# Kaldi 介绍

Kaldi 是由 C++ 编写的语音识别工具，其目的在于为语音识别研究者提供一个使用平台。

## Docker 基础命令

本节简单介绍在安装 Kaldi 过程中使用到的 Docker 基础命令，如读者需要更加详细的学习 Docker，请主动寻找相应的专业书籍、文档学习。

本章中所用到的 Docker 命令如下：

```
docker pull        从 Dockerhub 中拉去镜像
docker images      查看系统中所有系统镜像
docker run         运行系统镜像
docker exec        登陆系统镜像
docker commit      在原有镜像基础上创建新镜像
docker ps          查看镜像是否运行
```

# Kaldi 环境搭建

本文主要通过使用 Docker 构建 Ubuntu 环境对 Kaldi 进行搭建。Kaldi 的环境搭建分为两部分，一部分为依赖工具安装，另一部分为自身源码编译。

首先通过 Docker 获取相应的 Ubuntu 系统。

```
docker pull ubuntu:16.04
```

之后启动该镜像，并从安装相应相应软件

```
docker run -itd ubuntu:16.04 /bin/bash 启动镜像

docker ps 获取该启动镜像的 Container ID

docker exec -it <Container ID> /bin/bash 进入系统

apt-get update

apt-get install git vim

cd /opt && git clone https://github.com/kaldi-asr/kaldi.git && cd kaldi
```

接下来进行 Kaldi 的第一部分安装，第一部分主要是针对 Kaldi 依赖工具的安装比如 Openfst、Portaudio等。安装步骤如下：

```
cd tools && extras/check_dependencies.sh

apt-get install  g++ zlib1g-dev make automake autoconf bzip2 unzip wget sox
libtool subversion python2.7 python3 libatlas-dev libatlas-base-dev

make

extras/install_irstlm.sh
```

Kaldi 安装的第二部分为源码编译部分，这里的首要条件是第一部分正常安装之后，第二部分才能顺利完成。安装步骤如下：

```
cd /opt/kaldi/src

./configure --shared

make depend -j 8

make -j 8
```

# Kaldi 目录介绍

本节主要对 Kaldi 的目录进行详细介绍。

## Kaldi 一级目录

在Kaldi 的一级主目录中（也就是进入kaldi目录之后大家所看到的所有目录）包括的目录有：egs、misc、scripts、src、tools、windows。

- egs： 此目录为 Kaldi 的例子目录，其中包含了不乏 语音识别、语种识别、声纹识别、关键字识别等例子。
- misc : 此目录包含了一些pdf、以及相关docker、htk等资源的例子
- scripts: 此目录只用来存放 Rnnlm，以及相应的运行脚本。
- src : 此目录为 Kaldi 的源代码目录，Kaldi 的多数算法的源代码都存放于此，其中不乏GMM、Ivector、Nnet等一系列的算法。
- tools： 此目录主要存放 Kaldi 依赖库的安装脚本
- windows： 此目录为在windows平台运行所必须的脚本以及相关的执行程序。

## Kaldi 一级目录 egs

Egs 目录 主要用于存放 Kaldi 的所有例程，这里会统一罗列出相关文件所包含的相关例子。

- aishell : 此目录为中文语音识别和声纹识别相关例子
- aishell2： 此目录主要为中文语音识别例子，但是针对aishell在脚本方面更加规整。
- ami : 此目录主要涉及到多信道语音识别的例子
- an4 : 此例子为CMU提供语音识别例子，并没有涉及神经网络。
- apiai_decode: 此例子为解码器使用的例子，其中包含了如何使用预训练模型，这里主要针对的是Nnet3解码
- aspire: 此为ASpIRE 挑战赛的例子，其中包含了怎样使用噪声数据构建多条件数据的例子

