TRAIN))
IMG_F_TEST = np.load(ROOT_FOLDER+'data/test_img_front.npy')
IMG_F_TEST = np.unpackbits(IMG_S_TEST).reshape((-1,IMG_SIZE,IMG_SIZE))
KPT_F_TRAIN = np.load(ROOT_FOLDER+'data/train_kpt_front.npy')/IMG_SIZE - 0.5
print(np.shape(IMG_S_TRAIN))

# show one
idx = np.random.randint(0,1000)
draw_points(IMG_F_TRAIN[idx,:,:], KPT_F_TRAIN[idx,:,:])
```

```
(5000000,)
(5000000,)
(5000000,)
```



Criterion and optimizer for CNN

[0]:
```
batch_size = 32
valid_size = 0.3 # percentage of training set to use as validation

# Define a transform to normalize the data
tsfm_train = transforms.Compose([Rescale(205), RandomCrop(200), ToTensor()])
tsfm_test = transforms.Compose([ToTensor()])

# Load the training data and test data of FRONT view
trainset_front = FrontKeypointsDataset(IMG_F_TRAIN, KPT_F_TRAIN,␣
 ↪transform=tsfm_train)
testset_front = FrontKeypointsDataset(IMG_F_TEST, None, train=False,␣
 ↪transform=tsfm_test)

# prepare data loaders for front view
train_loader_front, valid_loader_front =␣
 ↪prepare_train_valid_loaders(trainset_front, valid_size, batch_size)
test_loader_front = torch.utils.data.DataLoader(testset_front,␣
 ↪batch_size=batch_size)
```

[0]:
```
# Cause there are 9 points in the front view. The output_size equal to 2 times␣
 ↪9.
output_size=18
model_front = CNN(output_size)
model_front = model.to(device)
```

```python
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.0005)
# optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.8)

train_losses, valid_losses = train(train_loader, valid_loader,
                                   model, criterion, optimizer,
                                   n_epochs=100,
                                   saved_model=ROOT_FOLDER+'model_front.pt')
```

```
Epoch: 1        Training Loss: 0.023119         Validation Loss: 0.016452
Validation loss decreased (inf --> 0.016452).  Saving model ...
Epoch: 2        Training Loss: 0.018246         Validation Loss: 0.015340
Validation loss decreased (0.016452 --> 0.015340).  Saving model ...
Epoch: 3        Training Loss: 0.017870         Validation Loss: 0.018035
Epoch: 4        Training Loss: 0.017744         Validation Loss: 0.015900
Epoch: 5        Training Loss: 0.017077         Validation Loss: 0.015712
Epoch: 6        Training Loss: 0.017071         Validation Loss: 0.014427
Validation loss decreased (0.015340 --> 0.014427).  Saving model ...
Epoch: 7        Training Loss: 0.018175         Validation Loss: 0.019984
Epoch: 8        Training Loss: 0.017554         Validation Loss: 0.015882
Epoch: 9        Training Loss: 0.017512         Validation Loss: 0.015024
Epoch: 10       Training Loss: 0.016794         Validation Loss: 0.016314
Epoch: 11       Training Loss: 0.016707         Validation Loss: 0.015510
Epoch: 12       Training Loss: 0.016827         Validation Loss: 0.017190
Epoch: 13       Training Loss: 0.016331         Validation Loss: 0.016315
Epoch: 14       Training Loss: 0.015667         Validation Loss: 0.016520
Epoch: 15       Training Loss: 0.015742         Validation Loss: 0.020218
Epoch: 16       Training Loss: 0.015887         Validation Loss: 0.016957
Epoch: 17       Training Loss: 0.015075         Validation Loss: 0.017836
Epoch: 18       Training Loss: 0.014465         Validation Loss: 0.017904
Epoch: 19       Training Loss: 0.014512         Validation Loss: 0.017517
Epoch: 20       Training Loss: 0.014684         Validation Loss: 0.016860
Epoch: 21       Training Loss: 0.014813         Validation Loss: 0.018632
Epoch: 22       Training Loss: 0.014224         Validation Loss: 0.017081
Epoch: 23       Training Loss: 0.013714         Validation Loss: 0.018436
Epoch: 24       Training Loss: 0.014019         Validation Loss: 0.018693
Epoch: 25       Training Loss: 0.014467         Validation Loss: 0.018817
Epoch: 26       Training Loss: 0.014122         Validation Loss: 0.017906
Epoch: 27       Training Loss: 0.013268         Validation Loss: 0.016555
Epoch: 28       Training Loss: 0.013324         Validation Loss: 0.016498
Epoch: 29       Training Loss: 0.013719         Validation Loss: 0.021386
Epoch: 30       Training Loss: 0.013666         Validation Loss: 0.018270
Epoch: 31       Training Loss: 0.012675         Validation Loss: 0.018171
Epoch: 32       Training Loss: 0.012783         Validation Loss: 0.017723
Epoch: 33       Training Loss: 0.012485         Validation Loss: 0.018061
Epoch: 34       Training Loss: 0.012419         Validation Loss: 0.017643
Epoch: 35       Training Loss: 0.012927         Validation Loss: 0.020225
```

