第12章 人类蛋白质图谱分类

12.1 案例简介

对人类蛋白质进行可视化成像是生物医学研究的常用手段，这些蛋白质可能是下一个医学突破的关键。然而，随着通量显微镜技术的进步，这些图像的生成速度远远超过了手动评估的速度，因此，如图12-1所示，开发一个自动化的人类蛋白质图谱分类工具，用以加速对人类细胞和疾病的理解，是一个极其迫切的需求。

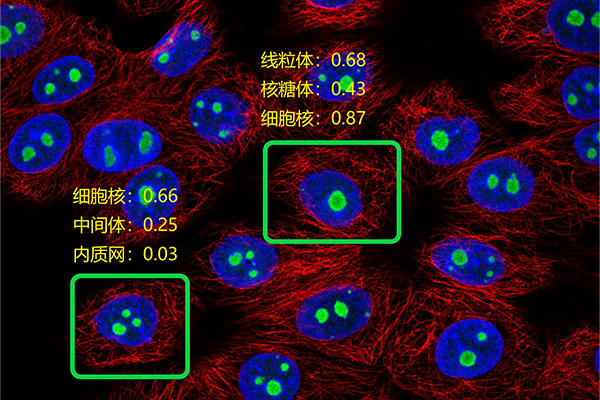


图 12-1 对蛋白质图像进行自动化的分类评估

本章案例属于对蛋白质图谱进行自动化分类评估的应用，旨在通过深度学习工具，对人类蛋白质荧光显微图片中的细胞器进行精准识别。该案例进一步将训练好的模型移植到华为Atlas 300AI加速卡上，对未标注的蛋白质荧光显微图片进行亚细胞位置的预测，以拓展昇腾（Ascend）芯片在生物领域的应用。

本案例完成的系统在华为Atlas 300AI加速卡上实现了对输入的任意人类蛋白质图谱进行多元分类预测，同时将预测结果可视化，并保存可视化的结果，具有完整性、代表性和实用性，满足了对生物医学图像进行自动化分析和标注的需求，具有很大的便捷性和应用场景。

12.2 系统总体设计

该系统使用深度学习框架Caffe来训练模型，然后再在Atlas 300AI环境上将训练好的Caffe模型转换成Atlas 300AI需要的om模型，之后将转换的模型和待推理的测试图片一同导入到Atlas 300AI环境中，使用Atlas 300AI环境进行模型的推理，以及可视化预测结果。

12.2.1 功能结构

整个系统可以划分为数据集处理、模型构建、用户交互等三个主要子系统。各个子系统相互关联，其中数据集处理子系统包括对数据集的收集、筛选、数据预处理以及候选区域提取等。模型构建子系统包括使用预训练的模型进行特征提取，使用提取的特征进行样本分类、网络定义、模型训练和模型推理等。用户交互子系统包括图像选择、启动预测模型和预测结果展示等。整体结构图如图12-2所示。

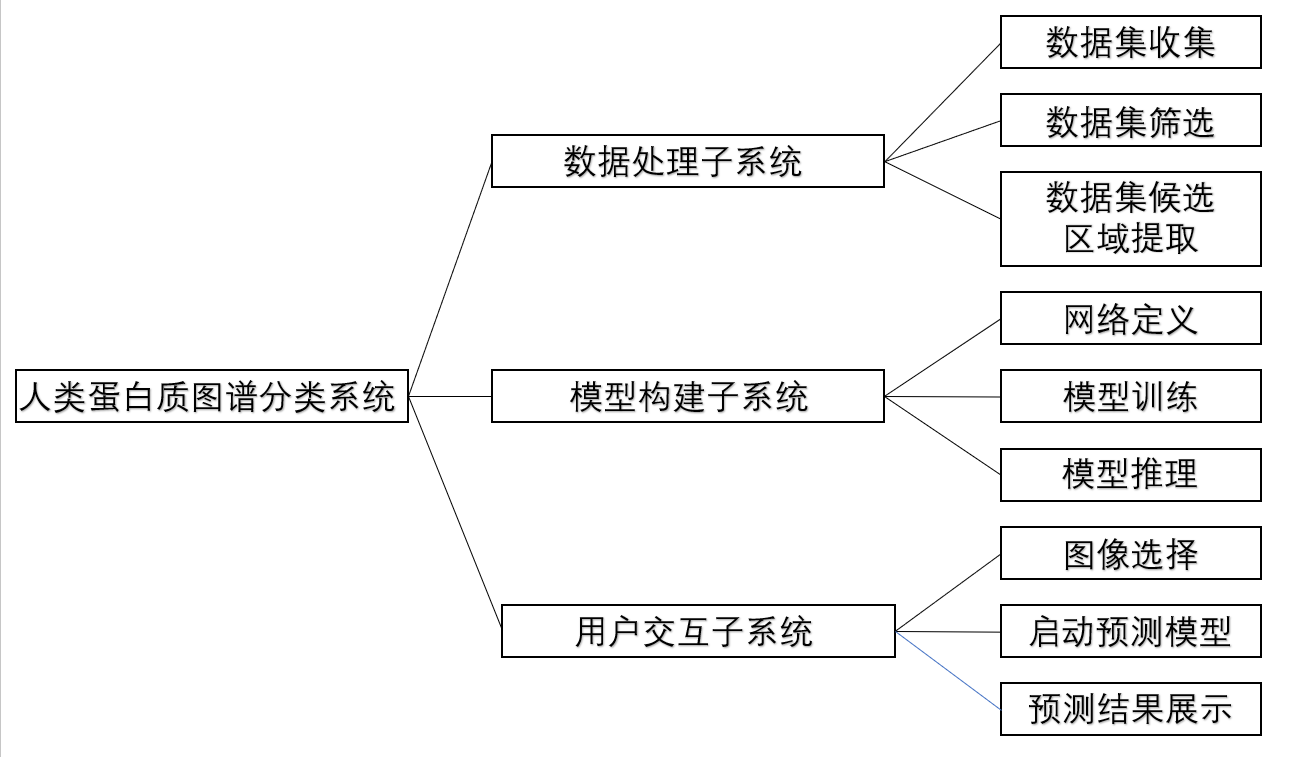


图 12-2 系统整体功能结构图

12.2.2 体系结构

按照运行流程划分，系统分成两个阶段，分别是训练阶段和预测阶段。训练阶段在Human Protein Atlas数据集上使用Caffe的resnet18模型进行训练。推理阶段包括可视化GUI工具开发，使用可视化GUI工具导入蛋白质图片、模型转换、导入模型、模型推测、预测结果展示等步骤，具体流程参照图12-3。

