Ravina G., Sidharth P., Narendra P., Devon F, Bonaventure R

Computer Vision Semester Project
Modeling Stage

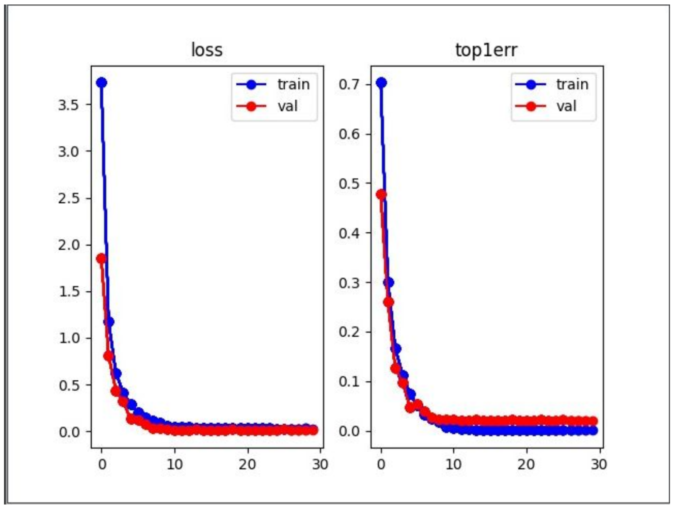
Overview:

We had decided earlier that we were going to follow a recently conducted research experiment on re-identification of individuals (https://arxiv.org/abs/1904.07223). This paper models re-identification of individuals after training an CNN to find important features. We decided to follow through with a pre-existing model that worked by building on the classic ResNet50 model which acted as a ConvNet with 50 layers and 1000 classes. However, since this model was trained on Market 1501's dataset from Tsinghua, this only has 751 class labels (see https://paperswithcode.com/sota/person-re-identification-on-market-1501). Thus, this model where the bottleneck is 751 rather than 1000. It is important to know that the class labels do not matter- i.e this could be applied to Duke MDMC's dataset and perform only slightly worse due to occlusions and difference in lightings. The model we chose only has 751 labels to increase rank accuracy and mAP.

Code Explanation:

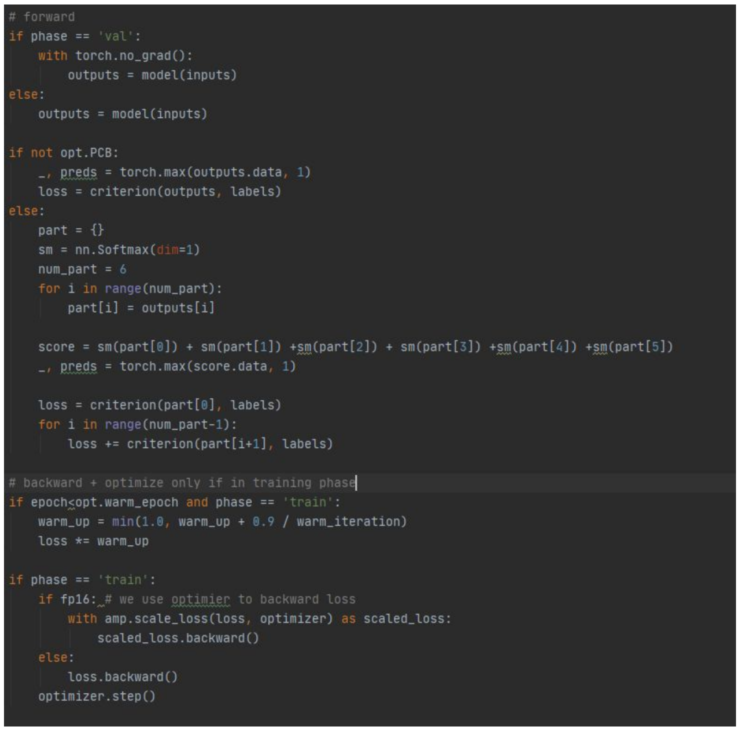
Training:

Our code follows heavily https://github.com/layumi/Person_reID_baseline_pytorch which covers the basis of training on Market 1501. Unfortunately for us, we could not download the pretrained model directly. Therefore, we had to train the models directly. Below are the visualization of training.



