

华为认证 HCIA-AI 系列教程

HCIA-AI

实验指导手册

版本:3.0



华为技术有限公司

第二章 机器学习概览实验	3
第四章 业界主流开发框架实验	18
第六章 Hello Davinci 实验	81
第九章 人工智能综合实验	102

华为认证 HCIA-AI 系列教程

HCIA-AI

机器学习概览

实验指导手册

版本:3.0



华为技术有限公司

版权所有 © 华为技术有限公司 2020。保留一切权利。

未经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编： 518129

网址：
<http://e.huawei.com>

华为认证体系介绍

基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，华为公司打造了业界唯一覆盖ICT全技术领域的认证体系，包含ICT技术架构认证、平台与服务认证和行业ICT认证三类认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代的新型ICT人才，构建良性的ICT人才生态。

华为认证HCIA-AI V3.0定位于培养和认证具备使用机器学习、深度学习等算法设计、开发AI产品和解决方案能力的工程师。

通过HCIA-AI V3.0认证，将证明您了解人工智能发展历史、华为昇腾AI体系和全栈全场景AI战略知识，掌握传统机器学习和深度学习的相关算法；具备利用TensorFlow开发框架和MindSpore开发框架进行搭建、训练、部署神经网络的能力；能够胜任人工智能领域销售、市场、产品经理、项目管理、技术支持等岗位。



前言

简介

本手册为 HCIA-AI 认证培训教程，适用于准备参加 HCIA-AI 考试的学员或者希望了解 AI 编程基础知识的读者。掌握本实验手册内容，您将能够进行基础的 AI 图像识别方面的编程。

内容描述

本实验指导书共包含 1 个实验，是基于如何利用 sklearn-learn, python 包利用不同的回归算法来对波士顿地区的房价来进行预测，希望学员或者读者能够入门机器学习，具备机器学习构建的基础编程能力。

读者知识背景

- 本课程为华为认证基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：
- 具有基本的 Python 语言编程能力，有一定的数据结构基础，同时对深度学习算法知识有一定的认识。

实验环境说明

Python 开发工具

本实验环境是基于 Python3.6 环境开发编译的

目录

前 言	2
简介	2
内容描述	2
读者知识背景	2
实验环境说明	2
1 波士顿房价预测	4
1.1 实验介绍	4
1.1.1 关于本实验	4
1.1.2 实验目的	4
1.1.3 本实验数据集与框架	4
1.2 实验代码	5
1.2.1 引入相关依赖的包	5
1.2.2 载入数据集， 查看数据属性，可视化	6
1.2.3 分割数据集，并对数据集进行预处理	8
1.2.4 利用各类回归模型，对数据集进行建模	8
1.2.5 利用网格搜索对超参数进行调节	10
1.3 本章总结	12

1 波士顿房价预测

1.1 实验介绍

1.1.1 关于本实验

本实验所使用的开发环境是基于 ModelArts 华为云平台操作，环境的具体搭建可以参照《HCIA-AI 实验环境搭建实验手册》；由于本案例使用的数据集样本量较小，且数据来自于 scikit-learn 自带的开源波士顿房价数据。波士顿房价预测项目是一个简单的回归模型，通过该项目的学习可以学会一些关于机器学习库 sklearn 的基本用法和一些基本的数据处理方法。

1.1.2 实验目的

- 利用网络公开的波士顿房价数据集，作为模型输入数据。
- 构建机器学习模型，并进行训练与评估。
- 了解机器学习模型搭建的总体流程。
- 掌握机器学习模型训练，网格搜索，评估指标的运用。
- 掌握相关 API 的使用。

1.1.3 本实验数据集与框架

该案例主要内容是进行波士顿数据集，共有 13 个特征，总共 506 条数据，每条数据包含房屋以及房屋周围的详细信息。其中包含城镇犯罪率，一氧化氮浓度，住宅平均房间数，到中心区域的加权距离以及自住房平均房价等等。具体如下：

- CRIM：城镇人均犯罪率。
- ZN：住宅用地超过 25000 sq.ft. 的比例。
- INDUS：城镇非零售商用土地的比例。
- CHAS：查理斯河空变量（如果边界是河流，则为 1；否则为 0）。
- NOX：一氧化氮浓度。
- RM：住宅平均房间数。

- AGE: 1940 年之前建成的自用房屋比例。
- DIS: 到波士顿五个中心区域的加权距离。
- RAD: 辐射性公路的接近指数。
- TAX: 每 10000 美元的全值财产税率。
- PTRATIO: 城镇师生比例。
- B: $1000 (B_k - 0.63)^2$, 其中 B_k 指代城镇中黑人的比例。
- LSTAT: 人口中地位低下者的比例。
- target: 自住房的平均房价, 以千美元计。

框架: Sklearn, 框架一方面提供波士顿房价数据, 并且提供用于分割数据集, 标准化, 评价函数, 另一方面集成了各类常规机器学习算法; 另外我们使用了 XGboost, 是集成算法中 GBDT 的优化版本。

1.2 实验代码

1.2.1 引入相关依赖的包

```
# 防止不必要的警告
import warnings
warnings.filterwarnings("ignore")

# 引入数据科学基础包
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import scipy.stats as st
import seaborn as sns

## 设置属性防止画图中文乱码
mpl.rcParams['font.sans-serif'] = [u'SimHei']
mpl.rcParams['axes.unicode_minus'] = False

# 引入机器学习, 预处理, 模型选择, 评估指标
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score

# 引入本次所使用的波士顿数据集
from sklearn.datasets import load_boston

# 引入算法
from sklearn.linear_model import RidgeCV, LassoCV, LinearRegression, ElasticNet
#对比 SVC，是 svm 的回归形式
from sklearn.svm import SVR

# 集成算法
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
```

1.2.2 载入数据集， 查看数据属性，可视化

```
# 载入波士顿房价数据集
boston = load_boston()

# x 是特征，y 是标签
x = boston.data
y = boston.target

# 查看相关属性
print('特征的列名')
print(boston.feature_names)
print("样本数据量:%d, 特征个数: %d" % x.shape)
print("target 样本数据量:%d" % y.shape[0])

输出：
特征的列名
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO']
```

'B' 'LSTAT']

样本数据量:506, 特征个数: 13

target 样本数据量:506

转化为 dataframe 形式

```
x = pd.DataFrame(boston.data, columns=boston.feature_names)
```

```
x.head()
```

输出:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

对标签的分布进行可视化

```
sns.distplot(tuple(y), kde=False, fit=st.norm)
```

输出:

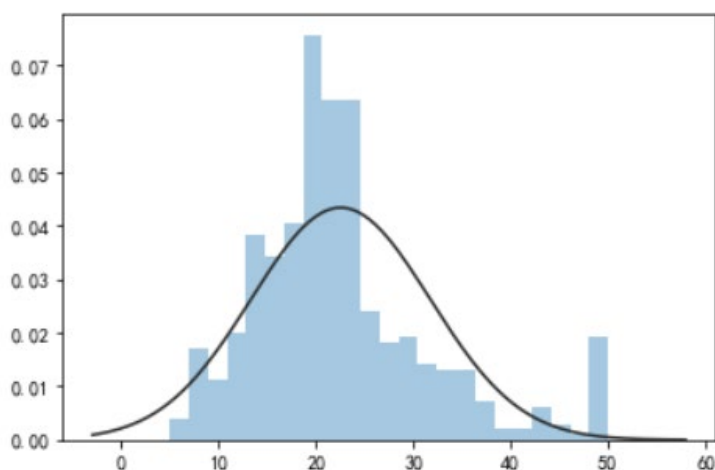


图1-1 目标数据分布

1.2.3 分割数据集，并对数据集进行预处理

```
# 数据分割
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=28)
# 标准化数据集
ss = StandardScaler()
x_train = ss.fit_transform(x_train)
x_test = ss.transform(x_test)
x_train[0:100]
```

输出：

```
array([[ -0.35451414, -0.49503678, -0.15692398, ..., -0.01188637,
         0.42050162, -0.29153411],
       [ -0.38886418, -0.49503678, -0.02431196, ..., 0.35398749,
         0.37314392, -0.97290358],
       [ 0.50315442, -0.49503678, 1.03804143, ..., 0.81132983,
         0.4391143 , 1.18523567],
       ...,
       [ -0.34444751, -0.49503678, -0.15692398, ..., -0.01188637,
         0.4391143 , -1.11086682],
       [ -0.39513036, 2.80452783, -0.87827504, ..., 0.35398749,
         0.4391143 , -1.28120919],
       [ -0.38081287, 0.41234349, -0.74566303, ..., 0.30825326,
         0.19472652, -0.40978832]])
```

1.2.4 利用各类回归模型，对数据集进行建模

```
# 模型的名字
names = ['LinerRegression',
         'Ridge',
         'Lasso',
         'Random Forrest',
         'GBDT',
         'Support Vector Regression',
```

```
'ElasticNet',  
'XgBoost']
```

```
# 定义模型
```

```
# cv 在这里是交叉验证的思想
```

```
models = [LinearRegression(),  
          RidgeCV(alphas=(0.001,0.1,1),cv=3),  
          LassoCV(alphas=(0.001,0.1,1),cv=5),  
          RandomForestRegressor(n_estimators=10),  
          GradientBoostingRegressor(n_estimators=30),  
          SVR(),  
          ElasticNet(alpha=0.001,max_iter=10000),  
          XGBRegressor())]
```

```
# 输出所有回归模型的 R2 评分
```

```
# 定义 R2 评分的函数
```

```
def R2(model,x_train, x_test, y_train, y_test):
```

```
    model_fitted = model.fit(x_train,y_train)  
    y_pred = model_fitted.predict(x_test)  
    score = r2_score(y_test, y_pred)  
    return score
```

```
# 遍历所有模型进行评分
```

```
for name,model in zip(names,models):  
    score = R2(model,x_train, x_test, y_train, y_test)  
    print("{}: {:.6f}, {:.4f}".format(name,score.mean(),score.std()))
```

输出：

LinerRegression: 0.564144, 0.0000

Ridge: 0.563700, 0.0000

Lasso: 0.564078, 0.0000

Random Forrest: 0.646657, 0.0000

GBDT: 0.725883, 0.0000

Support Vector Regression: 0.517310, 0.0000

ElasticNet: 0.564021, 0.0000

XgBoost: 0.765266, 0.0000

1.2.5 利用网格搜索对超参数进行调节

```
# 模型构建
```

```
'''
```

```
'kernel': 核函数
```

```
'C': SVR的正则化因子,
```

```
'gamma': 'rbf', 'poly' and 'sigmoid'核函数的系数, 影响模型性能
```

```
'''
```

```
parameters = {
```

```
    'kernel': ['linear', 'rbf'],
```

```
    'C': [0.1, 0.5, 0.9, 1, 5],
```

```
    'gamma': [0.001, 0.01, 0.1, 1]
```

```
}
```

```
# 使用网格搜索, 以及交叉验证
```

```
model = GridSearchCV(SVR(), param_grid=parameters, cv=3)
```

```
model.fit(x_train, y_train)
```

输出:

```
GridSearchCV(cv=3, error_score='raise',
```

```
    estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',
```

```
kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False),
```

```
    fit_params={}, iid=True, n_jobs=1,
```

```
    param_grid={'kernel': ['linear', 'rbf'], 'C': [0.1, 0.5, 0.9, 1, 5], 'gamma': [0.001, 0.01, 0.1, 1]},
```

```
    pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
```

```
    scoring=None, verbose=0)
```

```
## 获取最优参数
```

```
print("最优参数列表:", model.best_params_)
```

```
print("最优模型:", model.best_estimator_)
```

```
print("最优 R2 值:", model.best_score_)
```

输出：

最优参数列表: {'C': 5, 'gamma': 0.1, 'kernel': 'rbf'}

最优模型: SVR(C=5, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.1, kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)

最优 R2 值: 0.797481706635164

```
## 可视化
```

```
ln_x_test = range(len(x_test))
```

```
y_predict = model.predict(x_test)
```

```
# 设置画布
```

```
plt.figure(figsize=(16,8), facecolor='w')
```

```
# 用红实线画图
```

```
plt.plot(ln_x_test, y_test, 'r-', lw=2, label=u'真实值')
```

```
# 用绿实线画图
```

```
plt.plot(ln_x_test, y_predict, 'g-', lw = 3, label=u'SVR 算法估计值,$R^2$=%.3f' % (model.best_score_))
```

```
# 图形显示
```

```
plt.legend(loc = 'upper left')
```

```
plt.grid(True)
```

```
plt.title(u"波士顿房屋价格预测(SVM)")
```

```
plt.xlim(0, 101)
```

```
plt.show()
```

输出：

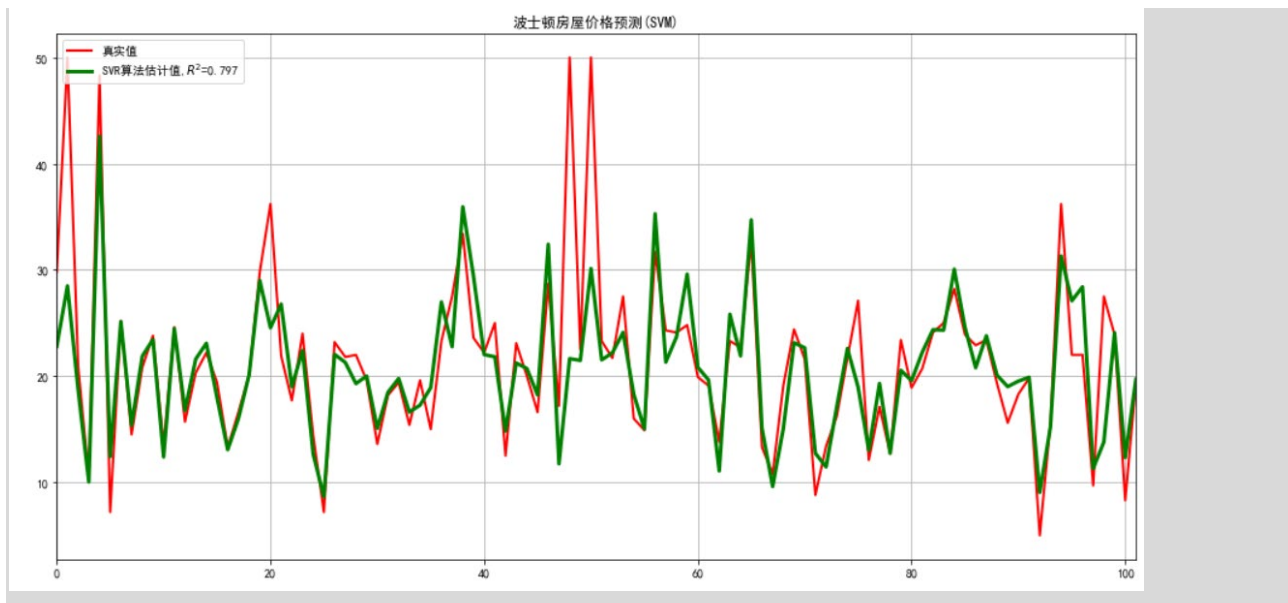


图1-2 可视化结果

1.3 本章总结

本章主要基于 sklearn 构建波士顿房价回归模型的学习，主要包含导入数据，分割数据，数据标准化，定义模型以及设置相关超参数等方面。使学员整体上对机器学习模型的构建有了一个基本的概念。

华为认证 AI 系列教程

HCIA-AI

业界主流开发框架

实验指导手册

版本:3.0



华为技术有限公司

版权所有 © 华为技术有限公司 2020。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： <http://e.huawei.com>

华为认证体系介绍

于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，华为公司打造了业界唯一覆盖ICT全技术领域的认证体系，包含ICT技术架构认证、平台与服务认证和行业ICT认证三类认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代的新型ICT人才，构建良性的ICT人才生态。

华为认证HCIA-AI V3.0定位于培养和认证具备使用机器学习、深度学习等算法设计、开发AI产品和解决方案能力的工程师。

通过HCIA-AI V3.0认证，将证明您了解人工智能发展历史、华为昇腾AI体系和全栈全场景AI战略知识，掌握传统机器学习和深度学习的相关算法；具备利用TensorFlow开发框架和MindSpore开发框架进行搭建、训练、部署神经网络的能力；能够胜任人工智能领域销售、市场、产品经理、项目管理、技术支持等岗位。



前言

简介

本书为 HCIA-AI 认证培训教程，适用于准备参加 HCIA-AI 考试的学员或者希望了解 AI 基础知识及 TensorFlow 编程基础的读者。

内容描述

本实验指导书共包 3 个实验：

- 实验一为 TensorFlow 基础，本实验会主要介绍 TensorFlow 2 的基本语法。
- 实验二为 TensorFlow 2 的常用模块介绍，主要会介绍 keras 接口。
- 实验三为手写字体图像识别实验，通过基本的代码，帮助读者了解如何通过 TensorFlow2.0 实现手写字体的识别。

读者知识背景

本课程为华为认证基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

具有基本的 Python 知识背景，同时熟悉 TensorFlow 基本概念，了解基本 Python 编程知识。

目录

前 言	3
简介	3
内容描述	3
读者知识背景	3
1 TensorFlow2 基础	6
1.1 实验介绍	6
1.1.1 关于本实验	6
1.1.2 实验目的	6
1.2 实验步骤	6
1.2.1 tensor 介绍	6
1.2.2 TensorFlow2 Eager Execution 模式	28
1.2.3 TensorFlow2 AutoGraph	30
2 TensorFlow 2 常用模块介绍	33
2.1 实验介绍	33
2.2 实验目的	33
2.3 实验步骤	33
2.3.1 模型构建	33
2.3.2 训练与评估	40
2.3.3 模型保存与恢复	46
3 利用 TensorFlow 进行手写数字识别	48
3.1 实验介绍	48
3.2 实验目的	48

3.3 实验步骤	48
3.3.1 项目描述和数据集获取	48
3.3.2 数据集预处理及可视化	51
3.3.3 DNN 网络构建	52
3.3.4 构建 CNN 网络	55
3.3.5 预测结果可视化	58

1 TensorFlow2 基础

1.1 实验介绍

1.1.1 关于本实验

本实验主要是 TensorFlow 2 的张量操作，通过对张量的一系列操作介绍，可以使学员对 TensorFlow 2 的基本语法有所了解。，包括张量的创建、切片、索引、张量维度变化、张量的算术运算、张量排序中介绍 TensorFlow 2 的语法。

1.1.2 实验目的

掌握张量的创建方法。

掌握张量的切片与索引方法。

掌握张量维度变化的语法。

掌握张量的算术运算操作。

掌握张量的排序方法。

通过代码，深入了解 Eager Execution 和 AutoGraph。

1.2 实验步骤

1.2.1 tensor 介绍

TensorFlow 中，tensor 通常分为：常量 tensor 与变量 tensor：

- 常量 tensor 定义后值和维度不可变，变量定义后值可变而维度不可变。
- 在神经网络中，变量 tensor 一般可作为储存权重和其他信息的矩阵，是可训练的数据类型。而常量 tensor 可作为储存超参数或其他结构信息的变量

1.2.1.1 创建 tensor

1.2.1.1.1 创建常量 tensor

常量 tensor 的创建方式比较多，常见的有以下几种方式：

- `tf.constant()`：创建常量 tensor；
- `tf.zeros()`, `tf.zeros_like()`, `tf.ones()`, `tf.ones_like()`：创建全零或者全一的常量 tensor；
- `tf.fill()`：创建自定义数值的 tensor；
- `tf.random`：创建已知分布的 tensor；
- 从 `numpy`, `list` 对象创建，再利用 `tf.convert_to_tensor` 转换为类型。

步骤 1 `tf.constant()`

`tf.constant(value, dtype=None, shape=None, name='Const', verify_shape=False)`：

- `value`：值；
- `dtype`：数据类型；
- `shape`：张量形状；
- `name`：常量名称；
- `verify_shape`：布尔值，用于验证值的形状，默认 `False`。`verify_shape` 为 `True` 的话表示检查 `value` 的形状与 `shape` 是否相符，如果不符合会报错。

代码：

```
const_a = tf.constant([[1, 2, 3, 4]], shape=[2, 2], dtype=tf.float32) # 创建 2x2 矩阵, 值 1, 2, 3, 4
const_a
```

输出：

```
<tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[1., 2.],
       [3., 4.]], dtype=float32)>
```

代码：

#查看常见属性

```
print("常量 const_a 的数值为: ", const_a.numpy())
print("常量 const_a 的数据类型为: ", const_a.dtype)
print("常量 const_a 的形状为: ", const_a.shape)
print("常量 const_a 将被产生的设备名称为: ", const_a.device)
```

输出:

```
常量 const_a 的数值为: [[1. 2.]
 [3. 4.]]
常量 const_a 的数据类型为: <dtype: 'float32'>
常量 const_a 的形状为: (2, 2)
常量 const_a 将被产生的设备名称为: /job:localhost/replica:0/task:0/device:CPU:0
```

步骤 2 tf.zeros(), tf.zeros_like(), tf.ones(), tf.ones_like()

因为 `tf.ones()`, `tf.ones_like()` 与 `tf.zeros()`, `tf.zeros_like()` 的用法相似, 因此下面只演示前者的使用方法。

创建一个值为 0 的常量。

```
tf.zeros(shape, dtype=tf.float32, name=None):
```

- shape: 张量形状;
- dtype: 类型;
- name: 名称。

代码:

```
zeros_b = tf.zeros(shape=[2, 3], dtype=tf.int32) # 创建 2x3 矩阵, 元素值均为 0
```

根据输入张量创建一个值为 0 的张量, 形状和输入张量相同。

```
tf.zeros_like(input_tensor, dtype=None, name=None, optimize=True):
```

- input_tensor: 张量;
- dtype: 类型;
- name: 名称;
- optimize: 优化。

代码：

```
zeros_like_c = tf.zeros_like(const_a)
#查看生成数据
zeros_like_c.numpy()
```

输出：

```
array([[0., 0.],
       [0., 0.]], dtype=float32)
```

步骤 3 tf.fill()

创建一个张量，用一个具体值充满张量。

tf.fill(dims, value, name=None):

- dims：张量形状，同上述 shape；
- vlaue：张量数值；
- name：名称。

代码：

```
fill_d = tf.fill([3,3], 8) # 2x3 矩阵，元素值均为为 8
#查看数据
fill_d.numpy()
```

输出

```
array([[8, 8, 8],
       [8, 8, 8],
       [8, 8, 8]], dtype=int32)
```

步骤 4 tf.random

用于产生具体分布的张量。该模块中常用的方法包括：tf.random.uniform(), tf.random.normal()和 tf.random.shuffle()等。下面演示 tf.random.normal()的用法。

创建一个符合正态分布的张量。

```
tf.random.normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None,
name=None):
```

