華中科技大學

软件课程设计

（基于 ResNet 的行人重识别系统）

院　　系　电子信息与通信学院

专业班级　电信卓越 1501

姓　　名　林俊宏

学　　号　U201513247

指导教师　刘文予

2018 年 02 月 28 日

目　录

摘　要

行人重识别通常被认为是图片检索问题，需要将不同摄像机拍到的行人进行匹配。给出待匹配的行人图片，行人重识别系统将判断此人是否被其他摄像机拍到。我们采用了卷积神经网络来实现行人重识别算法。我们基于残差神经网络原理实现了训练模型，并在额外引入的 Market1501 数据库中测试得到 Top 1 为 87.4703%，Top 5 为 95.22%的结果。同时，我们用 C++实现了友好的图形界面，便于用户操作。

关键词：python; C++; 残差神经网络;

1 项目描述

1.1 问题背景

行人重识别通常被认为是图片检索问题，需要将不同摄像机拍到的行人进行匹配。给出待匹配的行人图片(query)，行人重识别系统将判断此人是否被其他摄像机拍到。我们采用了卷积神经网络(convolutional neural network, CNN)来实现行人重识别算法。目前，主要有两种 CNN 模型被用于此问题，分别是验证模型(verification model)和识别模型(identification model)。训练时，这两种模型的输入、输出、特征提取和损失函数都不相同（如图 1 所示）。

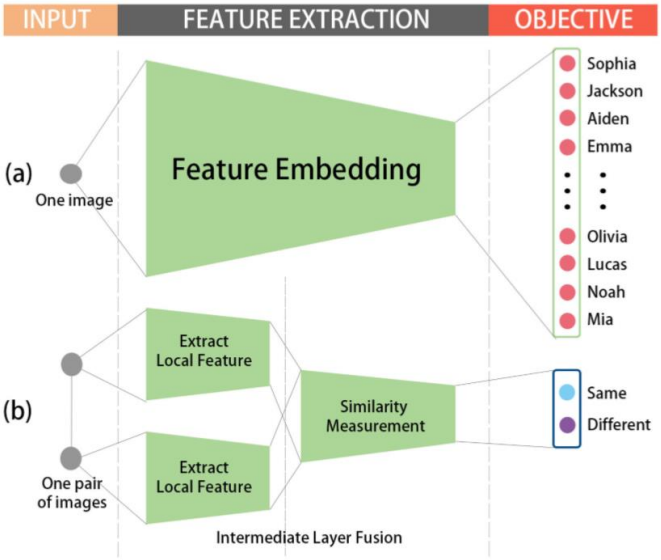

图 1 (a)为识别模型，(b)为验证模型

其中，验证模型的输入是一对图片（图 1 (b)），输出为判断它们是否属于同一个人。验证网络将属于同一个人的两张图片映射到特征空间中相近的地方，而不属于同一个人的两张图片映射到特征空间中距离较远的地方。验证模型的问题是只使用了弱标签，即没有考虑输入和数据集中其他图片的关系。

为了更好的使用数据集标签，识别模型将行人重识别问题当成分类问题来做（图 1 (a)）。训练时，输入图片并输出它所属的行人编号，在最后使用交叉熵损失。测试时，由全连接层提取特征，再计算归一化特征的欧氏距离。识别模型的主要问题是训练过程和测试过程不同，训练时没有考虑图片相似度，从而在检索时产生问题。最终我们决定以识别模型为基础搭建行人重识别系统。

1.2 项目分工

考虑到最终将实现一个较复杂的系统，且生成一个 exe 执行文件，我们决定将算法和界面的实现分开，使用 python 实现卷积神经网络算法，C++实现交互界面。

- 算法实现：罗天琦
- 界面实现：林俊宏

2 系统描述

2.1 算法部分

2.1.1 平台版本

本系统的算法部分使用 python 实现，版本为 python 3。为了更好地实现卷积神经网络模型，使用了 PyTorch 和 NumPy 库。由于训练所需计算量较大，我们在服务器上训练，服务器系统为 Ubuntu 5.4.0，Linux version 4.8.0-36-generic。服务器仅进行网络训练，训练结束后可以将模型参数和数据特征下载到本地进行测试。

最终实现的可执行程序在 Windows10 上运行，可以使用 CPU 或者 GPU 运行。

2.1.2 卷积神经网络库

PyTorch 是 Facebook 的 AI 研究团队发布了一个 Python 工具包，专门针对 GPU 加速的深度神经网络（DNN）编程。Torch 是一个经典的对多维矩阵数据进行操作的张量（tensor ）库，在机器学习和其他数学密集型应用有广泛应用。但由于 Torch 语言采用 Lua，导致在国内一直很小众，并逐渐被支持 Python 的 Tensorflow 抢走用户。作为经典机器学习库 Torch 的端口，PyTorch 为 Python 语言使用者提供了舒适的写代码选择。