We only ran it on roughly 30 epochs to increase to an accuracy of 99.7% on a train/validation set (also provided via the market dataset). We can see from the image above, the loss function dropped to a low of 0.04.

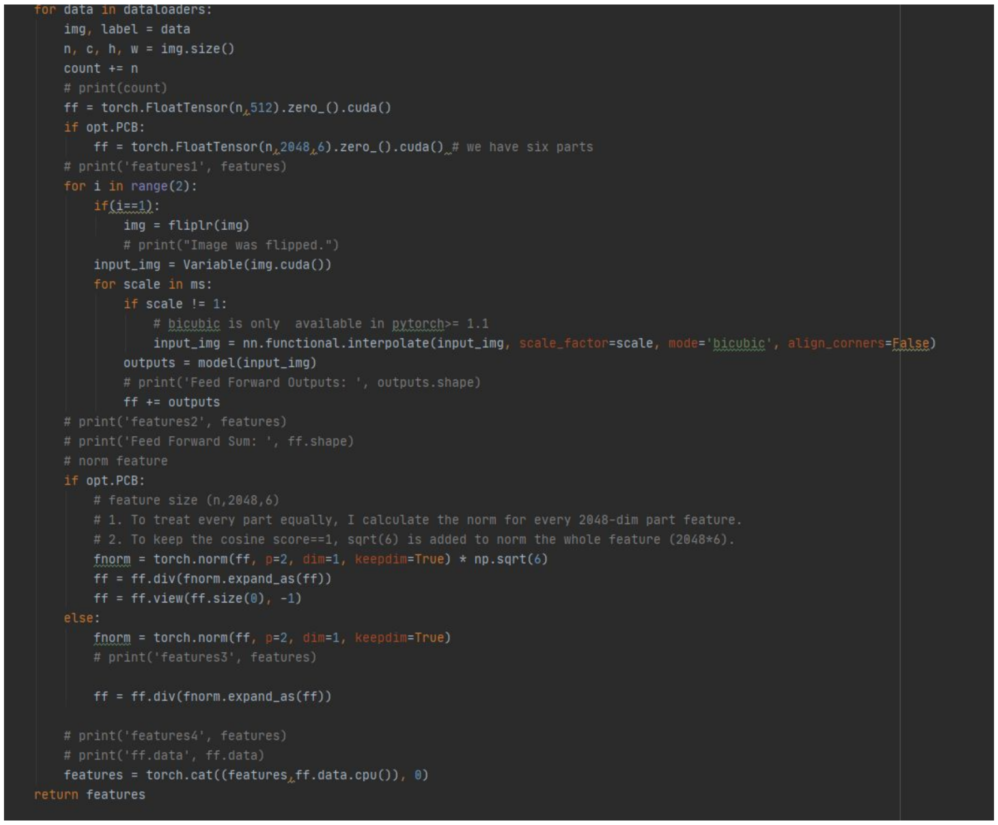
The bulk of our training comes from this portion specifically. This code is directly supplied from the github. Essentially, for training you want to use forward and backward propagation to train the neural network (as standard protocol). However, for the validation set, we only take the vector outputted via the forward propagation.