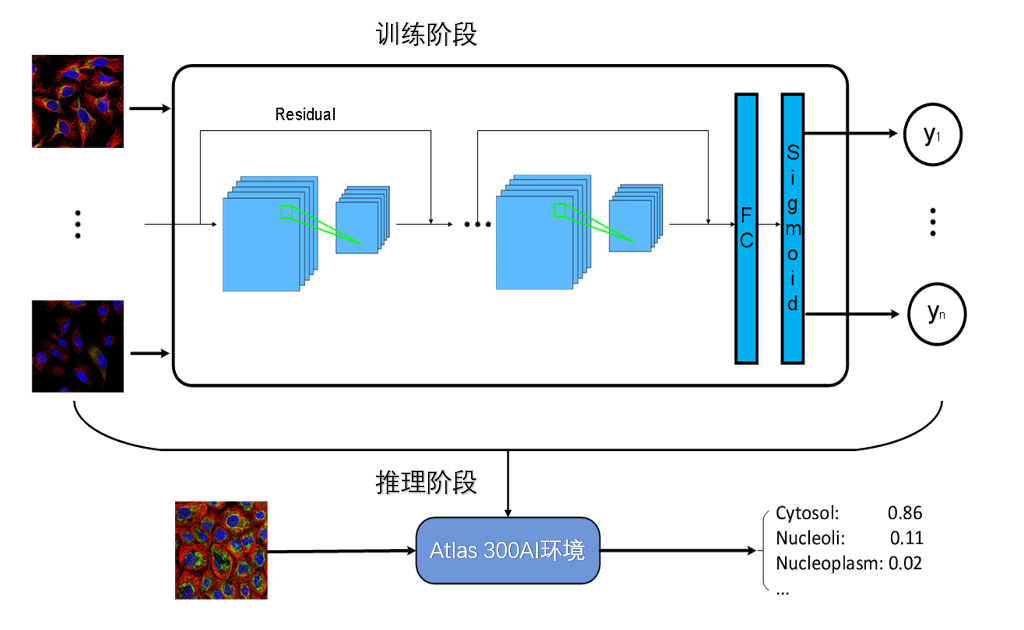


图 12-3 系统流程图

12.3 系统设计与实现

本节将详细介绍系统的设计与实现。12.3.1节给出人类蛋白质图谱数据集的下载与筛选等预处理过程。12.3.2节介绍对下载的数据进行挑选和截取有效区域部分的过程；12.3.3节介绍最终使用的数据集的制作过程，数据的存储采用Caffe支持的高效数据结构LMDB（Lightning Memory Mapped Database）； 12.3.4节介绍如何基于Caffe训练一个ResNet模型； 12.3.5节则介绍如何进行模型的测试，以及相应测试结果的展示。

12.3.1 数据集的下载与筛选

本案例所使用的图像数据来自人类蛋白质图谱数据库（human protein altas），目的是利用各种组学技术（包括抗体成像，质谱分析，蛋白质组学等）绘制所有人类蛋白质和中的表达和空间分布图。该数据库可免费使用，有助于加速生命科学研究和药物发现。该数据库收集了主要的蛋白质图谱数据，将其划分为了10个类别，分别表示不同的细胞器标签。

在收集得到人类蛋白质图谱数据库之后，对得到的数据进行一轮简单的筛选。由于数据是使用网络爬虫从网站上爬取下来的，因而数据集的质量良莠不济，甚至一些图片并没有下载完整。因而对下载之后的图片数据进行一轮筛选，剔除掉那些下载不完全、图片模糊、图片对比度低、图片分辨率低的数据，提高整体的数据的质量。

