# 手写汉字拍照识别系统技术说明书 —单字版

哈尔滨工业大学

2019/9/15

## 1 项目开发目的和意义

书写汉字是几乎所有国人的最自然技能之一。日常生活中有很多场景需要对写在纸上的汉字进行识别或者录入电脑。如下图1-1是一位同事咨询该字如何发音，如果能够识别它就能达到认识它的目的。

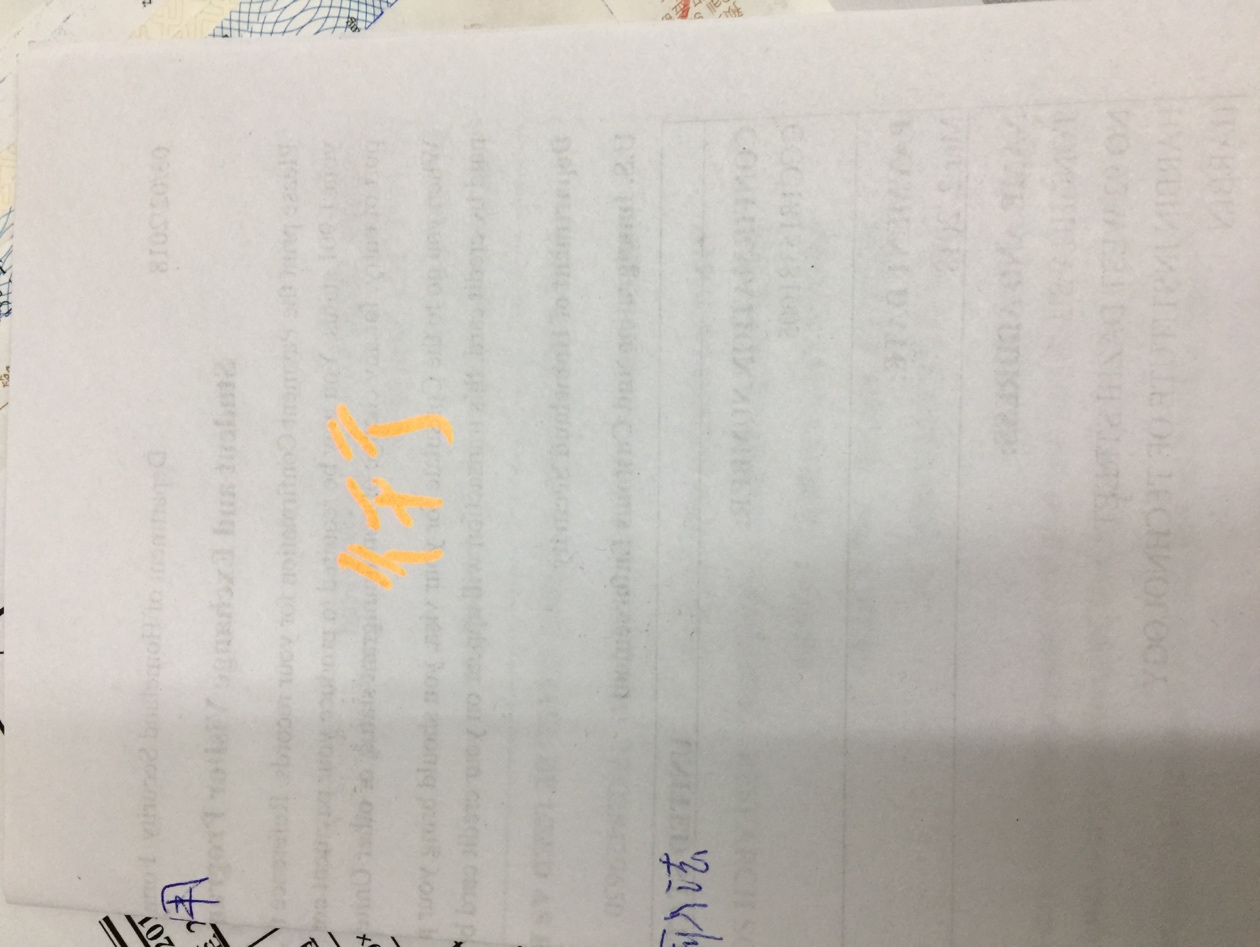


图1-1 生活中的一个字如何发音？

本项目是属于手写体文字识别应用，旨在华为Atlas200DK上实现通过拍照所书写汉字进行识别的系统。该系统能够对写在纸上的多个汉字，使用摄像头捕获视频/图像，实时检测手写文字区域并给出识别类别。该系统与用户交互的部分包括从摄像头图像采集、模型推断到输出结果的完整流程。其中模型推断部分采用的是深度神经网络，目前深度学习在文字识别方面有着广泛的应用，多分类问题是其中重要的一类。然而，深层网络模型的结构通常很复杂，对于一般的多类别分类任务，所需的深度网络参数通常随着类别数量的增加而呈现超线性增长。本项目需要识别字库中字的类别数高达3,755类，而且模型推断的速度在整个识别流程中占据较大部分，所以其速度对于用户体验至关重要。如何研究一个速度快、模型小、实用性强的方案变得极具挑战性。

本项目的单字版（对应任务书中中期任务）在华为Atlas200DK上实现了借助摄像头对少量书写汉字的实时检测和识别，具有完整性、代表性和实用性，满足了在实际场景下用摄像头进行文字的拍照感知、实时检测和识别的需求。

## 2 总体设计

系统可以划分为数据处理、模型构建和训练、文字识别等三个主要子系统，各子系统相对独立，但存在数据关联。其中数据处理包括汉字集划分、新数据集制作、图像增强等；模型构建和训练包括网络构建、模型训练等；文字识别包括单字检测和识别。为了说明各模块之间的结构关系，细化的整体结构图如2-1所示。



图2-1 系统整体功能结构图

## 3 算法设计

整个算法包括训练和推断两个阶段。训练阶段在HITHCD-2018数据子集上借助Caffe生成定制版ResNet模型。推断阶段包括摄像头图像采集、字符检测、图像预处理、文字识别等模块后给出输出反馈，具体流程见下图3-1所示。



图3-1 系统流程

### 3.1 手写数据集

使用HITHCD-2018数据集的子数据集。HITHCD-2018是哈尔滨工业大学收集的、用于手写汉字识别（HCCR）的大型数据库，有超过5346书写者书写，是目前规模最大、字类最多的数据库（**Tonghua Su et al**. HITHCD–2018: Handwritten Chinese Character Database of 21K-Category, ICDAR, 2019）。我们使用子数据集，共563,250个样本，它覆盖了3755个类别的国标第一级字符（GB2312-1980 Level 1）。其中训练数据中卫每个字符类提供了120个样本，测试集每类提供30个样本，后者可以用于超参数验证。

### 3.2 数据集制作

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ACID性等特点。由于本项目所用HITHCD数据集存储形式为gnt，我们需要对其进行格式转换，输出成一个lmdb库文件。gnt的存储形式如3-2所示，前4个字节是当前图片所占的字节数，紧跟的2个字节是图片对应标签的ASCII编码，再往后4个字节分别是宽和高，最后是图片具体信息，如此往复。



图3-2 gnt文件存储情况

生成lmdb需要准备的条件如下：1）编译好caffe并编写好convert\_imageset; 2）被转换gnt文件，注意需要为图3-2所示标准格式; 3）一个标签文件lexicon3755.txt; 4）用命令编辑好的shell脚本create-lmdb.sh。下面依次进行说明。