- aurora4: 此例子主要介绍RBM 预训练
- babel: 此例子主要是用来训练 KWS （Key Word Search）
- babel_multilang: 此例子为训练多语音 KWS
- bentham: 手写笔识别的例子
- bn_music_speech : 音乐与语音区分的例子
- callhome_diarization : 说话人分割的例子
- callhome_etyptian: 埃及语语音识别例子
- chime1-5 : 主要针对CHiME 竞赛开放的例子
- cigar : 图像分类的例子
- commonvoice: Mozilla Common Voice 语音识别的例子
- csj : 日语 语音识别例子
- dihard_2018 : DiHARD Speech Diarization CHALLENGE 的例子
- fame : 富里西语语音识别和声纹识别的例子。
- farsdat: 主要用来声学语音研究和语音识别的例子
- fisher_callhome_spanish : 使用Callhome预料进行语音识别的例子
- fisher_english:
- fisher_swbd:
- gale_arabic: 阿拉伯语语音识别例子
- gale_mandarin: 普通话语音识别例子
- gp: 全球电话语音识别例子（多语种语音识别例子）
- heroico: 西班牙语音识别例子
- houst: 普通话电话语音识别例子
- hub4_english : 英语新闻广播语音识别例子
- hub4_spanish: 西班牙新闻广播语音识别例子
- iam: IAM 手写笔识别例子
- iban: 语音识别例子
- ifnenit: 阿拉伯语手写笔识别例子
- librispeech: 英语语音识别例子
- lre/lre07 : 语种识别例子
- madcat_ar : 手写笔识别例子
- madcat_zh: 中文手写笔识别例子
- mini_librispeech: 英语语音识别例子
- mult_en: 英语LVCSR 例子
- pub: RNNLM 模型构建例子
- reverb: REVERB 挑战赛例子
- rimes: 法语手写笔识别例子
- rm: 英语语音识别例子，包含了如何进行迁移学习
- sitw: sitw 说话人识别挑战赛的例子
- sprakbanken: 丹麦语语音识别例子
- sprakbanken_swe: 瑞典语语音识别例子
- sre08/10/16: 说话人识别的例子
- svhn: 图像分类的例子
- swahili: 班图人语 语音识别例子
- swab: 双声道对电话语音识别例子
- tedium: 英语语音识别例子
- thchs30: 普通话语音识别例子
- tidigits: 基础语音识别的例子

- timit: 主要是GMM/HMM 语音识别例子
- tunisian_msa: 阿拉伯语音识别例子
- uw3: OCR 识别例子
- voxceleb: 说话人识别例子
- vystadial_cz: 捷克语语音识别例子
- voxforge: 基础语音识别例子，以及对应的在线demo的例子
- vystadial_en: 英文语音识别例子
- wsj: wsj 英文语音识别例子
- yesno: 独立词语音识别例子
- yomdle_fa/korean/russian/tamil/zh: OCR 识别例子
- zeroth_korean: 朝鲜语语音识别例子

