```
from __future__ import print_function
import argparse
import os
import shutil
import time
import glob
import os, re, os.path

import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.optim
import torch.utils.data
import torch.nn.functional as F
import torchvision.transforms as transforms
import torchvision.datasets as datasets

import numpy as np
from numpy import dot
from numpy.linalg import norm
import cv2

from light_cnn import LightCNN_9Layers, LightCNN_29Layers, LightCNN_29Layers_v2
from load_imglist import ImageList
#from sklearn.metrics.pairwise import cosine_similarity


parser = argparse.ArgumentParser(description='PyTorch ImageNet Feature Extracting')
parser.add_argument('--arch', '-a', metavar='ARCH', default='LightCNN')
parser.add_argument('--cuda', '-c', default=True)
parser.add_argument('--resume', default='', type=str, metavar='PATH',
                    help='path to latest checkpoint (default: none)')
parser.add_argument('--model', default='', type=str, metavar='Model',
                    help='model type: LightCNN-9, LightCNN-29')
parser.add_argument('--root_path', default='', type=str, metavar='PATH',
                    help='root path of face images (default: none).')
parser.add_argument('--img_list', default='', type=str, metavar='PATH',
                    help='list of face images for feature extraction (default:
none).')
parser.add_argument('--save_path', default='', type=str, metavar='PATH',
                    help='save root path for features of face images.')
parser.add_argument('--num_classes', default=79077, type=int,
                    metavar='N', help='mini-batch size (default: 79077)')

def main():

    mypath = "test_feat"
    for root, dirs, files in os.walk(mypath):
        for file in files:
            os.remove(os.path.join(root, file))

    global args
    args = parser.parse_args()

    if args.model == 'LightCNN-9':
        model = LightCNN_9Layers(num_classes=args.num_classes)
    elif args.model == 'LightCNN-29':
        model = LightCNN_29Layers(num_classes=args.num_classes)
```

```
    elif args.model == 'LightCNN-29v2':
        model = LightCNN_29Layers_v2(num_classes=args.num_classes)
    else:
        print('Error model type\n')

    model.eval()
    if args.cuda:
        model = torch.nn.DataParallel(model).cuda()

    if args.resume:
        if os.path.isfile(args.resume):
            print("=> loading checkpoint '{}'".format(args.resume))
            checkpoint = torch.load(args.resume)
            model.load_state_dict(checkpoint['state_dict'])
    else:
        print("=> no checkpoint found at '{}'".format(args.resume))


    script(args.root_path)

    img_list  = read_list(args.img_list)
    #print(args.img_list)
    #print("_____")
    #print(img_list)
    transform = transforms.Compose([transforms.ToTensor()])
    count     = 0
    input     = torch.zeros(1, 1, 128, 128)
    for img_name in img_list:
        #print(img_name)
        count = count + 1
        img   = cv2.imread(os.path.join(args.root_path, img_name),
cv2.IMREAD_GRAYSCALE)
        #img   = np.reshape(img, (128, 128, 1))
        img = cv2.resize(img,(128,128))
        img   = transform(img)
        input[0,:,:,:] = img

        start = time.time()
        if args.cuda:
            input = input.cuda()
        input_var   = torch.autograd.Variable(input, volatile=True)
        _, features = model(input_var)
        end         = time.time() - start
        print("{}({}/{}). Time: {}".format(os.path.join(args.root_path, img_name),
count, len(img_list), end))




        save_feature(args.save_path, img_name, features.data.cpu().numpy()[0])
        cos_sim_cal(img_name)

def script(root_path):
      x=os.listdir(root_path)
      f=open('test.txt','w+')
      for i in x:
         print(i)
         f.write(str(i)+"\n")
      f.close()
```

```
def read_list(list_path):
    img_list = []
    with open(list_path, 'r') as f:
        for line in f.readlines()[0:]:
            img_path = line.strip().split()
            if img_path:
               img_list.append(img_path[0])
    print('There are {} images..'.format(len(img_list)))
    return img_list

def save_feature(save_path, img_name, features):

    img_path = os.path.join(save_path, img_name)
    img_dir  = os.path.dirname(img_path) + '/';
    if not os.path.exists(img_dir):
        os.makedirs(img_dir)
    fname = os.path.splitext(img_path)[0]
    fname = fname + '.feat'
    fid   = open(fname, 'wb')
    fid.write(features)
    fid.close()

def cos_sim_cal(img_name):

    #for root,directory,files in os.walk('test_feat/'):

        #for each_file in files:
    a=[]
    reverse=[]
    dict ={}
    img=img_name.replace('.png','.feat')
    #img=img_name.replace('.jpeg','.feat')
    #print(img)
    path="test_feat/"+img
    #path=os.path.join(root,each_file)
    #print(root)
    feat1=list(np.fromfile(path,dtype='<f',count=-1))
    for root_2,directory,files_2 in os.walk('saved_2/'):
        i=1

        for each_file_2 in files_2:
            path_2=os.path.join(root_2,each_file_2)
            feat2=list(np.fromfile(path_2,dtype='<f',count=-1))

            #cossim=str(cosine_similarity([feat1], [feat2]))[1:-1][1:-1]
            product=dot(feat1,feat2)/(norm(feat1)*norm(feat2))
            cossim= 1-np.arccos(min(1,product))/np.pi
            print(i)
            i=i+1
            #print(str(cossim)[1:-1][1:-1]  )
            a.append(cossim)
            dict[cossim]=each_file_2.replace('.feat','.png')

        a.sort()
        reverse=a[::-1]
        #print(reverse)
        print("Top 3 similar images are :")
```

```
        for i in range(0,3):
            if reverse[i]:
               print(reverse[i])
               print(dict[reverse[i]])

    #with open('output'+str(img_name)+'.html','w') as fh:
    with open('output.html','w') as fh:
            message = """<!DOCTYPE html>
<html>

<head>
<style>
h1 {text-align: center;}

.img-container{
text-align:center;
}

</style>
<title>Displaying top 10 images </title>
</head>

<body>

<h1> Top 3 images for the given image </h2>
<div class ="img-container"> <img id="input" src="test/
"""

            fh.write(message)
            fh.write(img_name)
            input_tag='" border="5" style="width:190px;height: 190px;"> <table
align="center"> <tbody> <tr>'
            fh.write(input_tag)
            img_tag = '<td> <img src="KVQArefImgs_cropped_aligned_mtcnn_2/'
            img_tag_2 = '" border="5" alt =" Couldnot display image"> </td>'
            for i in range(0,3):
                fh.write(img_tag + str(dict[reverse[i]]) +img_tag_2)
            msg_2= ' </tr> <tr> '
            fh.write(msg_2)
            text_tag = '<td> <h4 style="color:red"> '
            for i in range(0,3):
                fh.write(text_tag+str(reverse[i])+"</h4> \n </td> \n")
            fh.write(msg_2)
            link_tag= '<td><a href="https://www.wikidata.org/wiki/'
            link_tag2='" target="_blank" >Who is this!!!</a> </td>  '
            for i in range(0,3):
                #print(str(dict[reverse[i]].replace(".png" , "")) )
                fh.write(link_tag + str(dict[reverse[i]].replace(".png" , ""))
+link_tag2)
            message2="""</tr></tbody></table></div> </body></html>"""
            fh.write(message2)
            fh.close()



if __name__ == '__main__':
    main()
```