该部分核心代码封装在convert\_imageset程序中，主要实现gnt转lmdb。考虑到现有caffe框架仅提供jpg,png等格式转lmdb，若以此制作数据，需将gnt转为单张图片过渡，存储和I/O效率成本较高，我们对其进行了重写，其流程图如3-3所示。



图3-3 制作LMDB流程图

具体的，主要是解析gnt文件，并将其转存为lmdb格式，供后续caffe卷积神经网络网络训练使用。首先，载入gnt文件，将其保存为char\* buffer。接下来将图片名(从1到563250)和label(one-hot编码格式)存取到std::vector<std::pair<std::string, int>>中。进一步考虑是否需要打乱训练样本，若需要，则打乱vector并记录每张图片所在字节位置。记录字节需新建start数组，维度为样本数n+1，情况如图3-4所示，start[0]=0，start[1]=4906，start[2]=16136，start[3]=23202，…，start[n]=gnt所占字节数。据此根据打乱后的vector顺序，再逐个到start数组中寻址，将原有图片信息copy到新buffer指针中，如此往复，直到读到原buffer指针末尾。

解析完gnt后，需经过图像预处理(3.3节)，并且由于部署环境(atlas 200dk)仅支持彩色图像输入，因此图片应转为rgb存入lmdb。随后遍历新buffer，判断是否读到末尾。若否，则往下读取4个字节的图片长度信息，再读取2个字节的图片标签信息(以ASCII码存储)，紧接着是4个字节的图片长宽信息，最后是图片的实际内容。以图3-4图片1为例，所占字节为4906，其尺寸为68\*72，因图片总长、label、长宽占了共4+4+2=10个字节，总计4896+10=4906字节。这样循环往复，一直读到文件末尾。读取完毕后，还需要对读入信息序列化，写入为lmdb格式，即完成了整个制作流程。



图3-4 start数组保存信息示意图

lexicon3755.txt为第0-3754类汉字的label，按行保存即可，如3-5所示。

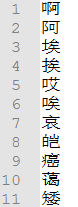


图3-5 lexicon3755.txt示意图

最后，编写shell文件create\_lmdb.sh，用来生成lmdb文件**3755\_train\_RGB/data.mdb和3755\_test\_RGB/data.mdb**，内容如下所示。其中shuffle设置是否打乱样本，默认为不打乱，resize\_height和resize\_width设置resize的图片长宽，默认保持原图大小不变，lexicon3755.txt为字典路径，hwtrn.gnt为输入gnt文件路径，./3755\_train\_RGB为输出lmdb路径。

#!/usr/bin/en sh

convert\_imageset --shuffle=true –resize\_height=112 –resize\_width=112 \

lexicon3755.txt \

hwtrn.gnt \

./3755\_train\_RGB\

Pause

### 3.3 图像预处理

图像的亮度、对比度等属性对识别的影响很大，书写的同个汉字在不同环境下也有不同。然而，在识别问题中，这些因素不应该影响最后的识别结果。为了尽可能减少无关因素的影响，我们对原始数据进行了预处理和增强，提高了网络的泛化能力。该部分流程如图3-6所示：



图3-6 数据增强流程

大律法二值化主要是利用最大类间方差，将图片分为前景和背景两部分。本项目中，它的目的是为了保持手写汉字灰度不变，将背景统一为纯白底色，增加识别的鲁棒性。调用threshold(img, img, 0, 255, THRESH\_TOZERO | THRESH\_OTSU)函数，可实现项目需求。示意图如3-7所示。

D:\1.pngD:\2.png

图3-7 大律法二值化效果图(左为原始图像，右为大律法校正图像)

灰度均衡法由Cheng-Lin Liu, Fei Yin等人在“Online and offline handwritten Chinese character recognition: Benchmarking on new databases”中提出，目的是为了尽可能使得训练样本汉字灰度值相近，提高识别的准确率。对于给定的像素值在0到255之间的汉字样本，首先进行灰度均值计算，若大于110，即样本图片更接近白色、笔画颜色偏浅，则对其进行笔画增粗、增黑，具体代码在第5章进行说明，其前后效果如图3-8所示。

图3-8 灰度均衡效果图(左为原始图像，右为灰度均衡校正图像)

对于给定的caffe网络，训练的样本需为统一尺寸。因此，在预处理过程中，还需要对汉字进行居中padding和大小归一化。该部分的主要步骤为:1.根据长宽比，将汉字resize到尽可能接近目标尺寸；2.采用邻接线性插值法，将样本padding为正方形。如图3-9所示的“知”字，我们将原有68\*72大小处理为了112\*112的标准图片，并采用cvtColor(img, img, COLOR\_GRAY2BGR)将其转为三通道，完成了整个预处理流程。

D:\2.png

图3-9 居中padding及归一化示意图(左为原始图像，右为预处理后标准图像)

### 3.4 文字检测

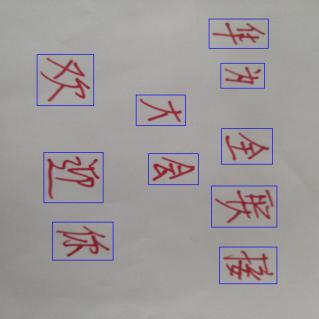
针对onCameraFrame里内容进行单字检测，整个过程见图3-10所示，关键步骤的效果见图3-11。首先考虑到摄像头中真实场景的复杂背景信息，以及opencv有限的区域提取能力，我们固定手写汉字颜色为红色，以简化轮廓提取难度。因为红色在BGR颜色空间是不连续的，将图片转为HSV颜色空间进行颜色过滤操作。具体做法为：首先接收摄像头发送的格式为YUV420SP的图片，对该图片转为BGR格式，转换后如图3-11(a)所示。



图3-10 文字检测流程

1. 原图 (b) hsv图 (c) 轮廓提取图

(d) 轮廓膨胀图 (e) 区域提取图

图 3-11 单字检测关键过程的示意图

接着使用H(170,180)， S(100,255), V(100,255)，做为颜色阈值对图像进行颜色提取，结果如图3-11(b)和3-11(c)所示。

随后使用opencv的膨胀方法对提取的文本区域进行膨胀处理，以便于更明显的区分文本区域和背景，结果如图3-11(d)所示。

接下来在膨胀后的图像上提取轮廓，并针对该轮廓求最小水平矩形。考虑到存在可能的误差区域以及一个字分成多个区域，我们使用轮廓间的相对距离（即轮廓间距离/图像对角线距离）进行是否属于同一区域的判断，具体做法是设定距离阈值，计算两两轮廓间距离除以图像对角线距离得到相对距离，该距离小于距离阈值时，属于同一区域，该距离大于等于距离阈值时，属于不同区域，最后使用交并集算法进行区域合并。然后设定面积阈值，计算合并后的每个水平矩形的面积，并除以图像面积得到相对面积，当相对面积在距离阈值区间时，判定为文字区域，否则，舍弃，最后返回我们标定的文字区域坐标范围。最后的提取区域如图3-11(c)所示。