**注意：** 虽然 Egs 中存放了大量的例子，但由于某些外部原因并不是所有例子的数据都能免费获得。

## Kaldi 一级目录 src

src目录为 Kaldi 的源码目录，主要保存了包括GMM、HMM等在内的大部分 Kaldi 语音项目源代码。这里分别对相关算法目录进行介绍。

在 src 目录中，有两类文件夹，一类是算法原目录，一类为算法组合生成bin（可执行程序）目录。

```
base:        Kaldi 的基础目录，主要包括对与 Kaldi 项目相关的基础宏定义、类型定义等

bin:         Kaldi 的基础bin目录，主要是包括基础的执行只需。例如，查看tree 信息、矩阵拷贝等基础操作。

cudamatrix: GPU 版本 Kaldi 相关矩阵计算

matrix :     CPU 版本的 Kaldi 相关矩阵计算

itf:         Kaldi 相关的interface

hmm :        Kaldi 相关的隐马尔可夫算法的代码目录

utils:       Kaldi 相关，语音算法无关的工具库，例如，线程操作、io操作、文本操作等。

probe:       Kaldi 相关的exp 测试目录

transform:   Kaldi 相关的特征转换算法目录

fstext:     Kaldi 中fst 相关的算法基础库

fstbin:      Kaldi 中 fst 对应的算法执行文件夹

feat:       Kaldi 相关的特征提取算法目录

featbin:    Kaldi 相关的特征提取可执行目录

gmm:        Kaldi 相关的GMM算法基础目录
```

```
gmmbin:    Kaldi 相关的GMM算法可执行文件目录

ivector:   Kaldi 相关的ivector 算法基础目录

ivectorbin: Kaldi 相关的 ivector 算法的可执行目录，以及基于能量的vad 执行目录。

kws:       Kaldi 相关的关键字搜索基础算法目录

kwsbin:    Kaldi 相关的关键字搜索执行目录

lat:       Kaldi 相关的网格生成基础算法目录

latbin:    Kaldi 相关的网格生成算法的可执行文件目录

lm:       Kaldi 自带的语言模型基础算法目录

lmbin:  Kaldi 相关语音模型的可执行文件目录

nnet:    Kaldi 相关的 nnet1 基础算法实现目录

nnetbin: Kaldi nnet1相关的算法可执行文件目录

nnet2:   Kaldi nnet2 相关的基础算法实现目录

nnet2bin: Kaldi nnet2 相关的算法可执行文件目录

nnet3:   Kaldi nnet3 相关基础算法实现目录

nnet3bin:  Kaldi nnet3 相关实现算法的可执行文件目录
online:   Kaldi online1 相关解码算法的实现目录

onlinebin: Kaldi online1 相关解码器算法的可执行目录

online2:   Kaldi online2 相关解码器算法的实现目录

online2bin: kaldi online2 相关解码器算法的可执行目录

rnnlm:     Kaldi rnnlm 相关的语音模型基础算法实现目录

rnnlmbin:  Kaldi rnnlm 相关的语音模型的可执行目录

sgmm2:    Kaldi sgmm2 相关的子空间GMM基础算法实现目录

sgmm2bin:  Kaldi sgmm2 相关的子空间GMM基础算法可执行目录

tfrnnlm:   Kaldi 相关的 Tensorflow rnnlm的基础算法目录

tfrnnlmbin: Kaldi 相关的 Tensorflow rnnlm 的基础算法实现的可执行目录
```

# Kaldi Egs Aishell 例子黑箱运行

本节的目的在于，在读者不了解 Kaldi 语音识别相关的算法的情况下，能过顺利运行 Aishell 的例子。

首先需要有几个注意点：

- 如果需要运行神经网络相关的算法，确保机器拥有GPU运算能力
- 确保以上 Kaldi 编译成功
- 如果网络下载速度有限，确保数据集存放位置正确
- 确定运行代码机器内存和硬盘大小。

本例子完全运行需要硬盘资源为76G，所以需要确定硬盘是否有足够的空间。

## Aishell 例子运行

为了能过顺利黑箱运行 Aishell 语音识别的例子。需要对 /opt/kaldi/egs/aishell/s5 中的 run.sh 脚本进行几点修改。

假设 Aishell 的语音数据已经全部下载，目录存放于/newdata/corpus/aishell , 那么对应的 run.sh 脚本修改如下：

```
15 data=/newdata/corpus/aishell
16 #data_url=www.openslr.org/resources/33
```

同时，由于我们的数据集已经下载，故需要注视掉aishell 下载脚本，修改如下：

```
19 #local/download_and_untar.sh $data $data_url data_aishell || exit 1;
20 #local/download_and_untar.sh $data $data_url resource_aishell || exit 1;
21
```

由于大家机器不同，内存和CPU数量的不同，在部分脚本相对应的运行进程上也要进行部分修改，这里需要大家根据自己实际机器情况。假设运行的机器为16G 内存，run.sh 脚本修改为，vim 打开 run.sh ，进入命令模式：

```
:%s/--nj 10/--nj 5/g
```

最后，需要注意，如果需要进行神经网络相关训练操作，需要对 local/chain/run_tdnn.sh 进行修改，修改操作为将 num_jobs_initial 和 num_jobs_final 统一修改成该训练机器所对应的GPU个数。修改如下图：

```
21 num_jobs_initial=2
22 num_jobs_final=2
```

最后，回到 run.sh 的目录中，使用命令：

```
nohup sh run.sh &
```

确保 run.sh 能够在后台运行。直至出现错误或者运行成功自动结束。

# 如何在黑箱情况下，使用独立语音进行模型模型构建

本节假设读者已经成功运行上一节黑箱的例子，相信大家做语音识别的初衷亦或者兴趣点在于如何使用自己的语音训练模型，那么本节的主要目的就在于告诉大家如何在aishell的基础上，使用自己的语音构建模型。

学习本节，读者依旧不需要了解任何的 Kaldi 语音识别相关内容。此时，读者可能会有疑问，既然是黑箱，那如何能够使用独立语音训练集训练出自己的模型呢?

