手指互动相关工作整理

1. 数据
   1. 数据来源

目前手指检测的数据来源有两个：公司自己采集和数据公司采集。

* 1. 数据标注

公司自己采集的数据由公司外包团队标注；数据公司采集数据由数据公司标注。公司自己采集数据标注完后通过硬盘直接拷贝到数据存放目录；外包公司标注完成的数据通过百度网盘获取。

标注要求：https://yiqixie.com/d/home/fcABe2RZOpl8QFFdaKynqWiDT。

标注需要注意事项：

1. 标注框要基本贴合手和指尖；
2. 若图像中有多个符合要求的手及指尖，每个手和指尖都需要做标注；
3. 图像中存在过于模糊导致无法准确定位的手或者指尖时，整张图均不做标注；
   1. 数据存放

标注完成的数据存放地址：/mnt/cv-samba/dataset/finger/。

数据转化record格式存放地址：

/mnt/cv-samba/dataset/finger\_record/ 【最终的数据存放】

10.0.12.147:/home/dev/fingerData/zh/data\_record

10.0.12.148:/home/alg/fingerData/data\_record【数据比较全，但包含很多标注不全或者未删除重复数据的record】

数据整理完成后，信息跟新：http://gitlab.ling.ai:10080/pbr/cv-tasks/issues/190。

数据处理脚本Gitlab地址：http://gitlab.ling.ai:10080/zhenghui/finger\_tools/tree/master/1\_data\_process

create\_filelist.py: 生成图片列表文件，参数为1-23，分别表示不同的数据集

dataset\_check.py: 检查标注文件是否存在（不存在的生成一个无Object的xml），检查标注是否正确，e.g：python dataset\_check.py [img\_path\_dir] [dataset\_list\_file]

dataset\_infor.py: 生成文件信息，包含图片数和标注的图片数，参数为1-23，表示不同数据集

create\_finger\_tf\_record.py: 生成record文件，需要在代码中修改待生成的文件列表（dataset = 'neg\_2'），e.g：python create\_finger\_tf\_record.py –data\_dir=/mnt/cv-module/dataset/finger --label\_map\_path=label\_map.pbtxt

* 1. 数据清洗

鉴于目前标注数据会存在错误（如类型名字/位置精确度等），要求对标注的数据进行检查清洗。数据清洗可以分为自动清洗和手动筛查：

自动清洗：通过脚本实现规则化的信息筛查，比如类型名称是否正确，位置信息是否合理等；

手动筛查：随机挑选数据，利用标注软件[[1]](#footnote-2)，人工确认标注结果是否满足要求。

数据清洗脚本：http://gitlab.ling.ai:10080/zhenghui/finger\_tools/blob/master/1\_data\_process/dataset\_check.py

e.g：python dataset\_check.py [img\_path\_dir] [dataset\_list\_file]

会生成一个txt文件，列出出错的的文件名（如：error\_trainval\_dataset\_5.txt）。

1. 模型训练和转换
   1. 模型训练

手和指尖模型训练采用基于tensorflow的深度学习的SSD检测算法，基础网络目前采用的mobilenet1.0/2.0、inceptionV2。

* + 1. 训练设备

1. 设备1：gpu@10.0.5.7

训练代码位置：/home/gpu/zh/finger\_tensorflow/models/research/，仅支持mobilenet1.0。

1. 设备2：dev@10.0.12.147

docker镜像：tensorflow\_object\_detection

可以利用docker开展相关训练，共享的数据位置/home/dev/fingerData。

训练代码[[2]](#footnote-3)位置：/workspace/models/research (for mobile1.0. or /work/models/research for mobilenet2.0)，支持mobilenet1.0/2.0。