### 3.5 文字识别

本节按照模型训练、模型转换、推断时的数据结构设计以及部署流程设计等四个方面展开。

**3.5.1 模型训练**

文字识别部分采用定制版ResNet来完成模型的学习与推断，它们的作用是用来对检测出来的文字进行分类。ResNet的网络结构在ResNet-50的基础上进行了简化，共有18层构成，如图3-12所示。其中，resnet2b模块下方绿色省略部分为pool到resnet2b的重复构建，共有5个主体卷积池化结构。



图3-12 Resnet-18网络结构图

Caffe的网络定义主要在prototxt中完成，其网络的输入定义如下:

#训练集

layer {

name: "data"

type: "Data"

top: "data"

top: "label"

include {

phase: TRAIN

}

transform\_param {

mirror: false

mean\_value: 0

mean\_value: 0

mean\_value: 0

}

data\_param {

#训练集data.mdb路径

source: "L:/vs\_projects/gnt2lmdb/gnt2lmdb/3755\_train\_RGB"

batch\_size: 64

backend: LMDB

}

}

#测试集

layer {

name: "data"

type: "Data"

top: "data"

top: "label"

include {

phase: TEST

}

transform\_param {

mirror: false

mean\_value: 0

mean\_value: 0

mean\_value: 0

}

data\_param {

#测试集data.mdb路径

source: "L:/vs\_projects/gnt2lmdb/gnt2lmdb/3755\_test\_RGB"

batch\_size: 100

backend: LMDB

}

}

输出定义如下：

layer {

bottom: "pool5"

top: "fc3755"

name: "fc3755"

type: "InnerProduct"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 1

}

inner\_product\_param {

num\_output: 3755

weight\_filler {

type: "xavier"

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

bottom: "fc3755"

bottom: "label"

name: "loss"

type: "SoftmaxWithLoss"

top: "loss"

}

网络中间层配置及完整resnet18.prototxt脚本另见项目文件。

写好深度学习网络prototxt文件后，还需在solver.prototxt中定义训练的各种参数，如下所示：

#网络模型prototxt路径

net: "resnet18.prototxt"

#测试时迭代次数

test\_iter: 1126

#每隔多少个step跑一次测试集

test\_interval: 7040

#All parameters are from the cited paper above

base\_lr: 0.001

momentum: 0.9

momentum2: 0.999

#since Adam dynamically changes the learning rate, we set the base learning

#rate to a fixed value

lr\_policy: "step"

gamma: 0.9

stepsize: 7040

display: 100

#The maximum number of iterations

max\_iter: 352000

snapshot: 7040

snapshot\_prefix: "models3755rgb/resnet\_gemfield\_cls3755\_RGB\_adam"

type: "Adam"

solver\_mode: GPU

最后，定义如下resnet.bat文件，双击即可开始训练。

caffe-master\Build\x64\Release\caffe.exe train --solver=solver2.prototxt

pause

训练过程如图3-13所示：

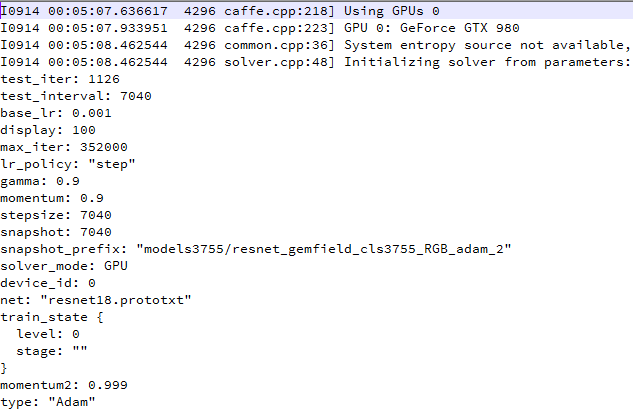


图3-13 训练过程展示

测试集验证情况如图3-14所示：

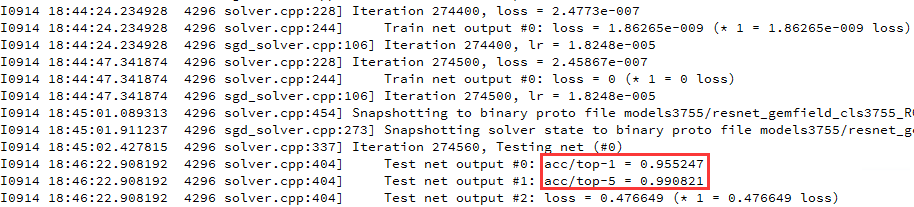


图3-14测试集top-1&top-5展示

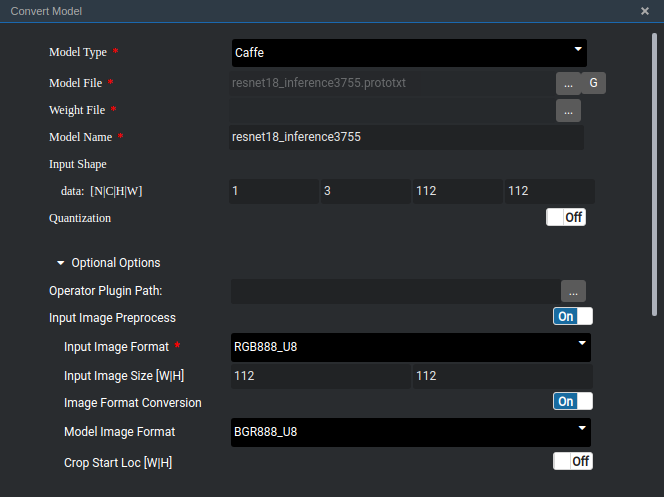
网络每隔7040个step会保存一次模型文件，最终训练出的部分文件如图3-15所示：



图3-15 网络模型展示

**3.5.2 模型转换**

要将训练好的Caffe模型应用到Atlas200DK上，首先要将其转换为Ascend 310芯片支持的离线模型，模型转换的参数如图3-13所示。Model File为记录网络结构的prototxt文件，Weight File为记录参数值的caffemodel文件推理过程的batch size需要设为1，图像的宽、高和通道数不变，因此输入的维度N、C、H、W分别对应为1、3、112、112。为了使输入图像的格式符合模型输入要求，还需在模型转换中设置图像预处理的参数，其中输入图像的格式Input Image Format需调整为RGB888\_U8， 输入的图像大小Input Image Size为112\*112，其他选项一律设为off。模型转换过程尤为重要，参数需要按照caffe模型要求来调整，否则模型的输出会与期望不符，转换之后的模型为om文件，将其添加入工程中。



3-13 模型转换时的关键配置

**3.5.3 推断时的数据结构设计**

模型部署模块数据结构设计参考了人脸检测的数据类型，汉字识别在其基础上添加了如下数据类型：

// 每个汉字的矩形框

struct CRect {

hiai::Point2D lt; // left top

hiai::Point2D rb; // right bottom

};

// 每张图片的检测与识别结果

struct ImageResults{

int num;// 每帧图像中的汉字数量

std::vector<OutputT> output\_datas;//每帧图像中汉字输出向量集合

std::vector<CRect> rects;//每帧图像中汉字的矩形框集合

};

// 多**帧**图像检测与识别结果集合

struct CEngineTransT{

bool status;

std::string msg; // error message

hiai::BatchInfo b\_info;

std::vector<NewImageParaT> imgs;//每帧图像集合

std::vector<ImageResults> results;//每帧识别结果集合

};