- shape: 数据形状;
- mean: 高斯分布均值;
- stddev: 高斯分布标准差;
- dtype: 数据类型;
- seed: 随机种子
- name: 名称。

代码:

```
random_e = tf.random.normal([5,5],mean=0,stddev=1.0, seed = 1)
#查看创建数据
random_e.numpy()
```

输出:

```
array([[ -0.8521641,  2.0672443, -0.94127315,  1.7840577,  2.9919195 ],
       [ -0.8644102,  0.41812655, -0.85865736,  1.0617154,  1.0575105 ],
       [ 0.22457163, -0.02204755,  0.5084496, -0.09113179, -1.3036906 ],
       [ -1.1108295, -0.24195422,  2.8516252, -0.7503834,  0.1267275 ],
       [ 0.9460202,  0.12648873, -2.6540542,  0.0853276,  0.01731399]],
      dtype=float32)
```

步骤 5 从 numpy, list 对象创建, 再利用 tf.convert_to_tensor 转换为类型。

将给定值转换为张量。可利用这个函数将 python 的数据类型转换成 TensorFlow 可用的 tensor 数据类型。

```
tf.convert_to_tensor(value,dtype=None,dtype_hint=None,name=None):
```

- value: 需转换数值;
- dtype: 张量数据类型;
- dtype_hint: 返回张量的可选元素类型, 当 dtype 为 None 时使用。在某些情况下, 调用者在 tf.convert_to_tensor 时可能没有考虑到 dtype, 因此 dtype_hint 可以用作为首选项。

代码:

```
#创建一个列表
list_f = [1,2,3,4,5,6]
#查看数据类型
type(list_f)
```

输出:

```
list
```

代码:

```
tensor_f = tf.convert_to_tensor(list_f, dtype=tf.float32)
tensor_f
```

输出:

```
<tf.Tensor: shape=(6,), dtype=float32, numpy=array([1., 2., 3., 4., 5., 6.], dtype=float32)>
```

1.2.1.1.2 创建变量 tensor

TensorFlow 中, 变量通过 tf.Variable 类进行操作。tf.Variable 表示张量, 其值可以通过在其上运行算术运算更改。可读取和修改变量值。

代码:

```
# 创建变量, 只需提供初始值
var_1 = tf.Variable(tf.ones([2,3]))
var_1
```

输出:

```
<tf.Variable 'Variable:0' shape=(2, 3) dtype=float32, numpy=
array([[1., 1., 1.],
       [1., 1., 1.]], dtype=float32)>
```

代码：

```
#变量数值读取
print("变量 var_1 的数值: ",var_1.read_value())
#变量赋值
var_value_1=[[1,2,3],[4,5,6]]
var_1.assign(var_value_1)
print("变量 var_1 赋值后的数值: ",var_1.read_value())
```

输出：

```
变量 var_1 的数值:  tf.Tensor(
[[1. 1. 1.]
 [1. 1. 1.]], shape=(2, 3), dtype=float32)
变量 var_1 赋值后的数值:  tf.Tensor(
[[1. 2. 3.]
 [4. 5. 6.]], shape=(2, 3), dtype=float32)
```

代码：

```
#变量加法
var_1.assign_add(tf.ones([2,3]))
var_1
```

输出：

```
<tf.Variable 'Variable:0' shape=(2, 3) dtype=float32, numpy=
array([[2., 3., 4.],
       [5., 6., 7.]], dtype=float32)>
```

1.2.1.2 tensor 切片与索引

1.2.1.2.1 切片

切片的方式主要有：

- [start: end]: 从 tensor 的开始位置到结束位置的数据切片；
- [start :end :step]或者[:, :, step]: 从 tensor 的开始位置到结束位置每隔 step 的数据切片；
- [::-1]: 负数表示倒序切片；
- ‘...’ : 任意长。

代码：

```
#创建一个 4 维 tensor。tensor 包含 4 张图片，每张图片的大小为 100*100*3
tensor_h = tf.random.normal([4,100,100,3])
tensor_h
```

输出：

```
<tf.Tensor: shape=(4, 100, 100, 3), dtype=float32, numpy=
array([[[[ 1.68444023e-01, -7.46562362e-01, -4.34964240e-01],
          [-4.69263226e-01,  6.26460612e-01,  1.21065331e+00],
          [ 7.21675277e-01,  4.61057723e-01, -9.20868576e-01],
          ...,

```

代码：

```
#取出第一张图片
tensor_h[0,:,:,:]
```

输出：

```
<tf.Tensor: shape=(100, 100, 3), dtype=float32, numpy=
array([[[[ 1.68444023e-01, -7.46562362e-01, -4.34964240e-01],
          [-4.69263226e-01,  6.26460612e-01,  1.21065331e+00],
          [ 7.21675277e-01,  4.61057723e-01, -9.20868576e-01],
          ...,

```


代码：

```
#每两张图片取出一张的切片
```

```
tensor_h[::2,...]
```

输出：

```
<tf.Tensor: shape=(2, 100, 100, 3), dtype=float32, numpy=
array([[[[ 1.68444023e-01, -7.46562362e-01, -4.34964240e-01],
          [-4.69263226e-01,  6.26460612e-01,  1.21065331e+00],
          [ 7.21675277e-01,  4.61057723e-01, -9.20868576e-01],
          ...,

```

代码：

```
#倒序切片
```

```
tensor_h[::-1]
```

输出：

```
<tf.Tensor: shape=(4, 100, 100, 3), dtype=float32, numpy=
array([[[[-1.70684665e-01,  1.52386248e+00, -1.91677585e-01],
          [-1.78917408e+00, -7.48436213e-01,  6.10363662e-01],
          [ 7.64770031e-01,  6.06725179e-02,  1.32704067e+00],
          ...,

```

1.2.1.2.2 索引

索引的基本格式：a[d1][d2][d3]

代码：

```
#取出第一张图片第二个通道中在[20,40]位置的像素点
```

```
tensor_h[0][19][39][1]
```

输出：

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.38231283>
```

如果要提取的索引不连续的话，在 TensorFlow 中，常见的用法为 `tf.gather` 和 `tf.gather_nd`。

在某一维度进行索引。

`tf.gather(params, indices, axis=None):`

- `params`: 输入张量;
- `indices`: 取出数据的索引;
- `axis`: 所取数据所在维度。

代码:

```
#取出 tensor_h ( [4,100,100,3] ) 中, 第 1, 2, 4 张图像。
indices = [0,1,3]
tf.gather(tensor_h,axis=0,indices=indices,batch_dims=1)
```

输出:

```
<tf.Tensor: shape=(3, 100, 100, 3), dtype=float32, numpy=
array([[[[ 1.68444023e-01, -7.46562362e-01, -4.34964240e-01],
          [-4.69263226e-01,  6.26460612e-01,  1.21065331e+00],
          [ 7.21675277e-01,  4.61057723e-01, -9.20868576e-01],
          ...,

```

`tf.gather_nd` 允许在多维上进行索引: `tf.gather_nd(params, indices):`

- `params`: 输入张量;
- `indices`: 取出数据的索引, 一般为多维列表。

代码:

```
#取出 tensor_h([4,100,100,3])中, 第一张图像第一个维度中[1,1]的像素点; 第二张图片第一像素点中[2,2]的像素点
indices = [[0,1,1,0],[1,2,2,0]]
tf.gather_nd(tensor_h,indices=indices)
```

输出:

```
<tf.Tensor: shape=(2,), dtype=float32, numpy=array([0.5705869, 0.9735735], dtype=float32)>
```

1.2.1.3 张量的维度变化

1.2.1.3.1 维度查看

代码：

```
const_d_1 = tf.constant([[1, 2, 3, 4]],shape=[2,2], dtype=tf.float32)
#查看维度常用的三种方式
print(const_d_1.shape)
print(const_d_1.get_shape())
print(tf.shape(const_d_1))#输出为张量，其数值表示的是所查看张量维度大小
```

输出：

```
(2, 2)
(2, 2)
tf.Tensor([2 2], shape=(2,), dtype=int32)
```

可以看出.shape 和.get_shape()都是返回 TensorShape 类型对象，而 tf.shape(x)返回的是 Tensor 类型对象。

1.2.1.3.2 维度重组

tf.reshape(tensor,shape,name=None):

- tensor：输入张量；
- shape：重组后张量的维度。

代码：

```
reshape_1 = tf.constant([[1,2,3],[4,5,6]])
print(reshape_1)
tf.reshape(reshape_1, (3,2))
```

输出：

```
<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
array([[1, 2],
       [3, 4],
       [5, 6]], dtype=int32)>
```

1.2.1.3.3 维度增加

`tf.expand_dims(input,axis,name=None):`

- `input`: 输入张量;
- `axis`: 在第 `axis` 维度后增加一个维度。在输入 `D` 尺寸的情况下, 轴必须在 $[-(D + 1), D]$ (含) 范围内。负数代表倒序。

代码:

```
#生成一个大小为 100*100*3 的张量来表示一张尺寸为 100*100 的三通道彩色图片
expand_sample_1 = tf.random.normal([100,100,3], seed=1)
print("原始数据尺寸: ",expand_sample_1.shape)
print("在第一个维度前增加一个维度(axis=0): ",tf.expand_dims(expand_sample_1, axis=0).shape)
print("在第二个维度前增加一个维度(axis=1): ",tf.expand_dims(expand_sample_1, axis=1).shape)
print("在最后一个维度后增加一个维度(axis=-1): ",tf.expand_dims(expand_sample_1, axis=-1).shape)
```

输出:

```
原始数据尺寸: (100, 100, 3)
在第一个维度前增加一个维度(axis=0): (1, 100, 100, 3)
在第二个维度前增加一个维度(axis=1): (100, 1, 100, 3)
在最后一个维度后增加一个维度(axis=-1): (100, 100, 3, 1)
```

1.2.1.3.4 维度减少

`tf.squeeze(input,axis=None,name=None):`

- `input`: 输入张量;
- `axis`: `axis=1`, 表示要删掉的为 1 的维度。

代码：

```
#生成一个大小为 100*100*3 的张量来表示一张尺寸为 100*100 的三通道彩色图片
squeeze_sample_1 = tf.random.normal([1,100,100,3])
print("原始数据尺寸: ",squeeze_sample_1.shape)
squeezed_sample_1 = tf.squeeze(expand_sample_1)
print("维度压缩后的数据尺寸: ",squeezed_sample_1.shape)
```

输出：

```
原始数据尺寸: (1, 100, 100, 3)
维度压缩后的数据尺寸: (100, 100, 3)
```

1.2.1.3.5 转置

`tf.transpose(a,perm=None,conjugate=False,name='transpose')`:

- a: 输入张量；
- perm: 张量的尺寸排列；一般用于高维数组的转置。
- conjugate: 表示复数转置；
- name: 名称。

代码：

```
#低维的转置问题比较简单，输入需转置张量调用 tf.transpose
trans_sample_1 = tf.constant([1,2,3,4,5,6],shape=[2,3])
print("原始数据尺寸: ",trans_sample_1.shape)
transposed_sample_1 = tf.transpose(trans_sample_1)
print("转置后数据尺寸: ",transposed_sample_1.shape)
```

输出：

```
原始数据尺寸: (2, 3)
转置后数据尺寸: (3, 2)
```

代码：

"高维数据转置需要用到 perm 参数，perm 代表输入张量的维度排列。

对于一个三维张量来说，其原始的维度排列为[0,1,2]（perm）分别代表高维数据的长宽高。

通过改变 perm 中数值的排列，可以对数据的对应维度进行转置"

#生成一个大小为4*100*200*3的张量来表示4张尺寸为100*200的三通道彩色图片

```
trans_sample_2 = tf.random.normal([4,100,200,3])
```

```
print("原始数据尺寸: ",trans_sample_2.shape)
```

#对4张图像的长宽进行对调。原始perm为[0,1,2,3]，现变为[0,2,1,3]

```
transposed_sample_2 = tf.transpose(trans_sample_2,[0,2,1,3])
```

```
print("转置后数据尺寸: ",transposed_sample_2.shape)
```

输出：

原始数据尺寸： (4, 100, 200, 3)

转置后数据尺寸： (4, 200, 100, 3)

1.2.1.3.6 广播（broadcast_to）

利用把 broadcast_to 可以将小维度推广到大维度。

tf.broadcast_to(input,shape,name=None):

- input：输入张量；
- shape：输出张量的尺寸。

代码：

```
broadcast_sample_1 = tf.constant([1,2,3,4,5,6])
```

```
print("原始数据: ",broadcast_sample_1.numpy())
```

```
broadcasted_sample_1 = tf.broadcast_to(broadcast_sample_1,shape=[4,6])
```

```
print("广播后数据: ",broadcasted_sample_1.numpy())
```

输出：

原始数据: [1 2 3 4 5 6]

广播后数据: [[1 2 3 4 5 6]

[1 2 3 4 5 6]

[1 2 3 4 5 6]

[1 2 3 4 5 6]]

代码:

#运算时, 当两个数组的形状不同时, 与 numpy 一样, TensorFlow 将自动触发广播机制。

```
a = tf.constant([[ 0, 0, 0],
                 [10,10,10],
                 [20,20,20],
                 [30,30,30]])
b = tf.constant([1,2,3])
print(a + b)
```

输出:

```
tf.Tensor(
[[ 1  2  3]
 [11 12 13]
 [21 22 23]
 [31 32 33]], shape=(4, 3), dtype=int32)
```

1.2.1.4 张量的算术运算

1.2.1.4.1 算术运算符

算术运算主要包括了: 加(tf.add)、减(tf.subtract)、乘(tf.multiply)、除(tf.divide)、取对数 (tf.math.log) 和指数 (tf.pow) 等。因为调用比较简单, 下面只演示一个加法例子。

代码:

```
a = tf.constant([[3, 5], [4, 8]])
b = tf.constant([[1, 6], [2, 9]])
print(tf.add(a, b))
```

输出：

```
tf.Tensor(  
[[ 4 11]  
 [ 6 17]], shape=(2, 2), dtype=int32)
```

1.2.1.4.2 矩阵乘法运算

矩阵乘法运算的实现通过调用 `tf.matmul`。

代码：

```
tf.matmul(a,b)
```

输出：

```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=  
array([[13, 63],  
       [20, 96]], dtype=int32)>
```

1.2.1.4.3 张量的数据统计

张量的数据统计主要包括：

- `tf.reduce_min/max/mean()`：求解最小值最大值和均值函数；
- `tf.argmax()/tf.argmin()`：求最大最小值位置；
- `tf.equal()`：逐个元素判断两个张量是否相等；
- `tf.unique()`：除去张量中的重复元素。
- `tf.nn.in_top_k(prediction, target, K)`：用于计算预测值和真是值是否相等，返回一个 `bool` 类型的张量。

下面演示 `tf.argmax()` 的用法：

返回最大值所在的下标

- `tf.argmax(input,axis)`：
- `input`：输入张量；
- `axis`：按照 `axis` 维度，输出最大值。

代码：

```
argmax_sample_1 = tf.constant([[1,3,2],[2,5,8],[7,5,9]])
print("输入张量: ",argmax_sample_1.numpy())
max_sample_1 = tf.argmax(argmax_sample_1, axis=0)
max_sample_2 = tf.argmax(argmax_sample_1, axis=1)
print("按列寻找最大值的位置: ",max_sample_1.numpy())
print("按行寻找最大值的位置: ",max_sample_2.numpy())
```

输出：

```
输入张量: [[1 3 2]
[2 5 8]
[7 5 9]]
按列寻找最大值的位置: [2 1 2]
按行寻找最大值的位置: [1 2 2]
```

1.2.1.5 基于维度的算术操作

TensorFlow 中，`tf.reduce_*`一系列操作等都造成张量维度的减少。这一系列操作都可以对一个张量在维度上的元素进行操作，如按行求平均，求取张量中所有元素的乘积等。

常用的包括：`tf.reduce_sum`(加法)、`tf.reduce_prod`（乘法）、`tf.reduce_min`（最小）、`tf.reduce_max`（最大）、`tf.reduce_mean`（均值）、`tf.reduce_all`（逻辑和）、`tf.reduce_any`（逻辑或）和 `tf.reduce_logsumexp`（`log(sum(exp))`操作）等。

这些操作的使用方法都相似，下面只演示 `tf.reduce_sum` 的操作案例。

计算一个张量的各个维度上元素的总和

`tf.reduce_sum(input_tensor, axis=None, keepdims=False, name=None)`:

- `input_tensor`: 输入张量；
- `axis`: 指定需要计算的轴，如果不指定，则计算所有元素的均值；
- `keepdims`: 是否降维度，设置为 `True`，输出的结果保持输入 `tensor` 的形状，设置为 `False`，输出结果会降低维度；
- `name`: 操作名称。

代码：

```
reduce_sample_1 = tf.constant([1,2,3,4,5,6],shape=[2,3])
print("原始数据",reduce_sample_1.numpy())
print("计算张量中所有元素的和 ( axis=None ) : ",tf.reduce_sum(reduce_sample_1,axis=None).numpy())
print("按列计算，分别计算各列的和 ( axis=0 ) : ",tf.reduce_sum(reduce_sample_1,axis=0).numpy())
print("按行计算，分别计算各列的和 ( axis=1 ) : ",tf.reduce_sum(reduce_sample_1,axis=1).numpy())
```

输出：

```
原始数据 [[1 2 3]
 [4 5 6]]
计算张量中所有元素的和 ( axis=None ) :  21
按行计算，取出行，分别计算各列的和 ( axis=0 ) :  [5 7 9]
按列计算，取出列，分别计算各列的和 ( axis=1 ) :  [ 6 15]
```

1.2.1.6 张量的拼接与分割

1.2.1.6.1 张量的拼接

TensorFlow 中，张量拼接的操作主要包括：

- `tf.concat()`：将向量按指定维连起来，其余维度不变。
- `tf.stack()`：将一组 R 维张量变为 R+1 维张量，拼接前后维度变化。

`tf.concat(values, axis, name='concat')`:

- `values`：输入张量；
- `axis`：指定拼接维度；
- `name`：操作名称。

代码：

```
concat_sample_1 = tf.random.normal([4,100,100,3])
concat_sample_2 = tf.random.normal([40,100,100,3])
print("原始数据的尺寸分别为: ",concat_sample_1.shape,concat_sample_2.shape)
concat_sample_1 = tf.concat([concat_sample_1,concat_sample_2],axis=0)
print("拼接后数据的尺寸: ",concat_sample_1.shape)
```

输出：

```
原始数据的尺寸分别为: (4, 100, 100, 3) (40, 100, 100, 3)
拼接后数据的尺寸: (44, 100, 100, 3)
```

在原来矩阵基础上增加了一个维度，也是同样的道理，axis 决定维度增加的位置。

tf.stack(values, axis=0, name='stack'):

- values: 输入张量；一组相同形状和数据类型的张量。
- axis: 指定拼接维度；
- name: 操作名称。

代码：

```
stack_sample_1 = tf.random.normal([100,100,3])
stack_sample_2 = tf.random.normal([100,100,3])
print("原始数据的尺寸分别为: ",stack_sample_1.shape, stack_sample_2.shape)
#拼接后维度增加。axis=0，则在第一个维度前增加维度。
stacked_sample_1 = tf.stack([stack_sample_1, stack_sample_2],axis=0)
print("拼接后数据的尺寸: ",stacked_sample_1.shape)
```

输出：

```
原始数据的尺寸分别为: (100, 100, 3) (100, 100, 3)
拼接后数据的尺寸: (2, 100, 100, 3)
```

1.2.1.6.2 张量的分割

TensorFlow 中，张量分割的操作主要包括：

- `tf.unstack()`: 将张量按照特定维度分解。
- `tf.split()`: 将张量按照特定维度划分为指定的分数。

与 `tf.unstack()` 相比，`tf.split()` 更佳灵活。

```
tf.unstack(value,num=None,axis=0,name='unstack'):
```

- `value`: 输入张量;
- `num`: 表示输出含有 `num` 个元素的列表, `num` 必须和指定维度内元素的个数相等。通常可以忽略不写这个参数。
- `axis`: 指明根据数据的哪个维度进行分割;
- `name`: 操作名称。

代码：

#按照第一个维度对数据进行分解，分解后的数据以列表形式输出。

```
tf.unstack(stacked_sample_1,axis=0)
```

输出:

```
[<tf.Tensor: shape=(100, 100, 3), dtype=float32, numpy=
array([[[[ 0.0665694 ,  0.7110351 ,  1.907618 ],
          [ 0.84416866,  1.5470593 , -0.5084871 ],
          [-1.9480026 , -0.9899087 , -0.09975405],
          ...,
```

```
tf.split(value, num_or_size_splits, axis=0):
```

- value: 输入张量;
- num_or_size_splits: 准备切成几份
- axis: 指明根据数据的哪个维度进行分割。

tf.split()的分割方式有两种:

1. 如果 num_or_size_splits 传入的是一个整数，那直接在 axis=D 这个维度上把张量平均切分成几个小张量。
2. 如果 num_or_size_splits 传入的是一个向量，则在 axis=D 这个维度上把张量按照向量的元素值切分成几个小张量。

代码：

```
import numpy as np
split_sample_1 = tf.random.normal([10,100,100,3])
print("原始数据的尺寸为: ",split_sample_1.shape)
splited_sample_1 = tf.split(split_sample_1, num_or_size_splits=5,axis=0)
print("当 m_or_size_splits=10，分割后数据的尺寸为: ",np.shape(splited_sample_1))
splited_sample_2 = tf.split(split_sample_1, num_or_size_splits=[3,5,2],axis=0)
print("当 num_or_size_splits=[3,5,2]，分割后数据的尺寸分别为: ",
      np.shape(splited_sample_2[0]),
      np.shape(splited_sample_2[1]),
      np.shape(splited_sample_2[2]))
```

输出：

```
原始数据的尺寸为: (10, 100, 100, 3)
当 m_or_size_splits=10，分割后数据的尺寸为: (5, 2, 100, 100, 3)
当 num_or_size_splits=[3,5,2]，分割后数据的尺寸分别为: (3, 100, 100, 3) (5, 100, 100, 3) (2, 100, 100, 3)
```

1.2.1.7 张量排序

TensorFlow 中，张量排序的操作主要包括：

- tf.sort(): 按照升序或者降序对张量进行排序，返回排序后的张量。
- tf.argsort(): 按照升序或者降序对张量进行排序,但返回的是索引。
- tf.nn.top_k(): 返回前 k 个最大值。

tf.sort/argsort(input, direction, axis):

- input: 输入张量；

- direction: 排列顺序, 可为 DESCENDING 降序或者 ASCENDING (升序)。默认为 ASCENDING (升序);
- axis: 按照 axis 维度进行排序。默认 axis=-1 最后一个维度。

代码:

```
sort_sample_1 = tf.random.shuffle(tf.range(10))
print("输入张量: ",sort_sample_1.numpy())
sorted_sample_1 = tf.sort(sort_sample_1, direction="ASCENDING")
print("生序排列后的张量: ",sorted_sample_1.numpy())
sorted_sample_2 = tf.argsort(sort_sample_1,direction="ASCENDING")
print("生序排列后, 元素的索引: ",sorted_sample_2.numpy())
```

输出:

```
输入张量: [1 8 7 9 6 5 4 2 3 0]
生序排列后的张量: [0 1 2 3 4 5 6 7 8 9]
生序排列后, 元素的索引: [9 0 7 8 6 5 4 2 1 3]
```

tf.nn.top_k(input,K,sorted=TRUE):

- input: 输入张量;
- K: 需要输出的前 k 个值及其索引。
- sorted: sorted=TRUE 表示升序排列; sorted=FALSE 表示降序排列。

返回两个张量:

- values: 也就是每一行的最大的 k 个数字
- indices: 这里的下标是在输入的张量的最后一个维度的下标

代码:

```
values, index = tf.nn.top_k(sort_sample_1,5)
print("输入张量: ",sort_sample_1.numpy())
print("升序排列后的前 5 个数值: ", values.numpy())
print("升序排列后的前 5 个数值的索引: ", index.numpy())
```

输出:

```
输入张量: [1 8 7 9 6 5 4 2 3 0]
升序排列后的前 5 个数值: [9 8 7 6 5]
升序排列后的前 5 个数值的索引: [3 1 2 4 5]
```

1.2.2 TensorFlow2 Eager Execution 模式

Eager Execution 介绍:

TensorFlow 的 Eager Execution 模式是一种命令式编程 (imperative programming)，这和原生 Python 是一致的，当你执行某个操作时，可以立即返回结果的。

Graph 模式介绍:

TensorFlow1.0 一直是采用 Graph 模式，即先构建一个计算图，然后需要开启 Session，喂进实际的数据才真正执行得到结果。

Eager Execution 模式下，我们可以更容易 debug 代码，但是代码的执行效率更低。

下面我们在 Eager Execution 和 Graph 模式下，用 TensorFlow 实现简单的乘法，来对比两个模式的区别。

代码:

```
x = tf.ones((2, 2), dtype=tf.dtypes.float32)
y = tf.constant([[1, 2],
                 [3, 4]], dtype=tf.dtypes.float32)
z = tf.matmul(x, y)
print(z)
```

输出：

```
tf.Tensor(  
[[4. 6.]  
 [4. 6.]], shape=(2, 2), dtype=float32)
```

代码：

```
#在 TensorFlow 2 版本中使用 1.X 版本的语法；可以使用 2.0 中的 v1 兼容包来沿用 1.x 代码，并在代码中关闭 eager  
运算。  
import TensorFlow.compat.v1 as tf  
tf.disable_eager_execution()  
#创建 graph，定义计算图  
a = tf.ones((2, 2), dtype=tf.dtypes.float32)  
b = tf.constant([[1, 2],  
                 [3, 4]], dtype=tf.dtypes.float32)  
c = tf.matmul(a, b)  
#开启绘画，进行运算后，才能取出数据。  
with tf.Session() as sess:  
    print(sess.run(c))
```

输出：

```
[[4. 6.]  
 [4. 6.]]
```

首先重启一下 kernel，使得 TensorFlow 恢复到 2.0 版本并打开 eager execution 模式。Eager Execution 模式的另一个优点是可以使用 Python 原生功能，比如下面的条件判断：

代码：


```
import TensorFlow as tf
thre_1 = tf.random.uniform([], 0, 1)
x = tf.reshape(tf.range(0, 4), [2, 2])
print(thre_1)
if thre_1.numpy() > 0.5:
    y = tf.matmul(x, x)
else:
    y = tf.add(x, x)
```

输出：

```
tf.Tensor(0.11304152, shape=(), dtype=float32)
```

这种动态控制流主要得益于 eager 执行得到 Tensor 可以取出 numpy 值，这避免了使用 Graph 模式下的 tf.cond 和 tf.while 等算子。

1.2.3 TensorFlow2 AutoGraph

当使用 tf.function 装饰器注释函数时，可以像调用任何其他函数一样调用它。它将被编译成图，这意味着可以获得更高效地在 GPU 或 TPU 上运行。此时函数变成了一个 TensorFlow 中的 operation。我们可以直接调用函数，输出返回值，但是函数内部是在 graph 模式下执行的，无法直接查看中间变量数值

代码：

```
@tf.function
def simple_nn_layer(w,x,b):
    print(b)
    return tf.nn.relu(tf.matmul(w, x)+b)

w = tf.random.uniform((3, 3))
x = tf.random.uniform((3, 3))
b = tf.constant(0.5, dtype='float32')

simple_nn_layer(w,x,b)
```

输出：

```
Tensor("b:0", shape=(), dtype=float32)
<tf.Tensor: shape=(3, 3), dtype=float32, numpy=
array([[1.4121541, 1.1626956, 1.2527422],
       [1.2903953, 1.0956903, 1.1309073],
       [1.1039395, 0.92851776, 1.0752096]], dtype=float32)>
```

通过输出结果可知，无法直接查看函数内部 b 的数值，而返回值可以通过.numpy()查看。

通过相同的操作（执行一层 lstm 计算），比较 graph 和 eager execution 模式的性能。

代码：

```
#timeit 测量小段代码的执行时间
import timeit
#创建一个卷积层。
CNN_cell = tf.keras.layers.Conv2D(filters=100,kernel_size=2,strides=(1,1))

#利用@tf.function，将操作转化为 graph。
@tf.function
def CNN_fn(image):
    return CNN_cell(image)

image = tf.zeros([100, 200, 200, 3])

#比较两者的执行时间
CNN_cell(image)
CNN_fn(image)
#调用 timeit.timeit，测量代码执行 10 次的时间
print("eager execution 模式下做一层 CNN 卷积层运算的时间:", timeit.timeit(lambda: CNN_cell(image),
number=10))
print("graph 模式下做一层 CNN 卷积层运算的时间:", timeit.timeit(lambda: CNN_fn(image), number=10))
```

输出：

```
eager execution 模式下做一层 CNN 卷积层运算的时间: 18.26327505100926
graph 模式下做一层 CNN 卷积层运算的时间: 6.740775318001397
```

通过比较，我们可以发现 graph 模式下代码执行效率要高出许多。因此我们以后，可以多尝试用 @tf.function 功能，提高代码运行效率。

2 TensorFlow 2 常用模块介绍

2.1 实验介绍

本节将为大家介绍 TensorFlow 2 常用模块，主要包括：

- `tf.data`：实现对数据集的操作；
包括读取从内存中直接读取数据集、读取 CSV 文件、读取 `tfrecord` 文件和数据增强等。
- `tf.image`：实现对图像处理的操作；
包括图像亮度变换、饱和度变换、图像尺寸变换、图像旋转和边缘检测等操作。
- `tf.gfile`：实现对文件的操作；
包括对文件的读写操作、文件重命名和文件夹操作等。
- `tf.keras`：用于构建和训练深度学习模型的高阶 API；
- `tf.distributions` 等等。

本节我们将重点聚焦到 `tf.keras` 模块，为后面深度学习建模打下基础。

2.2 实验目的

掌握 `tf.keras` 中常用的深度学习建模接口。

2.3 实验步骤

2.3.1 模型构建

2.3.1.1 模型堆叠（`tf.keras.Sequential`）

最常见的模型构建方法是层的堆叠，我们通常会使用 `tf.keras.Sequential`。

代码：

```
import TensorFlow.keras.layers as layers
model = tf.keras.Sequential()
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

2.3.1.2 函数式模型构建

函数式模型主要利用 `tf.keras.Input` 和 `tf.keras.Model` 构建，比 `tf.keras.Sequential` 模型要复杂，但是效果很好，可以同时/分阶段输入变量，分阶段输出数据； 你的模型需要多于一个的输出，那么需要选择函数式模型。

模型堆叠（`.Sequential`）vs 函数式模型（`Model`）：

`tf.keras.Sequential` 模型是层的简单堆叠，无法表示任意模型。使用 Keras 的函数式模型可以构建复杂的模型拓扑，例如：

- 多输入模型；
- 多输出模型；
- 具有共享层的模型；
- 具有非序列数据流的模型（例如，残差连接）。

代码：

```
# 以上一层的输出作为下一层的输入
x = tf.keras.Input(shape=(32,))
h1 = layers.Dense(32, activation='relu')(x)
h2 = layers.Dense(32, activation='relu')(h1)
y = layers.Dense(10, activation='softmax')(h2)
model_sample_2 = tf.keras.models.Model(x, y)

#打印模型信息
model_sample_2.summary()
```

输出：

Model: "model"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 32)]	0
dense_3 (Dense)	(None, 32)	1056
dense_4 (Dense)	(None, 32)	1056
dense_5 (Dense)	(None, 10)	330
=====		
Total params: 2,442		
Trainable params: 2,442		
Non-trainable params: 0		

2.3.1.3 网络层构建（tf.keras.layers）

tf.keras.layers 模块的主要作用为配置神经网络层。其中常用的类包括：

- tf.keras.layers.Dense：构建全连接层；
- tf.keras.layers.Conv2D：构建 2 维卷积层；
- tf.keras.layers.MaxPooling2D/AveragePooling2D：构建最大/平均池化层；
- tf.keras.layers.RNN：构建循环神经网络层；
- tf.keras.layers.LSTM/tf.keras.layers.LSTMCell：构建 LSTM 网络层/LSTM unit；
- tf.keras.layers.GRU/tf.keras.layers.GRUCell：构建 GRU unit/GRU 网络层；
- tf.keras.layers.Embedding 嵌入层将正整数（下标）转换为具有固定大小的向量，如
[[4],[20]]->[[0.25,0.1],[0.6,-0.2]]。Embedding 层只能作为模型的第一层；
- tf.keras.layers.Dropout：构建 dropout 层等。

下面主要讲解 `tf.keras.layers.Dense`、`tf.keras.layers.Conv2D`、

`tf.keras.layers.MaxPooling2D/AveragePooling2D` 和

`tf.keras.layers.LSTM/tf.keras.layers.LSTMCell`。

`tf.keras.layers` 中主要的网络配置参数如下：

- `activation`：设置层的激活函数。默认情况下，系统不会应用任何激活函数。
- `kernel_initializer` 和 `bias_initializer`：创建层权重（核和偏置）的初始化方案。默认为 "Glorot uniform" 初始化器。
- `kernel_regularizer` 和 `bias_regularizer`：应用层权重（核和偏置）的正则化方案，例如 L1 或 L2 正则化。默认情况下，系统不会应用正则化函数。

2.3.1.3.1 `tf.keras.layers.Dense`

`tf.keras.layers.Dense` 可配置的参数，主要有：

- `units`：神经元个数；
- `activation`：激活函数；
- `use_bias`：是否使用偏置项。默认为使用；
- `kernel_initializer`：创建层权重核的初始化方案；
- `bias_initializer`：创建层权重偏置的初始化方案；
- `kernel_regularizer`：应用层权重核的正则化方案；
- `bias_regularizer`：应用层权重偏置的正则化方案；
- `activity_regularizer`：施加在输出上的正则项，为 `Regularizer` 对象；
- `kernel_constraint`：施加在权重上的约束项；
- `bias_constraint`：施加在权重上的约束项。

代码：

```
#创建包含 32 个神经元的全连接层，其中的激活函数设置为 sigmoid。
#activation 参数可以是函数名称字符串，如'sigmoid'；也可以是函数对象，如 tf.sigmoid。
layers.Dense(32, activation='sigmoid')
layers.Dense(32, activation=tf.sigmoid)

#设置 kernel_initializer 参数
layers.Dense(32, kernel_initializer=tf.keras.initializers.he_normal)
#设置 kernel_regularizer 为 L2 正则
layers.Dense(32, kernel_regularizer=tf.keras.regularizers.l2(0.01))
```

输出：

```
<TensorFlow.python.keras.layers.core.Dense at 0x130c519e8>
```

2.3.1.3.2 tf.keras.layers.Conv2D

tf.keras.layers.Conv2D 可配置的参数，主要有：

- filters：卷积核的数目（即输出的维度）；
- kernel_size：卷积核的宽度和长度；
- strides：卷积的步长。
- padding：补 0 策略。
 - padding=“valid”代表只进行有效的卷积，即对边界数据不处理。padding=“same”代表保留边界处的卷积结果，通常会导致输出 shape 与输入 shape 相同；
- activation：激活函数；
- data_format：数据格式，为“channels_first”或“channels_last”之一。以 128x128 的 RGB 图像为例，“channels_first”应将数据组织为（3,128,128），而“channels_last”应将数据组织为（128,128,3）。该参数的默认值是~/.keras/keras.json 中设置的值，若从未设置过，则为“channels_last”。
- 其他参数还包括：use_bias；kernel_initializer；bias_initializer；kernel_regularizer；bias_regularizer；activity_regularizer；kernel_constraints；bias_constraints。

代码：

```
layers.Conv2D(64,[1,1],2,padding='same',activation="relu")
```

输出：

```
<TensorFlow.python.keras.layers.convolutional.Conv2D at 0x106c510f0>
```

2.3.1.3.3 tf.keras.layers.MaxPooling2D/AveragePooling2D

tf.keras.layers.MaxPooling2D/AveragePooling2D 可配置的参数，主要有：

- pool_size：池化 kernel 的大小。如取矩阵（2，2）将使图片在两个维度上均变为原长的一半。为整数意为各个维度值都为该数字。
- strides：步长值。
- 其他参数还包括：padding；data_format。

代码：

```
layers.MaxPooling2D(pool_size=(2,2),strides=(2,1))
```

输出：

```
<TensorFlow.python.keras.layers.pooling.MaxPooling2D at 0x132ce1f98>
```

2.3.1.3.4 tf.keras.layers.LSTM/tf.keras.layers.LSTMCell

tf.keras.layers.LSTM/tf.keras.layers.LSTMCell 可配置的参数，主要有：

- units：输出维度；
- input_shape (timestep, input_dim), timestep 可以设置为 None, input_dim 为输入数据维度；
- activation：激活函数；
- recurrent_activation：为循环步施加的激活函数；
- return_sequences：=True 时，返回全部序列；=False 时，返回输出序列中的最后一个 cell 的输出；
- return_state：布尔值。除了输出之外是否返回最后一个状态；
- dropout：0~1 之间的浮点数，控制输入线性变换的神经元断开比例；

- recurrent_dropout: 0~1 之间的浮点数，控制循环状态的线性变换的神经元断开比例。

代码：

```
import numpy as np
inputs = tf.keras.Input(shape=(3, 1))
lstm = layers.LSTM(1, return_sequences=True)(inputs)
model_lstm_1 = tf.keras.models.Model(inputs=inputs, outputs=lstm)

inputs = tf.keras.Input(shape=(3, 1))
lstm = layers.LSTM(1, return_sequences=False)(inputs)
model_lstm_2 = tf.keras.models.Model(inputs=inputs, outputs=lstm)

# t1, t2, t3 序列
data = [[[0.1],
         [0.2],
         [0.3]]]
print(data)
print("当 return_sequences=True 时的输出", model_lstm_1.predict(data))
print("当 return_sequences=False 时的输出", model_lstm_2.predict(data))
```

输出：

```
[[[0.1], [0.2], [0.3]]]
当 return_sequences=True 时的输出 [[[-0.0106758 ]
  [-0.02711176]
  [-0.04583194]]]
当 return_sequences=False 时的输出 [[0.05914127]]
```

LSTMcell 是 LSTM 层的实现单元。

- LSTM 是一个 LSTM 网络层
- LSTMCell 是一个单步的计算单元，即一个 LSTM UNIT。

```
#LSTM
tf.keras.layers.LSTM(16, return_sequences=True)

#LSTMCell
x = tf.keras.Input((None, 3))
y = layers.RNN(layers.LSTMCell(16))(x)
model_lstm_3= tf.keras.Model(x, y)
```

2.3.2 训练与评估

2.3.2.1 模型编译，确定训练流程。

构建好模型后，通过调用 compile 配置该模型的学习流程：

- compile(optimizer='rmsprop', loss=None, metrics=None, loss_weights=None):
- optimizer: 优化器；
- loss: 损失函数，对于二分类任务就是交叉熵，回归任务就是 mse 之类的；
- metrics: 在训练和测试期间的模型评估标准。比如 metrics = ['accuracy']。指定不同的评估标准，需要传递一个字典，如 metrics = {'output_a': 'accuracy'}。
- loss_weights: 如果的模型有多个任务输出，在优化全局 loss 的时候，需要给每个输出指定相应的权重。

代码：

```
model = tf.keras.Sequential()
model.add(layers.Dense(10, activation='softmax'))
#确定优化器（optimizer）、损失函数（loss）、模型评估方法（metrics）
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss=tf.keras.losses.categorical_crossentropy,
              metrics=[tf.keras.metrics.categorical_accuracy])
```

2.3.2.2 模型训练

`fit(x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None, validation_split=0.0, validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None, validation_steps=None):`

- `x`: 输入训练数据;
- `y`: 目标 (标签) 数据;
- `batch_size`: 每次梯度更新的样本数。如果未指定, 默认为 32;
- `epochs`: 训练模型迭代轮次;
- `verbose`: 0, 1 或 2。日志显示模式。 0 = 不显示, 1 = 进度条, 2 = 每轮显示一行;
- `callbacks`: 在训练时使用的回调函数;
- `validation_split`: 验证集与训练数据的比例;
- `validation_data`: 验证集; 这个参数会覆盖 `validation_split`;
- `shuffle`: 是否在每轮迭代之前混洗数据。当 `steps_per_epoch` 非 None 时, 这个参数无效;
- `initial_epoch`: 开始训练的轮次, 常用于恢复之前的训练权重;
- `steps_per_epoch`: `steps_per_epoch = 数据集大小 / batch_size`;
- `validation_steps`: 只有在指定了 `steps_per_epoch` 时才有用。停止前要验证的总步数 (批次样本)。

代码:

```
import numpy as np

train_x = np.random.random((1000, 36))
train_y = np.random.random((1000, 10))

val_x = np.random.random((200, 36))
val_y = np.random.random((200, 10))

model.fit(train_x, train_y, epochs=10, batch_size=100,
          validation_data=(val_x, val_y))
```

输出：

```
Train on 1000 samples, validate on 200 samples
Epoch 1/10
1000/1000 [=====] - 0s 488us/sample - loss: 12.6024 -
categorical_accuracy: 0.0960 - val_loss: 12.5787 - val_categorical_accuracy: 0.0850
Epoch 2/10
1000/1000 [=====] - 0s 23us/sample - loss: 12.6007 -
categorical_accuracy: 0.0960 - val_loss: 12.5776 - val_categorical_accuracy: 0.0850
Epoch 3/10
1000/1000 [=====] - 0s 31us/sample - loss: 12.6002 -
categorical_accuracy: 0.0960 - val_loss: 12.5771 - val_categorical_accuracy: 0.0850
...
Epoch 10/10
1000/1000 [=====] - 0s 24us/sample - loss: 12.5972 -
categorical_accuracy: 0.0960 - val_loss: 12.5738 - val_categorical_accuracy: 0.0850

<TensorFlow.python.keras.callbacks.History at 0x130ab5518>
```

对于大型数据集可以使用 `tf.data` 构建训练输入。

代码：

```
dataset = tf.data.Dataset.from_tensor_slices((train_x, train_y))
dataset = dataset.batch(32)
dataset = dataset.repeat()
val_dataset = tf.data.Dataset.from_tensor_slices((val_x, val_y))
val_dataset = val_dataset.batch(32)
val_dataset = val_dataset.repeat()

model.fit(dataset, epochs=10, steps_per_epoch=30,
          validation_data=val_dataset, validation_steps=3)
```

输出：

```
Train for 30 steps, validate for 3 steps
Epoch 1/10
30/30 [=====] - 0s 15ms/step - loss: 12.6243 - categorical_accuracy:
0.0948 - val_loss: 12.3128 - val_categorical_accuracy: 0.0833
...
30/30 [=====] - 0s 2ms/step - loss: 12.5797 - categorical_accuracy:
0.0951 - val_loss: 12.3067 - val_categorical_accuracy: 0.0833
<TensorFlow.python.keras.callbacks.History at 0x132ab48d0>
```

2.3.2.3 回调函数

回调函数是传递给模型以自定义和扩展其在训练期间的行为的对象。我们可以编写自己的自定义回调，或使用

`tf.keras.callbacks` 中的内置函数，常用内置回调函数如下：

`tf.keras.callbacks.ModelCheckpoint`：定期保存模型。

`tf.keras.callbacks.LearningRateScheduler`：动态更改学习率。

`tf.keras.callbacks.EarlyStopping`：提前终止。

`tf.keras.callbacks.TensorBoard`：使用 TensorBoard。

代码：

#超参数设置

Epochs = 10

#定义一个学习率动态设置函数

def lr_Scheduler(epoch):

if epoch > 0.9 * Epochs:

lr = 0.0001

elif epoch > 0.5 * Epochs:

lr = 0.001

elif epoch > 0.25 * Epochs:

lr = 0.01

else:

lr = 0.1

print(lr)

return lr

callbacks = [

#早停:

tf.keras.callbacks.EarlyStopping(

#不再提升的关注指标

monitor='val_loss',

#不再提升的阈值

min_delta=1e-2,

#不再提升的轮次

patience=2),

#定期保存模型:

tf.keras.callbacks.ModelCheckpoint(

#模型路径

filepath='testmodel_{epoch}.h5',

#是否保存最佳模型

save_best_only=True,

#监控指标

monitor='val_loss'),

#动态更改学习率

tf.keras.callbacks.LearningRateScheduler(lr_Scheduler),

#使用 TensorBoard

tf.keras.callbacks.TensorBoard(log_dir='./logs'))

输出：

```
Train on 1000 samples, validate on 200 samples
0
0.1
Epoch 1/10
1000/1000 [=====] - 0s 155us/sample - loss: 12.7907 -
categorical_accuracy: 0.0920 - val_loss: 12.7285 - val_categorical_accuracy: 0.0750
1
0.1
Epoch 2/10
1000/1000 [=====] - 0s 145us/sample - loss: 12.6756 -
categorical_accuracy: 0.0940 - val_loss: 12.8673 - val_categorical_accuracy: 0.0950
...
0.001
Epoch 10/10
1000/1000 [=====] - 0s 134us/sample - loss: 12.3627 -
categorical_accuracy: 0.1020 - val_loss: 12.3451 - val_categorical_accuracy: 0.0900

<TensorFlow.python.keras.callbacks.History at 0x133d35438>
```

2.3.2.4 评估与预测

评估和预测函数：tf.keras.Model.evaluate 和 tf.keras.Model.predict 方法。

代码：

```
# 模型评估
test_x = np.random.random((1000, 36))
test_y = np.random.random((1000, 10))
model.evaluate(test_x, test_y, batch_size=32)
```