12.3.2 选择性搜索算法

在完成了第一轮的数据筛选之后，得到了一批相对精良的数据。样例数据图片如图12-4所示。

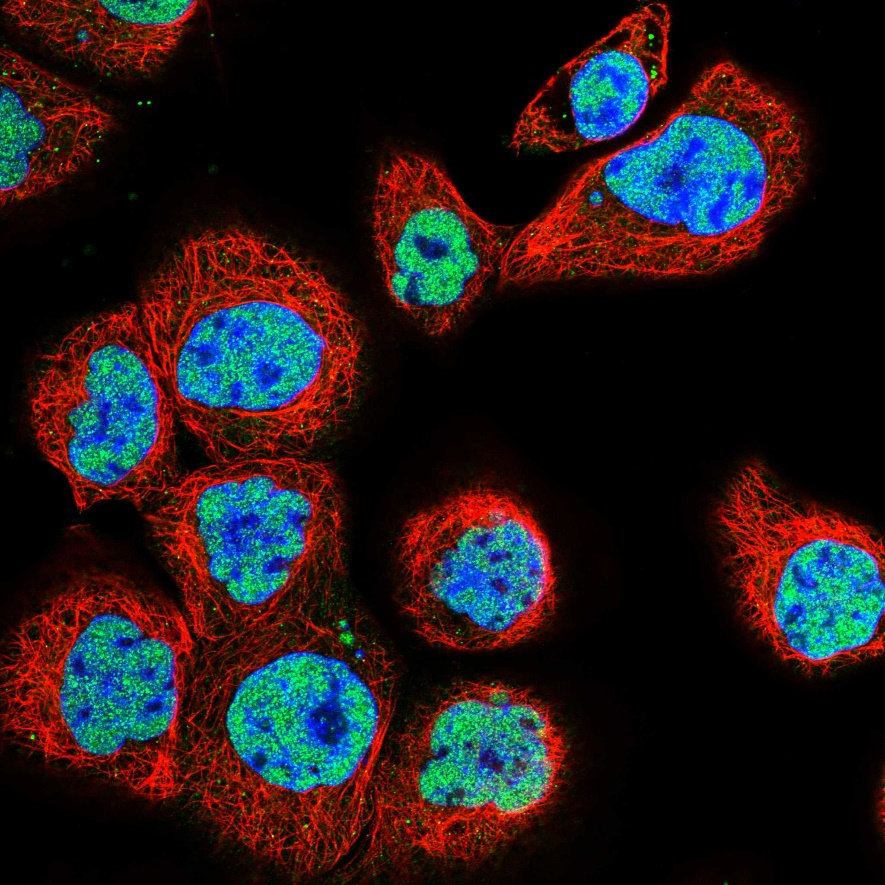


图12-4 挑选之后的图谱图片

以图12-4为例，其中每张图片都是2048×2048尺寸的人类蛋白质细胞图片。在这张图片中包含了众多的细胞器，如果直接将这样的图片输入的网络模型中进行训练，将会给网络训练带来极大的负担。同时为了提高网络模型的泛化能力，本案例在上一轮完成的数据筛选的基础上，对筛选之后的数据执行了选择性搜索算法。

选择性搜索算法目前主要运用于目标检测和目标识别领域。在目标检测时，为定位到目标的具体位置，通常将图像分为许多子块，然后把子块作为输入，送入到目标识别模型中。划分子块的最直接的方法是滑动窗口法（sliding window approach）。滑动窗口的方法是按子块的大小在整幅图像上穷举所有的子图像块，算法数据量非常大。和滑动窗口相对的是另一类基于候选区域(region proposal）的方法。候选区域算法用分割不同区域的办法来识别潜在的物体。在分割的时候，要合并那些在某些方面（如颜色、纹理）类似的小区域。相比滑动窗口法在不同位置和大小的穷举，候选区域算法将像素分配到少数的分割区域中。所以最终候选区域算法产生的数量比滑窗法少的多，从而大大减少运行物体识别算法的次数。同时候选区域算法所选定的范围天然兼顾了不同的大小和长宽比。

区域合并算法是总体思路是，假设现在图像上有*n*个预分割的区域,表示为*R*={*R*1, *R*2, ..., *Rn*}, 计算每个区域与它相邻区域(注意是相邻的区域)的相似度,这样会得到一个*n*×*n*的相似度矩阵(同一个区域之间和一个区域与不相邻区域之间的相似度可设为NaN)。从矩阵中找出最大相似度值对应的两个区域，将这两个区域合二为一，这时图像上还剩下*n*-1个区域; 重复以上过程(只需要计算新的区域与它相邻区域的新相似度，其他的不用重复计算)，重复一次,区域的总数目就少1，直到所有的区域都合并成一个区域(即此过程进行了*n*-1次，区域总数目最后变成了1)。算法的设计如图12-5所示。

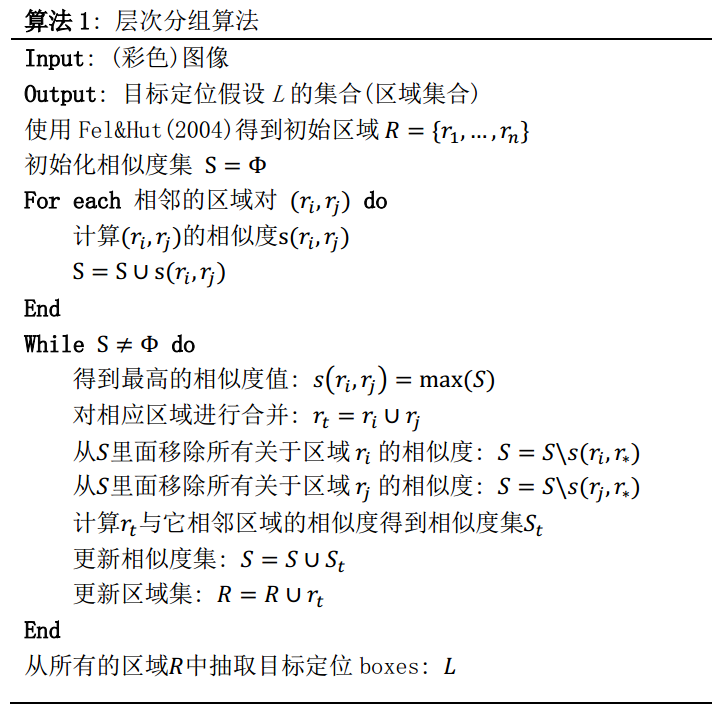


图 12-5 层次算法设计

在相似度计算上，选择性搜索算法主要是通过区域之间在颜色/纹理/大小和形状交叠四个方面的加权和进行衡量的。

为了计算颜色相似度，将色彩空间转为HSV，每个通道下以bins=25计算直方图，这样每个区域的颜色直方图有25×3=75个区间。对直方图除以区域尺寸做归一化后使用式(12-1)计算颜色相似度（color similarity）：

 (12-1)

为计算纹理相似度，采用方差为1的高斯分布在8个方向做梯度统计，然后将统计结果（尺寸与区域大小一致）以bins=10计算直方图，直方图区间数为8×3×10=240（使用RGB色彩空间）。纹理相似度（texture similarity）的计算公式如式(12-2)：

 (12-2)

尺寸相似度（size similarity）的计算公式如式(12-3)：

 (12-3)

交叠相似度（shape compatibility measure）的计算公式如式(12-4)：

 (12-4)