**3.5.4 部署流程设计**

根据汉字检测与识别的需求，共设计了三个Engine，分别为摄像头模块、推理模块、后处理模块，部署流程图如图3-14所示。

摄像头模块与Camera驱动进行交互，设置摄像头的帧率、图像分辨率、图像格式等相关参数，从摄像头中获取YUV420SP格式的视频数据，每一帧传给推理引擎进行计算。以此工程为例，其中帧率fps为5，图像分辨率取1280x720，摄像头图像格式为默认的YUV420SP。

推理模块接收摄像头数据，对YUV420SP格式的每帧图像进行以下两方面的处理：一方面将其转为RGB格式的图像，使用OpenCV对图像进行处理，检测出汉字的矩形框集合，接下来依次对每个汉字子图像通过模型进行推理，得到输出向量的结果集合；另一方面还需将每帧图像转换JPEG格式，以便于查看摄像头图像。将JPEG格式的每帧图像集合和每帧识别结果集合作为输入传给后处理引擎模块。

后处理模块接收上一个引擎的推理结果与摄像头JPEG图像，将矩形框集合添加到Presenter Server记录检测目标位置信息的数据结构DetectionResult类中，作为摄像头图像的检测结果，通过调用Presenter Agent的API发送到UI Host上部署的Presenter Server服务进程。Presenter Server根据接收到的推理结果，求出汉字最大预测概率值所对应的索引，在索引表中查找对应汉字，在JPEG图像上进行汉字矩形框位置及汉字识别结果的标记，并将图像信息发送给Web UI。

索引表为一个记录汉字与其对应索引值的表，为txt文件，在Ubuntu系统下以UTF-8的格式存储，其中每一行对应一个汉字。



3-14 部署流程图

## 4 结果

我们首先在HITHCD-2018数据子集上实验。图4-1是ResNet在3755类手写汉字数据下的训练过程和验证过程。表4-1为 ResNet网络训练3755类手写汉字的关键网络参数表和测试结果。

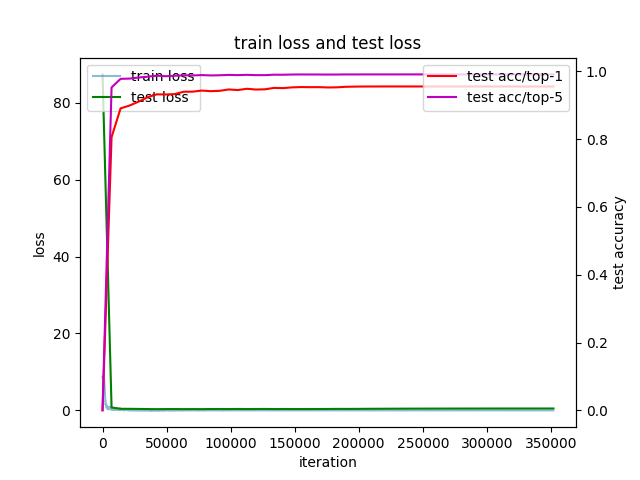


图4-1 ResNet在3755类手写汉字训练集下的accuracy和loss

表4-1 ResNet网络在3755类手写汉字识别任务中的网络参数和测试结果

|  |  |
| --- | --- |
| **参数** | **具体值** |
| image\_size | 112\*112\*3 |
| decay\_steps | 7040 |
| eval\_steps | 7040 |
| learning\_rate | 1e-3 |
| lr\_policy | step |
| gamma | 0.9 |
| stepsize | 7040 |
| epoch | 50 |
| batch\_size | 64 |
| 优化器 | Adam |
| Top-1 accuracy | **95.5247%** |
| Top-5 accuracy | **99.0821%** |

然后针对拍照识别的两种实际场景，进行了测试。第一种场景的硬件布局图见图4-2所示，第二种场景的布局如图4-3。第一种场景多见于感知外界已有（已经张贴好的）手写文字，第二种场景可以用于实时书写实时识别的场合。



图4-2 场景1：摄像头的平视布局

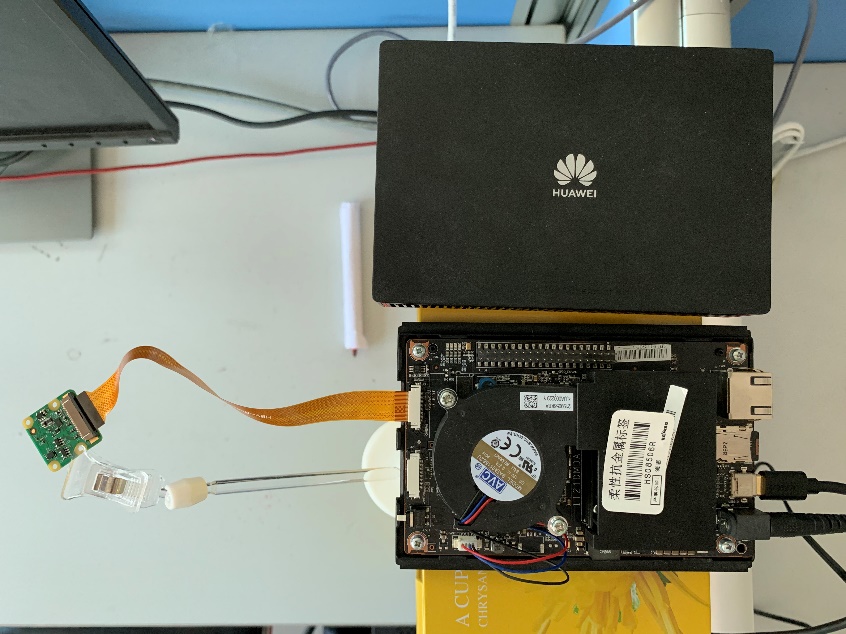


图4-3 场景2：摄像头的俯视布局

我们测试了两种场景，给出典型的效果图。对于场景1的识别效果如图4-4所示，对于场景2样例如图4-5所示。

最后，我们测算了系统的主要时间消耗情况。一帧图片的整图字符检测约60毫秒，识别阶段每个字的平均识别时间约为3毫秒。在光线稳定的情况下，单字识别准确率90%以上，符合项目要求。