NumPy 系统是 Python 的一种开源的数值计算扩展。这种工具可用来存储和处理大型矩阵，比 Python 自身的嵌套列表（nested list structure)结构要高效的多（该结构也可以用来表示矩阵（matrix））。NumPy（Numeric Python）提供了许多高级的数值编程工具，如：矩阵数据类型、矢量处理，以及精密的运算库。专为进行严格的数字处理而产生。多为很多大型金融公司使用，以及核心的科学计算组织如：Lawrence Livermore，NASA 用其处理一些本来使用 C++，Fortran

或 Matlab 等所做的任务。

2.1.3 深度残差网络（ResNet）模型

本系统使用的识别算法基于深度残差网络（Deep Residual Network, ResNet），深度残差网络是卷积神经网络的一种有效实现。具体的数据处理过程和网络结构将在第三节（算法模型描述）中解释。

卷积神经网络（Convolutional Neural Network, CNN）是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于大型图像处理有出色表现。它包括卷积层(convolutional layer)和池化层(pooling layer)。

深度残差网络在 2015 的 ILSVRC 比赛中获得取得第一的成绩，ICLR2016 上也是重点议题之一。它主要思想很简单，就是在标准的前馈卷积网络上，加一个跳跃绕过一些层的连接。每绕过一层就产生一个残差块(residual block)，卷积层预测加输入张量的残差（如图 4 所示）。

2.2 界面部分

2.2.1 自绘界面

界面的开发基于 C++，使用 Visual Studio Enterprise 2017 开发平台进行开发。基于自己设计开发的 DirectUI 界面绘制平台(DUI Lib)，对本软件用户界面进行了美观的设计。为了方便迭代开发和编程操作，采用了面向对象的方式对 DUI Lib 进行开发。

该界面绘制平台的基本开发思路为先根据用户界面控件的通用性质，如都为子窗口、都具有文本属性、都具有控件坐标属性等等，为控件开发出一个通用类(CDUIControlBase)。该类实现控件所需要的所有基本操作，如设置控件的位置大小、设置控件的文本等。据此，再以该通用类为父类进行继承，依次根据各控件的特异属性，如按钮的鼠标点按操作、滚动条的拖动操作等，开发各控件的相应子类(CDUIButton、CDUILabel、CDUIPictureBox 等)。开发完毕后，对所有类进行封装，封装为动态链接库后，在程序执行过程中动态调用该库。

自绘界面的显示，采取读取外部图像并经过适当拉伸放大后显示在窗口正确位置的方法实现。每当用户与界面进行交互后，程序会将控件新状态刷新显示在窗口上，实现人机交互。本界面绘制平台在实现上由于只需要更改外部图像源就能实现不同形式的界面展示，因此非常方便地实现换肤等提高用户体验的功能。但由于时间较紧，本软件仅实现了查找图片等功能，尚未实现设置部分的功能。

最终本软件运行过程的截图如下，可以看到显示了相似程度由高到低的 15

张照片，并能根据用户选择显示每张图片的匹配度与编号，实现了与用户良好的交互：

图 2 界面实现效果

2.2.2 界面的重设计

在设计之初，我们就充分考虑到了用户界面的可重设计性。用户 UI 在产品迭代过程中经常根据需求会发生很大的变动，如果在代码设计过程中就已经将每个空间的位置与大小等数据写入固化于代码中，在产品迭代过程中必然会带来很大的升级难度。

因此我们基于 XML 文件存放用户界面所有控件的必要信息，如位置、大小、样式、文本内容等，并在每次打开软件时读取该文件以确定用户界面的样式。因此在升级迭代过程中只需要修改该 XML 文件就能方便地实现空间的增删和修改。

2.2.3 界面与 Python 脚本调用

由于界面采用 C++语言实现，为方便实现对搜索部分 Python 脚本的调用，此处采用线程调用方法。由于我们在设计程序时已经方便地将 Python 脚本设计为可利用命令行方式通过传参调用，因此用户界面只需借用命令行方式对 Python 脚本进行运行，达到实现简单，方便用户操作的目的。

在查找图片界面，每当用户点击搜索按钮以后，采用 CreateProcess()方法运行 Python，将 Stdin 与 Stdout 数据流重定向至用户界面上，并将用户选定文件与目录作为命令参数传入，最后通过 Stdout 数据流与 GetExitCodeProcess()方法获取脚本命令行输出与脚本运行状态将脚本运行后查找到的相似图片显示在用户界面上。经过这一流程，用户操作被大大简化，并且实现了数据可视化。