使用以上算法和相似度计算公式，通过不断的迭代，区域合并效果如图12-6所示。

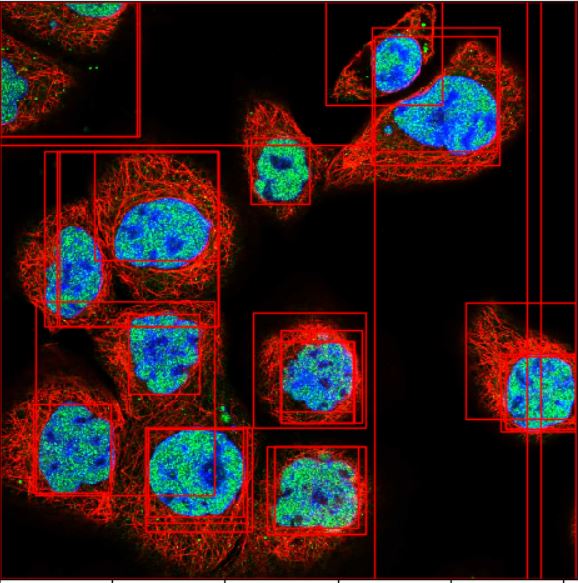
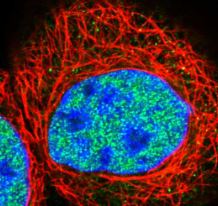
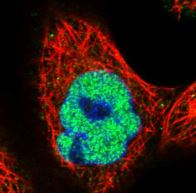
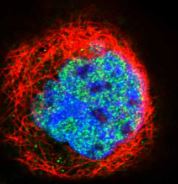


图12-6 选择性搜索算法搜索的候选框

12.3.3 数据集的制作

在使用上一节选择性搜索算法之后，在一个蛋白质图像上面框选出了众多的候选区域。然后对这些候选区域进行截取，获得较小规模的蛋白质图像，如图12-7所示，这些较小规模的蛋白质图像包含单个细胞器，便于模型进行精准的学习训练与推理。

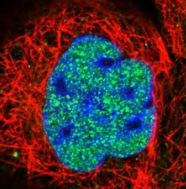
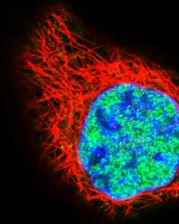
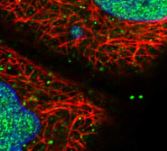
  

图12-7 较小规模的蛋白质图像

在Caffe中经常使用的数据类型是LMDB，不是常见的jpg/jpeg/png/tif等格式。比起单张图片，它具有I/O效率高、支持多线程并发读写、节省内存、语义完全符合数据库ACID性（原子性、一致性、隔离性和持久性）等特点。由于人类蛋白质图谱数据集存储形式为jpg，需要对其进行格式转换，得到LMDB格式的文件。

生成LMDB格式的数据，需要准备的条件有：（1）编译好Caffe并编写好convert\_imageset; （2）准备好需要进行转换的图片数据，将其统一放置到文件夹中; （3）编写好每张图片对应的标签文件如train.txt，在该文件中，每行记录代表一个样本，每行记录分别包含图片名称和相应label; （4）编辑shell脚本create-lmdb.sh。（5加快模型的训练以及归一化输入的数据，还需要对图片数据进行去均值的预处理，这一步在convert\_imageset方法中可以通过提供均值数据来完成，因此需要先制备均值数据。

制备均值数据的脚本create\_train\_val\_mean.sh的内容如程序清单12-1所示。

程序清单12-1 create\_train\_val\_mean.sh脚本

#!/usr/bin/env sh

# Compute the mean image from the imagenet training lmdb

# N.B. this is available in data/ilsvrc12

EXAMPLE=/home/leihouchao/leihouchao/caffeHuaWei/lmdb\_data/

DATA=/home/leihouchao/leihouchao/caffeHuaWei/lmdb\_data/

TOOLS=/home/leihouchao/caffe/build/tools/

echo "\nprocess train mean lmdb..."

$TOOLS/compute\_image\_mean $EXAMPLE/train\_lmdb \

$DATA/train\_mean.binaryproto

echo "\nprocess test mean lmdb..."

$TOOLS/compute\_image\_mean $EXAMPLE/test\_lmdb \

$DATA/test\_mean.binaryproto

echo ""

在获得均值文件之后，就可以开始制作LMDB格式的数据集了，制作数据集的脚本create\_train\_val\_lmbd.sh的源代码为程序清单12-2所示。

程序清单12-2 制作lmbd格式的数据集

echo "create train lmdb for resnet18..."

/home/leihouchao/caffe/build/tools/convert\_imageset \

--resize\_height=224 \

--resize\_width=224 \

--backend="lmdb" \

--shuffle \

/home/leihouchao/leihouchao/caffeHuaWei/data/train/ \

/home/leihouchao/leihouchao/caffeHuaWei/data/train.txt \

/home/leihouchao/leihouchao/caffeHuaWei/lmdb\_data/train\_lmdb

echo "done"

echo ""

echo ""

echo "create val lmdb for resnet18..."

/home/leihouchao/caffe/build/tools/convert\_imageset \

--resize\_height=224 \

--resize\_width=224 \

--backend="lmdb" \

--shuffle \

/home/leihouchao/leihouchao/caffeHuaWei/data/test/ \

/home/leihouchao/leihouchao/caffeHuaWei/data/test.txt \

/home/leihouchao/leihouchao/caffeHuaWei/lmdb\_data/test\_lmdb

echo "done"

echo ""

echo ""

12.3.4 模型训练

本案例在搭建网络模型时，采用的是基于Caffe的resnet18模型，使用该模型来对输入的蛋白质图片数据进行特征的提取，ResNet（Residual Neural Network）由微软研究院的Kaiming He等四名华人提出，在ILSVRC2015比赛中取得冠军，在top5上的错误率为3.57%，同时参数量比VGGNet低，效果非常突出。ResNet的结构可以极快的加速神经网络的训练，模型的准确率也有比较大的提升。