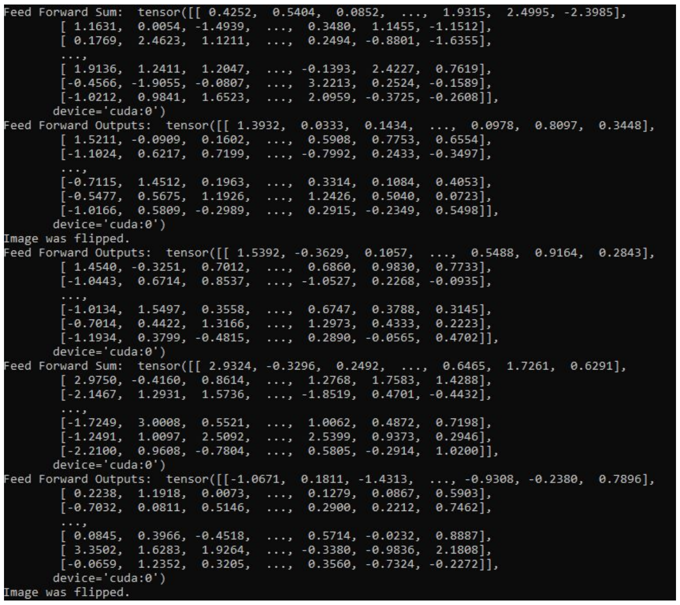
```
# forward
if phase == 'val':
    with torch.no_grad():
        outputs = model(inputs)
else:
    outputs = model(inputs)

if not opt.PCB:
    _, preds = torch.max(outputs.data, 1)
    loss = criterion(outputs, labels)
else:
    part = {}
    sm = nn.Softmax(dim=1)
    num_part = 6
    for i in range(num_part):
        part[i] = outputs[i]

    score = sm(part[0]) + sm(part[1]) +sm(part[2]) + sm(part[3]) +sm(part[4]) +sm(part[5])
    _, preds = torch.max(score.data, 1)

    loss = criterion(part[0], labels)
    for i in range(num_part-1):
        loss += criterion(part[i+1], labels)

# backward + optimize only if in training phase
if epoch<opt.warm_epoch and phase == 'train':
    warm_up = min(1.0, warm_up + 0.9 / warm_iteration)
    loss *= warm_up

if phase == 'train':
    if fp16: # we use optimier to backward loss
        with amp.scale_loss(loss, optimizer) as scaled_loss:
            scaled_loss.backward()
    else:
        loss.backward()
    optimizer.step()
```

Testing:

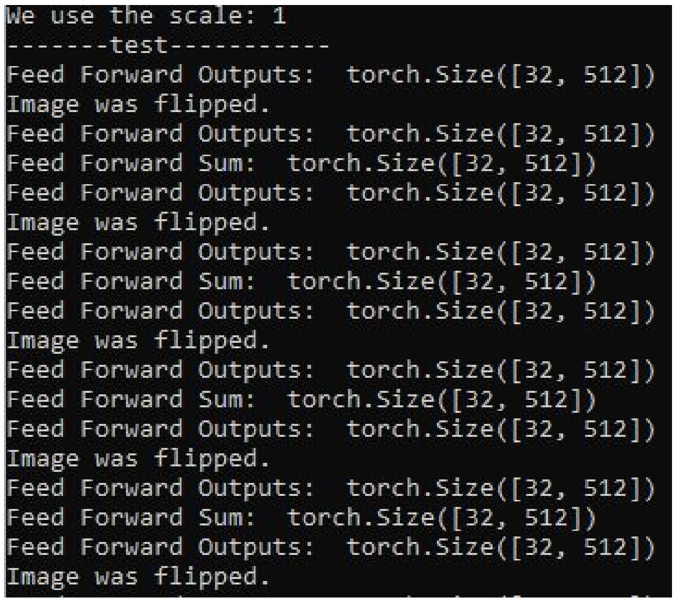
The testing was done in a peculiar way. Each image in the test set was flipped horizontally (indicated in the image below at fliplr(img)). The model adds the forward outputs of the image passed in + the diagonally flipped image vectors from the forward pass. These two vector sums make up the output from a test image. This test image, then has the L2-Norm (easier to take the euclidean/cosine distance) and has that passed as the important feature vector.



```python
for data in dataloaders:
    img, label = data
    n, c, h, w = img.size()
    count += n
    # print(count)
    ff = torch.FloatTensor(n,512).zero_().cuda()
    if opt.PCB:
        ff = torch.FloatTensor(n,2048,6).zero_().cuda() # we have six parts
    # print('features1', features)
    for i in range(2):
        if(i==1):
            img = fliplr(img)
            # print("Image was flipped.")
        input_img = Variable(img.cuda())
        for scale in ms:
            if scale != 1:
                # bicubic is only  available in pytorch>= 1.1
                input_img = nn.functional.interpolate(input_img, scale_factor=scale, mode='bicubic', align_corners=False)
            outputs = model(input_img)
            # print('Feed Forward Outputs: ', outputs.shape)
            ff += outputs
    # print('features2', features)
    # print('Feed Forward Sum: ', ff.shape)
    # norm feature
    if opt.PCB:
        # feature size (n,2048,6)
        # 1. To treat every part equally, I calculate the norm for every 2048-dim part feature.
        # 2. To keep the cosine score==1, sqrt(6) is added to norm the whole feature (2048*6).
        fnorm = torch.norm(ff, p=2, dim=1, keepdim=True) * np.sqrt(6)
        ff = ff.div(fnorm.expand_as(ff))
        ff = ff.view(ff.size(0), -1)
    else:
        fnorm = torch.norm(ff, p=2, dim=1, keepdim=True)
        # print('features3', features)

        ff = ff.div(fnorm.expand_as(ff))

    # print('features4', features)
    # print('ff.data', ff.data)
    features = torch.cat((features,ff.data.cpu()), 0)
return features
```