而在生成 XML 界面，同样通过这一调用流程实现 Python 脚本的调用，

并且将脚本生成的.xml 文件显示在界面文本框中，用户可以实时对该文件内容进行复制修改，编辑完毕后点击保存按钮可以实现.xml 文件的保存。

2.3 数据预处理

先将所有图片调整大小，使短边等于 144，再随机截取为 256*128，然后将图片以 50%的概率水平翻转。最后进行数据归一化。

随机截取（random crop）和随机水平翻转（random flip horizontal）是为了进行数据增强，添加合成数据，使模型接触到更多训练数据。这样做减少过度拟合并提高模型的泛化能力。随机截取是为了去除行人图片中的环境影响，减少背景和行人位置在 CNN 模型决策中的贡献。水平翻转的原理是翻转后的行人图片应该和翻转前一样被识别。

归一化公式如下：

$$N = \frac{X - mean}{standard\ \ deviation}$$

其中 X 表示图片集，大小为[n,3,256,128]。其中 n 是图片张数，3 代表 RGB 三个通道。Mean 为整个图片集的均值，*standard deviation* 为图片集的标准差，N 表示归一化后的图片。

2.4 网络结构

2.4.1 深度残差网络

在了解了 AlexNet，VGG，GoogLeNet，ResNet 等多种经典模型之后，选择了 ResNet 进行实现和验证。深度残差网络（Deep Residual Network, ResNet）是卷积神经网络的一种有效实现。如图 3 所示，ResNet 的主要思想就是在标准的前馈卷积网络上，加一个跳跃绕过一些层的连接。每绕过一层就产生一个残差块(residual block)，卷积层预测加输入张量的残差。

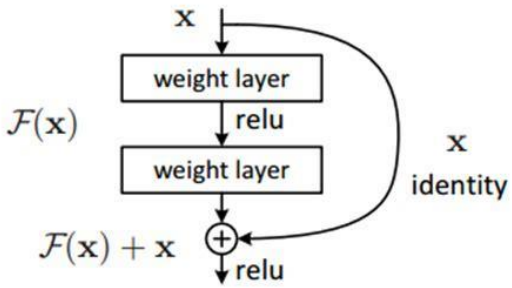

图 3 残差神经网络

我们在[15]的 resnet-50 的基础上实现算法，其网络结构如图 4 所示。左边为 ResNet50 的整体结构，其中 Layer 1 至 Layer 4 分别由多个残差块组成。图 4 右边为 Layer 1 的内部结构

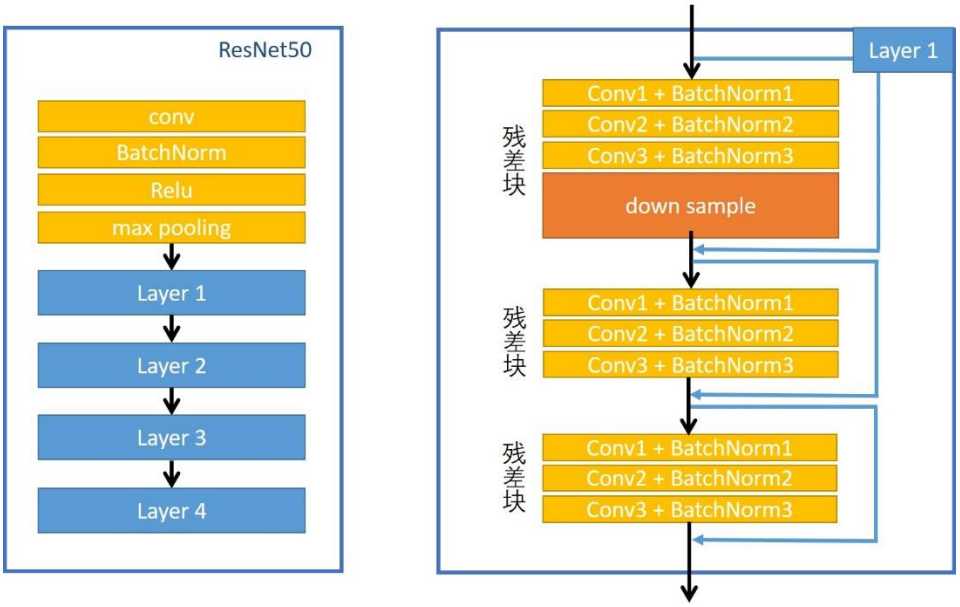

图 4 ResNet50

数据通过 ResNet50 之后，通过一个平均池化层（average pooling）。接下来又加了一层 Linear，一层 batchnorm，一层 ReLu。这时得到的就是图片特征（feature）。最后再接分类器。整体网络结构如图 5 所示。