其实很简单，我们这里有两种方法提供给大家：

**方法一：** 我们只需要模仿 Aishell 的数据存放规律进行存放即可。

**方法二：** 如果我们现有的独立数据集有自己的规律存放，那么只需要修改针对性的修改 run.sh 中的两个脚本即可。

方法一这里不做详细说明，大家根据上一节使用的aishell的数据方式模仿存储即可。这里针对方法二进行示范。

再次回到 run.sh 脚本中，我们可以发现，Aishell 的数据处理逻辑主要存在于两个子脚本中。他们分别为如图所示：

```
23 # Lexicon Preparation,
24 local/aishell_prepare_dict.sh $data/resource_aishell || exit 1;
25
26 # Data Preparation,
27 local/aishell_data_prep.sh $data/data_aishell/wav $data/data_aishell/transcript
   || exit 1;
28
```

这两个脚本即是我们需要修改的数据处理脚本。

首先看第一个脚本： local/aishell_prepare_dict.sh，此脚本的作用在于处理aishell 语音数据集所包含的发音字典信息。由于我们使用的是我们自己的独立数据集合，因此，我们需要通过我们自己的独立语音数据集获取相对应的发音词典。对应于aishell 数据集中的lexicon.txt 文件。

第二步：修改local/aishell_data_prep.sh 脚本。该脚本主要是用来读取训练数据集、验证集以及测试集相对应的语音和语音相对应文本的对应关系。

由于第一步只需要进行匹配和人工标注，这里不多赘述。这里简单说明一下第二部分如何修改。

假设我们的语音数据集的分布为如图所示：

| 一级目录 | 二级目录 | 三级目录 | 四级目录 | 五级目录 |
|---|---|---|---|---|
| our_wav_dir | | | | |
| | train | | | |
| | | S0001 | session | .wav |
| | | | | .wav |
| | | S0003 | session | .wav |
| | | | | .wav |
| | dev | | | |
| | | S0005 | sessioin | .wav |
| | | | | .wav |
| | | S0006 | session | .wav |
| | | | | .wav |
| | test | | | |
| | | S0007 | session | .wav |
| | | | | .wav |

同时，我们可以通过aishell 的数据，看出数据存放格式如下图所示：

| 一级目录 | 二级目录 | 三级目录 | 四级目录 |
| --- | --- | --- | --- |
| our_wav_dir | | | |
| | train | | |
| | | S0001 | .wav |
| | | S0002 | .wav |
| | | S0003 | .wav |
| | dev | | |
| | | S0005 | .wav |
| | | S0006 | .wav |
| | test | | |
| | | S0007 | .wav |
| | | | |

那么，我们只需要将我们的数据集中session文件夹去除掉，亦或者修改脚本中如下图所示的代码即可：

```
45 # Transcriptions preparation
46 for dir in $train_dir $dev_dir $test_dir; do
47   echo Preparing $dir transcriptions
48   sed -e 's/\.wav//' $dir/wav.flist | awk -F '/' '{print $NF}' > $dir/utt.list
49   sed -e 's/\.wav//' $dir/wav.flist | awk -F '/' '{i=NF-1;printf("%s %s\n",$NF,$
   i)}' > $dir/utt2spk_all
50   paste -d' ' $dir/utt.list $dir/wav.flist > $dir/wav.scp_all
51   utils/filter_scp.pl -f 1 $dir/utt.list $aishell_text > $dir/transcripts.txt
52   awk '{print $1}' $dir/transcripts.txt > $dir/utt.list
53   utils/filter_scp.pl -f 1 $dir/utt.list $dir/utt2spk_all | sort -u > $dir/utt2s
   pk
54   utils/filter_scp.pl -f 1 $dir/utt.list $dir/wav.scp_all | sort -u > $dir/wav.s
   cp
55   sort -u $dir/transcripts.txt > $dir/text
56   utils/utt2spk_to_spk2utt.pl $dir/utt2spk > $dir/spk2utt
57 done
```

只需要将49行的NF-1 修改为 NF-2即可。 这样就可以顺利运行我们自己的独立语音数据集合。

## 总结

本章主要介绍了Kaldi 相关的 Docker 基础操作，并介绍了如何正确安装 Kaldi 以及 Kaldi 相关的目录结构，之后介绍了黑盒运行 AIshell 的例子，最后简单介绍了如何修改脚本来运行自己的独立语音数据集。