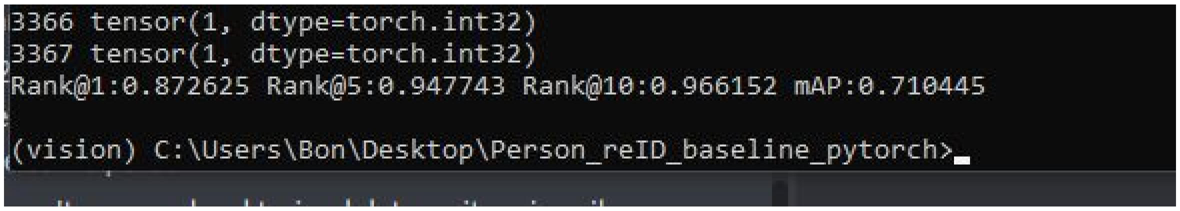


```
Feed Forward Sum:  tensor([[ 0.4252,  0.5404,  0.0852,  ...,  1.9315,  2.4995, -2.3985],
        [ 1.1631,  0.0054, -1.4939,  ...,  0.3480,  1.1455, -1.1512],
        [ 0.1769,  2.4623,  1.1211,  ...,  0.2494, -0.8801, -1.6355],
        ...,
        [ 1.9136,  1.2411,  1.2047,  ..., -0.1393,  2.4227,  0.7619],
        [-0.4566, -1.9055, -0.0807,  ...,  3.2213,  0.2524, -0.1589],
        [-1.0212,  0.9841,  1.6523,  ...,  2.0959, -0.3725, -0.2608]],
       device='cuda:0')
Feed Forward Outputs:  tensor([[ 1.3932,  0.0333,  0.1434,  ...,  0.0978,  0.8097,  0.3448],
        [ 1.5211, -0.0909,  0.1602,  ...,  0.5908,  0.7753,  0.6554],
        [-1.1024,  0.6217,  0.7199,  ..., -0.7992,  0.2433, -0.3497],
        ...,
        [-0.7115,  1.4512,  0.1963,  ...,  0.3314,  0.1084,  0.4053],
        [-0.5477,  0.5675,  1.1926,  ...,  1.2426,  0.5040,  0.0723],
        [-1.0166,  0.5809, -0.2989,  ...,  0.2915, -0.2349,  0.5498]],
       device='cuda:0')
Image was flipped.
Feed Forward Outputs:  tensor([[ 1.5392, -0.3629,  0.1057,  ...,  0.5488,  0.9164,  0.2843],
        [ 1.4540, -0.3251,  0.7012,  ...,  0.6860,  0.9830,  0.7733],
        [-1.0443,  0.6714,  0.8537,  ..., -1.0527,  0.2268, -0.0935],
        ...,
        [-1.0134,  1.5497,  0.3558,  ...,  0.6747,  0.3788,  0.3145],
        [-0.7014,  0.4422,  1.3166,  ...,  1.2973,  0.4333,  0.2223],
        [-1.1934,  0.3799, -0.4815,  ...,  0.2890, -0.0565,  0.4702]],
       device='cuda:0')
Feed Forward Sum:  tensor([[ 2.9324, -0.3296,  0.2492,  ...,  0.6465,  1.7261,  0.6291],
        [ 2.9750, -0.4160,  0.8614,  ...,  1.2768,  1.7583,  1.4288],
        [-2.1467,  1.2931,  1.5736,  ..., -1.8519,  0.4701, -0.4432],
        ...,
        [-1.7249,  3.0008,  0.5521,  ...,  1.0062,  0.4872,  0.7198],
        [-1.2491,  1.0097,  2.5092,  ...,  2.5399,  0.9373,  0.2946],
        [-2.2100,  0.9608, -0.7804,  ...,  0.5805, -0.2914,  1.0200]],
       device='cuda:0')
Feed Forward Outputs:  tensor([[-1.0671,  0.1811, -1.4313,  ..., -0.9308, -0.2380,  0.7896],
        [ 0.2238,  1.1918,  0.0073,  ...,  0.1279,  0.0867,  0.5903],
        [-0.7032,  0.0811,  0.5146,  ...,  0.2900,  0.2212,  0.7462],
        ...,
        [ 0.0845,  0.3966, -0.4518,  ...,  0.5714, -0.0232,  0.8887],
        [ 3.3502,  1.6283,  1.9264,  ..., -0.3380, -0.9836,  2.1808],
        [-0.0659,  1.2352,  0.3205,  ...,  0.3560, -0.7324, -0.2272]],
       device='cuda:0')
Image was flipped.
```



```
We use the scale: 1
-------test-----------
Feed Forward Outputs:  torch.Size([32, 512])
Image was flipped.
Feed Forward Outputs:  torch.Size([32, 512])
Feed Forward Sum:  torch.Size([32, 512])
Feed Forward Outputs:  torch.Size([32, 512])
Image was flipped.
Feed Forward Outputs:  torch.Size([32, 512])
Feed Forward Sum:  torch.Size([32, 512])
Feed Forward Outputs:  torch.Size([32, 512])
Image was flipped.
Feed Forward Outputs:  torch.Size([32, 512])
Feed Forward Sum:  torch.Size([32, 512])
Feed Forward Outputs:  torch.Size([32, 512])
Image was flipped.
Feed Forward Outputs:  torch.Size([32, 512])
Feed Forward Sum:  torch.Size([32, 512])
Feed Forward Outputs:  torch.Size([32, 512])
Image was flipped.
```