1. 设备3：alg@10.0.12.148

docker 镜像：tf\_research:1.12.0-gpu

工作中的docker 容器：90113f76804c

训练代码[[3]](#footnote-4)位置：/work/fingerData/tf\_models/models/research。

训练脚本位置：/work/fingerData/train/

表 1 训练模型docker创建

| 基础模型 | 命令 |
| --- | --- |
| 创建Docker+  mobilenet1.0 | sudo nvidia-docker run -it -p [portNum1]:8888 -p [portNum2]:6006 -v /home/dev/fingerData:/workspace/fingerData –name [my\_name] tensorflow\_object\_detection |
| 创建Docker+  mobilenet2.0 | sudo nvidia-docker run -it -p [portNum1]:8888 -p [portNum2]:6006 -v /home/dev/fingerData:/work/fingerData –name [my\_name] tf\_mobilev2:1.7.0-gpu-v2 |
| 进入容器 | sudo docker exec -it [containerId] /bin/bash |

其中，[portNum1]，[portNum2]，[my\_name]由人工指定，不同容器应不同。可以在本地利用浏览器登录http://10.0.12.147:portNum1进入notebook，方便可视化调试（不能设置代理，第一次登录需输入token，在运行nvidia-docker run会输出）；登录http://10.0.12.147:portNum2，利用tensorboard看训练状态。[containerId]为容器ID。

1. 美国x8

可以利用docker开展相关训练，共享的数据位置/home/dev/fingerData。

训练代码位置：/work/finger\_detect/models/research。

表 2 训练模型docker创建（X8）

| 基础模型 | 命令 |
| --- | --- |
| 创建Docker+  mobilenet1.0 | nvidia-docker run -it tf\_mobilev1:1.3.0-gpu |
| 创建Docker+  mobilenet2.0 | nvidia-docker run -it tf\_mobilev2:1.7.0-cpu  暂未实现GPU版本（需要实现GPU驱动升级） |

* + 1. 训练脚本

训练脚本Gitlab地址：http://gitlab.ling.ai:10080/zhenghui/finger\_tools/tree/master/2\_train

代码支持Mobilenet2.0时，需要对配置文件进行修改：

1. 配置文件中feature\_extractor的min\_depth改为0 （原为16）；
2. [CodePath]/slim/nets/mobilenet/mobilenet\_v2.py中，修改V2\_DEF.spec最后一个op：

修改：

op(slim.conv2d, stride=1, kernel\_size=[1, 1], num\_outputs=1280)

为：

op(slim.conv2d, stride=1, kernel\_size=[1, 1],num\_outputs=3657)

但这种修改仅针对depth\_multiplier=0.35的情况；

10.0.12.147设备有4个卡可以用于训练；10.0.12.148设备有4个卡可以用于训练；x8有8个卡可以用于训练。通过调参可以使得一个卡执行一个任务，需要指定CUDA\_VISIBLE\_DEVICES=1可以将训练任务指定给卡1，否则会用所有卡训练。

操作步骤：

1. 修改config文件中的参数各个（如：ssd\_mobilenet\_v1\_0.25\_224.config）；
2. 执行./1\_train.sh，训练完成后，会在./train\_log目录中保存一系列checkpoint文件，该文件可以用来生成模型；
3. 执行./2\_out.sh，在./output目录生成tensorflow模型文件在output文件夹，并且在当前目录生成tmp.txt文件（用于进行模型转换为ncnn模型）；
4. 执行./3\_test-presion.sh，验证模型性能；

模型训练时，调整各配置参数训练模型并测试，分析各参数对模型的影响，并选择最优参数。

部分参数的调整需要修改代码，具体操作如下[[4]](#footnote-5)：

1. data augmentation中增加random\_dist\_colors时，需要修改random\_distort\_color函数；
2. data augmentation中增加random\_pixel\_value\_scale时，需要修改random\_distort\_color函数函数；
   1. 模型转换

模型转换代码Gitlab地址：http://gitlab.ling.ai:10080/zhenghui/finger\_tools/tree/master/3\_trans

设备端采用的NCNN框架，所以需要对训练模型进行转换，转换步骤如下：

1. 修改[tensorflowPath]/graph\_util\_impl.py[[5]](#footnote-6)使得输出模型时生成tmp.txt（现有代码均已修改）
2. 训练完成后执行../2\_train/2\_out.sh，调用export\_inference\_graph.py导出模型（注意设置CUDA\_VISIBLE\_DEVICES），生成tmp.txt文件；
3. 运行model\_transfer\_mobilenetv1.py（或者model\_transfer\_mobilenetv1\_dropout.py），实现ssd+mobilenetv1的模型转换，即生成ncnn所需文件mobilenet\_ssd\_ncnn\_v1.bin和mobilenet\_ssd\_ncnn\_v1.param（可选4）；
4. 运行model\_transfer\_mobilenetv2.py可以实现ssd+mobilenetv2的模型转换，生成mobilenet\_ssd\_ncnn\_v2.bin和mobilenet\_ssd\_ncnn\_v2.param（可选3）；
5. 若训练时修改了配置文件中的anchor生成参数（如：min\_size/ratio），则ncnn配置文件也需人工修改对应参数；