输出：


```
1000/1000 [=====] - 0s 45us/sample - loss: 12.2881 -  
categorical_accuracy: 0.0770  
[12.288104843139648, 0.077]
```

代码：

```
# 模型预测  
pre_x = np.random.random((10, 36))  
result = model.predict(test_x,)  
print(result)
```

输出：

```
[[0.04431767 0.24562006 0.05260926 ... 0.1016549  0.13826898 0.15511878]  
 [0.06296062 0.12550288 0.07593573 ... 0.06219672 0.21190381 0.12361749]  
 [0.07203944 0.19570401 0.11178136 ... 0.05625525 0.20609994 0.13041474]  
 ...  
 [0.09224506 0.09908539 0.13944311 ... 0.08630784 0.15009451 0.17172746]  
 [0.08499582 0.17338121 0.0804626  ... 0.04409525 0.27150458 0.07133815]  
 [0.05191234 0.11740112 0.08346355 ... 0.0842929  0.20141983 0.19982798]]
```

2.3.3 模型保存与恢复

2.3.3.1 保存和恢复整个模型

代码：

```
import numpy as np
# 模型保存
model.save('./model/the_save_model.h5')
# 导入模型
new_model = tf.keras.models.load_model('./model/the_save_model.h5')
new_prediction = new_model.predict(test_x)
#np.testing.assert_allclose: 判断两个对象的近似程度是否超出了指定的容差限。若是，则抛出异常。:
#atol:指定的容差限
np.testing.assert_allclose(result, new_prediction, atol=1e-6) # 预测结果一样
```

模型保存后可以在对应的文件夹中找到对应的权重文件。

2.3.3.2 只保存和加载网络权重

若权重名后有.h5 或.keras 后缀，则保存为 HDF5 格式文件，否则默认为 TensorFlow Checkpoint 格式文件。

代码：

```
model.save_weights('./model/model_weights')
model.save_weights('./model/model_weights.h5')
#权重加载
model.load_weights('./model/model_weights')
model.load_weights('./model/model_weights.h5')
```

3 利用 TensorFlow 进行手写数字识别

3.1 实验介绍

手写数字识别是常见的图像识别任务，计算机通过手写体图片来识别图片中的字，与印刷字体不同的是，不同人的手写体风格迥异，大小不一，造成了计算机对手写识别任务的困难，此项目通过应用深度学习和 tensorflow 工具对 MNIST 手写数据集进行训练并建模。

所以本章主要介绍 TensorFlow 计算的基本流程，以及构建网络的基本要素。

3.2 实验目的

掌握 TensorFlow 计算基本流程；

熟悉构建网络的基本要素：数据集、网络模型构建、模型训练、模型验证。

3.3 实验步骤

读取 mnist 手写数字数据集；

利用简单数学模型入门 tensorflow；

高级 API 实现 softmax 回归；

构建多层卷积网络 CNN；

高级 API 实现卷积网络 CNN；

预测结果可视化；

3.3.1 项目描述和数据集获取

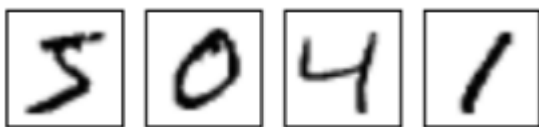
3.3.1.1 项目描述

手写数字识别是常见的图像识别任务，计算机通过手写体图片来识别图片中的字，与印刷字体不同的是，不同人的手写体风格迥异，大小不一，造成了计算机对手写识别任务的困难，此项目通过应用深度学习和 tensorflow 工具对 MNIST 手写数据集进行训练并建模。

3.3.1.2 数据获取以及数据处理

3.3.1.2.1 数据集介绍

- 1). MNIST 数据集来自美国国家标准与技术研究所 (National Institute of Standards and Technology , 简称 NIST);
- 2). 该数据集由来自 250 个不同人手写的数字构成, 其中 50%是高中学生, 50%来自人口普查局的工组人员;
- 3). 数据集可在 <http://yann.lecun.com/exdb/mnist/> 获取, 它包含了四个部分:
 - Training set images: train-images-idx3-ubyte.gz (9.9 MB, 解压后 47 MB, 包含 60,000 个样本)
 - Training set labels: train-labels-idx1-ubyte.gz (29 KB, 解压后 60 KB, 包含 60,000 个标签)
 - Test set images: t10k-images-idx3-ubyte.gz (1.6 MB, 解压后 7.8 MB, 包含 10,000 个样本)
 - Test set labels: t10k-labels-idx1-ubyte.gz (5KB, 解压后 10 KB, 包含 10,000 个标签)
- 4). mnist 是一个入门级的计算机视觉数据集, 它包含各种手写数字图片:



它也包含每一张图片对应的标签, 告诉我们这个是数字几, 比如说, 上面这四张图片的标签分别是 5,0,4,1。

3.3.1.2.2 mnist 数据集读取

从 tensorflow 直接读取数据集, 联网下载解压;

代码:

```
import os
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, optimizers, datasets
from matplotlib import pyplot as plt
import numpy as np

(x_train_raw, y_train_raw), (x_test_raw, y_test_raw) = datasets.mnist.load_data()

print(y_train_raw[0])
print(x_train_raw.shape, y_train_raw.shape)
print(x_test_raw.shape, y_test_raw.shape)

#将分类标签变为 onehot 编码
num_classes = 10
y_train = keras.utils.to_categorical(y_train_raw, num_classes)
y_test = keras.utils.to_categorical(y_test_raw, num_classes)
print(y_train[0])
```

输出：

```
5
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

在 mnist 数据集中, images 是一个形状为[60000,28,28]的张量, 第一个维度数字用来索引图片, 第二、三个维度数字用来索引每张图片中的像素点。在此张量里的每一个元素, 都表示某张图片里的某个像素的强度值, 介于 0,255 之间。

标签数据是"one-hot vectors", 一个 one-hot 向量除了某一位数字是 1 之外, 其余各维度数字都是 0, 如标签 1 可以表示为([0,1,0,0,0,0,0,0,0,0]), 因此, labels 是一个 [60000, 10] 的数字矩阵。

3.3.2 数据集预处理及可视化

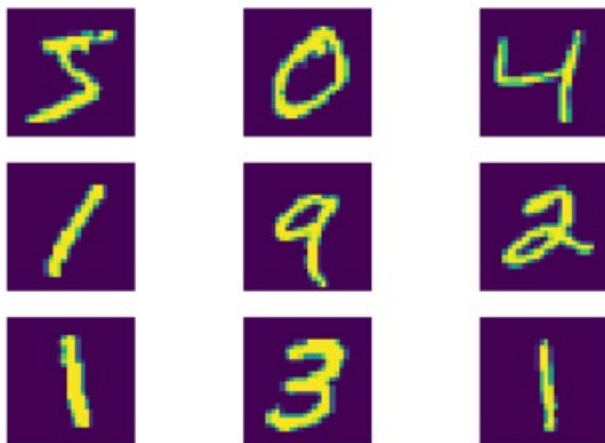
3.3.2.1 数据可视化

绘制前 9 张图片

代码：

```
plt.figure()
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.imshow(x_train_raw[i])
    #plt.ylabel(y[i].numpy())
    plt.axis('off')
plt.show()
```

输出：



数据处理，因为我们构建的是全连接网络所以输出应该是向量的形式，而非现在图像的矩阵形式。因此我们需要把图像整理成向量。

代码：

```
#将 28*28 的图像展开成 784*1 的向量
x_train = x_train_raw.reshape(60000, 784)
x_test = x_test_raw.reshape(10000, 784)
```

现在像素点的动态范围为 0 到 255。处理图形像素值时，我们通常会把图像像素点归一化到 0 到 1 的范围内。

代码：

代码：

```
#将图像像素值归一化
x_train = x_train.astype('float32')/255
x_test = x_test.astype('float32')/255
```

3.3.3 DNN 网络构建

3.3.3.1 DNN 构建网络

代码：

```
# 创建模型。模型包括 3 个全连接层和两个 RELU 激活函数
model = keras.Sequential([
    layers.Dense(512, activation='relu', input_dim = 784),
    layers.Dense(256, activation='relu'),
    layers.Dense(124, activation='relu'),
layers.Dense(num_classes, activation='softmax')])

model.summary()
```

输出：

```
Model: "sequential"
Layer (type)                 Output Shape                 Param #
=====
dense (Dense)                 (None, 512)                  401920
dense_1 (Dense)               (None, 256)                  131328
dense_2 (Dense)               (None, 124)                  31868
dense_3 (Dense)               (None, 10)                   1250
=====
Total params: 566,366
Trainable params: 566,366
Non-trainable params: 0
```

其中 layer.Dense()表示全连接层，activation 参数表示使用的激活函数。

3.3.3.2 编译 DNN 模型

代码：

```
Optimizer = optimizers.Adam(0.001)
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=Optimizer,
              metrics=['accuracy'])
```

以上定义了模型的损失函数为“交叉熵”，优化算法为“Adam”梯度下降方法。

3.3.3.3 DNN 模型训练

代码：

```
# 使用 fit 方法使模型对训练数据拟合
model.fit(x_train, y_train,
          batch_size=128,
          epochs=10,
          verbose=1)
```

输出：


```
Epoch 1/10
60000/60000 [=====] - 7s 114us/sample - loss: 0.2281 - acc: 0.9327s -
loss: 0.2594 - acc: 0. - ETA: 1s - loss: 0.2535 - acc: 0.9 - ETA: 1s - loss:
Epoch 2/10
60000/60000 [=====] - 8s 129us/sample - loss: 0.0830 - acc: 0.9745s -
loss: 0.0814 - ac
Epoch 3/10
60000/60000 [=====] - 8s 127us/sample - loss: 0.0553 - acc: 0.9822
Epoch 4/10
60000/60000 [=====] - 7s 117us/sample - loss: 0.0397 - acc: 0.9874s -
los
Epoch 5/10
60000/60000 [=====] - 8s 129us/sample - loss: 0.0286 - acc: 0.9914
Epoch 6/10
60000/60000 [=====] - 8s 136us/sample - loss: 0.0252 - acc: 0.9919
Epoch 7/10
60000/60000 [=====] - 8s 129us/sample - loss: 0.0204 - acc: 0.9931s -
lo
Epoch 8/10
60000/60000 [=====] - 8s 135us/sample - loss: 0.0194 - acc: 0.9938
Epoch 9/10
60000/60000 [=====] - 7s 109us/sample - loss: 0.0162 - acc: 0.9948
Epoch 10/10
60000/60000 [=====] - ETA: 0s - loss: 0.0149 - acc: 0.994 - 7s
117us/sample - loss: 0.0148 - acc: 0.9948
```

其中 epoch 表示批次，表示将全量的数据迭代 10 次。

3.3.3.4 DNN 模型评估

代码：

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

输出：

```
Test loss: 0.48341113169193267
```

```
Test accuracy: 0.8765
```

经过评估，模型准确率为 0.87，迭代了 10 次训练。

3.3.3.5 保存模型

代码：

```
model.save('./mnist_model/final_DNN_model.h5')
```

3.3.4 构建 CNN 网络

之前用传统方法构建 CNN 网络，可以更清楚的了解内部的网络结构，但是代码量比较多，所以我们尝试用高级 API 构建网络，以简化构建网络的过程。

3.3.4.1 CNN 构建网络

代码：

```
import tensorflow as tf
from tensorflow import keras
import numpy as np

model=keras.Sequential() # 创建网络序列
## 添加第一层卷积层和池化层
model.add(keras.layers.Conv2D(filters=32,kernel_size = 5,strides = (1,1),
                                padding = 'same',activation = tf.nn.relu,input_shape = (28,28,1)))
model.add(keras.layers.MaxPool2D(pool_size=(2,2), strides = (2,2), padding = 'valid'))
## 添加第二层卷积层和池化层
model.add(keras.layers.Conv2D(filters=64,kernel_size = 3,strides = (1,1),padding = 'same',activation =
tf.nn.relu))
model.add(keras.layers.MaxPool2D(pool_size=(2,2), strides = (2,2), padding = 'valid'))
## 添加 dropout 层 以减少过拟合
model.add(keras.layers.Dropout(0.25))
model.add(keras.layers.Flatten())
## 添加两层全连接层
model.add(keras.layers.Dense(units=128,activation = tf.nn.relu))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(units=10,activation = tf.nn.softmax))
```

以上网络中，我们利用 `keras.layers` 添加了两个卷积池化层，之后又添加了 dropout 层，防止过拟合，最后添加了两层全连接层。

3.3.4.2 CNN 网络编译和训练

代码：

```
# 将数据扩充维度，以适应 CNN 模型
X_train=x_train.reshape(60000,28,28,1)
X_test=x_test.reshape(10000,28,28,1)
model.compile(optimizer=tf.train.AdamOptimizer(),loss="categorical_crossentropy",metrics=['accuracy'])
model.fit(x=X_train,y=y_train,epochs=5,batch_size=128)
```

输出：

```
Epoch 1/5
55000/55000 [=====] - 49s 899us/sample - loss: 0.2107 - acc: 0.9348
Epoch 2/5
55000/55000 [=====] - 48s 877us/sample - loss: 0.0793 - acc: 0.9763
Epoch 3/5
55000/55000 [=====] - 52s 938us/sample - loss: 0.0617 - acc: 0.9815
Epoch 4/5
55000/55000 [=====] - 48s 867us/sample - loss: 0.0501 - acc: 0.9846
Epoch 5/5
55000/55000 [=====] - 50s 901us/sample - loss: 0.0452 - acc: 0.9862

<tensorflow.python.keras.callbacks.History at 0x214bbf34ac8>
```

在训练时，网络训练数据只迭代了 5 次，可以再增加网络迭代次数，自行尝试看效果如何。

3.3.4.3 CNN 模型验证

代码：

```
test_loss,test_acc=model.evaluate(x=X_test,y=mnist.test.labels)
print("Test Accuracy %.2f"%test_acc)
```

输出：

```
10000/10000 [=====] - 2s 185us/sample - loss: 0.0239 - acc: 0.9921
Test Accuracy 0.99
```

最终结果也达到了 99%的准确率。

3.3.4.4 CNN 模型保存

代码：

```
test_loss,test_acc=model.evaluate(x=X_test,y=y_test)
print("Test Accuracy %.2f"%test_acc)
```

输出：

```
10000/10000 [=====] - 5s 489us/sample - loss: 0.0263 - acc: 0.9920s -  
loss: 0.0273 - ac  
Test Accuracy 0.99
```

3.3.5 预测结果可视化

3.3.5.1 加载 CNN 保存模型

代码：

```
from tensorflow.keras.models import load_model  
new_model = load_model('./mnist_model/final_CNN_model.h5')  
new_model.summary()
```

输出：

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 28, 28, 32)	832
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
dropout (Dropout)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense_4 (Dense)	(None, 128)	401536
dropout_1 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 10)	1290
=====		
Total params: 422,154		
Trainable params: 422,154		
Non-trainable params: 0		

用将预测结果可视化

代码：