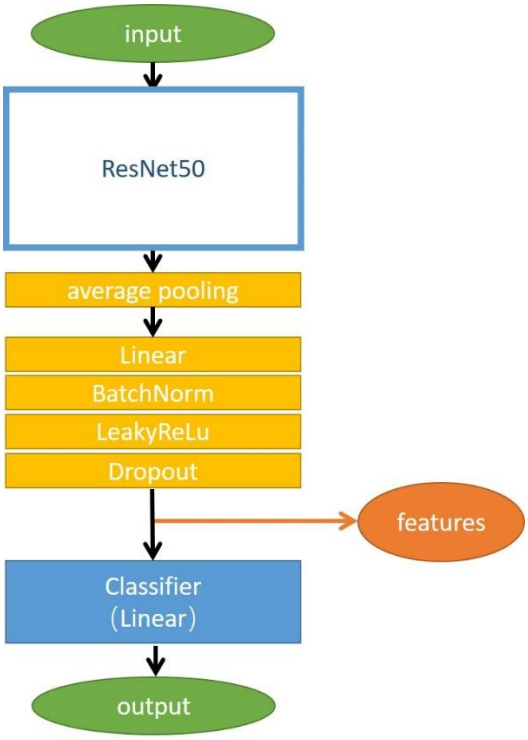

图 5 整体网络结构

2.4.2 损失函数

通过分类器之后，我们选择了交叉熵损失函数（cross entropy loss）。函数公式如下：

$$loss(x, label) = -\log \frac{e^{x_{label}}}{\sum_{j=1}^{N} e^{x_j}}$$

$$= -x_{label} + \log \sum_{j=1}^{N} e^{x_j}$$

2.5 测试过程

测试时输入图像调整大小为 288x144，并进行水平翻转。对于每张图片 $x_{3,256,144}$ 提取特征为 $f_{1,2048}$。对于每个 query 输入，将它的特征向量与每个 reference 图片的特征向量计算欧氏距离，再将结果排序得到前 15 张符合要求的图片。

3 软件设计

3.1 建立网络模型

使用 pytorch 建立 resnet-50 模型，设置平均池化层。

```
class ft_net(nn.Module):
    def __init__(self, class_num ):
        super(ft_net, self).__init__()
        model_ft = models.resnet50(pretrained=True)
        # avg pooling to global pooling

        model_ft.avgpool = nn.AdaptiveAvgPool2d((1,1))
```

在网络最后最后增加 fc 层。

```
        num_ftrs = model_ft.fc.in_features
        add_block = []
        num_bottleneck = 512
        add_block += [nn.Linear(num_ftrs, num_bottleneck)]
        add_block += [nn.BatchNorm1d(num_bottleneck)]
        add_block += [nn.LeakyReLU(0.1)]
        add_block += [nn.Dropout(p=0.5)]  #default dropout rate 0.5
        add_block = nn.Sequential(*add_block)
        add_block.apply(weights_init_kaiming)

        model_ft.fc = add_block
```

增加分类器 classifier 层。

```
        classifier = []
        classifier += [nn.Linear(num_bottleneck, class_num)]
        classifier = nn.Sequential(*classifier)
        classifier.apply(weights_init_classifier)

        self.classifier = classifier
```

设置前向传播行为。

```
    def forward(self, x):
        x = self.model(x)
        x = self.classifier(x)

        return x
```

3.2 训练过程

数据导入。

```
image_datasets['trimage_dataain'] =
datasets.ImageFolder(os.path.join(data_dir, 'train' + train_all),
data_transforms['train'])
image_datasets['val'] = datasets.ImageFolder(os.path.join(data_dir,
'val'),data_transforms['val'])
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x],
batch_size=opt.batchsize,shuffle=True, num_workers=4) for x in ['train',
'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}

class_names = image_datasets['train'].classes
```

数据预处理。

```
transform_train_list = [transforms.Resize(144, interpolation=3),
        transforms.RandomCrop((256,128)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
0.225])]
if opt.erasing_p>0:
    transform_train_list = transform_train_list +
[RandomErasing(opt.erasing_p)]
if opt.color_jitter:

    transform_train_list = [transforms.ColorJitter(brightness=0.1,
contrast=0.1, saturation=0.1, hue=0)] + transform_train_list
```

设置损失函数和 optimizer。

```
criterion = nn.CrossEntropyLoss()
ignored_params = list(map(id, model.model.fc.parameters() )) +
list(map(id, model.classifier.parameters() ))
base_params = filter(lambda p: id(p) not in ignored_params,
model.parameters())

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD([{'params': base_params, 'lr': 0.01},
           {'params': model.model.fc.parameters(), 'lr': 0.1},
           {'params': model.classifier.parameters(), 'lr': 0.1}
       ], momentum=0.9, weight_decay=5e-4, nesterov=True)
# Decay LR by a factor of 0.1 every 40 epochs

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=40,
gamma=0.1)
```

开始训练。

```
    for data in dataloaders[phase]:
            inputs, labels = data # get the inputs
            inputs, labels = Variable(inputs), Variable(labels)
            optimizer.zero_grad()  # zero the parameter gradients

            # forward
            outputs = model(inputs)
            _, preds = torch.max(outputs.data, 1)
            loss = criterion(outputs, labels)
            # backward + optimize only if in training phase
            if phase == 'train':
                loss.backward()

                optimizer.step()
```