设备端采用的TFLITE框架，模型转换步骤如下：

1. 执行export\_lite.sh，实现ckpt到pb的转换；
2. 执行convert\_tflite.sh，实现pb到tflite的转换；
   1. 模型性能

手和指尖检测需要关注几个关键性能指标：召回率，准确率，误报率，设备端内存占用和处理时间。我们的目标是在高召回率和准确率的前提下，模型可以满足设备内存占用和处理时间要求。

表 3 手和指尖检测性能要求

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 召回率 | 准确率 | 内存占用 | 处理时间 |
| 手 | 95% | 95% | <10M | <200ms |
| 指尖 | 95% | 95% |

* + 1. 测试数据

测试数据分为两类，一类是record文件格式，一类图片格式。目前的测试数据是用户采集的1000张负样本和394张正样本（数据位置：/mnt/cv-samba/dataset/finger/finger\_test/）。

最新的测试数据是用户采集的2100张正样本和1000张负样本（数据位置：/mnt/cv-samba/dataset/finger/finger\_test），其中子文件夹fingerX是正样本，neg文件夹是负样本。

* + 1. 模型精度性能测试

精度性能包括两个类别的召回率和准确率。

测试脚本Gitlab地址：http://gitlab.ling.ai:10080/zhenghui/finger\_tools/tree/master/2\_train

测试的和分析Gitlab地址：http://gitlab.ling.ai:10080/pbr/cv-tasks/issues/177。

测试方式有两种，采用tensorflow验证脚本和自己写的测试脚本：

1. 直接采用tensorflow测试脚本进行测试，要求测试数据为record文件格式，步骤如下：
   * 1. 修改[CodePath]/utils/metrics.py， 增加compute\_precision\_recall\_with\_score\_th[[6]](#footnote-7)；
     2. 修改[CodePath]/utlis/object\_detection\_evaluation.py的evaluate函数，增加对compute\_precision\_recall\_with\_score\_th的调用和输出[[7]](#footnote-8)；
     3. 确认配置文件中eval\_config的num\_examples为测试样本数量，max\_evals为1；
     4. 运行4\_test-tf.sh脚本；
2. 利用自己的测试脚本进行测试（推荐该方式），测试数据通过txt格式的文件列表提供，支持图像保存，执行步骤如下：
   * 1. 执行3\_test-presion.sh；
     2. 设备端性能测试

设备端性能包括内存占用和检测耗时，该部分代码可以编译PC端（PC端主要用来验证转换为ncnn版本模型是否有效）和设备端应用程序，分别在不同设备上进行模型测试。

设备端相关代码：http://gitlab.ling.ai:10080/zhenghui/finger\_tools/tree/master/4\_test/test\_on\_device（或者test\_on\_device\_tfl）。

测试和分析Gitlab地址：http://gitlab.ling.ai:10080/pbr/cv-tasks/issues/212，或者

<http://gitlab.ling.ai:10080/luka/cv/cv-tasks/issues/177>。

测试步骤如下：

1. 生成a33上手指检测程序；
2. 准备ncnn[[8]](#footnote-9)和android-ndk编译环境；
3. 执行./build.sh(build\_pc.sh)，生成build-android-armv7/bin/test\_src\_model(/test\_trans\_model，改程序测试经过加密的模型)，注意DANDROID\_NATIVE\_API\_LEVEL=android-19
4. PC端测试可以直接执行应用程序4\_run.py
5. 设备端测试，需要利用adb push将检测程序、配置文件和模型、图片数据等推至设备端/data目录下（注意配置文件、图片数据名称需与源文件对应）；
6. 内存占用查看：
7. 在一个终端中运行adb shell，执行top -d 0 | grep test\_src\_model；
8. 另一终端中运行adb shell，进入data目录，执行./test\_src\_model；
9. top命令输出的rss项的最大值即为内存占用情况；
10. 检测时间查看：
    1. 执行./ test\_src\_model >info.txt可将时间信息输出到info.txt，传到PC端；
    2. PC端利用脚本scripts/time\_stat.py可统计平均时间；