ResNet的主要思想是在网络中增加了直连通道，即Highway Network的思想。此前的网络结构是对输入做一个非线性变换，而Highway Network则允许保留之前网络层的一定比例的输出。ResNet的思想和Highway Network的思想也非常类似，允许原始输入信息直接传到后面的层中，如图12-8所示。

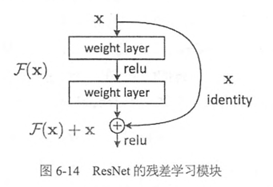


图12‑8 ResNet残差学习模块

由此，该层的神经网络可以不用学习整个的输出，而是学习上一个网络输出的残差，因此ResNet又叫做残差网络。Resnet网络是当前应用最为广泛的CNN特征提取网络，采用它作为网络模型的特征提取单元，在众多的ResNet网络结构中，选择的是resnet18，主要原因是resnet18的网络规模较小，容易进行训练和微调，在Caffe中，resnet18的网络架构图如图12-9所示。

Caffe的网络定义主要在prototxt中完成。在设计时，在resnet18的基础上做了些许的修改，以便搭建起适合于本项目的网络架构。具体而言，将resnet18的最后一层的全连接层去掉，替换为一个新的未训练的全连接层，该全连接层的输出为10个单元，对应10个细胞器标签的分类。其中，对train.prototxt做的主要的修改是将resnet18的最后一层替换为10分类的输出，具体修改如程序清单12-3所示。

在使用Caffe训练模型时，还需要solve.prototxt文件来设置训练模型相关的细节，比如batch\_size、学习率、模型保存位置等相关参数。本章具体使用的solver.prototxt文件的内容如程序清单12-4所示。



图 12-9 resnet18网络结构图

程序清单12-3 对train.prototxt的修改

layer {

name: "data"

type: "Data"

top: "data"

top: "label"

include {

phase: TRAIN

}

transform\_param {

mirror: true

crop\_size: 224

mean\_file: "/home/leihouchao/leihouchao/caffeHuaWei/lmdb\_data/train\_mean.binaryproto"

}

data\_param {

source: "/home/leihouchao/leihouchao/caffeHuaWei/lmdb\_data/train\_lmdb"

batch\_size: 16

backend: LMDB

}

}

...

layer {

bottom: "pool5"

top: "score"

name: "score"

type: "InnerProduct"

param {

lr\_mult: 10

decay\_mult: 1

}

param {

lr\_mult: 20

decay\_mult: 1

}

inner\_product\_param {

num\_output: 10

weight\_filler {

type: "xavier"

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "score"

name: "prob"

type: "Sigmoid"

top: "prob"

}

程序清单12-4 solver.prototxt文件的配置信息

# 定义的网络

net: "/home/leihouchao/leihouchao/caffeHuaWei/model/train.prototxt"

test\_net : "/home/leihouchao/leihouchao/caffeHuaWei/model/test.prototxt"

# 测试时的batch\_size是 16 ,故测试时跑的迭代次数是: 10925 / 16 =683次

test\_iter: 683

# 每训练1000次进行一次测试

test\_interval: 10000

test\_initialization: false

#初始的学习率

base\_lr: 0.0001

type : "Adam"

# 每 100个 训练batch 显示一下结果

display: 100

#学习率的下降策略

lr\_policy: "fixed"

# 训练集迭代的次数： 27540 \* 16 = 44069 \* 10 即训练10个epochs左右

max\_iter: 27540

power: 1

#梯度下降策略，90%依赖这次，10%依赖上次

momentum: 0.9

#正则项的比重

weight\_decay: 0.0005

#迭代多少次保存一次中间的caffemodel

snapshot: 1000

#caffemodel的前缀

snapshot\_prefix: "saved\_model"

random\_seed: 0

# 使用gpu训练

solver\_mode: GPU

程序清单12-5 模型训练的代码

import caffe

import numpy as np

caffe.set\_mode\_gpu()

caffe.set\_device(0)

from sklearn.metrics import recall\_score, f1\_score, precision\_score

import warnings

warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt;

import os

# 训练

def train():

# 1,定义solver

solver = caffe.get\_solver("./model/solver.prototxt")

# 2,定义参数

weights = "./model/resnet-18.caffemodel"

train\_preds = []

train\_gts = []

epochs = 82620 # 迭代次数30 epochs

display\_epochs = 100

# 3,加载预训练模型的参数

solver.net.copy\_from(weights)

print("\n\n\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\* start training \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n")

# 4,开始训练

for it in range(1, epochs + 1):

# (1)执行一次前向计算和反向传播

solver.step(1)

# print("loss: ", solver.net.blobs['loss'].data)

# (2)获取预测值和真实值

train\_gt = (solver.net.blobs['label'].data)[:,:,0,0]

train\_pred = solver.net.blobs['score'].data

train\_pred[train\_pred >= 0] = 1

train\_pred[train\_pred < 0] = 0

train\_gts.extend(train\_gt)

train\_preds.extend(train\_pred)

# (3) 进行100次之后统计计算 准确率，召回率，f1值等指标

if it % display\_epochs == 0:

train\_gts = np.array(train\_gts)

train\_preds = np.array(train\_preds)

train\_precision = precision\_score(train\_gts, train\_preds, average="macro")

train\_recall = recall\_score(train\_gts, train\_preds, average="macro")

train\_f1 = f1\_score(train\_gts, train\_preds, average="macro")

print("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* train集指标 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*")

print("\*")

print ('\* iteration:%d\n\t查准率: %.4f\t查全率: %.4f\tF1值: %.4f' %(it, train\_precision, train\_recall, train\_f1))

train\_gts = []

train\_preds = []

print("\*")

print("\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n")

print("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\* end training \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n")