3.3 测试过程

(1) 提取特征

```
def extract_feature(model,dataloaders):
    features = torch.FloatTensor()
    for data in dataloaders:
        img, label = data
        ff = torch.FloatTensor(n,2048).zero_()
        img = fliplr(img)
        input_img = Variable(img)
        f = model(input_img).data
        ff +=f
        fnorm = torch.norm(ff, p=2, dim=1, keepdim=True) # norm feature
        ff = ff.div(fnorm.expand_as(ff))
        features = torch.cat((features,ff), 0)

    return features
```

(2) 分析结果

```
def evaluate(qf, qn, gf, gn):
    score = np.dot(gf,qf) #点乘计算
    sorted_list = []
    for i in range(0,len(score)):
        sorted_list.append((score[i], gn[i]))
    sorted_list.sort() #对匹配度从小到大排序
    sorted_list.reverse() #顺序翻转
    result_name = []
    for i in range(0, len(sorted_list)):
        score, gn = sorted_list[i]
        result_name.append(gn)

    return result_name
```

(3)生成 XML 文件

从 xml.dom.minidom 库中导入 Document 类，使数据输出为 xml 文件。

3.4 用户界面初始化

初始时，直接读取资源文件夹中的 XML 文件以读取用户界面的样式，并对据此对资源文件夹中的皮肤文件进行读取。读取完毕后，即可建立父窗口，并处理窗口消息，实现主界面的显示与交互。

```
int APIENTRY WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR     lpCmdLine,
    int       nCmdShow)
{
    IDUIApp* pDUIApp = DUIGetAppInstance();
    if (pDUIApp == NULL) return -1;

    CResLoader loader;
    CDUIString strXMLEntry;
    loader.LoadXMLFromFile(_T(".\\res\\xml\\dui_control_demo.xml"), strXMLEntry);
    //pDUIApp->Init(strXMLEntry);

    BOOL bRet = pDUIApp->Init(strXMLEntry);//载入 XML 文件初始化皮肤部分
    DUI_ASSERT(bRet);
    if (!bRet) return -1;

    PopFileInitialize(0);
    IDUISkinMgr* pSkinMgr = pDUIApp->GetSkinMgr();
    IDUIControlFactory* pControlFactory = pDUIApp->GetControlFactory();

    CDUIString strTitle;
    pSkinMgr->GetStringText(_T("ids_title"), strTitle);//查询 XML 文件中关键字为 ids_title 的部
分获取窗口标题
    CMainWnd main;
    RECT rtMain = { 100, 100, 700, 700 };
    main.CreateFromResource(_T("main_wnd"), NULL, rtMain, strTitle.c_str()
        , WS_VISIBLE | WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN | WS_CLIPSIBLINGS);

    SetTimer(main, 1, 20, NULL);
    pDUIApp->GetMessageLoop()->Run();
    pDUIApp->Term();
    return 0;

}
```

3.5 Python 脚本调用

Python 脚本的调用采用 CreateProcess()方法进行线程生成，并重定向该现成的 Stdin 和 Stdout 管道，以软件实现模拟命令行调用实现自动化，简化用户的操作。

```
void CreateChildProcess(int choice)
    {
        CDUIString szCmdline = _T("python ");
        STARTUPINFO siStartInfo;
        BOOL bSuccess = FALSE;

        CResLoader mResLoad;
        CDUIString path, Current;
        CDUIString temp = _T("");
        mResLoad.ToAbsolutePath(temp);
        Current = temp;
        if (choice == 1)
        {
            temp = _T("test_single.py");
            //mResLoad.ToAbsolutePath(temp);
            path = szCmdline.append(temp);
            path = path.append(_T(" --pic_path \""));
            path.append(choose);
            path.append(_T("\""));
            mode = 1;
        }
```

```
    else if (choice == 2)
    {
        temp = _T("test_multiple.py ");
        //mResLoad.ToAbsolutePath(temp);
        path = szCmdline.append(temp);
        path = path.append(_T(" --query_dir \""));
        path.append(choose path);
        path.append(_T("\""));
        mode = 2;
    }

    ZeroMemory(&piProcInfo, sizeof(PROCESS_INFORMATION));

    ZeroMemory(&siStartInfo, sizeof(STARTUPINFO));
    siStartInfo.cb = sizeof(STARTUPINFO);
    siStartInfo.hStdError = g_hChildStd_OUT_Wr;
    siStartInfo.hStdOutput = g_hChildStd_OUT_Wr;
    siStartInfo.hStdInput = g_hChildStd_IN_Rd;
    siStartInfo.dwFlags |= STARTF_USESTDHANDLES;

    // Create the child process.

    bSuccess = CreateProcess(NULL,
        (LPWSTR)path.c_str(),     // command line
        NULL,          // process security attributes
        NULL,          // primary thread security attributes
        TRUE,          // handles are inherited
        CREATE_NO_WINDOW,           // creation flags
        NULL,          // use parent's environment
        Current.c_str(),         // use parent's current directory
        &siStartInfo,  // STARTUPINFO pointer
        &piProcInfo);  // receives PROCESS_INFORMATION

    if (!bSuccess)
        ErrorExit(TEXT("CreateProcess"));
}
```