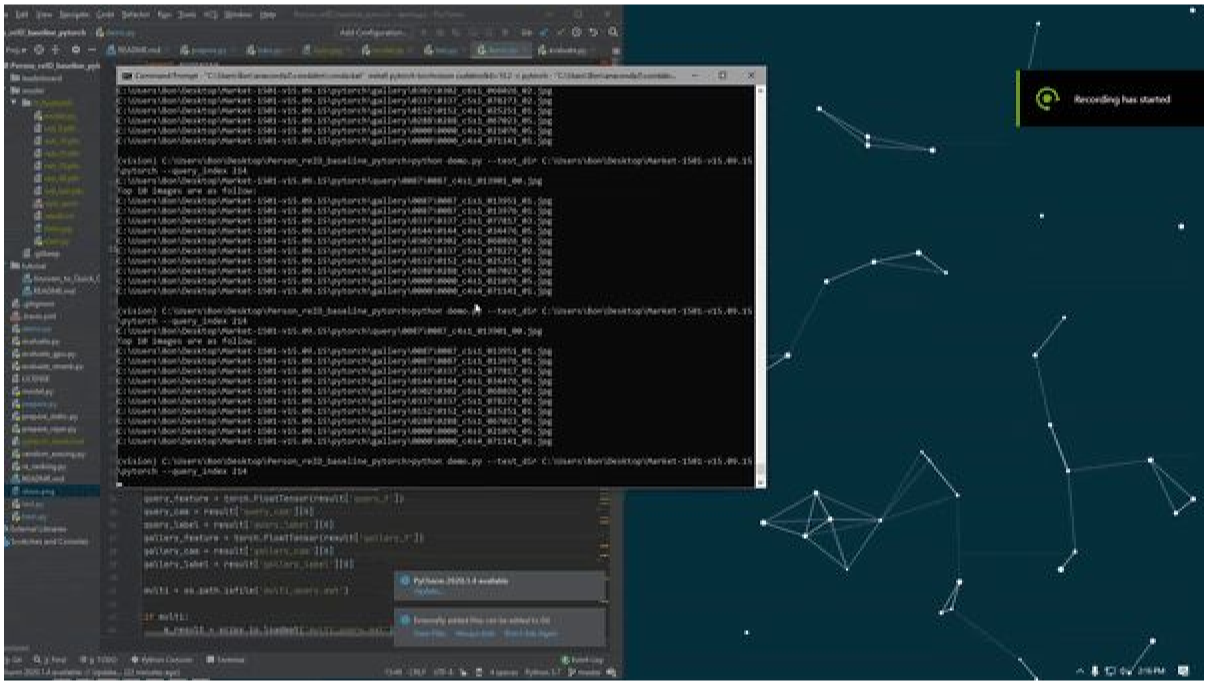
Explanations above



```
3366 tensor(1, dtype=torch.int32)
3367 tensor(1, dtype=torch.int32)
Rank@1:0.872625 Rank@5:0.947743 Rank@10:0.966152 mAP:0.710445

(vision) C:\Users\Bon\Desktop\Person_reID_baseline_pytorch>
```