```
Epoch: 36        Training Loss: 0.012529         Validation Loss: 0.017898
Epoch: 37        Training Loss: 0.012275         Validation Loss: 0.018561
Epoch: 38        Training Loss: 0.012633         Validation Loss: 0.018106
Epoch: 39        Training Loss: 0.012058         Validation Loss: 0.017677
Epoch: 40        Training Loss: 0.012544         Validation Loss: 0.016737
Epoch: 41        Training Loss: 0.012377         Validation Loss: 0.018440
Epoch: 42        Training Loss: 0.013659         Validation Loss: 0.018620
Epoch: 43        Training Loss: 0.013642         Validation Loss: 0.017465
Epoch: 44        Training Loss: 0.012283         Validation Loss: 0.018056
Epoch: 45        Training Loss: 0.012373         Validation Loss: 0.019222
Epoch: 46        Training Loss: 0.012485         Validation Loss: 0.016636
Epoch: 47        Training Loss: 0.011531         Validation Loss: 0.017768
Epoch: 48        Training Loss: 0.012666         Validation Loss: 0.017883
Epoch: 49        Training Loss: 0.012555         Validation Loss: 0.017792
Epoch: 50        Training Loss: 0.012618         Validation Loss: 0.019276
Epoch: 51        Training Loss: 0.012188         Validation Loss: 0.018606
Epoch: 52        Training Loss: 0.012073         Validation Loss: 0.017685
Epoch: 53        Training Loss: 0.011618         Validation Loss: 0.018737
Epoch: 54        Training Loss: 0.011935         Validation Loss: 0.018660
Epoch: 55        Training Loss: 0.012178         Validation Loss: 0.018013
Epoch: 56        Training Loss: 0.011954         Validation Loss: 0.019369
Epoch: 57        Training Loss: 0.011393         Validation Loss: 0.017753
Epoch: 58        Training Loss: 0.011263         Validation Loss: 0.018224
Epoch: 59        Training Loss: 0.012430         Validation Loss: 0.018116
Epoch: 60        Training Loss: 0.011520         Validation Loss: 0.017410
Epoch: 61        Training Loss: 0.012253         Validation Loss: 0.018830
Epoch: 62        Training Loss: 0.012442         Validation Loss: 0.017672
Epoch: 63        Training Loss: 0.013319         Validation Loss: 0.019171
Epoch: 64        Training Loss: 0.012352         Validation Loss: 0.017542
Epoch: 65        Training Loss: 0.011328         Validation Loss: 0.017910
Epoch: 66        Training Loss: 0.010818         Validation Loss: 0.017726
Epoch: 67        Training Loss: 0.010556         Validation Loss: 0.018570
Epoch: 68        Training Loss: 0.010998         Validation Loss: 0.017359
Epoch: 69        Training Loss: 0.010789         Validation Loss: 0.018945
Epoch: 70        Training Loss: 0.010499         Validation Loss: 0.017689