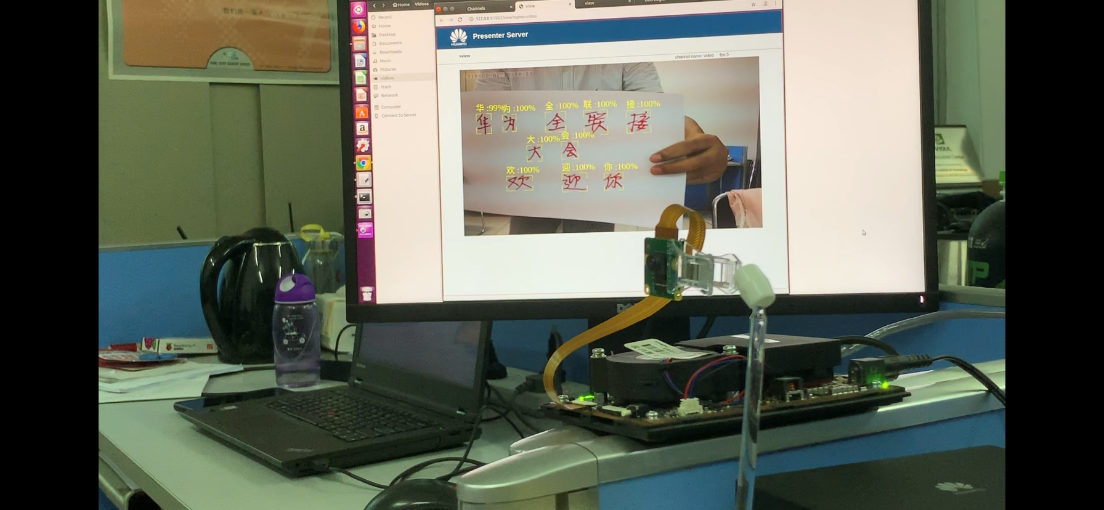
 

图4-4 场景一通过摄像头感知拿在手中的手写文字

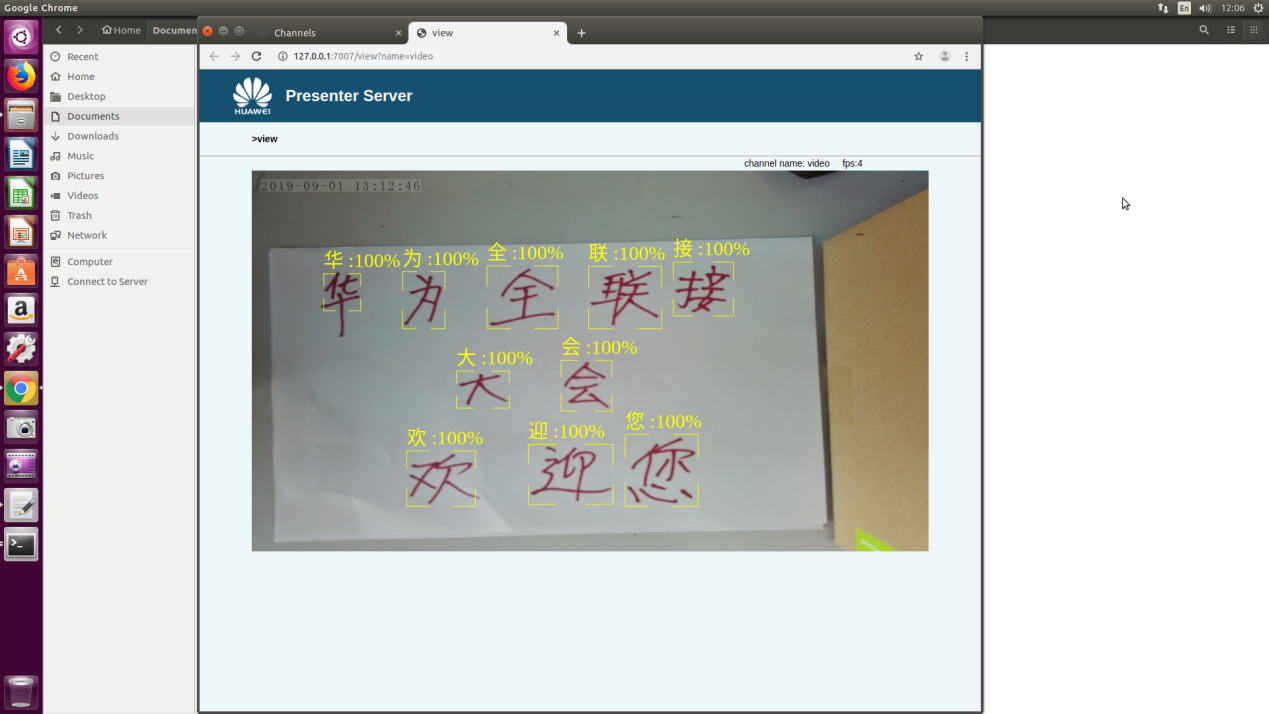
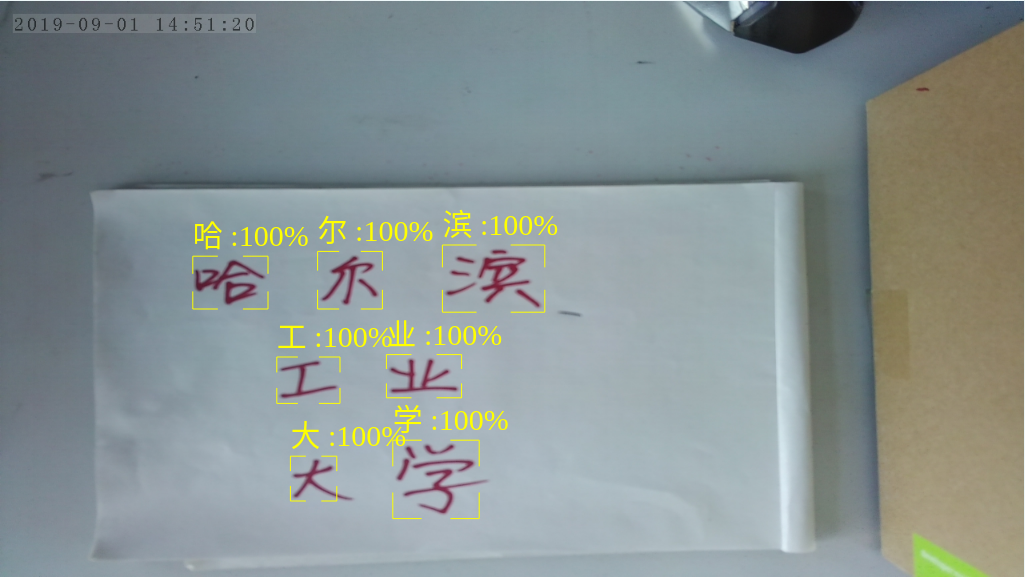
 

图4-5 场景二通过摄像头感知书桌上正在书写的文字

## 5 关键代码说明

### 5.1 gnt转lmdb核心代码(含图像处理)

if (argc < 4) {

//gflags::ShowUsageWithFlagsRestrict(argv[0], "gnt2lmdb");

argv[1] = "./lexicon3755.txt";//字典路径

argv[2] = "./hwtst.gnt";//gnt路径

argv[3] = "./3755\_test\_RGB2/";//输出路径

}

FILE \*fpIn;//定义文件句柄

char fname[30];

strcpy(fname, argv[2]);

fopen\_s(&fpIn, fname, "rb");

if (fpIn == 0)

{

printf("Cannot open the file %s\n", fname);

exit(1);

}

unsigned char\* buff;

long long fpLen;

\_fseeki64(fpIn, 0, SEEK\_END);

fpLen = \_ftelli64(fpIn);

\_fseeki64(fpIn, 0, SEEK\_SET);

long long gntlen;

long count = 0;

std::vector<std::pair<std::string, int> > lines;

std::ifstream infile(argv[1]);

std::string line;

string \*line\_ch = new string[3755];

size\_t pos;

int label;

int line\_count = 0;

while (std::getline(infile, line)) {

//cout << line << endl;

line\_ch[line\_count] = line;

line\_count++;

}

while (\_ftelli64(fpIn) < fpLen){

int temp = sizeof(unsigned long int);

fread(&gntlen, temp, 1, fpIn);

buff = new unsigned char[gntlen];

\_fseeki64(fpIn, -temp, SEEK\_CUR);

fread(buff, gntlen, 1, fpIn);

//int len\_all, len\_label, len\_height, len\_width;

int len\_all, len\_label;

len\_label = 2 \* sizeof(char);

len\_all = sizeof(unsigned long int);