最终训练生成的网络模型如图12-10所示：

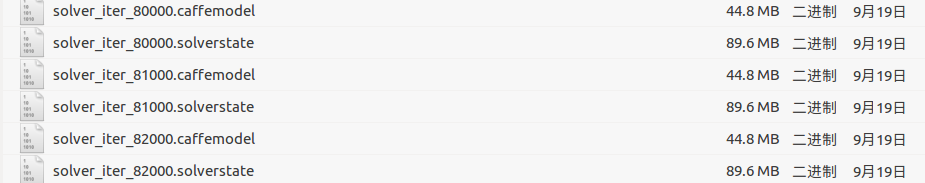


图 12-10 网络模型展示

12.3.5 模型推理

12.3.4节已完成了模型训练，本节将使用测试集数据对模型的性能进行测试。测试时，和训练时一样，也需要配置prototxt文件来设置网络结构以及数据集的相关信息。这里将该文件命名为test.prototxt，test.prototxt文件的主要修改部分如程序清单12-6所示:

程序清单12-6 test.prototxt文件的主要部分

layer {

name: "data"

type: "Data"

top: "data"

top: "label"

include {

phase: TEST

}

transform\_param {

mirror: false

crop\_size: 224

mean\_file: "/home/leihouchao/leihouchao/caffeHuaWei/lmdb\_data/test\_mean.binaryproto"

}

data\_param {

source: "/home/leihouchao/leihouchao/caffeHuaWei/lmdb\_data/test\_lmdb"

batch\_size: 16

backend: LMDB

}

}

...

layer {

bottom: "pool5"

top: "score"

name: "score"

type: "InnerProduct"

param {

lr\_mult: 10

decay\_mult: 1

}

param {

lr\_mult: 20

decay\_mult: 1

}

inner\_product\_param {

num\_output: 10

weight\_filler {

type: "xavier"

}

bias\_filler {

type: "constant"

value: 0

}

}

}

对网络模型进行测试的源代码如程序清单12-7所示:

程序清单12-7 对网络模型进行测试

import caffe

import numpy as np

caffe.set\_mode\_gpu()

caffe.set\_device(0)

from sklearn.metrics import recall\_score, f1\_score, precision\_score

import warnings

warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt;

import os

# 定义测试函数,计算训练好的模型的测试性能指标

def test():

# 1,定义网络

net = caffe.Net("./model/test.prototxt", "./saved\_model/solver\_iter\_70000.caffemodel", caffe.TEST)

print("\n\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* start testing \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

# 2, 定义计算指标所需参数

num\_batches = 683

ys = []

preds = []

for \_ in range(num\_batches):

# 3,前向计算

net.forward()

# 4,取出预测输出和真实值

y = net.blobs['label'].data[:, :, 0, 0]

pred = net.blobs['score'].data

pred[pred >= 0] = 1

pred[pred < 0] = 0

ys.extend(y)

preds.extend(pred)

# 5,计算指标

ys = np.array(ys)

preds = np.array(preds)

test\_precision = precision\_score(ys, preds, average="macro")

test\_recall = recall\_score(ys, preds, average="macro")

test\_f1 = f1\_score(y, pred, average="macro")

print("\n\n查准率: %.4f\n" % test\_precision)

print("查全率: %.4f\n"% test\_recall)

print("F1值: %.4f\n" % test\_f1)

print("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* end testing \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n")

网络模型的测试指标主要是分类相关的指标。本案例采用召回率、准确率和*f*1值这3个指标来衡量模型精度，测试的结果如图12-11所示:

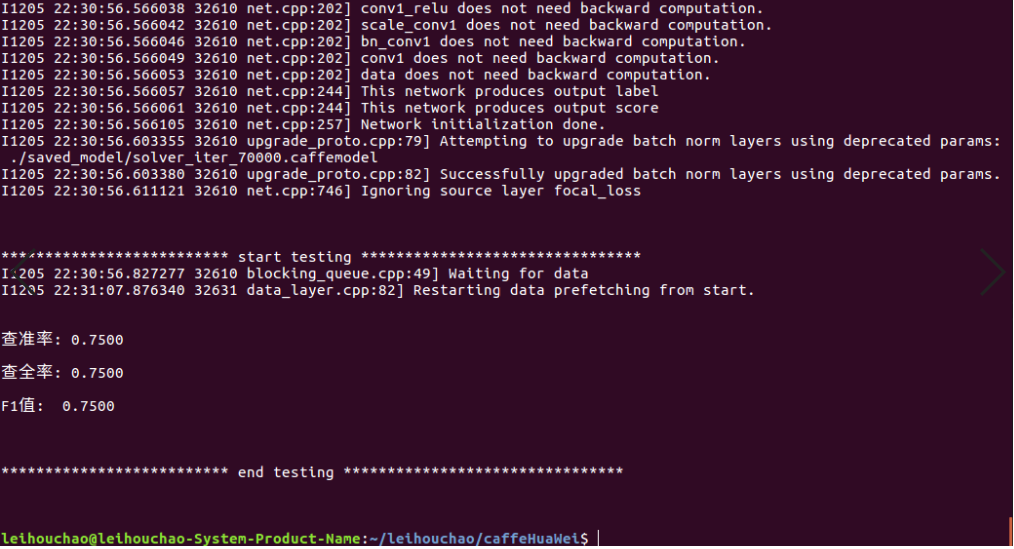


图12-11 测试结果图

12.4 案例系统部署

本案例系统最终运行在Atlas 300AI平台上，本章将会分为两个小节来介绍如何在Atlas 300AI加速卡环境下部署和运行本案例的实验。本案例是基于server/client模式进行部署的。12.4.1节介绍如何借助Atlas 300AI环境进行Caffe模型文件的转换，以及进行server端部署的详细介绍。12.4.2节介绍如何进行client端的部署。

12.4.1 Server端部署

Server端的所有代码在本案例的配套程序中的server文件夹下面。进行server端部署时，需要将server文件夹下的所有文件复制到Atlas 300AI环境/home /HwHiAiUser /HIAI\_PROJECTS /hpa 目录下。因为本程序是基于socket进行通信的，所有在运行程序之前需要修改程序 server.py中的ip地址和端口，ip是Atlas 300AI环境下的ip，可采用如图12-12的方式查询。