注：由于用户界面代码繁琐，主函数消息处理部分就已经多达一千行代码，而界面库中每个.cpp 文件也有一千多行代码，由于控件类型繁多，均通过界面库实现了对控件的处理，此处无法展示界面库由此复杂的代码模块，因此仅展示主消息处理模块的部分代码。

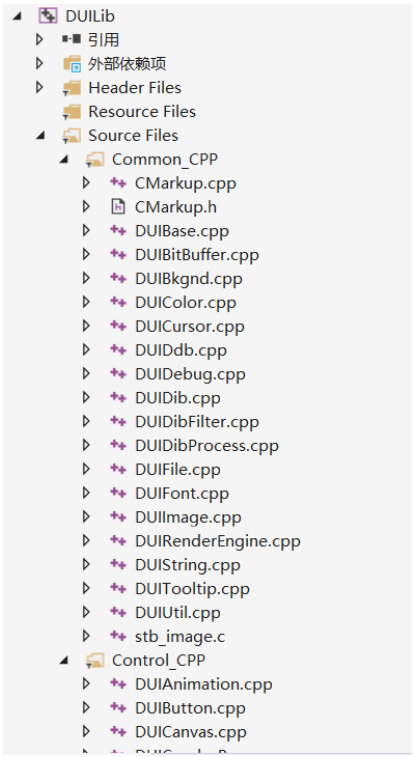

图 6 界面库部分代码文件

4 技术报告

4.1 训练结果

为了使训练结果更好并且易于检测效果，我们使用了 Market1501 数据集进行训练。训练过程如图 7 所示，其中 train 曲线对应 train 训练集，数据预处理时经过了随机截取和随机镜像，val 曲线对应 valid 集，数据预处理时没有经过随机截取和随机镜像，直接调整大小为 256*128。

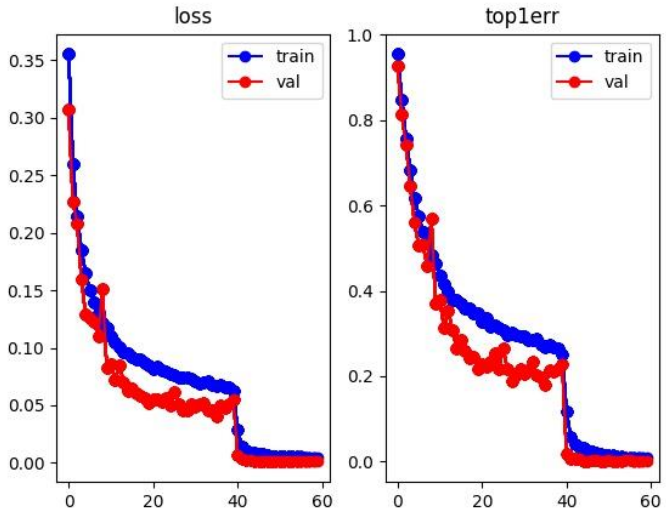

图 7 训练过程

4.2 测试结果

由于所给的测试集没有标签，我们在训练结束后使用 Market1501 测试集进行测试，结果如表 1。

表格 1 数据集 Market1501 测试结果

dataset	Market 1501
Top 1	87.4703 %
Top 5	95.2197 %
Top10	96.7637 %
mAP	69.6224 %

5 个人总结

本次课程设计，我与我的队友协同配合，我负责制作界面实现对神经网络部分的数据可视化。在这一过程中，我学习了 Python 语言，通过 C++在底层实现了对 Python 脚本的调用，提前对操作系统的知识进行了了解。而在编写代码的过程中我还学习到了神经网络的基本知识，感受到了通过机器学习能解决很多问题的魅力。C++与 DirectUI 的方式实现用户界面是非常繁琐的过程，我利用以前还只是一知半解的面向对象编程方式，最终设计出了上下层次清晰，功能多样的界面显示平台。虽然这次的软件课设花费了很多的时间进行调试，但通过这种基于项目导向的方式进行编程学习，我感觉效率非常的高，这是自己第一次实现了如此庞大的界面绘制平台，并且意识到了代码规范性对编程的方便性的影响，这对我未来在编程道路上的影响是非常的大。