//len\_height = sizeof(unsigned short int);

//len\_width = sizeof(unsigned short int);

short width, height;

char tagcode[3];

tagcode[2] = '\0';

//memcpy(&gntlen, buff, len\_all);

memcpy(tagcode, buff + len\_all, len\_label);

//memcpy(&width, buff + len\_label + len\_all, len\_width);

//memcpy(&height, buff + len\_label + len\_all + len\_width, len\_height);

//cout << tagcode << endl;

int position;

for (int l = 0; l < 3755; l++){

position = line\_ch[l].find(tagcode);

if (position != line\_ch[l].npos){

position = l;

break;

}

}

//cout << position << endl;

//cout << tagcode << endl;

delete buff;

//cout << to\_string(count) + ".png" << endl;

lines.push\_back(std::make\_pair(to\_string(count) + ".png", position));

count++;

}

// randomly shuffle data

if (FLAGS\_shuffle == TRUE){

cout << "Shuffling data" << endl;

shuffle(lines.begin(), lines.end());

cout << "Shuffling finished" << endl;

}

cout << "A total of " << lines.size() << " images." << endl;

if (fpIn == 0)

{

printf("Cannot open the file %s\n", fname);

exit(1);

}\*/

\_fseeki64(fpIn, 0, SEEK\_END);

fpLen = \_ftelli64(fpIn);

cout << "len:" << fpLen << endl;

\_fseeki64(fpIn, 0, SEEK\_SET);

long count\_flag = 0;

long long \*start = new long long[lines.size() + 1];

cout << "linesize:" << lines.size() << endl;

long long sum\_len = 0;

for (int m = 0; m < (lines.size() + 1); m++) start[m] = -1;

gntlen = 0;

cout << "Loading start array..." << endl;

int temp = sizeof(unsigned long int);

//记录start数组

while (\_ftelli64(fpIn) < fpLen)

{

fread(&gntlen, temp, 1, fpIn);

buff = new unsigned char[gntlen];

\_fseeki64(fpIn, -temp, SEEK\_CUR);

fread(buff, gntlen, 1, fpIn);

start[count\_flag] = sum\_len;

sum\_len += gntlen;

count\_flag++;

delete buff;

}

start[lines.size()] = fpLen;

\_fseeki64(fpIn, 0, SEEK\_SET);

//根据start数组和打乱后的vector，copy建立新buffer

cout << "Start finished" << endl;

cout << "Shuffling buffer..." << endl;

cout << fpLen << endl;

char \*new\_gnt = new char[fpLen];

cout << "\*\*\*" << endl;

//cout << fpLen << endl;

long long flag\_len = 0;

for (long m = 0; m < lines.size(); m++){

string cut = lines[m].first;

int fin = cut.find\_first\_of(".");

int n = atoi(cut.substr(0, fin).c\_str());

\_fseeki64(fpIn, start[n], SEEK\_SET);//移动到该图片所在指针

fread(new\_gnt + flag\_len, start[n + 1] - start[n], 1, fpIn);

flag\_len += start[n + 1] - start[n];

if (flag\_len < 0) {

cout << "n:" << n << endl;

cout << flag\_len << endl;

cout << "start[n]:" << start[n] << endl;

cout << "start[n+1]:" << start[n + 1] << endl;

return 0;

}

if (m % 10000 == 0) cout << m << endl;

}

cout << "Shuffling buffer finished" << endl;

//long gntlen;//记录单张图指针长度

count = 0;//记录第几幅图

// Create new DB

scoped\_ptr<db::DB> db(db::GetDB("lmdb"));

db->Open(argv[3], db::NEW);

scoped\_ptr<db::Transaction> txn(db->NewTransaction());

long long pos\_ptr = 0;//记录指针位置

Datum datum;

while (pos\_ptr<fpLen)//遍历new\_gnt

{

memcpy(&gntlen, new\_gnt + pos\_ptr, temp);

buff = new unsigned char[gntlen];

\_fseeki64(fpIn, -temp, SEEK\_CUR);

memcpy(buff, new\_gnt + pos\_ptr, gntlen);

pos\_ptr += gntlen;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*grayNorm\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

short height, width;

int len\_all, len\_label, len\_height, len\_width;

len\_label = 2 \* sizeof(char);

len\_all = sizeof(unsigned long int);

len\_height = sizeof(unsigned short int);

len\_width = sizeof(unsigned short int);

char tagcode[3];

tagcode[2] = '\0';

memcpy(tagcode, buff + len\_all, len\_label);

memcpy(&width, buff + len\_label + len\_all, len\_height);

memcpy(&height, buff + len\_label + len\_all + len\_height, len\_width);

Mat img(height, width, CV\_8UC1, buff + len\_label + len\_all + len\_height + len\_width);

img = 255 - img;

threshold(img, img, 0, 255, THRESH\_TOZERO | THRESH\_OTSU);

img = 255 - img;

float x = 0.0;

if ((height > width) & (height < 224)){

x = 224.0 / height;

cv::resize(img, img, cv::Size(int(width \* x), 224), 0, 0, cv::INTER\_LINEAR);

int padding\_left = int((224 - int(width \* x)) / 2);

int padding\_right = 224 - padding\_left - int(width \* x);

copyMakeBorder(img, img, 0, 0, padding\_left, padding\_right, BORDER\_CONSTANT, Scalar(255, 255, 255));

}

else{

x = 224.0 / width;

cv::resize(img, img, cv::Size(224, (224, int(height \* x))), 0, 0, cv::INTER\_LINEAR);

int padding\_top = int((224 - int(height \* x)) / 2);

int padding\_bottom = 224 - padding\_top - int(height \* x);

copyMakeBorder(img, img, padding\_top, padding\_bottom, 0, 0, BORDER\_CONSTANT, Scalar(255, 255, 255));

}

cv::resize(img, img, cv::Size(112, 112), 0, 0, cv::INTER\_LINEAR);

bitwise\_not(img, img);

float meanValue = 0.0f;

float stdValue = 0.0f;

int cnt = 0;

for (size\_t i = 0; i < img.cols; i++)

{

for (size\_t j = 0; j < img.rows; j++)

if (img.at<uchar>(j, i) > 0)

{

cnt++;

meanValue += img.at<uchar>(j, i);

}

}

meanValue /= (cnt + 0.1f);

if (meanValue < 110)

{

for (size\_t i = 0; i < img.cols; i++)

{

for (size\_t j = 0; j < img.rows; j++)

if (img.at<uchar>(j, i) > 0)

{

stdValue += (img.at<uchar>(j, i) - meanValue)\*(img.at<uchar>(j, i) - meanValue);

}

}

GaussianBlur(img, img, Size(3, 3), 0, 0);

stdValue /= (cnt + 0.1f);

stdValue = sqrt(stdValue);

float power = log(185.0f / (185.0f + 40.0f)) / log(meanValue / (meanValue + 2 \* stdValue));// method from Prof Liu

float alpha = 185.0f / pow(meanValue, power);// method from Prof Liu

for (size\_t i = 0; i < img.cols; i++)

{

for (size\_t j = 0; j < img.rows; j++)

{

int grayValue = img.at<uchar>(j, i);

if (grayValue > 0)

{

img.at<uchar>(j, i) = min(255.0, alpha\*pow(grayValue, power));// method from Prof Liu