注意事项：

1. 评估检测时间时，不能运行top -d 0 | grep test\_src\_model，否则会影响速度；
2. 评估检测时间时，需要关注开启的线程数（在test\_src\_model .cpp 中的ex.set\_num\_threads进行设置，目前线程数为2较稳定）以及设备端是否开启android系统（adb shell后，stop停止系统，start开启系统）；
3. 评估检测时间时，测试需提供两个测试结果，分别是开启android和关闭android；

影响设备端性能的主要因素主要时模型容量和资源消耗，主要的方法是通过模型裁剪获取容量和性能均满足要求的模型。

对mobilenet2.0模型进行裁剪，需要修改部分代码，具体操作如下：

1. 裁剪expanded\_conv类型的operator时，需要对两个文件进行修改：
2. [CodePath]/slim/nets/mobilenet/mobilenet\_v2.py[[9]](#footnote-10)，删除V2\_DEF中spec的对应op；
3. [CodePath]/ object\_detection/models/ssd\_mobilenet\_v2\_feature\_extractor.py[[10]](#footnote-11)中，对\_extract\_features函数中的feature\_map\_layout进行修改。feature\_map\_layout反映了在哪些层上加入predictor以及相关参数，原始的from\_layer前两项为'layer\_15/expansion\_output'和'layer\_19'，裁剪op后，'layer\_19'要相应修改（裁剪2个op，则改为'layer\_17'），'layer\_15/expansion\_output'也可能要视情况修改；
4. 直接裁剪channel然后训练
5. 对mobilenet\_v2.py中V2\_DEF中spec的op的num\_outputs进行修改（如将最后一个op的 3657改回1280）；
6. 必要时利用前期较为复杂的模型裁剪方法裁剪channel；
   1. 模型加密

为了安全，对模型参数和weight加密，保存成二进制文件，并加入噪声，目前支持ncnn和tflite模型。

模型加密代码Gitlab地址：http://gitlab.ling.ai:10080/zhenghui/finger\_tools/tree/master/5\_model\_build。

加密操作步骤如下：

1. 运行./5\_run.py（./5\_run\_tflite.py），完成模型加密；
2. 位置映射

在实际应用中，手指检测算法被用于进行绘本阅读的手指互动，通过检测手指的位置来和用户进行交互。但是，手指检测算法得到的是在设备端原始图片上指尖的位置，必须通过图片匹配的方式进行坐标映射转换，得到指尖在原始扫描图中的位置坐标，才可以进行正常的交互活动。

* 1. 坐标变换算法

将在warp图上获取的特征点变换到原始图像上，变换到原始图像上的特征点再进行畸变矫正后与scan图的特征点进行匹配获取单应性矩阵H，手指位置进行畸变矫正，利用H矩阵将畸变后的手指位置映射到scan图，畸变参数通过相机标定获取。

由于绘本可能较大，或者左右页可能不在一个平面，增加了左右两个独立的映射矩阵计算，并通过手指位置和映射矩阵的距离关系，选择最佳变换矩阵，提升边缘处的变换精度。

* 1. 设备端测试程序

设备端测试程序实现的手指检测和位置映射，并保存对应结果图像，供调试使用。

设备端Gitlab地址：http://gitlab.ling.ai:10080/pbr/LukaBookRecognizer，finger分支，操作步骤：

1. 根目录执行./gradlew assemble生成libfeature.so；
2. 推送到设备：
   1. adb push ./cv-module/src/main/libs/armeabi-v7a/libfeature.so /system/vendor/lib；
   2. adb push ./cv-module/src/main/libs/armeabi-v7a/libfeature.so /system/lib
3. 模型推送到设备：adb push [model\_file].bin /sdcard/model/interaction/model；
4. 执行绘本阅读
5. 拉取相应信息：

adb pull /sdcard/ ./

* 1. PC端测试程序

PC端测试程序实现手指检测位置映射的代码调试功能，从实际拍摄图片中检测到手指位置，映射到扫描图中的位置。

PC端代码地址：http://gitlab.ling.ai:10080/zhenghui/finger\_tools/tree/master/7\_mapping。