6 附录（软件说明书）

6.1 搜索相似图片

本软件实现了对单张图片搜索相似照片的功能，该功能使用方法如下：

1、 打开软件，可以看到软件界面如下。如要直接搜索图片，可直接点击“打开文件按钮”。

图 8 软件启动主界面

2、 选择想要查找的图片，并点击打开按钮，如下图所示。

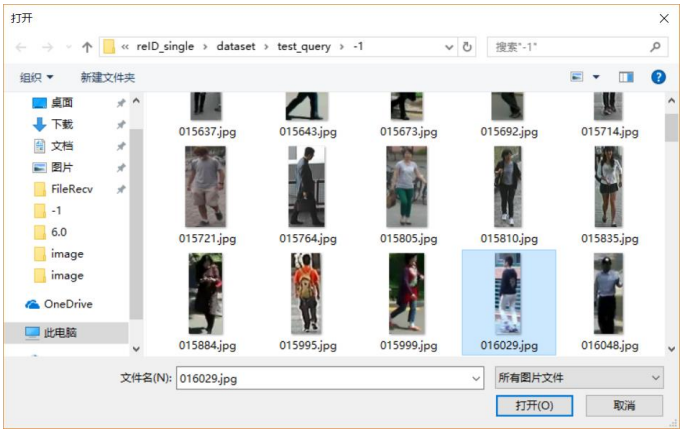

图 9 打开要搜索的图片

3、打开后，点击下面的搜索按钮，软件即开始在数据库中搜索与选中图片相似的照片，如下图所示。

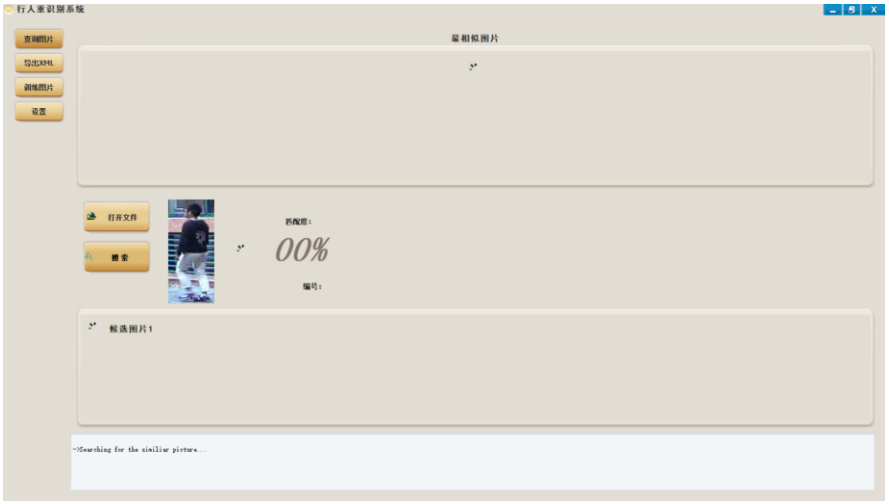

图 10 搜素相似图片

4、在搜索完毕以后，界面上显示出 15 张相似度由高到低的相似照片。在下面的候选图片选择栏中可以依次点击每张图片，可在界面正中央显示该图片以及匹配度、编号，以便用户了解该照片信息。