}

}

}

}

bitwise\_not(img, img);

if (FLAGS\_gray == FALSE)

cvtColor(img, img, COLOR\_GRAY2BGR);

CVMatToDatum(img, &datum);

datum.set\_label(lines[count].second);

img.release();

// Put in db

string out;

CHECK(datum.SerializeToString(&out));

string key\_str = caffe::format\_int(count, 8) + "\_" + lines[count].first;

txn->Put(key\_str, out);

//每10000个样本序列化写入一次

if (++count % 10000 == 0) {

// Commit db

txn->Commit();

txn.reset(db->NewTransaction());

cout << "Processed " << count << " files." << endl;;

}

delete buff;

}

if (count % 10000 != 0) {

txn->Commit();

cout << "Processed " << count << " files.";

}

### 5.2文本区域检测关键代码说明

std::vector<cv::Rect> detect(cv::Mat img)

{

std::vector<cv::Rect> result\_rects;

float benchmark\_len = img.rows;

float bench\_area = img.rows\*img.cols;

std::vector<cv::Rect> result\_rects;

cv::Mat hsv;

（1）采用opencv方法转换色彩空间

cv::cvtColor(img, hsv, cv::COLOR\_BGR2HSV);

cv::Mat mask;

（2）设定提取颜色阈值，这里提取红色区间

cv::inRange(hsv, cv::Scalar(156, 100, 100), cv::Scalar(180, 255, 255), mask);

（3）设定膨胀内核，膨胀形状为矩形，多次测试，设定内核大小为15\*15较为合适

cv::Mat element1 = cv::getStructuringElement(cv::MORPH\_RECT, cv::Size(15, 15));

cv::Mat dilate1;

cv::dilate(mask, dilate1, element1);

std::vector<cv::Rect> rects;

（4）在膨胀后的图像上提取轮廓，并对轮廓求最小水平矩形

std::vector<std::vector<cv::Point> > contours;

std::vector<cv::Vec4i> hierarchy;

std::vector <cv::Point> center\_point;

cv::findContours(dilate1, contours, hierarchy, cv::RETR\_EXTERNAL, cv::CHAIN\_APPROX\_SIMPLE);

for (int i = 0; i < contours.size(); i++)

{

cv::Rect rect = cv::boundingRect(contours[i]);

rects.push\_back(rect);

}

1. 使用相对距离判断是否属于同一文本区域

std::vector<std::vector<int> > dist\_list(rects.size());

for (int i = 0; i < rects.size(); i++)

{

dist\_list[i].push\_back(i);

for (int j = i + 1; j < rects.size(); j++)

{

Float two\_dist = pow((rects[i].x + 0.5\*rects[i].width) - (rects[j].x

+ 0.5\*rects[j].width), 2)+ pow((rects[i].y + 0.5\*rects[i].height) - (rects[j].y

+ 0.5\*rects[j].height), 2);

Float w\_scale = two\_dist / benchmark\_len;

if (w\_scale < 1)

{

dist\_list[i].push\_back(j);

}

}

1. 利用交并集算法提取同一区域，并计算最大包含矩形框作为新的文本区域

std::vector <int> dist\_result(dist\_list.size(), -1);

std::vector<int> dist;

for (int m = 0; m < dist\_list.size(); m++)

{

dist = dist\_list[m];

for (int j = 0; j < dist.size(); j++)

{

if (dist\_result[m] == -1)

{

dist\_result[m] = dist[j];

}

else

{

dist\_result[dist[j]] = dist\_result[m];

}

}

}

std::vector<std::vector<int> > un\_list(dist\_result.size());

std::vector<int> un\_dist;

std::vector<cv::Rect> inte\_area;

for (int i = 0; i < dist\_result.size(); i++)

{

un\_list[dist\_result[i]].push\_back(i);

}

for (int m = 0; m < un\_list.size(); m++)

{

if (un\_list[m].empty())

{

continue;

}

un\_dist = un\_list[m];

int xmin = rects[un\_dist[0]].x;

int ymin = rects[un\_dist[0]].y;

int xmax = rects[un\_dist[0]].x + rects[un\_dist[0]].width;

int ymax = rects[un\_dist[0]].y + rects[un\_dist[0]].height;

for (int j = 1; j < un\_dist.size(); j++)

{

if (xmin > rects[un\_dist[j]].x)

{

xmin = rects[un\_dist[j]].x;

}

if (ymin > rects[un\_dist[j]].y)

{

ymin = rects[un\_dist[j]].y;

}

if (xmax < rects[un\_dist[j]].x + rects[un\_dist[j]].width)

{

xmax = rects[un\_dist[j]].x + rects[un\_dist[j]].width;

}

if (ymax < rects[un\_dist[j]].y + rects[un\_dist[j]].height)

{

ymax = rects[un\_dist[j]].y + rects[un\_dist[j]].height;

}

}

double area = (xmax - xmin)\*(ymax - ymin) / bench\_area;

if (area>0.0015 && area<0.03)

{

cv::Rect roi = cv::Rect(xmin, ymin, xmax - xmin, ymax - ymin);

result\_rects.push\_back(roi);

}

### 5.3 推断阶段核心代码

HIAI\_IMPL\_ENGINE\_PROCESS("face\_detection\_inference",

FaceDetectionInference, INPUT\_SIZE) {

HIAI\_ENGINE\_LOG("Start process!");

if (arg0 == nullptr) {

return HIAI\_ERROR;

}

std::shared\_ptr<BatchImageParaWithScaleT> image\_handle =

std::static\_pointer\_cast<BatchImageParaWithScaleT>(arg0);

std::shared\_ptr<CEngineTransT> ctrans\_data = std::make\_shared<CEngineTransT>();

ctrans\_data->b\_info = image\_handle->b\_info;

//汉字提取

vector<ImageResults> b\_results;

for (uint32\_t k = 0; k < image\_handle->b\_info.batch\_size; k++) {

ImageData<u\_int8\_t> img = image\_handle->v\_img[k].img;

int img\_height = img.height;

int img\_width = img.width;

//将yuv格式的图像用opencv来表示

Mat src(img\_height \* 3 / 2,

img\_width, CV\_8UC1);

int copy\_size = img\_width \* img\_height \* 3 / 2;

int destination\_size = src.cols \* src.rows \* src.elemSize();

int ret = memcpy\_s(src.data, destination\_size, img.data.get(),

copy\_size);

//将yuv转为bgr

Mat dst\_temp;

cvtColor(src, dst\_temp, COLOR\_YUV420sp2BGR);

dst\_temp.convertTo(dst\_temp,CV\_32FC3);

//调用图像检测函数，获得汉字矩形框集合

std::vector<cv::Rect> rects = detect(dst\_temp);

ImageResults results;

results.num = 0;

for(int j=0; j<rects.size(); j++)

{

cv::Mat roi = dst\_temp(rects[j]);

//调用单字预处理函数

roi = PreProcess(roi);

cv::resize(roi,roi,Size(112,112));

uint32\_t size2 = roi.total() \* roi.channels();

u\_int8\_t \*image\_buf\_ptr = new (std::nothrow) u\_int8\_t[size2];

memcpy\_s(image\_buf\_ptr, size2, roi.ptr<u\_int8\_t>(),

roi.total() \* roi.channels());

ImageData<u\_int8\_t> temp\_img;

temp\_img.size = size2;

temp\_img.data.reset(image\_buf\_ptr,std::default\_delete<u\_int8\_t[]>());