The mAP on the specific dataset above, has a pretty high accuracy. However, we need to extend this testing to the Duke's Dataset and our own dataset. This is our next step.

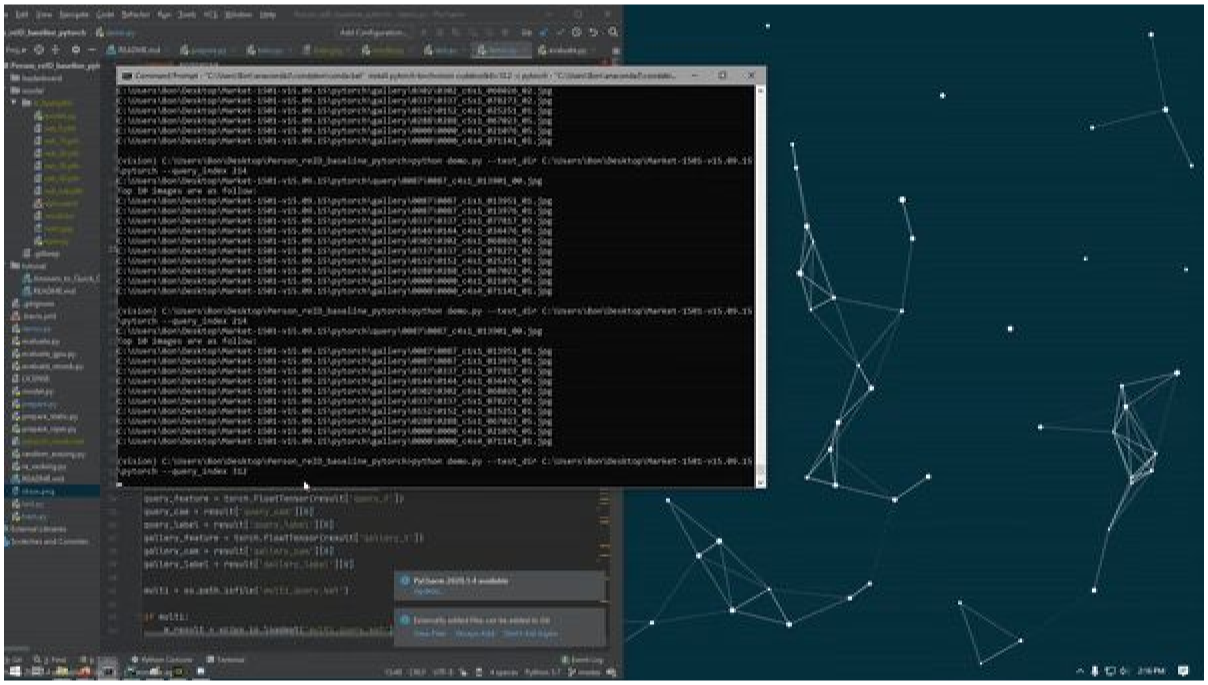
Below are a few demo's of the query image, and then a set of images that have a close cosine distance in the query image, and the subsequent images displayed.



```
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0302\0302_c6s1_068026_02.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0137\0137_c5s1_078273_02.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0152\0152_c4s1_025251_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0288\0288_c5s1_067023_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0000\0000_c4s1_021076_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0000\0000_c4s4_071141_05.jpg

(vision) C:\Users\Bon\Desktop\Person_reID_baseline_pytorch>python demo.py --test_dir C:\Users\Bon\Desktop\Market-1501-v15.09.15
\pytorch --query_index 214
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\query\0007\0007_c4s1_013901_00.jpg
Top 10 images are as follow:
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0007\0007_c1s1_013951_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0007\0007_c1s1_013976_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0137\0137_c3s1_077857_03.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0144\0144_c4s1_034476_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0302\0302_c6s1_068026_02.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0137\0137_c5s1_078273_02.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0152\0152_c4s1_025251_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0288\0288_c5s1_067023_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0000\0000_c4s1_021076_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0000\0000_c4s4_071141_05.jpg

(vision) C:\Users\Bon\Desktop\Person_reID_baseline_pytorch>python demo.py --test_dir C:\Users\Bon\Desktop\Market-1501-v15.09.15
\pytorch --query_index 214
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\query\0007\0007_c4s1_013901_00.jpg
Top 10 images are as follow:
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0007\0007_c1s1_013951_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0007\0007_c1s1_013976_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0137\0137_c3s1_077857_03.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0144\0144_c4s1_034476_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0302\0302_c6s1_068026_02.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0137\0137_c5s1_078273_02.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0152\0152_c4s1_025251_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0288\0288_c5s1_067023_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0000\0000_c4s1_021076_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0000\0000_c4s4_071141_05.jpg

(vision) C:\Users\Bon\Desktop\Person_reID_baseline_pytorch>python demo.py --test_dir C:\Users\Bon\Desktop\Market-1501-v15.09.15
\pytorch --query_index 214
```

```
query_feature = torch.FloatTensor(result['query_f'])
query_cam = result['query_cam'][0]
query_label = result['query_label'][0]
gallery_feature = torch.FloatTensor(result['gallery_f'])
gallery_cam = result['gallery_cam'][0]
gallery_label = result['gallery_label'][0]

multi = os.path.isfile('multi_query.mat')

if multi:
```