```
#测试集输出结果可视化
import matplotlib.pyplot as plt
%matplotlib inline
def res_Visual(n):
    final_opt_a=new_model.predict_classes(X_test[0:n])# 通过模型预测测试集
    fig, ax = plt.subplots(nrows=int(n/5),ncols=5 )
    ax = ax.flatten()
    print('前{}张图片预测结果为: '.format(n))
    for i in range(n):
        print(final_opt_a[i],end=',')
        if int((i+1)%5) ==0:
            print('\n')
    #图片可视化展示
    img = X_test[i].reshape((28,28))#读取每行数据，格式为 Narray
    plt.axis("off")
    ax[i].imshow(img, cmap='Greys', interpolation='nearest')#可视化
    ax[i].axis("off")
    print('测试集前{}张图片为: '.format(n))
res_Visual(20)
```

输出：

前 20 张图片预测结果为：

7,2,1,0,4,

1,4,9,5,9,

0,6,9,0,1,

5,9,7,3,4,

测试集前 20 张图片为：

7	2	1	0	4
1	4	9	5	9
0	6	9	0	1
5	9	7	3	4

华为认证 AI 系列教程

HCIA-AI

Hello Davinci

实验指导手册

版本:3.0



华为技术有限公司

版权所有 © 华为技术有限公司 2020。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129
网址： <http://e.huawei.com>

华为认证体系介绍

于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，华为公司打造了业界唯一覆盖ICT全技术领域的认证体系，包含ICT技术架构认证、平台与服务认证和行业ICT认证三类认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代的新型ICT人才，构建良性的ICT人才生态。

华为认证HCIA-AI V3.0定位于培养和认证具备使用机器学习、深度学习等算法设计、开发AI产品和解决方案能力的工程师。

通过HCIA-AI V3.0认证，将证明您了解人工智能发展历史、华为昇腾AI体系和全栈全场景AI战略知识，掌握传统机器学习和深度学习的相关算法；具备利用TensorFlow开发框架和MindSpore开发框架进行搭建、训练、部署神经网络的能力；能够胜任人工智能领域销售、市场、产品经理、项目管理、技术支持等岗位。



前言

简介

本书为 HCIA-AI 认证培训教程，适用于准备参加 HCIA-AI 考试的学员或者希望了解华为昇腾芯片相关知识的读者。

内容描述

本实验指导书共包含 1 个实验，主要分为三部分内容，主要是配合 HCIA-AI 认证课程第七章华为全栈全场景 AI 战略——昇腾理论教材。帮助学员更好的理解昇腾芯片，利用云上 AI 服务器开发。

- 第一部分为 AI 服务器昇腾芯片的驱动安装，可以获取昇腾芯片的状态。
- 第二部分为 AI 服务器上 DDK (Device Development Kit) 的安装，后续可以使用工具开发运行在昇腾芯片的程序。
- 第三部分为实战部分，通过 host 侧与 device 侧通信来使昇腾芯片输出内容并打印出来。

读者知识背景

本课程为华为认证基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具有基本的 Linux 基础。

背景介绍

实验目的

随着近些年深度学习的蓬勃发展，传统的 CPU 已经无法满足模型训练的需求，越来越多的厂商开始开发专门用于深度学习的芯片，而本次实验主要是教会大家如何在一台装有昇腾芯片的服务器上安装驱动和开发者工具，并使用一个小样例来查看是否安装成功，最终使大家可以利用昇腾芯片来进行高效的推理。

注意：现在服务器镜像中提供自动安装驱动和软件包的脚本，本实验则是一步步演示如何安装。

注意事项

实验资源一旦购买就开始计费，请合理安排时间进行实验，并注意以下几点：

- 本实验预计 2 小时完成，实验结束后请一定释放资源，并确认资源彻底删除；

- 若实验中途离开或中断，建议释放实验资源，否则将会按照购买的资源继续计费；
- 资源释放步骤请根据实验指导进行。

目录

前 言	3
简介.....	3
内容描述.....	3
读者知识背景.....	3
背景介绍.....	3
1 Atlas300 (3000) 加速卡驱动安装	7
1.1 AI 加速服务器购买.....	7
1.2 软件包获取.....	8
1.3 安装驱动.....	9
1.3.1 连接服务器.....	9
1.3.2 安装驱动包.....	12
2 DDK 软件安装	15
2.1 安装 DDK 依赖包.....	15
2.2 安装 DDK.....	16
2.3 使用自动安装脚本安装驱动和 DDK.....	16
3 Hello Davinci 编译运行	18
3.1 实验流程介绍.....	18
3.1.1 实验代码获取.....	18
3.1.2 实验流程介绍.....	19
3.2 实验运行.....	20

1

Atlas300（3000）加速卡驱动安装

1.1 AI 加速服务器购买

步骤 1 使用已实名的账号登录华为云网站（<https://www.huaweicloud.com/>），如果没有账号可参考网站引导进行注册实名。

步骤 2 账号登陆后选择产品-基础服务-弹性云服务器 ECS，点击立即购买。

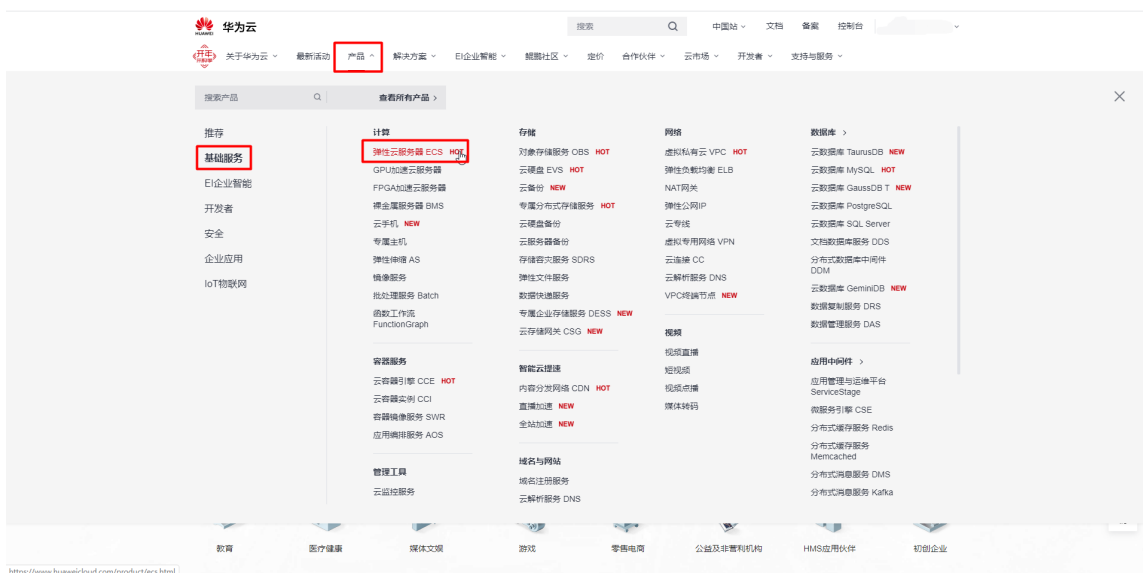


图1-1 华为云界面

步骤 3 选择按需计费、北京四区域、AI 加速型服务器，规格选择 ai1.large4，操作系统选择 Ubuntu。

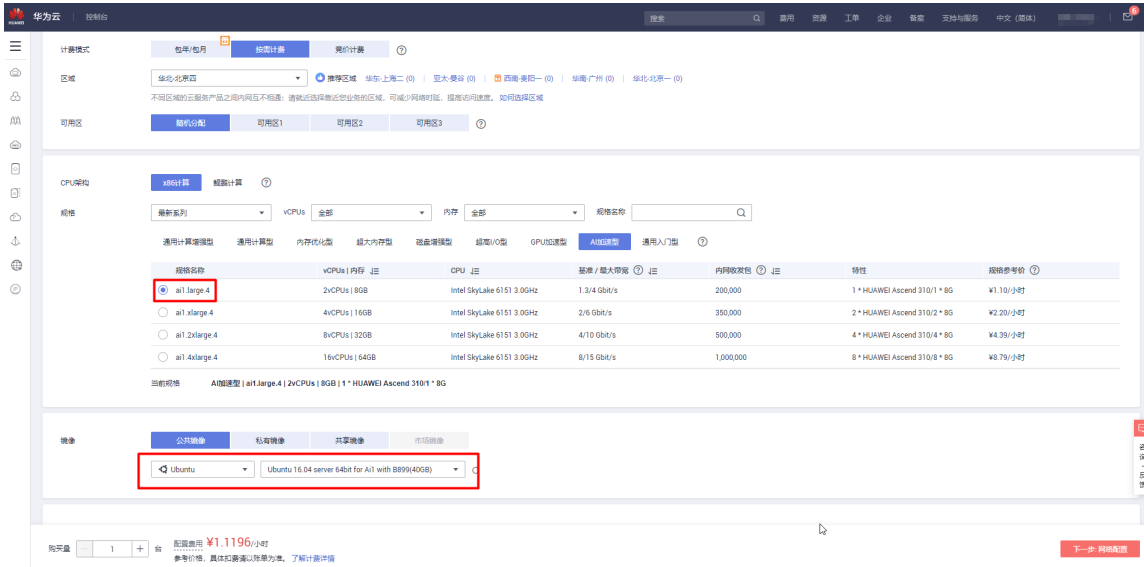


图1-2 云服务器购买界面

步骤 4 点击下一步，云服务器使用密码登录，完成购买。

1.2 软件包获取

注意：现在的 AI 加速型服务器镜像中已经预置的软件包，这一步可以跳过，如果镜像中不包含软件包或者希望使用最新的软件包则可以根据本部分下载最新的软件包。

步骤 1 登录华为企业业务网站（<https://e.huawei.com/cn/>），点击菜单，在技术支持、产品和解决方案支持、人工智能计算平台模块找到 Atlas300（3000），点击软件，然后点击最新版本的软件，选择 Ubuntu 版本的软件进行下载，下载时需要登录账号，可按网站引导进行注册登录。

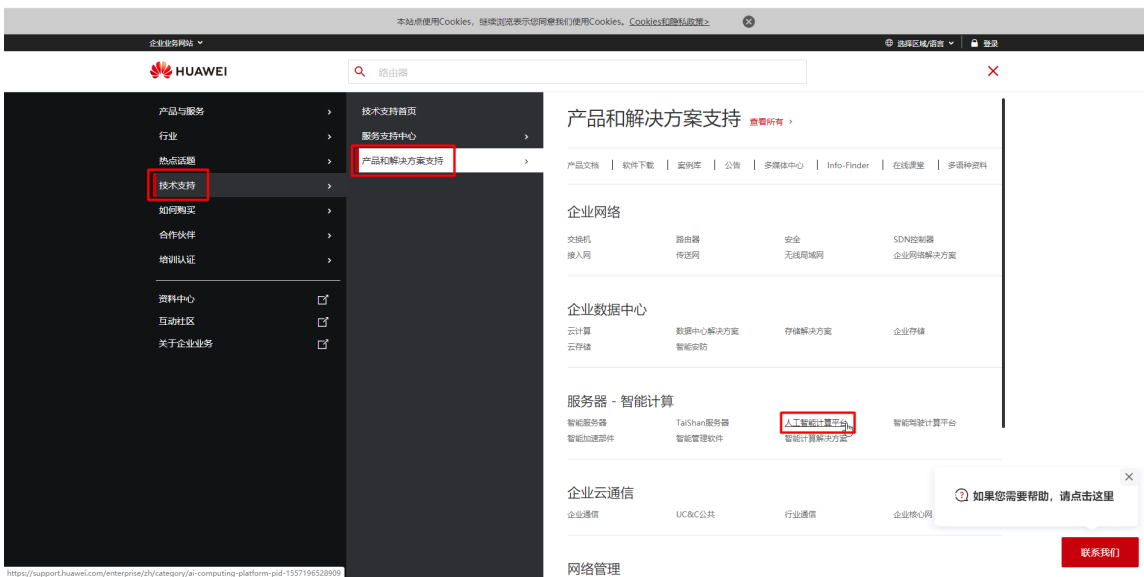


图1-3 华为企业业务网站界面

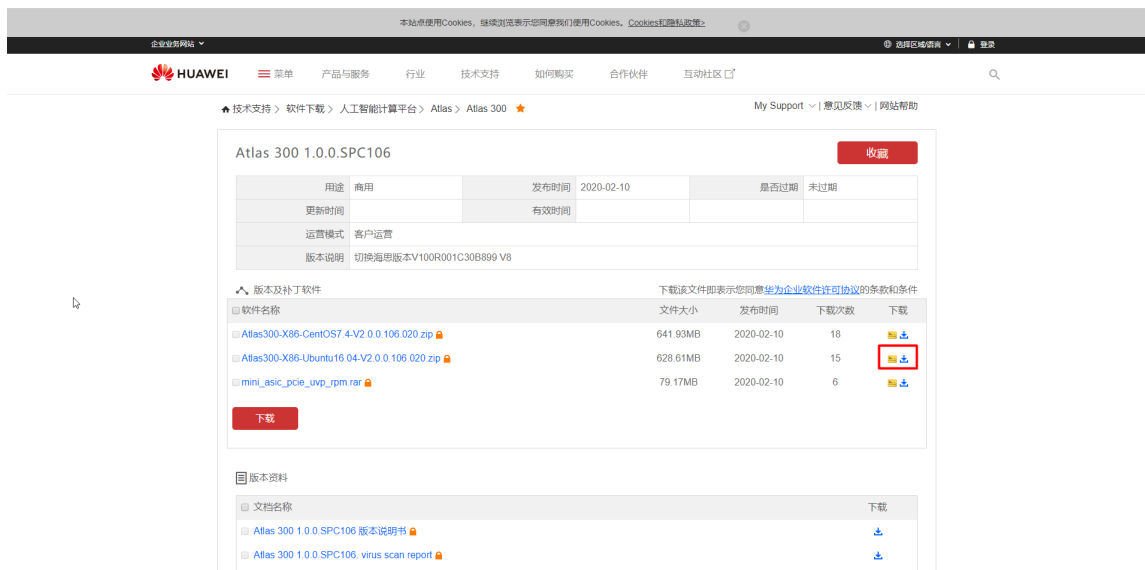


图1-4 软件包下载页面

步骤 2 解压后可得到四个文件，其中 MSpore 和 npu 打头的分别是 DDK (Device Development Kit) 安装包和驱动安装包。

名称	修改日期	类型	大小
MCU.zip	2020/1/9 19:14	WinRAR ZIP arch...	210 KB
MSpore_DDK-1.3.8.B899-x86_64.u...	2020/1/9 19:12	WinRAR archive	475,529 KB
MSpore_DDK-1.3.8.B899-x86_64.u...	2020/1/9 19:12	ASC 文件	1 KB
npu_ubuntu.x86_1.2.3.zip	2020/1/9 19:12	WinRAR ZIP arch...	167,893 KB

图1-5 软件包目录

1.3 安装驱动

1.3.1 连接服务器

注意：目前推荐使用普通用户进行 DDK (Device Development Kit) 的安装，因此需要先配置普通用户并配置相关权限。

步骤 1 自行搜索下载并安装 putty 和 winscp 工具，putty 可用于向服务器发送命令，winscp 可用于向服务器传输文件。

步骤 2 在华为云界面查看服务器 IP 地址，随后输入到 putty 中，进行登录。

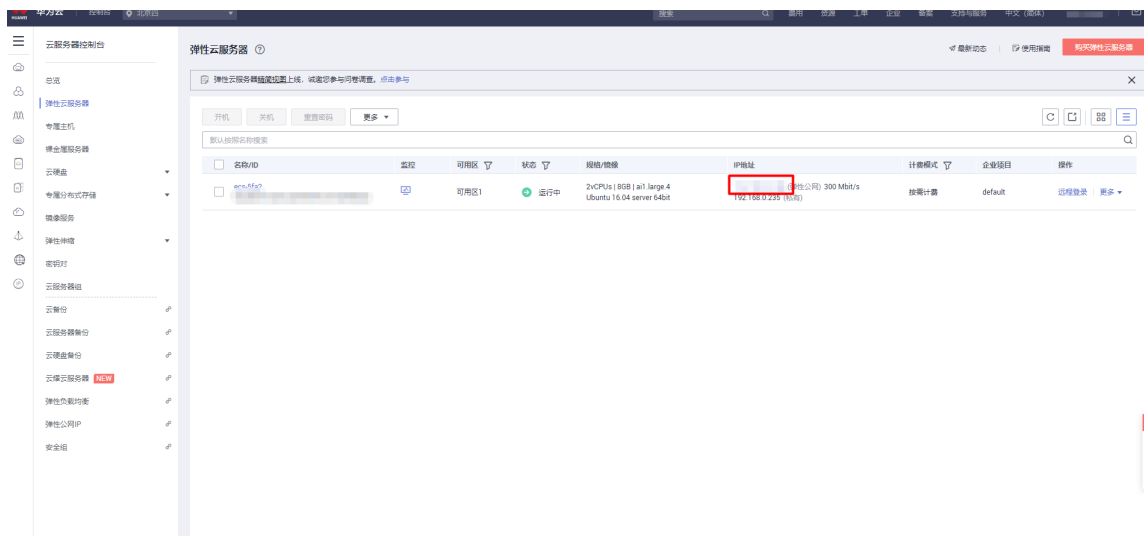


图1-6 IP 查看界面

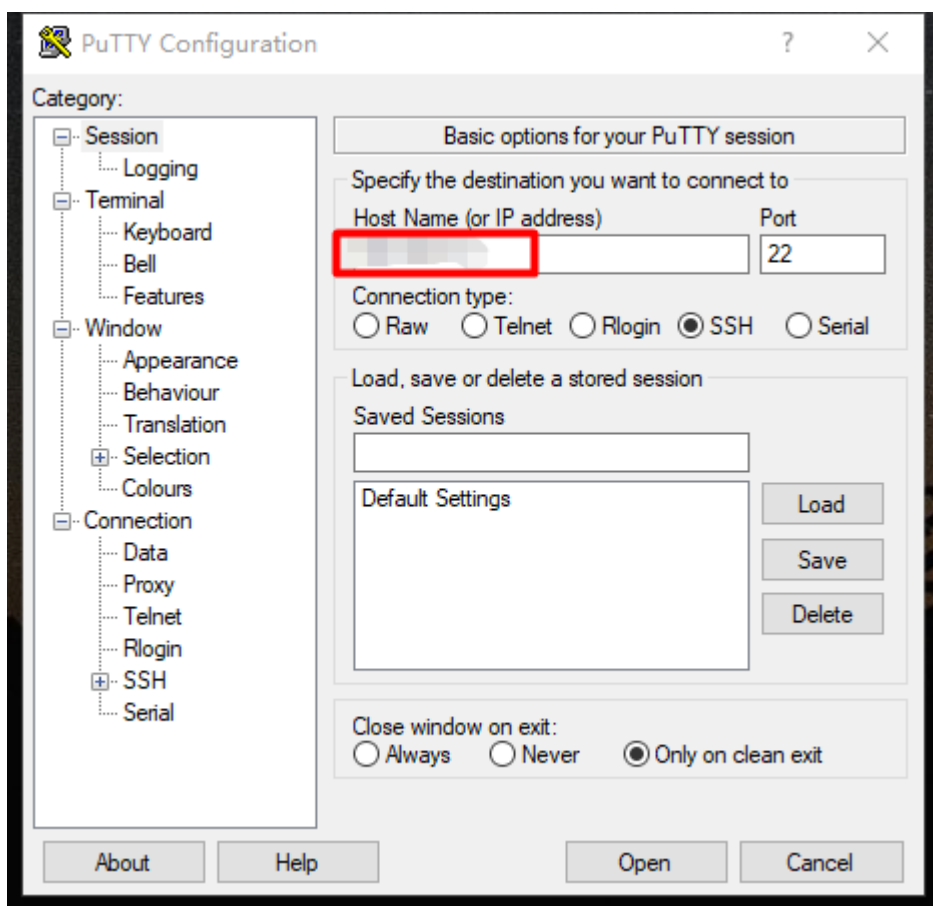


图1-7 软件登陆界面

步骤 3

软件登录成功后即可输入各种命令。

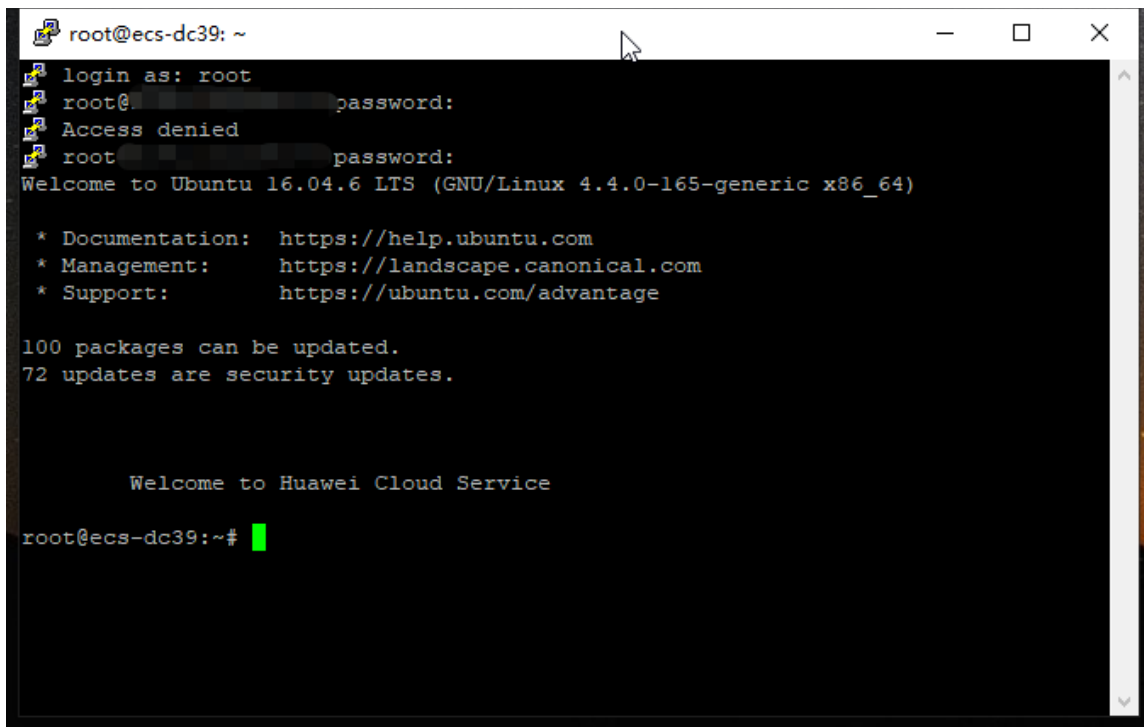


图1-8 Putty 工作界面

步骤 4 首先输入以下命令确认卡是否正常在位，输出回显中含“d100”，则表示正常在位。

```
lspci | grep 'd100'
```

```

root@ecs-5fa2:~# lspci | grep 'd100'
00:0d.0 Processing accelerators: Huawei Technologies Co., Ltd. Device d100 (rev 20)
root@ecs-5fa2:~#

```

图1-9 判断加速卡是否正常在位

步骤 5 输入以下命令创建普通用户并设置该用户\$HOME 目录，红色部分为用户名称，可根据自己喜好修改。

```
useradd -d /home/ascend1 -m ascend1
```

步骤 6 执行以下命令为用户设置登录密码。

```
passwd ascend1
```

步骤 7 执行以下命令设置权限，进入“/home”目录。

```
chmod 750 /home/ascend1
```

步骤 8 DDK 安装过程中需要使用 sudo apt-get 权限，下面继续为普通用户设置权限。

```
chmod u+w /etc/sudoers
```

```
vi /etc/sudoers
```

步骤 9 在该文件 “# User privilege specification” 下面增加如下内容：

```
ascend1 ALL=(ALL:ALL) NOPASSWD:SETENV:/usr/bin/apt-get, /usr/bin/pip, /bin/tar,  
/bin/mkdir, /bin/rm, /bin/sh, /bin/cp, /bin/bash
```

步骤 10 按 wq! 保存文件，随后执行以下命令取消 “/etc/sudoers” 文件的写权限。

```
chmod u-w /etc/sudoers
```

步骤 11 执行以下命令检查源是否可以用。

```
apt-get update
```

1.3.2 安装驱动包

步骤 1 打开 winscp 软件，输入 IP 地址和账号密码进行登录。

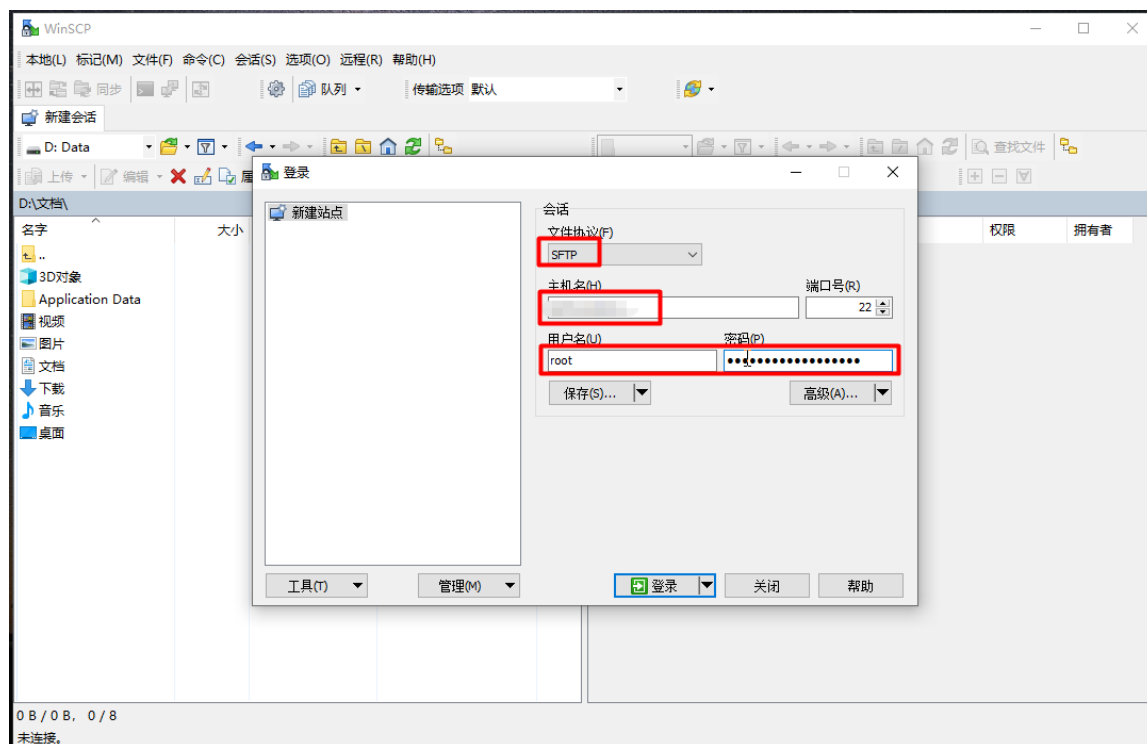


图1-10 Winscp 登录界面

步骤 2 登录成功后可以发现 root 目录下有软件包，如果没有的话可以返回 1.2 部分进行下载，将 MSpore***和 npu***两个文件复制到普通用户家目录（按 ctrl 可多选）。

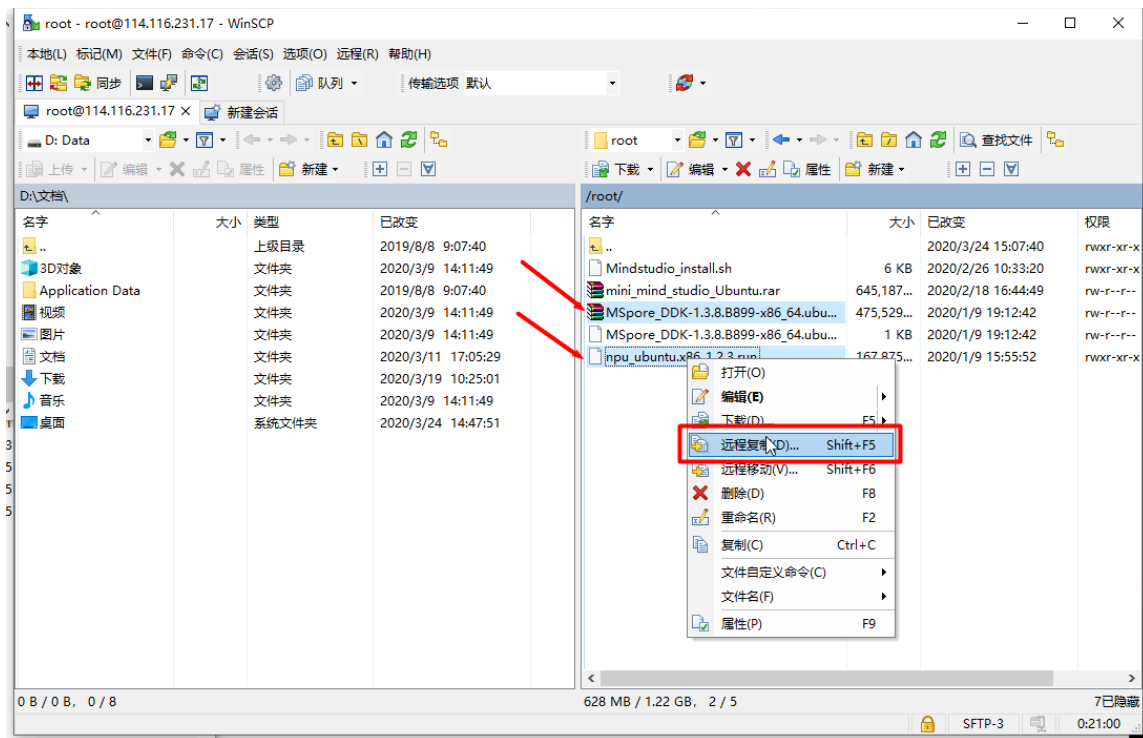


图1-11 镜像预置软件安装包

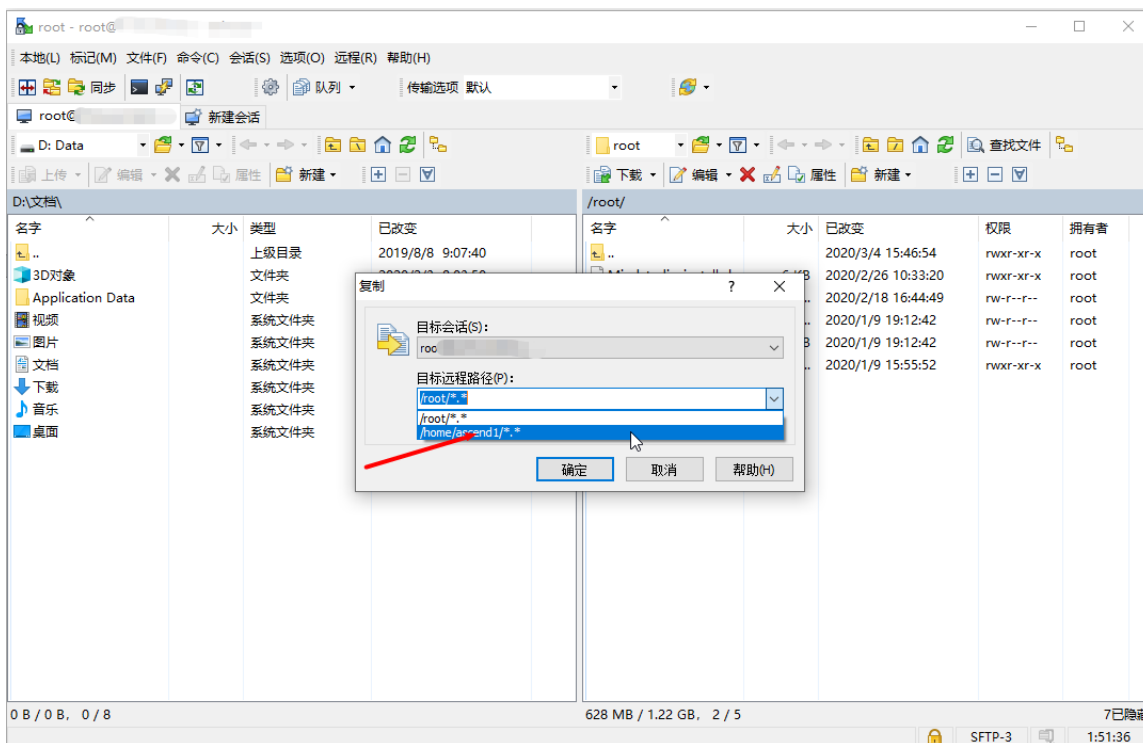


图1-12 移动软件包到普通用户家目录

步骤 3 软件包移动完成后，切换到 putty 窗口，执行以下命令安装依赖包，DKMS。