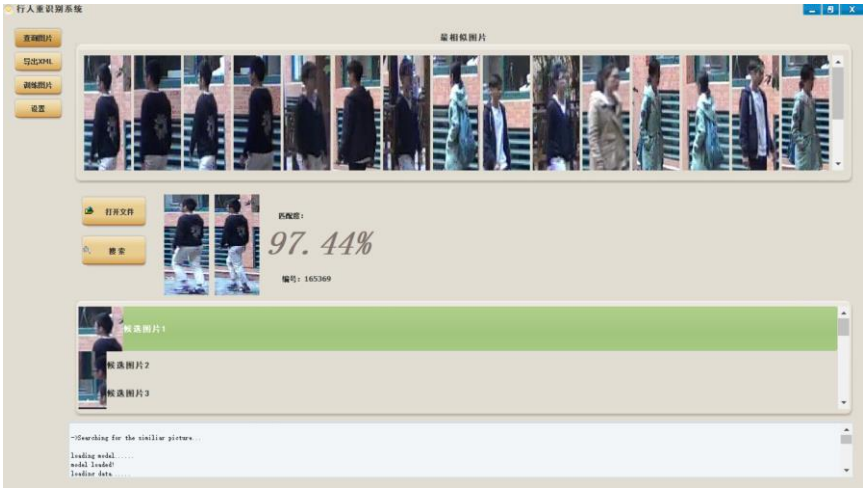

图 11 浏览搜索结果

6.2 导出所有查询结果

本软件还实现了对文件夹中（**注意：该文件夹中必须要有子文件夹**）所有待搜索图片进行搜索相似照片并导出结果的功能，该功能实现方法如下：

1、点击左边侧栏的“导出 XML”按钮，界面转换为导出结果页面，点击“打开”按钮选择一个需要检索的图片文件夹。

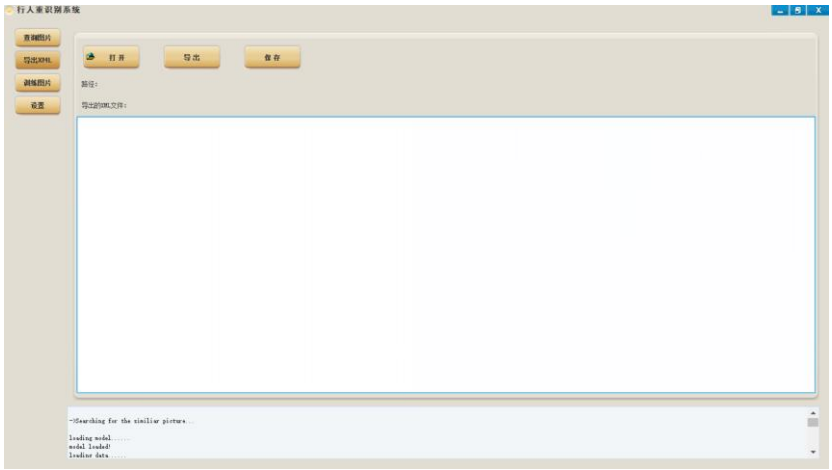

图 12 批量检索页面

2、选中一个包含了子文件夹（子文件夹中为待检索图片）的图片文件夹，并点击确定。

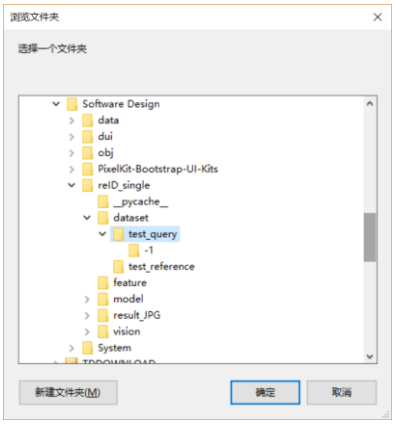

图 13 选中待检索文件夹

3、 点击“导出”按钮，下方命令行窗口显示软件已经开始批量检索，仅需稍等片刻即可得到检索结果，并可在文本框中编辑修改。

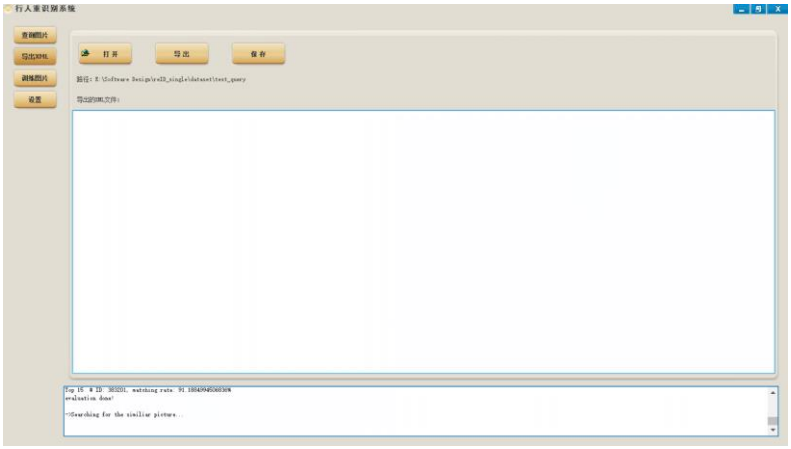

图 14 开始批量检索

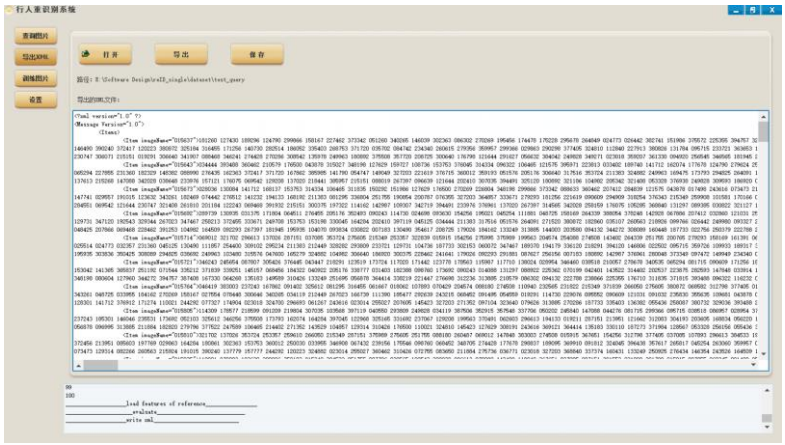

图 15 检索结果

4、点击“导出”按钮可以将检索结果导出为 XML 文件进行保存。