数据库：/mnt/cv-samba/dataset/finger/finger\_match/test/

/mnt/cv-samba/dataset/finger/finger\_match/scan/

操作步骤：

1. make，生成device\_test；
2. 执行：7\_run.py (./device\_test [test\_img\_num] [scan\_img\_path][[11]](#footnote-12))
3. 工作进度及计划
   1. 2018-06-30

根据目前手指检测工作的进展情况，需要在以下几个方面对手指检测工作继续跟进：

* + 1. 数据清洗和整理

1. 自动数据清洗脚本实现（已完成）；
2. 数据库规范化，指定标准的训练和测试数据集（已完成）；
   * 1. 手指模型分析
3. 测试对mobilenet2.0\_0.35进行裁剪后的精度和内存/耗时等性能指标（完成80%）；
4. 优化位置映射算法，建立规范测试评价方法，解决位置映射不准确问题（已完成）；
   * 1. 模型版本控制
5. 规范手指检测算法版本，进行输出版本控制；
   1. 2018-11-02

手指检测模型训练，采用了mobilenet1.0基础结构和SSD检测框架，由于硬件设备限制，模型scale采用了0.25来保证处理速度。本阶段主要工作内容包括：

1. 数据清洗

训练数据包括外包团队标注的儿童点读姿势（dataset13，共35340张标注图片）和公司内部数据标注（dataset1-dataset14），公司内部数据标注批次之间差异较大，本月完成了该dataset1和dataset5两个成人数据集的数据清洗工作，共生成4265张标注数据；

1. 测试数据整理

测试数据集采用用户数据，共395张点读数据（405个点读姿势）和1000张其它手指姿势图片，分别称为test\_hand和test\_nohand，利用test\_hand测试召回率和准确率，利用test\_nohand测试误报率；