```
apt-get install dkms
```

步骤 4 通过 cd 命令进入到普通用户家目录，输入以下命令使用默认路径安装驱动，红色部分替换为实际名称。

```
./npu_<host-os>.x86_<version>.run --full
```

步骤 5 输入以下命令重启服务器，完成安装。

```
reboot
```

步骤 6 输入以下命令查看是否安装成功，回显以下信息说明安装成功。

```
npu-smi info
```

```
root@ecs-5fa2:~# npu-smi info
+-----+
| npu-smi 1.2.3 | Version: 1.3.8.B899 |
+-----+
| NPU   Name   | Health   | Power(W)   | Temp(C)   |
| Chip  | Bus-Id    | AICore(%) | Memory-Usage(MB) |
+-----+
| 0     310    | OK       | 12.8       | 36        |
| 0     0000:00:0D.0 | 0        | 2375 / 8192 |
+-----+
root@ecs-5fa2:~#
```

图1-13 验证驱动是否安装成功

2 DDK 软件安装

2.1 安装 DDK 依赖包

步骤 1 新建一个 putty 窗口，输入以下命令切换到安装用户，接下来的操作需要 root 用户和普通用户分别执行，建议使用两个 putty 窗口。

```
su ascend1
```

步骤 2 使用普通用户输入以下命令安装依赖包，如果提示 not allowed 请重复 1.3.1 的步骤 8 到步骤 11。

```
sudo -E apt-get install gcc g++ cmake make python-pip python3-pip  
python3 python
```

步骤 3 使用 root 用户输入以下命令更新 pip，如果出现报错可执行步骤 5 重新安装 pip。

```
pip2 install -i https://pypi.tuna.tsinghua.edu.cn/simple --upgrade pip  
pip3 install -i https://pypi.tuna.tsinghua.edu.cn/simple --upgrade pip
```

步骤 4 （可选）使用 root 用户输入以下命令重装 pip。

```
sudo apt-get remove python-pip python3-pip  
wget https://bootstrap.pypa.io/get-pip.py  
python get-pip.py --user  
python3 get-pip.py --user
```

步骤 5 使用普通用户输入以下命令安装软件。

```
pip2 install -i https://pypi.tuna.tsinghua.edu.cn/simple setuptools==40.6.3 --user  
pip2 install -i https://pypi.tuna.tsinghua.edu.cn/simple numpy==1.16.0 --user  
pip2 install -i https://pypi.tuna.tsinghua.edu.cn/simple decorator --user  
pip3 install -i https://pypi.tuna.tsinghua.edu.cn/simple setuptools==20.7.0 --user  
pip3 install -i https://pypi.tuna.tsinghua.edu.cn/simple numpy --user
```



```
pip3 install -i https://pypi.tuna.tsinghua.edu.cn/simple decorator --user
```

步骤 6 使用普通用户输入以下命令查看依赖是否安装成功，返回信息没报错即可。

```
python2 --version
```

```
pip2 --version
```

```
easy_install --version
```

```
python3 --version
```

```
pip3 --version
```

```
easy_install3 --version
```

```
HwHiAiUser@ecs-5fa2:/root$ python2 --version
Python 2.7.12
HwHiAiUser@ecs-5fa2:/root$ pip2 --version
pip 20.0.2 from /usr/local/lib/python2.7/dist-packages/pip (python 2.7)
HwHiAiUser@ecs-5fa2:/root$ easy_install --version
setuptools 19.6.2 from /home/HwHiAiUser/.local/lib/python2.7/site-packages (Python 2.7)
HwHiAiUser@ecs-5fa2:/root$ python3 --version
Python 3.5.2
HwHiAiUser@ecs-5fa2:/root$ pip3 --version
pip 20.0.2 from /usr/local/lib/python3.5/dist-packages/pip (python 3.5)
HwHiAiUser@ecs-5fa2:/root$ easy_install3 --version
setuptools 20.7.0 from /home/HwHiAiUser/.local/lib/python3.5/site-packages (Python 3.5)
HwHiAiUser@ecs-5fa2:/root$
```

图2-1 查看依赖包是否安装成功

2.2 安装 DDK

步骤 1 进入到文件上传目录，使用普通用户执行以下命令解压 DDK 安装包。

```
tar -zxvf MSpore_DDK****tar.gz
```

步骤 2 使用普通用户执行以下命令安装 DDK，红色部分为安装目录。

```
bash install.sh $HOME/tools/che/ddk
```

2.3 使用自动安装脚本安装驱动和 DDK

前面的操作可以看出来安装过程比较繁琐，但可以清楚的知道整个软件的安装过程，现在我们可以通过自动安装脚本来实现驱动、DDK、Mindstudio 的安装。

如果希望体验自动安装脚本，建议本步骤重新申请一个服务器，因为自动安装脚本就是自动执行前面的所有操作。

注意：Mindstudio 是一个开发工具，本次实验并不涉及。

步骤 1 点击云服务器详情页面的远程登录。

注意：因为脚本执行过程较长，putty 可能会断开连接，丢失安装进度。

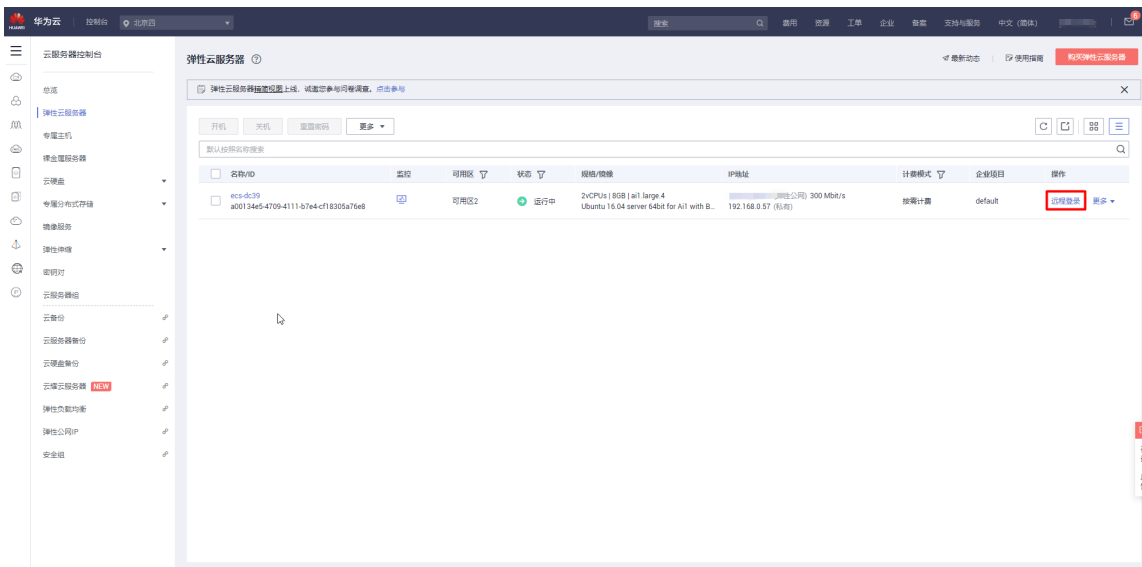


图2-2 华为云服务器管理界面

步骤 2 输入以下命令执行脚本安装命令，红色部分分别为普通用户名和普通用户密码，代码执行完成即可安装成功。

```
bash Mindstudio_install.sh ascend1 passwd
```

步骤 3 如果安装过程出现中断，可执行以下命令删除用户，执行步骤 2 命令重新安装，红色部分为用户名称。

```
userdel ascend1
```

3 Hello Davinci 编译运行

3.1 实验流程介绍

3.1.1 实验代码获取

实验代码可以在这个网址获取。

<https://gitee.com/HuaweiAtlas/samples/tree/master/>

本次实验用到的部分为这三个文件夹。

下载好代码之后将这个三个文件夹保持相对位置不变上传到服务器中。

← ...	
CMake	update
Common	Operated by Junhec
CompileDemo	update
DecodelImage	修改产品号，增加配套版本，更改部分日志级别
DecodeVideo	修改产品号，增加配套版本，更改部分日志级别
DvppCrop	修改产品号，增加配套版本，更改部分日志级别
DvppImagePerformance	修改产品号，增加配套版本，更改部分日志级别
DynamicGraph	修改产品号，增加配套版本，更改部分日志级别
EncodeJpeg	修改产品号，增加配套版本，更改部分日志级别
EncodeVideo	修改产品号，增加配套版本，更改部分日志级别
HaSample	增加HaSample的简介和支持产品说明
HelloDavinci	修改产品号，增加配套版本，更改部分日志级别
InferClassification	Operated by Junhec
InferObjectDetection	更新模型转换的对应产品号
InferOfflineVideo	Update README in InferOfflineVideo, and DstEngine in InferClassification

图3-1 Sample 界面

编译工具文件Cmake目录结构如下所示:

```

├── Ascend.cmake    //Device侧编译链
└── FindDDK.cmake  //cmake寻找DDK模块

```

HelloDavinci的目录结构如下所示:

```

├── build            //编译文件夹，包括Host和Device侧的编译
│   ├── CMakeLists.txt
│   ├── device
│   └── host
├── build.sh        //编译脚本
├── README.md       //README.md
├── main.cpp        //主函数入口
├── include         //HelloDavinci公共模块
├── DstEngine       //DstEngine(host侧)
│   ├── DstEngine.cpp
│   └── DstEngine.h
├── graph.config    //graph配置文件
├── HelloDavinci    //HelloDavinci Engine(device侧)
│   ├── HelloDavinci.cpp
│   └── HelloDavinci.h
└── SrcEngine       //SrcEngine Engine(host侧)
    ├── SrcEngine.cpp
    └── SrcEngine.h

```

图3-2 代码文件目录说明

3.1.2 实验流程介绍

本开发样例主要是演示从 Host 侧发送数据到 Device 侧，再从 Device 侧获取生成的字符串发送回 Host 侧，保存结果，并且打印到终端。如下图所示，整个程序分为两部分运行，Host 侧（包括 SrcEngine 和 DstEngine）和 Device 侧（包括 HelloDavinci），运行过程如下：

运行从 main 开始，向 SrcEngine 发数据。

SrcEngine 收到数据之后，转发给 HelloDavinci，HelloDavinci 在内部生成字符串 “This message is from HelloDavinci” 并发送到 DstEngine。

DstEngine 收到数据后将信息保存在目录 “\${workPath}/out/dacvinci_log_info.txt”（\${workPath}为工程根目录）下，并向 main 发送结束信号。

main 函数收到结束信号后，销毁 graph，在终端打印结束信息并退出程序。

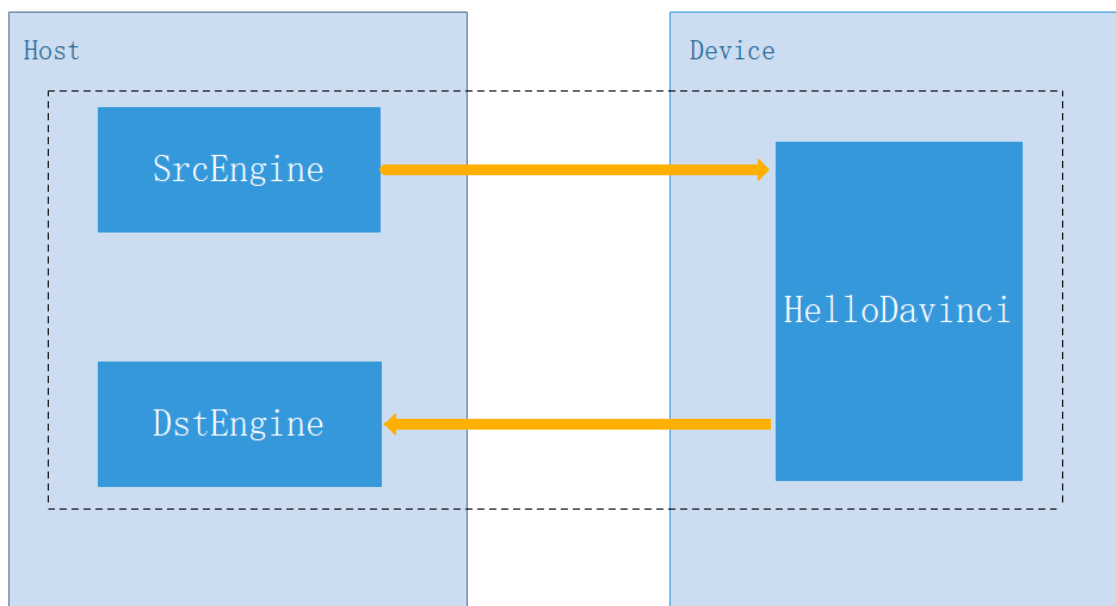


图3-3 实验流程框架

3.2 实验运行

步骤 1 实验在 root 用户下进行操作，执行以下命令配置 DDK 路径信息,红色部分根据实际情况进行修改，如果是使用自动安装脚本安装的，则不用执行这一步。

```
export DDK_HOME=/home/ascend1/tools/che/ddk/ddk/
```

步骤 2 进入到 Hello Davinci 文件夹，执行以下命令进行编译。

```
bash build.sh
```

步骤 3 执行以下命令，查看是否有结果输出。

```
./out/main
```

```
root@ecs-5fa2:/home/HwHiAiUser/sample/HelloDavinci# ./out/main
Hello Davinci!
The sample end!!
root@ecs-5fa2:/home/HwHiAiUser/sample/HelloDavinci#
```

图3-4 实验结果输出

华为认证 HCIA-AI 系列教程

HCIA-AI人工智能 综合实验指导手册

版本:3.0



华为技术有限公司

版权所有 © 华为技术有限公司 2020。保留一切权利。

未经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址：深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址：<http://e.huawei.com>

华为认证体系介绍

于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，华为公司打造了业界唯一覆盖ICT全技术领域的认证体系，包含ICT技术架构认证、平台与服务认证和行业ICT认证三类认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代的新型ICT人才，构建良性的ICT人才生态。

华为认证HCIA-AI V3.0定位于培养和认证具备使用机器学习、深度学习等算法设计、开发AI产品和解决方案能力的工程师。

通过HCIA-AI V3.0认证，将证明您了解人工智能发展历史、华为昇腾AI体系和全栈全场景AI战略知识，掌握传统机器学习和深度学习的相关算法；具备利用TensorFlow开发框架和MindSpore开发框架进行搭建、训练、部署神经网络的能力；能够胜任人工智能领域销售、市场、产品经理、项目管理、技术支持等岗位。



前言

简介

本书为 HCIA-AI 认证培训教程，适用于准备参加 HCIA-AI 考试的学员或者希望了解 AI 基础知识的读者。掌握本实验手册内容，您将能够了解利用华为云实现人脸识别（人脸检测、人脸比对与人脸搜索）、图像识别（图像标签、翻拍识别）、文字识别（图文识别、护照识别、表格识别）、图像内容审核、语音合成和语音识别功能的实际开发。实验操作主要是通过提交 Restful 请求调用华为公有云服务中的相关服务，华为企业云 EI 服务为大家提供了丰富的 AI 应用的接口。

内容描述

本实验指导书共包含 5 组实验，涉及到基于 Python 语言的华为云人脸识别、图像识别、文字识别、图像内容审核、语音合成和语音识别实验的内容，帮助学员提升 AI 的实践开发能力。

- 实验 1 掌握如何使用华为云的人脸识别服务。
- 实验 2 掌握如何使用华为云的图像识别服务。
- 实验 3 掌握如何使用华为云的文字识别服务。
- 实验 4 掌握如何使用华为云的图像内容审核服务。
- 实验 5 掌握如何使用华为云的语音合成和语音识别服务。

读者知识背景

本课程为华为认证开发课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具有基本的 Python 语言编程能力，会使用 Jupyter Notebook。
- 有一定的网络基础，能够理解华为云服务返回的状态信息。

实验环境说明

- Python3.6 Jupyter Notebook，建议通过 Anaconda3-5.2.0 安装。
- 实验需要访问网络，请保持网络连接。

目录

前 言	3
简介	3
内容描述	3
读者知识背景	3
实验环境说明	3
1 人脸识别	7
1.1 实验简介	7
1.2 实验目的	7
1.3 实验准备	8
1.3.1 环境准备	8
1.3.2 SDK 获取和配置	9
1.4 实验步骤	9
1.5 实验小结	12
2 图像识别	13
2.1 实验简介	13
2.2 实验目的	13
2.3 实验准备	14
2.3.1 环境准备	14
2.3.2 SDK 获取和配置	14
2.4 实验步骤	15
2.5 实验小结	18
3 文字识别	19

3.1 实验简介.....	19
3.2 实验目的.....	19
3.3 实验准备.....	20
3.3.1 环境准备.....	20
3.3.2 SDK 获取和配置.....	20
3.4 实验步骤.....	21
3.5 实验小结.....	29
4 内容审核.....	30
4.1 实验简介.....	30
4.2 实验目的.....	30
4.3 实验准备.....	30
4.3.1 环境准备.....	30
4.3.2 SDK 获取和配置.....	31
4.4 实验步骤.....	31
4.5 实验小结.....	33
5 语音合成和语音识别.....	34
5.1 实验简介.....	34
5.2 实验目的.....	34
5.3 实验准备.....	35
5.3.1 环境准备.....	35
5.3.2 SDK 获取和配置.....	35
5.4 实验步骤.....	36
5.4.1 语音合成.....	36
5.4.2 语音识别.....	38
5.5 实验小结.....	40

1 人脸识别

1.1 实验简介

人脸识别服务，是基于人的脸部特征信息，利用计算机对人脸图像进行处理、分析和理解，进行身份识别的一种智能服务。人脸识别以开放 API（Application Programming Interface，应用程序编程接口）的方式提供给用户，用户通过实时访问和调用 API 获取人脸处理结果，帮助用户自动进行人脸的识别、比对以及相似度查询等，打造智能化业务系统，提升业务效率。

人脸检测：人脸检测是在图像中准确识别出人脸的位置和大小。用户通过该服务，可以同时识别出图片中包含的不同倾角正脸及侧脸。该子服务是人脸识别领域的基础服务，适用于安防、电子身份、公安刑侦等众多应用场景。

人脸比对：通过对人脸区域的特征进行对比，该服务可以返回给用户两张图片中人脸的相似度。如果两张图片中包含多张人脸，则在两张图片中选取最大的人脸进行相似度比对。

人脸搜索：人脸搜索给用户提供了人脸集合操作相关的 API。用户可以通过创建人脸集合接口创建属于用户的人脸集；通过添加人脸接口向人脸集中添加图片；通过查询人脸接口，返回与输入人脸相似度最高的 N 张人脸图片；通过删除人脸接口从人脸集中删除用户不需要的人脸图片；通过删除人脸集接口删除用户创建的人脸集合。人脸搜索可用于企业、住宅的安全管理、公安刑侦等多种场景，但不可用于防翻拍场景。

本服务可以通过两种方式发布 POST 的 Restful HTTP 请求服务，一种是通过调用 SDK 封装的底层接口进行 Restful 服务发布，另一种是模拟前端浏览器访问的 Restful 服务发布方式。前者需要使用用户的 AK\SK 进行用户身份认证，后者需要获取用户 Token 进行用户身份认证。本实验将使用 AK\SK 认证方式来发布请求服务。

1.2 实验目的

本实验主要介绍了使用华为云服务中人脸识别的功能，通过本实验学员将了解如何结合华为云中的人脸识别服务进行通用人脸检测、人脸比对和人脸搜索。目前华为云提供了人脸识别的 SDK，本实验将指导

学员理解和掌握如何使用 Python 编程，在 Jupyter Notebook 中进行人脸识别业务开发的方法和技巧。

1.3 实验准备

1.3.1 环境准备

1. 注册并登录华为云管理控制台。
2. 了解人脸识别相关文档，详见 https://support.huaweicloud.com/api-face/face_02_0052.html。
3. 获取人脸识别服务的 Endpoint (<https://developer.huaweicloud.com/endpoint?all> 人脸识别服务“华北-北京四”对应的 Endpoint 为 `face.cn-north-4.myhuaweicloud.com`)。
4. 开通人脸识别服务：登录人脸识别管理控制台 (<https://console.huaweicloud.com/frs/?locale=zh-cn#/frs/management/faceRecognized>)，选择右上角的版本：版本二，选择对应的子服务，单击右侧的“开通服务”。服务开通一次即可，后续使用时无需再开通。
5. 获取华为云账号的AK/SK。如果之前没有生成过AK/SK，可登录华为云，在用户名处点击“我的凭证”，在“我的凭证”界面，选择“管理访问密钥 > 新增访问密钥”来获取，下载认证账号的AK/SK，请妥善保管AK/SK信息。之后的实验不用再新增，可以直接使用此AK/SK信息。
6. 获取project_id。在“我的凭证”界面的项目列表中查看项目ID，复制所属区域的项目ID为自己的project_id。

项目ID 项目	项目	所属区域
07693c7eba000fe22f9	cn-north-4	华北-北京四
07691909f080266a2f4	cn-east-2	华东-上海二

7. 已经安装好 Python 环境，Python SDK 适用于 Python3，推荐使用 Python3.6。
8. 实验数据获取：https://data-certification.obs.cn-east-2.myhuaweicloud.com/CHS/HCIA-AI/V3.0/face_data_demo.rar

1.3.2 SDK 获取和配置

1. 下载人脸识别服务的 Python SDK (<https://github.com/huaweicloud/huaweicloud-sdk-python-frs>) 并解压。
2. 请确认已安装 Python 包管理工具 setuptools, 请确认已安装 requests 和 websocket-client, 可通过 “pip list” 命令查看已安装列表。如果没有安装, 请使用以下命令安装:

```
pip install setuptools
pip install requests
pip install websocket-client
```

3. 命令行切换到 Python SDK 解压目录。
4. 在 SDK 目录中, 执行 python setup.py install 命令安装 Python SDK 到开发环境, 或者将.py 文件直接引入项目。

1.4 实验步骤

该实验需要在华为公有云服务上下载人脸识别的 SDK, 通过 AK\SK 信息进行身份认证从而调用 SDK 底层接口服务进行 Restful 服务请求的提交, 本实验就是通过 SDK 来调用人脸识别的服务的, 并在 Jupyter Notebook 中实验, 具体步骤如下:

步骤 1 导入所需要的包

```
# -*- coding: utf-8 -*-
from frsclient import AuthInfo
from frsclient import FrsClient
```

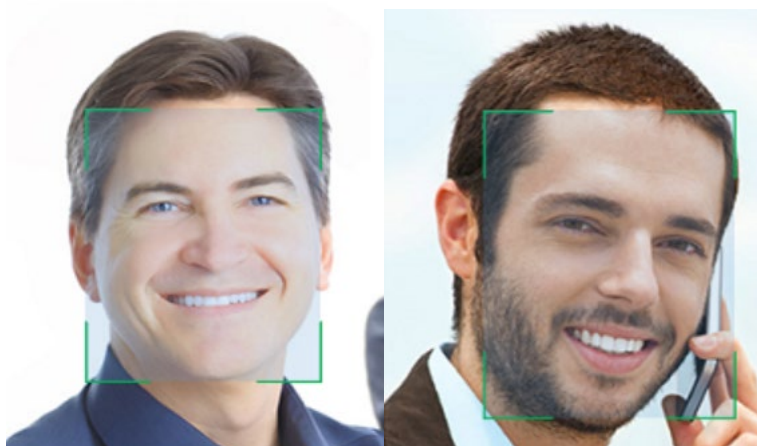
步骤 2 配置相关参数

```
ak = "****" #配置自己的 ak
sk = "****" #配置自己的 sk
project_id = "****" #配置自己的 project_id
region = "cn-north-4" #默认使用北京-4 区, 对应的区域代码即为 cn-north-4
end_point = "https://face.cn-north-4.myhuaweicloud.com" # 对应 cn-north-4
```

步骤 3 配置实验数据路径

```
#有人脸图像的数据路径
image1_path = "data/1.png"
image2_path = "data/2.png"
```

备注：由于过大图片对识别算法精度无明显提升，同时会导致时延较长，建议传入图片小于 1MB，一般 500KB 左右足够。图片中人脸像素建议 120*120 以上。下面分别是 1.png 和 2.png 的实验图片。



步骤 4 创建服务客户端

```
auth_info = AuthInfo(ak=ak, sk=sk, end_point=end_point)
frs_client = FrsClient(auth_info=auth_info, project_id=project_id)
```

步骤 5 人脸检测测试

```
# Detect face
result = frs_client.get_v2().get_detect_service().detect_face_by_file(image1_path)
print(result.get_eval_result())
```

输出结果：

```
{'faces': [{'bounding_box': {'top_left_x': 70, 'top_left_y': 83, 'width': 160, 'height': 206}]}}
```

返回的结果类似这样的，返回结果表明，1.png 这个测试图像也就只有一张人脸，bounding_box 指的是人脸在图像中的左上角坐标和人脸区域的宽度和高度。

步骤 6 人脸比对测试

```
# 人脸比对
result = frs_client.get_v2().get_compare_service().compare_face_by_file(image1_path, image2_path)
print(result.get_eval_result())
```

输出结果：

```
{'image1_face': {'bounding_box': {'width': 160, 'top_left_x': 70, 'top_left_y': 83, 'height': 206}}, 'similarity': 0.396217829, 'image2_face': {'bounding_box': {'width': 171, 'top_left_x': 79, 'top_left_y': 64, 'height': 235}}}
```


返回的结果里包含两个人脸图像的人脸区域位置以及两张人脸的相似度值，这里的'similarity'代表的是相似度值，这个数据为 0.396217829，相似度还是很低的。不过对比 1.png 和 2.png 也能看出来这两张图像确实不是同一个人，当然也就相似度比较低了。

步骤 7 人脸搜索测试

```
# 创建人脸集
result = frs_client.get_v2().get_face_set_service().create_face_set("face_set_name")
print(result.get_eval_result())
print("-----")
# 添加人脸
result = frs_client.get_v2().get_face_service().add_face_by_file("face_set_name", image1_path)
print(result.get_eval_result())
print("-----")
# 人脸搜索测试
result = frs_client.get_v2().get_search_service().search_face_by_file("face_set_name", image2_path)
print(result.get_eval_result())
```

输出结果：

```
{'face_set_info': {'face_number': 0, 'face_set_id': 'vLDdJC1S', 'face_set_name': 'face_set_name',
'create_date': '2020-02-12 14:41:17', 'face_set_capacity': 100000}}
-----
{'face_set_id': 'vLDdJC1S', 'face_set_name': 'face_set_name', 'faces': [{'face_id': 'h0uz2avG',
'external_image_id': 'TYCVStyx', 'bounding_box': {'width': 160, 'top_left_x': 70, 'top_left_y': 83, 'height':
206}, 'external_fields': {}}]}
-----
{'faces': [{'face_id': 'h0uz2avG', 'external_image_id': 'TYCVStyx', 'bounding_box': {'width': 160, 'top_left_x':
70, 'top_left_y': 83, 'height': 206}, 'similarity': 0.39641142}]}
```

第一个输出结果说明：创建人脸集容量是 10 万，人脸集名字为 'face_set_name' 以及人脸集 id 为 'vLDdJC1S'。

第二个输出结果说明：成功添加了一张人脸图像，且此时人脸集 'face_set_name' 中只有一张人脸图像，且人脸的 id 为 'h0uz2avG'，还会有这张人脸图像的人脸位置信息。

第三个输出结果说明：搜索出来的人脸 id 为 'h0uz2avG'，以及对应的人脸位置信息，并给出相似度值。

备注：

如果想删除某个人脸集下的人脸图像，我们可以使用这样的代码：

```
frs_client.get_v2().get_face_service().delete_face_by_face_id(face_set_name = "****", face_id = "****"), 其中 “****” 是填入实际的人脸集名称和人脸 id，我们实验使用的人脸集为 'face_set_name'，此次实验中的人脸 id 为 'h0uz2avG'，注意根据实际情况来修改。
```

如果想删除某个人脸集，我们可以使用这样的代码：

```
frs_client.get_v2().get_face_set_service().delete_face_set("****"), 其中 “****” 是填入实际的人脸集，我们实验中的人脸集为 'face_set_name'。
```

1.5 实验小结

本章主要介绍了应用华为公有云上的人脸识别服务进行实验的具体操作，主要是通过 SDK 发布 RestFul 请求进行相关功能的实现，而在使用 SDK 发布 RestFul 请求时，需要借助进行必要用户认证信息的配置，在本章中主要针对 AK\SK 的方式进行了人脸识别相关服务的介绍和说明，帮助学员使用人脸识别服务进行人脸检测、人脸比对以及人脸搜索业务的开发。

2 图像识别

2.1 实验简介

图像识别，是指利用计算机对图像进行处理、分析和理解，以识别各种不同模式的目标和对象的技术。图像识别以开放 API（Application Programming Interface，应用程序编程接口）的方式提供给用户，用户通过实时访问和调用 API 获取推理结果，帮助用户自动采集关键数据，打造智能化业务系统，提升业务效率。

图像识别（Image Recognition），基于深度学习技术，可准确识别图像中的视觉内容，提供多种物体、场景和概念标签，具备目标检测和属性识别等能力，帮助客户准确识别和理解图像内容。

图像标签-自然图像的语义内容非常丰富，一个图像包含多个标签内容，图像标签可识别三千多种物体以及两万多种场景和概念标签，更智能、准确的理解图像内容，让智能相册管理、照片检索和分类、基于场景内容或者物体的广告推荐等功能更加准确。

翻拍识别-零售行业通常根据零售店的销售量进行销售奖励，拍摄售出商品的条形码上传后台是常用的统计方式。翻拍识别利用深度神经网络算法判断条形码图片为原始拍摄，还是经过二次翻拍、打印翻拍等手法二次处理的图片。利用翻拍识别，可以检测出经过二次处理的不合规范图片，使得统计数据更准确、有效。

本服务可以通过两种方式发布 POST 的 Restful HTTP 请求服务，一种是通过调用 SDK 封装的底层接口进行 Restful 服务发布，另一种是模拟前端浏览器访问的 Restful 服务发布方式。前者需要使用用户的 AK\SK 进行用户身份认证，后者需要获取用户 Token 进行用户身份认证。本实验将使用 AK\SK 认证方式来发布请求服务。

图像识别 SDK 是对图像识别提供的 REST API 进行的封装，以简化用户的开发工作。用户直接调用 Image SDK 提供的接口函数即可实现使用图像识别业务能力的目的。

2.2 实验目的

本实验主要介绍了使用华为云服务中图像识别的功能，通过本实验学员将了解如何结合华为云中的图像识别服务进行图像内容标签和翻拍识别的功能。目前华为公有云提供了图像识别的 Restful API 和基于

Python 语言的图像识别的 SDK，本实验将指导学员理解和掌握如何使用 Python 进行图像标签业务以及翻拍识别业务的开发方法和技巧。

2.3 实验准备

2.3.1 环境准备

1. 注册并登录华为云管理控制台。
2. 了解图像识别相关文档，详见 https://support.huaweicloud.com/sdkreference-image/image_04_0011.html。
3. 开通图像识别服务：登录图像识别管理控制台
(https://console.huaweicloud.com/image_recognition/?region=cn-north-4), 依次选择左侧的“图像标签”、“翻拍识别”，分别在界面单击“开通服务”。服务开通一次即可，后续使用时无需再开通。
4. 准备华为云账号的AK/SK。如果之前可以获取过，可以继续使用之前的AK/SK。如果之前没有生成过AK/SK，可登录华为云，在用户名处点击“我的凭证”，在“我的凭证”界面，选择“管理访问密钥 > 新增访问密钥”来获取，下载认证账号的AK/SK，请妥善保管AK/SK信息。之后的实验不用再新增，可以直接使用此AK/SK信息。
5. 准备project_id。如果之前已经获取过，还可以继续使用之前的project_id。如果没有获取过，可在“我的凭证”界面的项目列表中查看项目ID，复制所属区域的项目ID为自己的project_id。





项目ID	项目	所属区域
07693c7eba00fe22f9	cn-north-4	华北-北京四
07691909f080266a2f4	cn-east-2	华东-上海二

6. 已经安装好 Python 环境，Python SDK 适用于 Python3，推荐使用 Python3.6。

2.3.2 SDK 获取和配置

1. 下载图像识别服务的 Python SDK (<https://image-sdk-static.obs.cn-north-4.myhuaweicloud.com/sdk/python/imagepython.zip>) 并解压。我们主要使用其中 data 文件夹中的数据和 image_sdk 这个文件夹中的基础库。我们也可以使用自己的数据放在 data 文件夹中。注意，

我们的 Jupyter Notebook 代码文件需要与 image_sdk 这个文件夹同级别，如下图所示这样：

	.ipynb_checkpoints	2020/2/5 15:59	文件夹
	data	2020/2/5 16:17	文件夹
	image_sdk	2020/2/5 14:46	文件夹
	image_tagging_aksk_demo.ipynb	2020/2/5 15:38	IPYNB 文件

2. 请确认已安装 Python 包管理工具 setuptools，请确认已安装 requests 和 websocket-client，可通过“pip list”命令查看已安装列表。如果没有安装，请使用以下命令安装：

```
pip install setuptools
pip install requests
pip install websocket-client
```

2.4 实验步骤

该实验需要在华为公有云服务上下载图像识别的 SDK，通过 AK\SK 信息进行身份认证从而调用 SDK 底层接口服务进行 Restful 服务请求的提交，本实验就是通过 SDK 来调用图像识别的服务的，并在 Jupyter Notebook 中实验，具体步骤如下：

步骤 1 导入所需要的包

```
# -*- coding:utf-8 -*-
from image_sdk.utils import encode_to_base64
from image_sdk.image_tagging import image_tagging_aksk
from image_sdk.recapture_detect import recapture_detect_aksk
from image_sdk.utils import init_global_env
```

步骤 2 配置相关参数

```
ak = "****" #配置自己的 ak
sk = "****" #配置自己的 sk
region = "cn-north-4" #默认使用北京-4 区，对应的区域代码即为 cn-north-4
```

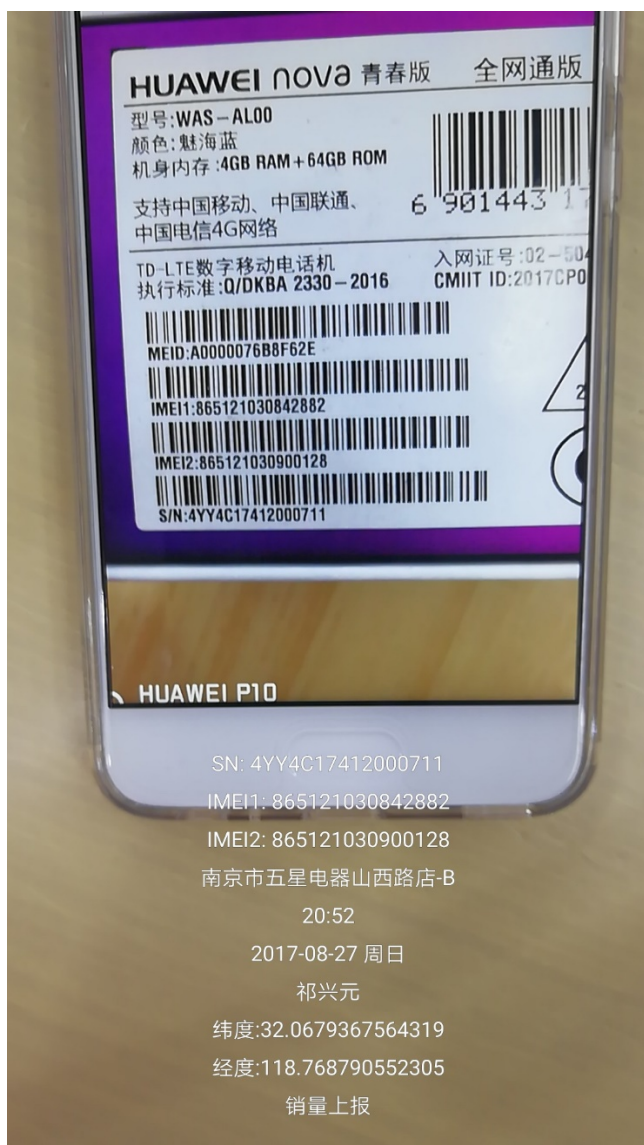
步骤 3 配置实验数据路径

```
img_tag_path="data/image-tagging-demo.jpg"
img_recapture_detect_path = "data/recapture-detect-demo.jpg"
```

image-tagging-demo.jpg 的数据图像如下：



recapture-detect-demo.jpg 的数据图像如下:



步骤 4 初始化全局环境

```
init_global_env(region)
```

步骤 5 图像标签测试

```
result = image_tagging_aksk(ak, sk, encode_to_base64(img_tag_path), "en", 5, 60)
print(result)
```

image_tagging_aksk 这个函数的前 2 个参数分别是 ak 和 sk，第 3 个参数是图像数据进行编码，第 4 个参数是 url 上的图片，如果使用本地图片，我们只需要这里传入空字符串即可，第 5 个参数是支持的语言，目前支持中文（“zh”）和英文（“en”），第 6 个参数表示最多返回的标签数，默认为-1 的话，代表返回所有标签，第 7 个参数指的是置信度的阈值（0~100），低于此置信度的标签，将不会返回，默认值为 0。

输出结果：

```
{"result":{"tags":[{"confidence":"100.0","i18n_tag":{"en":"Koala","zh":"考拉"},
{"tag":"Koala","type":"object"},{"confidence":"100.0","i18n_tag":{"en":"Marsupials","zh":"有袋目"},
{"tag":"Marsupials","type":"object"},{"confidence":"70.51","i18n_tag":{"en":"Veterinarians office","zh":"兽医办公室"},
{"tag":"Veterinarians office","type":"scene"}]}}
```

返回结果有 2 个标签，tags 代表标签列表集合，confidence 代表置信度，取值范围：0~100，i18n_tag 代表标签的国际化字段（zh 对应中文，en 对应英文），tag 代表标签名称，type 代表标签类别（object:实体标签，scene:场景标签，concept:概念标签）

步骤 6 图像翻拍检测测试

```
result = recapture_detect_aksk(ak, sk, encode_to_base64(img_recapture_detect_path), "0.75,
["recapture"])
print(result)
```

recapture_detect_aksk 这个函数的前 2 个参数分别是 ak 和 sk，第 3 个参数是图像数据进行编码，第 4 个参数是 url 上的图片，如果使用本地图片，我们只需要这里传入空字符串即可，第 5 个参数指的是置信度的阈值（0~100），达到此置信度才会被认为是翻拍的，第 6 个参数代表检测场景，当前仅支持翻拍照片场景：recapture，该参数可以为空，为空时返回所有的场景的检测结果。

输出结果：

```
{
  "result": {
    "suggestion": "false",
```

```
{
  "category": "recapture",
  "score": "1.0",
  "detail": [
    {
      "label": "recapture",
      "confidence": "1.0"
    }
  ]
}
```

suggestion 代表总体的结论，true：真实，false：虚假，uncertainty：不确定。category 代表标签（如果 suggestion 为真时，则该值为空字符串，否则不为空），recapture 代表是翻拍图，score 是总体置信度，取值范围 0–1.0，label 为标签值（original：原始图，recapture：翻拍图），confidence 代表置信度，取值范围：0–1.0。

2.5 实验小结

本章主要介绍了应用华为公有云上的图像标签和图像翻拍检测服务进行实验的具体操作，主要是通过 SDK 发布 RestFul 请求进行相关功能的实现，而在使用 SDK 发布 RestFul 请求时，需要借助进行必要用户认证信息的配置，在本章中主要针对 AK\SK 的方式进行了系统的介绍和说明，为使用图像标签和图像翻拍检测服务提供了实际的操作指导。

3 文字识别

3.1 实验简介

文字识别（Optical Character Recognition，OCR）是指对图像文件的打印字符进行检测识别，将图像中的文字转换成可编辑的文本格式。OCR 以开放 API（Application Programming Interface，应用程序编程接口）的方式提供给用户，用户通过实时访问和调用 API 获取推理结果，帮助用户自动采集关键数据，打造智能化业务系统，提升业务效率。

通用文字识别：提取图片内的文字及其对应位置信息，并能够根据文字在图片中的位置进行结构化整理工作。

通用表格识别：用于识别用户上传的通用表格图片（或者用户提供的华为云上 OBS 的通用表格图片文件的 URL）中的文字内容，并将识别的结果返回给用户。

护照识别：识别护照首页图片中的文字信息，并返回识别的结构化结果。当前版本支持中国护照的全字段识别。外国护照支持护照下方两行国际标准化的机读码识别，并可从中提取 6-7 个关键字段信息。

本服务可以通过两种方式发布 POST 的 Restful HTTP 请求服务，一种是通过调用 SDK 封装的底层接口进行 Restful 服务发布，另一种是模拟前端浏览器访问的 Restful 服务发布方式。前者需要使用用户的 AK\SK 进行用户身份认证，后者需要获取用户 Token 进行用户身份认证。本实验将使用 AK\SK 认证方式来发布请求服务。

3.2 实验目的

本实验主要介绍了使用华为云服务中文字识别的功能，通过本实验学员将了解如何结合华为云中的文字识别服务进行通用文字识别，护照识别和通用表格识别。本实验将指导学员理解和掌握如何基于 Python 进行文字识别相关业务的开发。

3.3 实验准备

3.3.1 环境准备

1. 注册并登录华为云管理控制台。

2. 了解文字识别相关文档，详见 https://support.huaweicloud.com/api-ocr/ocr_03_0047.html。

3. 开通文字识别服务：登录文字识别管理控制台

(<https://console.huaweicloud.com/ocr/?region=cn-north-4>), 依次选择左侧的“通用文字识别”、“通用表格识别”和“护照识别”，分别在界面单击“开通服务”。服务开通一次即可，后续使用时无需再开通。

4. 准备华为云账号的AK/SK。如果之前可以获取过，可以继续使用之前的AK/SK。如果之前没有生成过AK/SK，可登录华为云，在用户名处点击“我的凭证”，在“我的凭证”界面，选择“管理访问密钥 > 新增访问密钥”来获取，下载认证账号的AK/SK，请妥善保管AK/SK信息。之后的实验不用再新增，可以直接使用此AK/SK信息。

5. 准备project_id。如果之前已经获取过，还可以继续使用之前的project_id。如果没有获取过，可在“我的凭证”界面的项目列表中查看项目ID，复制所属区域的项目ID为自己的project_id。

项目ID	项目	所属区域
07693c7eba000fe22f9	cn-north-4	华北-北京四
07691909f080266a2f4	cn-east-2	华东-上海二

6. 已经安装好 Python 环境，Python SDK 适用于 Python3，推荐使用 Python3.6。

3.3.2 SDK 获取和配置

1. 下载文字识别服务的 Python SDK (<https://mirrors.huaweicloud.com/ocr-sdk/ocr-python-sdk/cloud-ocr-sdk-python-1.0.4.rar>) 并解压。我们主要使用其中 data 文件夹中的数据和 HWOcrClientAKSK.py 这个文件。我们也可以使用自己的数据放在 data 文件夹中。注意，我们的 Jupyter Notebook 代码文件需要与 HWOcrClientAKSK.py 这个文件和 data 文件夹同级。

2. 请确认已安装 Python 包管理工具 setuptools，请确认已安装 requests 和 websocket-client，可通过“pip list”命令查看已安装列表。如果没有安装，请使用以下命令安装：