Epoch: 71        Training Loss: 0.010790         Validation Loss: 0.018593
Epoch: 72        Training Loss: 0.010369         Validation Loss: 0.018038
Epoch: 73        Training Loss: 0.010285         Validation Loss: 0.018137
Epoch: 74        Training Loss: 0.010138         Validation Loss: 0.017727
Epoch: 75        Training Loss: 0.010642         Validation Loss: 0.018777
Epoch: 76        Training Loss: 0.010367         Validation Loss: 0.018716
Epoch: 77        Training Loss: 0.010555         Validation Loss: 0.017716
Epoch: 78        Training Loss: 0.010336         Validation Loss: 0.018462
Epoch: 79        Training Loss: 0.010857         Validation Loss: 0.018728
Epoch: 80        Training Loss: 0.011497         Validation Loss: 0.017964
Epoch: 81        Training Loss: 0.010844         Validation Loss: 0.018015
Epoch: 82        Training Loss: 0.010191         Validation Loss: 0.018876
Epoch: 83        Training Loss: 0.009854         Validation Loss: 0.018652
```

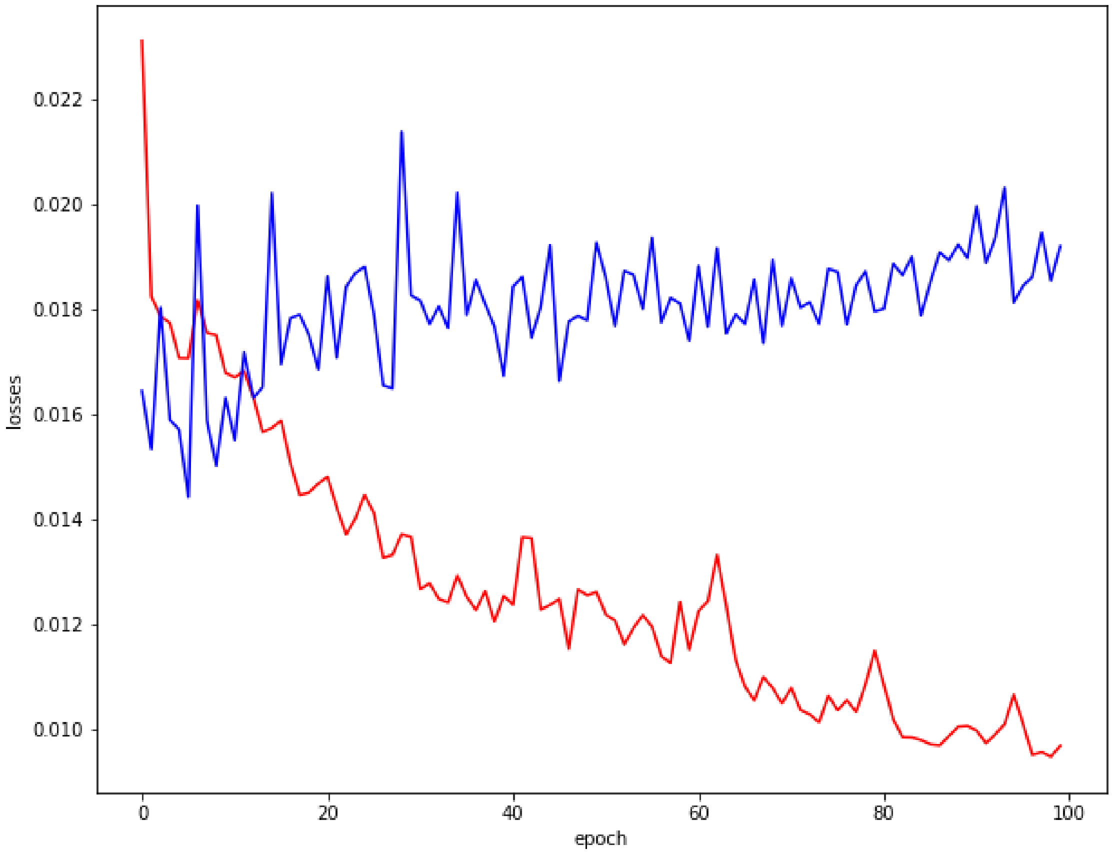
```
Epoch: 84       Training Loss: 0.009848        Validation Loss: 0.019007
Epoch: 85       Training Loss: 0.009798        Validation Loss: 0.017887
Epoch: 86       Training Loss: 0.009717        Validation Loss: 0.018521
Epoch: 87       Training Loss: 0.009694        Validation Loss: 0.019090
Epoch: 88       Training Loss: 0.009873        Validation Loss: 0.018938
Epoch: 89       Training Loss: 0.010051        Validation Loss: 0.019239
Epoch: 90       Training Loss: 0.010066        Validation Loss: 0.018980
Epoch: 91       Training Loss: 0.009975        Validation Loss: 0.019968
Epoch: 92       Training Loss: 0.009735        Validation Loss: 0.018887
Epoch: 93       Training Loss: 0.009907        Validation Loss: 0.019359
Epoch: 94       Training Loss: 0.010098        Validation Loss: 0.020317
Epoch: 95       Training Loss: 0.010661        Validation Loss: 0.018135
Epoch: 96       Training Loss: 0.010099        Validation Loss: 0.018455
Epoch: 97       Training Loss: 0.009515        Validation Loss: 0.018619
Epoch: 98       Training Loss: 0.009568        Validation Loss: 0.019466
Epoch: 99       Training Loss: 0.009483        Validation Loss: 0.018552
Epoch: 100      Training Loss: 0.009688        Validation Loss: 0.019205
```

Evaluation and structure for CNN

[0]:
```
# Draw the changing curve
n_epochs=100
x=range(0,n_epochs)
plt.figure(figsize=(10,8))
y1=train_losses
y2=valid_losses
print(np.shape(x))
print(np.shape(y1))
print(np.shape(y2))
plt.plot(x,y1,color="red")
plt.plot(x,y2,color="blue")
plt.xlabel('epoch')
plt.ylabel('losses')
```