//对图像数据进行推理

uint32\_t input\_size = temp\_img.size \* sizeof(uint8\_t);

std::shared\_ptr<uint8\_t> temp = std::shared\_ptr<uint8\_t>(

new uint8\_t[input\_size], std::default\_delete<uint8\_t[]>());

uint32\_t each\_size = temp\_img.size \* sizeof(uint8\_t);

errno\_t mem\_ret = memcpy\_s(temp.get(),

input\_size,

temp\_img.data.get(),

each\_size);

std::shared\_ptr<hiai::AINeuralNetworkBuffer> neural\_buf = std::shared\_ptr<

hiai::AINeuralNetworkBuffer>(

new hiai::AINeuralNetworkBuffer(),

std::default\_delete<hiai::AINeuralNetworkBuffer>());

neural\_buf->SetBuffer((void\*) temp.get(), input\_size);

std::shared\_ptr<hiai::IAITensor> input\_data = std::static\_pointer\_cast<

hiai::IAITensor>(neural\_buf);

std::vector<std::shared\_ptr<hiai::IAITensor>> input\_data\_vec;

input\_data\_vec.push\_back(input\_data);

hiai::AIContext ai\_context;

std::vector<std::shared\_ptr<hiai::IAITensor>> output\_data\_vector;

hiai::AIStatus ret = ai\_model\_manager\_->CreateOutputTensor(

input\_data\_vec, output\_data\_vector);

ret = ai\_model\_manager\_->Process(ai\_context, input\_data\_vec,

output\_data\_vector,

AI\_MODEL\_PROCESS\_TIMEOUT);

ctrans\_data->status = true;

std::shared\_ptr<hiai::AISimpleTensor> result\_tensor =

std::static\_pointer\_cast<hiai::AISimpleTensor>(output\_data\_vector[0]);

OutputT out;

out.size = result\_tensor->GetSize();

out.data = std::shared\_ptr<uint8\_t>(new uint8\_t[out.size],

std::default\_delete<uint8\_t[]>());

mem\_ret = memcpy\_s(out.data.get(), out.size,

result\_tensor->GetBuffer(),

result\_tensor->GetSize());

// a.OutputT

results.output\_datas.push\_back(out);

// b.CRect

CRect rect;

rect.lt.x = rects[j].x;

rect.lt.y = rects[j].y;

rect.rb.x = rects[j].x+rects[j].width;

rect.rb.y = rects[j].y+rects[j].height;

results.rects.push\_back(rect);

// c.num

results.num++;

}

b\_results.push\_back(results);

}//batch end

// 将yuv图像转为jpeg格式

for (uint32\_t index = 0; index < image\_handle->b\_info.batch\_size; index++){

HIAI\_StatusT convert\_ret = ConvertImage(image\_handle->v\_img[index]);

}

ctrans\_data->imgs = image\_handle->v\_img;

ctrans\_data->results = b\_results;

// 发送数据至下一个引擎

HIAI\_StatusT hiai\_ret = SendData(kSendDataPort, "CEngineTransT",

std::static\_pointer\_cast<void>(ctrans\_data));

return hiai\_ret;

}

## 6 重要问题及解决

问题1：caffe环境配置提示无法打开libcaffe.lib。

解决方法：提示此问题是因为编译未成功，应检查前面其他文件，可忽略此error。

问题2：glog依赖glag动态库出错：glog.targets error MSB4062: 未能从程序集 …\gflags.2.1.2.1\...\coapp.NuGetNativeMSBuildTasks.dll 加载任务“NuGetPackageOverlay”。

解决方法：可以在管理NuGet程序包中删除glog，再重装解决。

问题3：OpenCV找不到动态库：\\private\coapp.NuGetNativeMSBuildTasks.dll未能加载任务“NuGetPackageOverlay”。

解决方法：在路径NugetPackages/OpenCV.2.4.1x/build/native/中找到OpenCV.props第五行，去掉反斜杠。

问题4：提示无python的debug库。

解决方法：安装python35，把路径加到工程，并调整项目为release版本。

问题5：运行exe文件闪退，且不报错。

解决方法：检查依赖文件是否正确，x64系统应使用64位dll、lib文件。

问题6：已安装安装opencv，但仍提示opencv相关的error。

解决方法：将opencv\_core2411.dll，opencv\_highgui2411.dll，opencv\_imgproc2411.dll添加到项目依赖文件中。

问题7：打乱样本时程序中途退出。

解决方法：检查指针访问是否越界，以及start数组记载的字节地址是否与buffer指针对应。

问题8：94类测试时收敛迅速，扩展到3755类数据集上训练收敛慢。

解决方法：打乱训练样本。否则类别较大时，会遗忘前面训练的样本，导致部分样本对网络调整贡献过小。

问题9：atlas200DK配置usb ip地址后，每次开机都需要重新配置。

解决方案：其实电脑已经配置完成，在每次开机后执行网络重启，刷新网络即可。

问题10：ubuntu电脑连接开发板后不能上网问题。

解决方案：因为是实验室电脑，IP为192.168.1.\*，与开发板USB端口IP存在冲突，修改开发板USB端口IP为192.168.2.2，同时在ubuntu电脑中配置相同网段的USB端口虚拟IP，问题解决。

问题11：摄像头实时图像背景复杂，opencv针对文本区域提取困难。

解决方案：固定检测文本为红色，使用HSV颜色空间对红色区域进行提取，并依靠轮廓间的相对距离合并矩形框，依靠每个轮廓的相对面积，排除不合格的矩形框，最终得到理想区域。

问题12：图像从在YUV420SP转变为JPG后，使用opencv处理，出现乱码，结果图像不正确。

解决方案：图像在处理过程中，必须转换成浮点数格式，即CV\_32FC3，而如果要保存中间结果，则必须转换为整数型，即CV\_8UC3。

问题13：如何用opencv的Mat表示YUV420SP图像

解决方案：用Mat对象来表示YUV420SP格式的图像，高度为图像的的3/2倍，数据类型为CV\_8UC1。

问题14：配置开发板环境有哪些注意事项

解决方案：

1. ubuntu主机与开发板系统版本保持一致，均为16.04.3。
2. Ubuntu主机要求安装64位谷歌浏览器，且版本不低于67.0.3396。
3. Mindstudio的ip应设置为主机ip。
4. 每次开关机最运行stop.sh命令，关闭 mindstudio程序，避免下次使用出现数据库冲突等问题。
5. 首次启动或者升级atlas200DK开发者板时不能断电，以免对开发板造成损坏，再次上电与上次上电保持2s以上安全时间间隔。
6. 若ubuntu主机使用SSH登录开发板提示无信任关系，在ubuntu主机端执行以下命令回复信任关系sh-keygen -f "$HOME/.ssh/known\_hosts" -R *192.168.1.2。*
7. 若更换开发者板或者SD卡，或重置系统，则容易出现信任关系过期问题，无法登陆开发板，输入以下命令删除历史信任关系即可：ssh-keygen -R 192.168.1.2。
8. 对配置环境过程中以及随后登录过程中设置和修改的密码做记录，备份，防止忘记。

## 7 后续可扩展性

本项目中期任务聚焦于少量手写汉字的识别任务。后续可以扩展到包含复杂背景的大量手写汉字识别场景，比如手写作文文字的检测与识别等任务。