在修改ip完成之后，可按照以下步骤进行部署操作：

（1）转换hpa模型。

执行命令：

atc --model=caffe\_model/hpa.prototxt \

    --weight=caffe\_model/hpa.caffemodel \

    --framework=0 \

    --output=model/deploy\_vel \

    --soc\_version=Ascend310 \

    --input\_format=NCHW \

    --input\_fp16\_nodes=data \

     -output\_type=FP32 \

    --out\_nodes="score:0"

该命令将会把caffe\_model目录下的caffe模型转换为Atlas 300AI环境需要使用的om模型，并保存到 model/目录下，命名为deploy\_vel.om。

（2）编译调用om模型的程序。

执行命令：

cd build/intermediates/host

cmake ../../../src -DCMAKE\_CXX\_COMPILER=g++ -DCMAKE\_SKIP\_RPATH = TRUE

make

（3）修改编译出来的文件的权限。

执行命令：

cd ../../../out chmod 777 main

（4）运行程序。

cd ..

python3.7.5 server.py

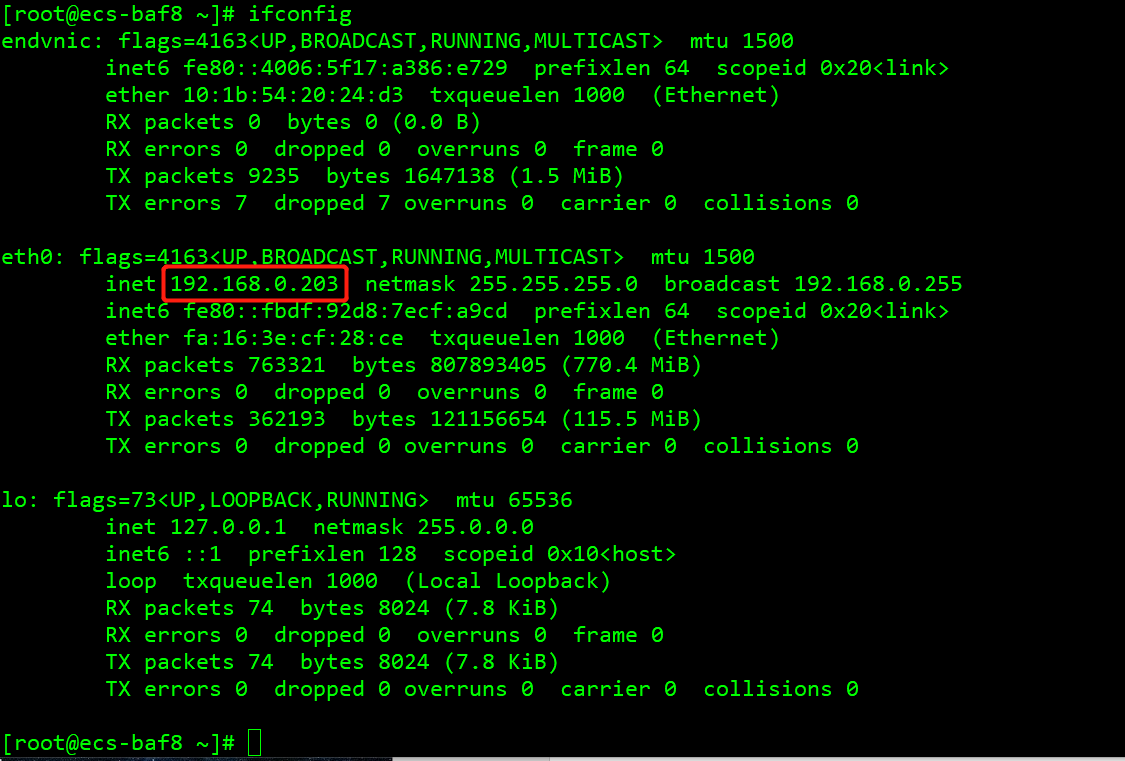
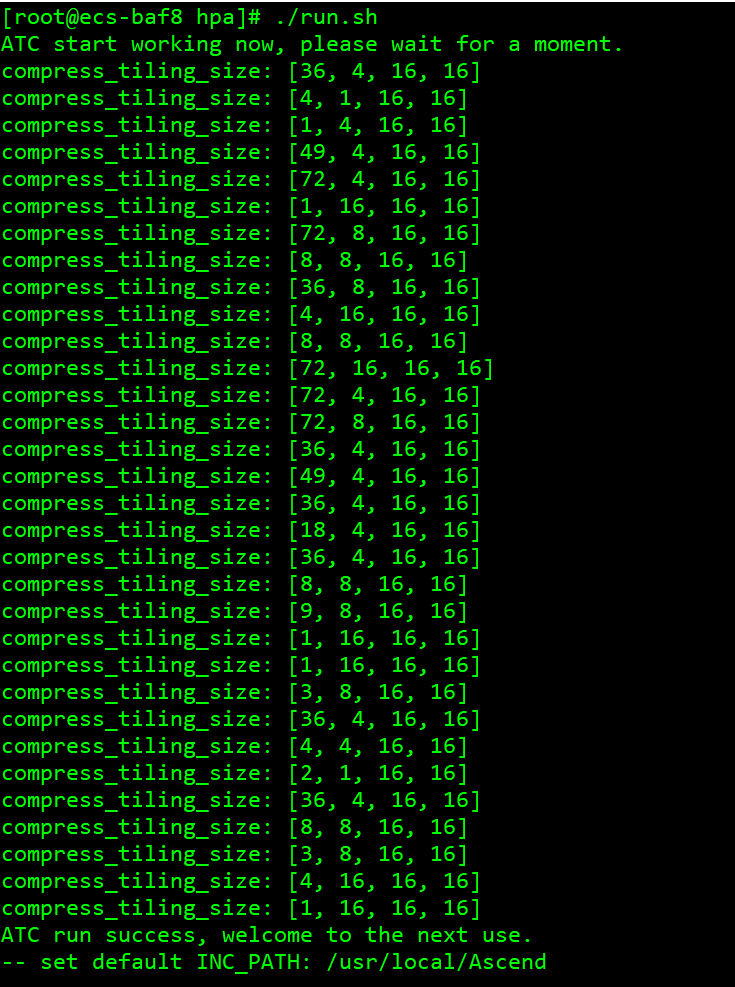