Ravina G., Sidharth P., Narendra P., Devon F, Bonaventure R



```
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0302\0302_c5s1_068026_02.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0337\0337_c5s1_078273_02.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0152\0152_c4s1_025251_01.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0288\0288_c3s1_067023_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0000\0000_c4s1_021076_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0000\0000_c4s4_071141_01.jpg

(vision) C:\Users\Bon\Desktop\Person_reID_baseline_pytorch>python demo.py --test_dir C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch --query_index 214
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\query\0087\0087_c4s1_013001_00.jpg
Top 10 images are as follow:
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0087\0087_c1s1_013951_01.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0087\0087_c1s1_013976_01.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0337\0337_c1s1_077817_03.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0144\0144_c4s1_034476_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0302\0302_c6s1_068026_02.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0337\0337_c5s1_078273_02.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0152\0152_c4s1_025251_01.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0288\0288_c3s1_067023_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0000\0000_c4s1_021076_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0000\0000_c4s4_071141_01.jpg

(vision) C:\Users\Bon\Desktop\Person_reID_baseline_pytorch>python demo.py --test_dir C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch --query_index 214
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\query\0087\0087_c4s1_013001_00.jpg
Top 10 images are as follow:
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0087\0087_c1s1_013951_01.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0087\0087_c1s1_013976_01.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0337\0337_c1s1_077817_03.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0144\0144_c4s1_034476_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0302\0302_c6s1_068026_02.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0337\0337_c5s1_078273_02.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0152\0152_c4s1_025251_01.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0288\0288_c3s1_067023_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0000\0000_c4s1_021076_05.jpg
C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch\gallery\0000\0000_c4s4_071141_01.jpg

(vision) C:\Users\Bon\Desktop\Person_reID_baseline_pytorch>python demo.py --test_dir C:\Users\Bon\Desktop\Market-1501-v15.09.15\pytorch --query_index 312
```

```python
query_feature = torch.FloatTensor(result['query_f'])
query_cam = result['query_cam'][0]
query_label = result['query_label'][0]
gallery_feature = torch.FloatTensor(result['gallery_f'])
gallery_cam = result['gallery_cam'][0]
gallery_label = result['gallery_label'][0]

multi = os.path.isfile('multi_query.mat')

if multi:
```

Next Steps:

Our next step is to implement this with our dataset. We expect it to have a lower accuracy than the one above because of the difference in lighting and angle placements of the cameras. However, this is something we are working on improving via tweaks to the model.

** Important **

Please refer to the github README for any setup information

We had to configure this to work with windows, so changes were made to the overall project to adapt to run on windows. However, for any ubuntu the code should be identical.