1. 模型压缩
2. 测试了mobilenet2.0\_s0.35模型压缩

mobilent2.0性能要优于mobilenet1.0，但除了速度慢，无法满足实际需要。

1. shufflenet为基本网络结构的模型训练

shufflenetV2\_s0.5的模型处理速度和mobilenet1.0\_s0.25相当，还未完成基础模型的训练，性能未知。

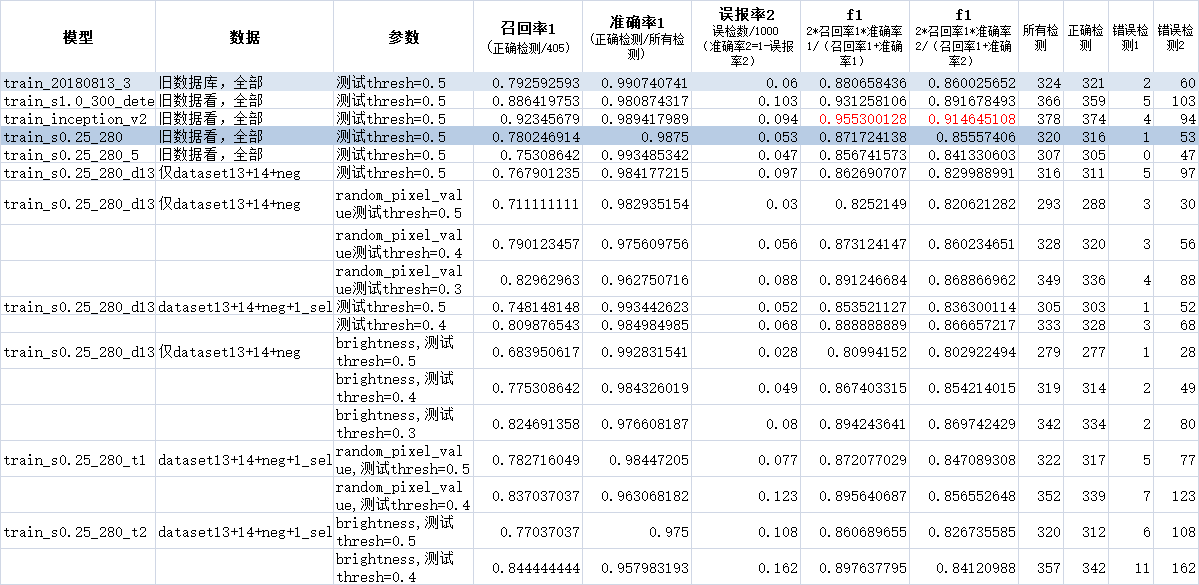
1. 位置匹配优化

针对边缘处位置匹配不准确的问题，增加左右页面独立建立变换矩阵，并通过手指位置和变换矩阵的位置关系，选择距离最近的变换矩阵，从而提升位置匹配准确性，改改进方法对速度影响不大。

1. 模型性能优化

模型性能优化途径主要有：

1. 修改训练数据，提升数据有效性，更好的训练数据对模型性能有提升；
2. 增加data\_argumentation参数，多次实验发现参数对模型的影响不确定；
3. 调整anchor参数，增加第一特征层的anchor数，实验证明该调整会导致性能下降；
4. 模型输入图像修改为280，性能下降可接受，速度可以提升25%；



从目前的测试结果看，在现有数据集上，通过参数调整对模型性能的提升有限，在误报率为5%时，召回率78%左右，建议从其它方面进行跟进。

1. 更换更有效的基础网络模型；
2. 利用更大的网络结构训练后进行压缩；
   1. 2019-08-06

本阶段主要工作内容：

1. 手指检测采用了tflite模型；
2. 测试了量化，warmup，mixup等策略；
3. 尝试了yolo3模型，尝试了inception基础网络等；
4. 更新了测试数据，扩大正样本到2100张；
5. 重新调整各超参，获取最优参数；

经过尝试，发现量化，warmup，cos学习率是比较有效的策略，清洗数据对模型性能有改善。目前模型最好的性能，召回率86.75%，准确率94.2%，f1=90.3%。

1. 附
   1. NCNN推理程序

设备端通过静态库方式调用NCNN推理程序，android编译静态库步骤如下：

1. 拉取镜像：docker pull lingcv/work:mobile；
2. 获取NCNN代码：ssh://git@gitlab.ling.ai:10022/chengpu/ncnn\_ling.git；
3. 编译：./build.sh，生成build-android-armv7/libs/armeabi-v7a/ libncnn.a；
   1. 设备端luka程序

Gitlab地址：<http://gitlab.ling.ai:10080/pbr/LukaBookRecognizer>，finger分支。

1. 下载地址为https://tzutalin.github.io/labelImg/ [↑](#footnote-ref-2)
2. 后文中用到“训练代码位置”用[CodePath]表示 [↑](#footnote-ref-3)
3. 后文中用到“训练代码位置”用[CodePath]表示 [↑](#footnote-ref-4)
4. 具体修改参见：Gitlab下的code\_edit/ [↑](#footnote-ref-5)
5. [tensorflowPath]: /usr/local/lib/python2.7/dist-packages/tensorflow/python/framework/,是安装路径，修改参见Gitlab代码。 [↑](#footnote-ref-6)
6. 参考Gitlab下的code\_edit/ metrics.py [↑](#footnote-ref-7)
7. 参考Gitlab下的code\_edit/object\_detection\_evaluation.py [↑](#footnote-ref-8)
8. http://gitlab.ling.ai:10080/zhenghui/finger\_tools/tree/master/6\_requirment [↑](#footnote-ref-9)
9. 代码位置：http://gitlab.ling.ai:10080/chengpu/ncnn\_ling/research/slim/nets/mobilenet/ [↑](#footnote-ref-10)
10. 代码位置：http://gitlab.ling.ai:10080/chengpu/ncnn\_ling/research/ object\_detection/models/ [↑](#footnote-ref-11)
11. 确定规范测试数据库和测试评价方法后，实现脚本调用 [↑](#footnote-ref-12)