图12-12 查询ip地址

为了方便整个部署操作，本案例还提供了一键式部署脚本 run.sh，在使用之前，需要使用命令 chmod 777 run.sh 给该脚本设置权限，该脚本会自动进行模型转换和编译运行等操作，并且在首次运行时，需要 ctrl + C 结束，然后来到 out/目录下给编译的结果main设置权限，使用命令 chmod 777 main设置权限即可。之后，重新运行run.sh脚本即可。运行方式如下: bash run.sh或者./run.sh。

Server端运行起来的状态如图12-13所示。



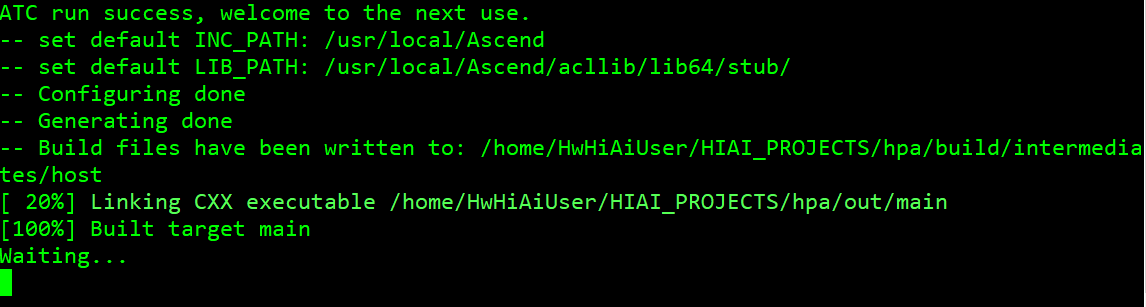


图12-13 Server端运行状态

12.4.2 Client端部署

在12.4.1节完成了对server端的部署了之后，这一节将会介绍如何在client端进行部署，并在client端进行推理以及推理结果的展示。

Client端部署所需要的所有代码在本案例提供的代码的client目录下。和server端部署类似，在部署之前，也需要进行ip地址的修改。具体的操作是修改 client目录下的client3.0.py文件中socket绑定的Ip地址。如图12-14所示。

Client端采用python3实现，运行所需要的库主要包括tkinter，PIL，numpy和socket，运行前需要确认这些库的安装。在完成依赖库的安装和ip地址的修改之后，就可以开始运行client端的程序了。运行方式：python client3.0.py。

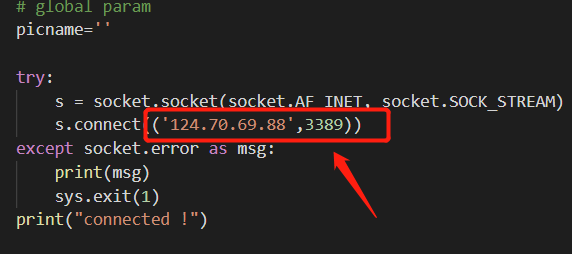


图12-14 client端修改ip

运行之后的主界面如图12-15所示。



图12-15 client端运行的主界面

之后选择图片，点击Process按钮进行推理，获得的可视化结果如图12-16所示。

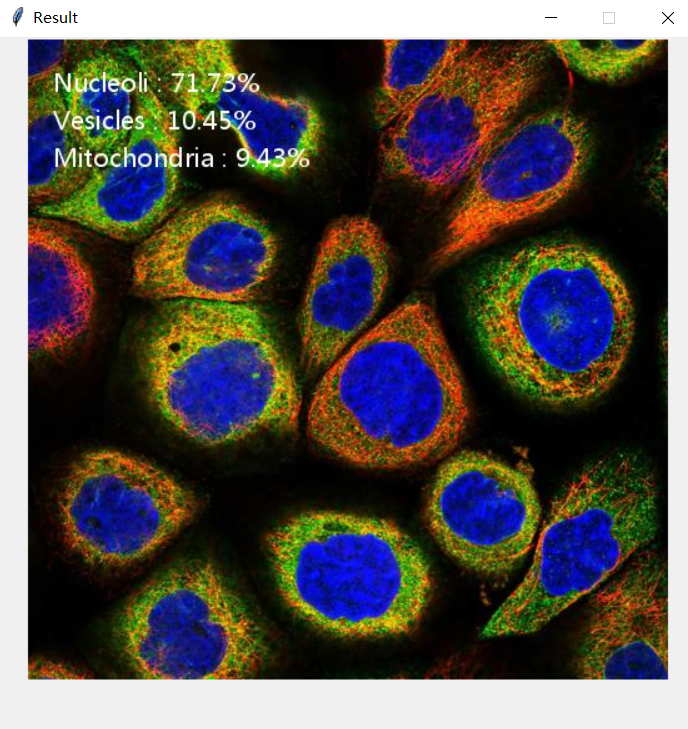


图12-16 client端预测结果的可视化结果

12.5 本章小结

本章提供了一个基于华为Atlas 300AI的人类蛋白质图谱分类案例。该案例演示了如何利用深度学习的工具开发一个人类蛋白质图谱的分类器。能够对任意输入的蛋白质图谱图片进行分类，并实时的给出分类的结果。

本章对案例系统做了详尽的剖析。阐明了整个系统功能、结构与流程是如何设计的，详细解释了如何下载筛选裁取数据、如何构建深度学习模型、如何开发分类算法、如何移植模型到Atlas 300AI端等内容。部署后的系统在Mind Studio平台和Atlas 300AI开发板上予以了测试，结果表明案例系统具有较快的推理速度和良好的分类性能。读者可以在该案例系统的基础上开发更有针对性的应用系统。