```
(100,)
(100,)
(100,)
```

[0]:
```
Text(0, 0.5, 'losses')
```



[0]:
```
# Evaluate this one
model_front.load_state_dict(torch.load(ROOT_FOLDER+'model_front.pt'))
predictions_front = predict(test_loader_front, model_front)
print(test_loader)
# Not provide the valid keypoints of front_dataset for training.
# Here just show the structure of CNN.
print(model_front)
print(output_size)
```

```
<torch.utils.data.dataloader.DataLoader object at 0x7f2c62138f60>
CNN(
  (conv1): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (pool1): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1,
ceil_mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv4): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2))
```

```
  (conv5): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1))
  (fc1): Linear(in_features=12800, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=12, bias=True)
  (drop1): Dropout(p=0.1, inplace=False)
  (drop2): Dropout(p=0.25, inplace=False)
  (drop3): Dropout(p=0.25, inplace=False)
  (drop4): Dropout(p=0.25, inplace=False)
  (drop5): Dropout(p=0.35, inplace=False)
  (drop6): Dropout(p=0.4, inplace=False)
)
18
```

[0]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
pandoc is already the newest version (1.19.2.4~dfsg-1build4).
texlive is already the newest version (2017.20180305-1).
texlive-latex-extra is already the newest version (2017.20180305-2).
texlive-xetex is already the newest version (2017.20180305-1).
0 upgraded, 0 newly installed, 0 to remove and 25 not upgraded.
Requirement already satisfied: pypandoc in /usr/local/lib/python3.6/dist-
packages (1.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-
packages (from pypandoc) (46.0.0)
Requirement already satisfied: wheel>=0.25.0 in /usr/local/lib/python3.6/dist-
packages (from pypandoc) (0.34.2)
Requirement already satisfied: pip>=8.1.0 in /usr/local/lib/python3.6/dist-
packages (from pypandoc) (19.3.1)
```

[0]:
```
# Export the notebook as pdf
!jupyter nbconvert --to PDF "./gdrive/My Drive/Colab Notebooks/
↪MAEG5735-2020-Assignment2/KeypointDetection_1155135359.ipynb"
```

```
[NbConvertApp] WARNING | pattern u'./gdrive/My Drive/Colab
Notebooks/MAEG5735-2020-Assignment2/KeypointDetection_1155135359.ipynb' matched
no files
This application is used to convert notebook files (*.ipynb) to various other
formats.

WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
-------
```

```
Arguments that take values are actually convenience aliases to full
Configurables, whose aliases are listed on the help line. For more information
on full configurables, see '--help-all'.

--execute
    Execute the notebook prior to export.
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
--no-input
    Exclude input cells and output prompts from converted document.
    This mode is ideal for generating code-free reports.
--stdout
    Write notebook output to stdout instead of files.
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
    relevant when converting to notebook format)
-y
    Answer yes to any questions instead of prompting.
--clear-output
    Clear output of current file and save in place,
    overwriting the existing notebook.
--debug
    set log level to logging.DEBUG (maximize logging output)
--no-prompt
    Exclude input and output prompts from converted document.
--generate-config
    generate default config file
--nbformat=<Enum> (NotebookExporter.nbformat_version)
    Default: 4
    Choices: [1, 2, 3, 4]
    The nbformat version to write. Use this to downgrade notebooks.
--output-dir=<Unicode> (FilesWriter.build_directory)
    Default: ''
    Directory to write output(s) to. Defaults to output to the directory of each
    notebook. To recover previous default behaviour (outputting to the current
    working directory) use . as the flag value.
--writer=<DottedObjectName> (NbConvertApp.writer_class)
    Default: 'FilesWriter'
    Writer class used to write the  results of the conversion
--log-level=<Enum> (Application.log_level)
    Default: 30
    Choices: (0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL')
```

```
    Set the log level by value or name.
--reveal-prefix=<Unicode> (SlidesExporter.reveal_url_prefix)
    Default: u''
    The URL prefix for reveal.js (version 3.x). This defaults to the reveal CDN,
    but can be any url pointing to a copy  of reveal.js.
    For speaker notes to work, this must be a relative path to a local  copy of
    reveal.js: e.g., "reveal.js".
    If a relative path is given, it must be a subdirectory of the current
    directory (from which the server is run).
    See the usage documentation
    (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-
    slideshow) for more details.
--to=<Unicode> (NbConvertApp.export_format)
    Default: 'html'
    The export format to be used, either one of the built-in formats
    ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf',
    'python', 'rst', 'script', 'slides'] or a dotted object name that represents
    the import path for an `Exporter` class
--template=<Unicode> (TemplateExporter.template_file)
    Default: u''
    Name of the template file to use
--output=<Unicode> (NbConvertApp.output_base)
    Default: ''
    overwrite base name use for output files. can only be used when converting
    one notebook at a time.
--post=<DottedOrNone> (NbConvertApp.postprocessor_class)
    Default: u''
    PostProcessor class used to write the results of the conversion
--config=<Unicode> (JupyterApp.config_file)
    Default: u''
    Full path of a config file.

To see all available configurables, use `--help-all`

Examples
--------

    The simplest way to use nbconvert is

    > jupyter nbconvert mynotebook.ipynb

    which will convert mynotebook.ipynb to the default format (probably HTML).

    You can specify the export format with `--to`.
    Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'rst', 'script', 'slides'].

    > jupyter nbconvert --to latex mynotebook.ipynb
```

```
Both HTML and LaTeX support multiple output templates. LaTeX includes
'base', 'article' and 'report'.  HTML includes 'basic' and 'full'. You
can specify the flavor of the format used.

> jupyter nbconvert --to html --template basic mynotebook.ipynb

You can also pipe the output to stdout, rather than a file

> jupyter nbconvert mynotebook.ipynb --stdout

PDF is generated via latex

> jupyter nbconvert mynotebook.ipynb --to pdf

You can get (and serve) a Reveal.js-powered slideshow

> jupyter nbconvert myslides.ipynb --to slides --post serve

Multiple notebooks can be given at the command line in a couple of
different ways:

> jupyter nbconvert notebook*.ipynb
> jupyter nbconvert notebook1.ipynb notebook2.ipynb

or you can specify the notebooks list in a config file, containing::

    c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

> jupyter nbconvert --config mycfg.py
```

1.3 Conclusion

During this assignment, I have learned a lot about the deep learning basic concepts including CNN structure, MLP structure, the measurement method about the conv2d and accumulated magical experience on adjusting parameters.

Besides, I have touched lots of the useful software about jupyter, colab, xshell and tested the codes in the command bar. Also now I have the experience to adjust the CUDA, CUDNN, tensorflow and torch by using conda command.

To sum up, TA Mr. Liu Zishun has drafted all the codes for us so the assignment could be solved smoothly. But it does not mean less chance to learn more. That varies on the student himself.

1.4 Reference

- Facial Keypoints Detection with PyTorch
- ImageNet Classification with Deep ConvolutionalNeural Networks

- LeNet-5 – A Classic CNN Architecture
- Gradient-Based Learning Applied to Document Recognition