```
pip install setuptools
pip install requests
pip install websocket-client
```

3.4 实验步骤

该实验需要在华为公有云服务上下载文字识别的 SDK，通过 AK\SK 信息进行身份认证从而调用 SDK 底层接口服务进行 Restful 服务请求的提交，本实验就是通过 SDK 来调用人脸识别的服务的，并在 Jupyter Notebook 中实验，具体步骤如下：

步骤 1 导入所需要的包

```
# -*- coding:utf-8 -*-
from HWOcrClientAKSK import HWOcrClientAKSK
```

步骤 2 配置相关参数

```
ak = "****" #配置自己的 ak
sk = "****" #配置自己的 sk
region = "cn-north-4" #默认使用北京-4 区，对应的区域代码即为 cn-north-4
req_text_uri = "/v1.0/ocr/general-text"#通用文字识别的 URI
req_table_uri = "/v1.0/ocr/general-table"#通用表格识别的 URI
req_passport_uri = "/v1.0/ocr/passport"#护照识别的 URI
```

步骤 3 配置数据路径

```
img_text_path = "data/general-text-demo.jpg"
img_table_path = "data/customs-form-en-demo.jpg"
img_passport_path = "data/passport-demo.jpg"
```

general-text-demo.jpg:

第 1 章 人际关系的构成 25

情感有关的“女人味”的技能称为表达性 (expressive) 特质。这两类特质在同一个人身上出现就一点也不奇怪了。双性化的人可能是这样一个人：在职场激烈的工资谈判中能有力、强悍地捍卫自己的利益；但回家后又能细腻、温柔地安慰刚刚失去宠物的孩子。大多数人只擅长于一种技能，在一种情境下显得游刃有余，另一种情境下就不会那么轻松。而双性化的人在两种情况下都能驾驭自如 (Cheng, 2005)。

实际上，最好把工具性和表达性看做男性和女性都拥有的、能高低变化的两组不同技能 (Choi et al., 2007)。符合传统期望的男人应该具有高工具性、低表达性特质，他们是坚忍、强悍的铁血真汉子；而符合传统期望的女人应该具有高表达性、低工具性特质，她们热情友好，但不够自信主动。双性化的人则既有工具性又有表达性特质。其余的人 (约 15%) 要么是在传统上属于异性的技能上高 (称为“跨类型”)，要么两组技能上都低 (称为“未分化”)。在双性化、跨类型和未分化各类型内的男女比例大体相当，所以就性别差异而言，把两性看做是具有不同特质的完全不同的两类人，既简单化又不准确 (Bem, 1993)。

不管怎样，人际关系的研究者都特别关注性认同差异。因为正是这些差异“实际上造成了许多不和谐”，最终导致人际关系失败，而不是让男性和女性变得更融洽 (Ickes, 1985, p. 188)。从相遇的那一刻起，传统的男性和女性并不如双性化的人那样欣赏和喜欢对方。在一项经典的实验中，研究者把男女配成对，一种条件下配对的男女都符合传统的性别角色，另一种条件下配对的双方至少有一个是双性化的。介绍两人认识后，他们有 5 分钟的时间在房间单独相处。研究者秘密地录下了这段时间男女双方的交往过程。结果令人吃惊，传统型的男女双方很少说话，很少注视对方，甚至看不到笑容，事后报告喜欢对方的程度也没有其他配对高 (Ickes & Barnes, 1978)。(请想一想：从行为方式来看，男子气十足的男性和女人味浓厚的女性有哪些共同点?) 如果双性化的男人遇上传统型的女人，或双性化的女人遇上传统型的男人，或者两个双性化的男女碰到一起，他们都比两个传统型的男女相处更为

更重要的是，传统型夫妇的这些缺憾并不会随着时间的流逝而消失。对婚姻满意度的调查发现，坚持传统刻板性别角色的夫妇一般不如非传统的夫妇婚姻幸福 (Helms et al., 2006)。强悍大丈夫和温柔小女人因为在处事风格和才干能力上差别很大，他们并不如那些不太传统、不太符合刻板印象的夫妇幸福快乐。

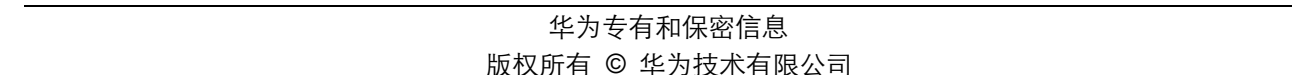
对此无需大惊小怪。人们一旦投身于亲密关系之中，就期望得到关爱、温情和理解 (Reis et al., 2000)。表达性低的人 (不怎么热情、温柔、敏感) 不太容易付出这些热情和温柔 (Basow & Rubenfeld, 2003)；也不怎么充满深情 (Miller et al., 2003)。因此，长期来看，与表达性低的配偶结婚的人就不如与那些更敏感、贴心和

customs-form-en-demo.jpg:

GOODS DECLARATION, GD-I										Custom File No.	
[] BILL OF ENTRY [] BILL OF EXPORT [] BAGGAGE DECLARATION										[] TRANSHIPMENT PERMIT	
1. EXPORTER'S/CONSIGNOR'S NAME AND ADDRESS HUAWEI TECHNOLOGIES CO LTD BANTIAN LONGGANG DISTRICT SHENZHEN P R CHINA				2003		2. DECLARATION TYPE HC		3. VALUATION METHOD 1		4. PREVIOUS REF 7. BANK CODE	
10. IMPORTER'S/CONSIGNEE'S/PASSENGER NAME & ADDRESS HUAWEI TECHNOLOGIES (PRIVATE) LIMITED				11. DECLARANT (OTHER THAN IMPORTER/EXPORTER) DHL GLOBAL FORWARDING # 189		9. DRY PORT IGM/EGM NO & DT RPAF-5161-2015 Date: 11-12-2015		INDEX 26			
14. NTN 1234567890		15. STR No / PASSPORT NO & DATE 1234567890		16. WAREHOUSE LIC NO		17. TRANSACTION TYPE False		18. COUNTRY OF DESTINATION HUAWEI		19. COUNTRY OF ORIGIN 176-2089	
18. DOCUMENTS ATTACHED E-Form No Date Value <input type="checkbox"/> INV <input type="checkbox"/> B/G <input type="checkbox"/> BL/AWB/ <input type="checkbox"/> IT EXMP <input type="checkbox"/> CO <input type="checkbox"/> <input type="checkbox"/> PL <input type="checkbox"/>				21. CURRENCY NAME & CODE US \$ 840		22. VESSEL MODE OF EK-612		23. BL AWL CON NO & DATE SZX60C8456		24. EXCHANGE RATE 104.000000	
25. PORT OF SHIPMENT Shenzhen				26. PAYMENT TERMS Without LC		27. PORT OF DISCHARGE		28. PLACE OF DELIVERY		29. DELIVERY TERMS CFR	
31. NUMBER OF PACKAGES 10.000		32. TYPE OF PACKAGE PACKAGES		33. GR OSS WT 0.09850 MT		34. Volume M3		35. GENERAL DESCRIPTION OF GOODS NET WT 0.09850 MT		36. IN THE CASE OF DANGEROUS GOODS INDICATE HAZARD CLASS/DIV/FLASH	
37. ITEM NO 1		38. QUANTITY (a) Unit Type KG		39. CO CODE China		40. SRO NO 5.5% Advance Income Tax u/s 148 in ce		41. HS Code 123456789.00		42. ITEM DESCRIPTION OF GOODS computer, phone	
43. UNIT VALUE Declared Assessed		44. TOTAL VALUE Declared Assessed		45. CUSTOM VALUE (PKR) Declared Assessed		46. LEVY CD 20.00 % ST 17.00 % ACD 1.00 % AST 3.00 % IT 5.50 %		47. RATE		48. SUM PAYABLE (PKR) 3968.0000 4081.0000 198.0000 720.0000 1584.0000	
46.0732 46.0732		188.90012 188.90012		19842.0000 19842.0000							
37. ITEM NO 2		38. QUANTITY (a) Unit Type NO		39. CO CODE China		40. SRO NO 5.5% Advance Income		41. HS Code 8517.7000		42. ITEM DESCRIPTION OF GOODS computer, phone, goods	
43. UNIT VALUE Declared Assessed		44. TOTAL VALUE Declared Assessed		45. CUSTOM VALUE (PKR) Declared Assessed		46. LEVY CD 5.00 % ST 17.00 % AST 3.00 % IT 5.50 %		47. RATE		48. SUM PAYABLE (PKR) 5667.0000 20231.0000 3570.0000 7854.0000	
59.9439 59.9439		1078.9902 1078.9902		113337.0000 113337.0000							
49. SRO / Test Report No & Dt		50. FOB VALUE 0.0000		51. FREIGHT 0.0000		52. CFR VALUE 2917.0191		53. INSURANCE 1.0000 %		54. LANDING CHARGES @ 1% False	
55. OTHER CHARGES 0.0000		56. ASSESSED VALUE PKR 306403.0000		57. TOTAL REBATE CLAIM/ PROV. ASSMNT US \$1							
58. MACHINE NO. & DATE RPAF-HC-8060-15-12-2015 104.000000 Appraiser: XXXXXX Examiner: XXXXXX		59. REVENUE RECOVER CODE LEVY CD 17222.00 ST 55114.00 ACD 577.00 AST 9726.00 IT 21387.00		60. AMOUNT (PKR)		61. A.O's name, sig & stamp XXXXXX		62. P.A.s name, sig & stamp XXXXXX		63. Out of Charge Sig & Stamp	
64. I declare that the above particulars are true & correct.		65. C/F/D NO & DATE C-RPAF-000730-15122015		66. Bank Stamp							

This is a system generated document, it does not require signature or stamp" as defined in sub section (kka) of Section 2 of Customs Act 1969

passport-demo.jpg:




```

        1712,
        39
    ],
    [
        2499,
        54
    ],
    [
        2499,
        115
    ],
    [
        1712,
        97
    ]
]
},
.....

```

Status code:200 代表访问服务正常；direction 代表图片朝向，当 detect_direction 为 true 时，该字段有效，返回图片逆时针旋转角度，值区间为[0,359]，当 detect_direction 为 false 时，该字段值为 -1；words_block_count 代表识别的文字块数目；words_block_list 代表识别文字块列表，输出顺序从左到右，先上后下；words 代表文字块识别结果，location 代表文字块的区域位置信息，列表形式，包含文字区域四个顶点的二维坐标（x,y），坐标原点为图片左上角，x 轴沿水平方向，y 轴沿竖直方向。

步骤 6 通用表格识别测试

```
response = ocr_client.request_ocr_service_base64(req_table_uri, img_table_path) # 调用 OCR API 识别表格图像.
```

```
print("Status code:" + str(response.status_code) + "\ncontent:" + response.text)
```

输出结果：

```
Status code:200
```

```
content:{
```

```
  "result": {
```

```
"words_region_count": 3,
"words_region_list": [
  {
    "type": "text",
    "words_block_count": 7,
    "words_block_list": [
      {
        "words": "GOODS DECLARATION. GD-I"
      },
      {
        "words": "Custom File No"
      },
      {
        "words": "[ ] BILL OF ENTRY"
      },
      {
        "words": "[ ]"
      },
      {
        "words": "BILL OF EXPORT"
      },
      {
        "words": "[ ] BAGGAGE DECLARATION"
      },
      {
        "words": "[ ] TRANSSHIPMENT PERMIT"
      }
    ]
  },
{
  "type": "table",
  "words_block_count": 72,
  "words_block_list": [
```



```
{
  "words": "1.EXPORTER'S/CONSIGNOR'S NAME AND ADDRESS\nHUAWEI
TECHNOLOGIES CO LTD \n2003 \nBANTIAN LONGGANG DISTRICT SHENZHEN P R CHINA",
  "rows": [
    0,
    1,
    2
  ],
  "columns": [
    0,
    1,
    2,
    3,
    4,
    5,
    6,
    7,
    8
  ]
},
{
  "words": "2.DECLARATION TYPE \nHC",
  "rows": [
    0
  ],
  "columns": [
    9,
    10
  ]
},
{
  "words": "3.VALUATION \nMETHOD",
  "rows": [
```

```

        0
    ],
    "columns": [
        11,
        12,
        13
    ]
} .....

```

Status code:200 代表访问服务正常； words_region_count 代表文字区域数目；

words_region_list 代表文字区域识别结果列表，输出顺序从左到右，先上后下； type 代表文字识别区域类型（ text：文本识别区域， table：表格识别区域）； words_block_count 代表子区域识别文字块数目； words_block_list 代表子区域识别文字块列表，输出顺序从左到右，先上后下； words 代表文字块识别结果； rows 代表文字块占用的行信息，编号从 0 开始，列表形式，仅在表格区域内有效，即 type 字段为"table"时该字段有效； columns 代表文字块占用的列信息，编号从 0 开始，列表形式，仅在表格区域内有效，即 type 字段为"table"时该字段有效。

步骤 7 护照识别测试

```
response = ocr_client.request_ocr_service_base64(req_passport_uri, img_passport_path) #调用 OCR
API 识别护照图像.
```

```
print("Status code:" + str(response.status_code) + "\ncontent:" + response.text)
```

输出结果：

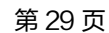
```
Status code:200
```

```
content:{
```

```

    "result": {
        "confidence": {
            "country_code": 0.9684,
            "sumame": 0.9684,
            "given_name": 0.9684,
            "passport_number": 0.9276,
            "date_of_birth": 0.9276,
            "sex": 0.9276,
            "date_of_expiry": 0.9276,
            "machine_code": 0.9684,

```



Status code:200 代表访问服务正常；country_code 代表护照签发国家的国家码；

passport_number 代表护照号码；nationality 代表持有人国籍；surname 代表姓；given_name 代表名字；sex 代表性别；date_of_birth 代表出生日期；date_of_expiry 护照有效期；confidence 代表相关字段的置信度信息，置信度越大，表示本次识别的对应字段的可靠性越高，在统计意义上，置信度越大，准确率越高，置信度由算法给出，不直接等价于对应字段的准确率；extra_info 默认为空，对于中国护照，extra_info 内会包含护照上由汉字描述的字段信息，如姓名、出生地等信息。

有些返回结果还有以下信息：passport_type 代表护照类型（P:普通因私护照、W:外交护照、G:公务护照），date_of_issue 代表护照签发日期，place_of_birth 代表出生地，place_of_issue 代表签发地，issuing_authority 代表签发机构，其中对中国的英文简写统一输出为 P.R.China。

本章主要介绍了应用华为公有云上的文字识别服务进行实验的具体操作，主要是通过 SDK 发布 Restful 请求进行相关功能的实现，而在使用 SDK 发布 Restful 请求时，需要借助进行必要用户认证信息的配置，在本章中主要针对 AK\SK 的方式进行了系统的介绍和说明，为使用文字识别服务进行通用文字识别、通用表格识别和护照识别业务提供了实际的操作指导。

4 内容审核

4.1 实验简介

随着互联网的飞速发展和信息量猛增，大量色情、暴力、政治敏感等不良信息夹杂其中，若不做好内容审核，不良内容会让用户产生反感，从而降低产品使用频率，最终远离产品。内容审核对用户上传的图片、文字、视频进行内容审核，自动识别涉黄、暴恐、政治敏感信息。内容审核以开放 API

(Application Programming Interface, 应用程序编程接口) 的方式提供给用户，用户通过实时访问和调用 API 获取推理结果，帮助用户自动采集关键数据，打造智能化业务系统，提升业务效率。

图像内容检测：利用深度神经网络模型对图片内容进行检测，准确识别图像中的涉政敏感人物、暴恐元素、涉黄内容等，帮助业务规避违规风险。

本服务可以通过两种方式发布 POST 的 Restful HTTP 请求服务，一种是通过调用 SDK 封装的底层接口进行 Restful 服务发布，另一种是模拟前端浏览器访问的 Restful 服务发布方式。前者需要使用用户的 AK\SK 进行用户身份认证，后者需要获取用户 Token 进行用户身份认证。本实验将使用 AK\SK 认证方式来发布请求服务。

4.2 实验目的

本实验主要介绍了使用华为云服务中内容审核的功能，通过本实验学员将了解如何结合华为云中的图像内容审核服务进行图像内容检测功能。本实验将指导学员理解和掌握如何使用 Python 进行图像内容审核业务的开发。

4.3 实验准备

4.3.1 环境准备

1. 注册并登录华为云管理控制台。
2. 了解图像内容审核相关文档，详见 https://support.huaweicloud.com/productdesc-moderation/moderation_01_0002.html。

3. 开通图像内容审核服务：登录图像内容审核管理控制台

(<https://console.huaweicloud.com/moderation/?region=cn-north-4>), 选择左侧的“图像内容检测”，在界面单击“开通服务”。服务开通一次即可，后续使用时无需再开通。

4. 准备华为云账号的AK/SK。如果之前可以获取过，可以继续使用之前的AK/SK。如果之前没有生成过AK/SK，可登录华为云，在用户名处点击“我的凭证”，在“我的凭证”界面，选择“管理访问密钥 > 新增访问密钥”来获取，下载认证账号的AK/SK，请妥善保管AK/SK信息。之后的实验不用再新增，可以直接使用此AK/SK信息。

5. 准备project_id。如果之前已经获取过，还可以继续使用之前的project_id。如果没有获取过，可在“我的凭证”界面的项目列表中查看项目ID，复制所属区域的项目ID为自己的project_id。

项目ID	项目	所属区域
07693c7eba000fe22f9	cn-north-4	华北-北京四
07691909f080266a2f4	cn-east-2	华东-上海二

6. 已经安装好 Python 环境，Python SDK 适用 Python3，推荐使用 Python3.6。

4.3.2 SDK 获取和配置

1. 下载内容审核服务的 Python SDK (<https://image-sdk-static.obs.cn-north-4.myhuaweicloud.com/sdk/python/moderationpython.zip>) 并解压。我们主要使用其中 data 文件夹中的数据 and moderation_sdk 这个文件夹。我们也可以使用自己的数据放在 data 文件夹中。注意，我们的 Jupyter Notebook 代码文件需要与 moderation_sdk 和 data 文件夹同级别。
2. 请确认已安装 Python 包管理工具 setuptools，请确认已安装 requests 和 websocket-client，可通过“pip list”命令查看已安装列表。如果没有安装，请使用以下命令安装：

```
pip install setuptools
pip install requests
pip install websocket-client
```

4.4 实验步骤

该实验需要在华为公有云服务上下载内容审核的 SDK，通过 AK\SK 信息进行身份认证从而调用 SDK 底层接口服务进行 Restful 服务请求的提交，本实验就是通过 SDK 来调用图像内容审核的服务的，并在 Jupyter Notebook 中实验，具体步骤如下：

步骤 1 导入所需要的包

```
# -*- coding:utf-8 -*-  
from moderation_sdk.utils import encode_to_base64  
from moderation_sdk.moderation_image import moderation_image_aksk  
from moderation_sdk.utils import init_global_env
```

步骤 2 配置相关参数

```
ak = "****" #配置自己的 ak  
sk = "****" #配置自己的 sk  
region = "cn-north-4" #默认使用北京-4 区，对应的区域代码即为 cn-north-4
```

步骤 3 配置数据路径

```
img_path = "data/moderation-terrorism.jpg"
```

步骤 4 初始化全局环境

```
init_global_env(region)
```

步骤 5 图像内容审核测试

```
result = moderation_image_aksk(ak, sk, encode_to_base64(img_path), "[porn', 'politics', 'terrorism', 'ad'])  
print(result)
```

输出结果：

```
{"result":{"suggestion":"review","category_suggestions":{"politics":"pass","ad":"pass","terrorism":"review","p  
orn":"pass"},"detail":{"politics":[],"ad":[{"confidence":0.0,"label":"ad"}, {"confidence":1.0,"label":"normal"}], "terr  
orism":[{"confidence":0.999,"label":"bloody"}, {"confidence":0.0001,"label":"terrorist"}, {"confidence":0.0,"label  
":"fascist"}, {"confidence":0.0,"label":"cult"}, {"confidence":0.0004,"label":"negative_politics"}, {"confidence":0.  
0,"label":"negative_political_events"}, {"confidence":0.0,"label":"special_characters"}, {"confidence":0.0,"labe  
l":"kidnap"}, {"confidence":0.0,"label":"corpse"}, {"confidence":0.0,"label":"riot"}, {"confidence":0.0,"label":"para  
de"}, {"confidence":0.001,"label":"normal"}], "porn":[{"confidence":1.0,"label":"normal"}, {"confidence":0.0,"label  
":"porn"}, {"confidence":0.0,"label":"sexy"}]}}
```

suggestion 代表检测结果是否通过（block：包含敏感信息，不通过，pass：不包含敏感信息，通过，review：需要人工复检）；category_suggestion 代表具体每个场景的检测结果（block：包含敏感信息，不通过，pass：不包含敏感信息，通过，review：需要人工复检）。

detail 是针对选定的每个检测场景列出结果列表；politics 为涉政敏感人物检测结果，terrorism 为涉政暴恐检测结果，porn 为涉黄检测结果，如果检测场景中的最高置信度也未达到 threshold 则结果列表为空；confidence 代表置信度，范围 0-1.0；face_detail 代表 politics 场景中的人物面部信息；label

代表每个检测结果的标签（politics：label 为对应的政治人物信息，terrorism：label 为对应的涉政暴恐元素信息，porn：label 为对应的涉黄分类（涉黄、性感等）信息，ad：label 为对应的广告识别结果（普通、广告）信息）。

从测试出的结果来看，category_suggestions 这一项中的 terrorism 为“review”，说明测试图像可能是有恐怖主义色彩的图像，需要人工复检。

4.5 实验小结

本章主要介绍了应用华为公有云上的图像内容审核服务进行实验的具体操作，主要是通过 SDK 发布 RestFul 请求进行相关功能的实现，而在使用 SDK 发布 RestFul 请求时，需要借助进行必要用户认证信息的配置，在本章中主要针对 AK\SK 的方式进行了系统的介绍和说明，帮助学员使用图像内容审核服务提供了实际的操作指导。

5 语音合成和语音识别

5.1 实验简介

华为云上的语音交互服务中，有语音合成和语音识别服务，本实验的内容是定制版语音合成和定制版的一句话识别服务。

语音合成（Text To Speech, TTS），又称文语转换，是一种将文本转换成逼真语音的服务。语音合成以开放 API（Application Programming Interface，应用程序编程接口）的方式提供给用户，用户通过实时访问和调用 API 获取语音合成结果，将用户输入的文字合成为音频。通过音色选择、自定义音量、语速，为企业和个人提供个性化的发音服务。

语音识别（Automatic Speech Recognition, ASR），将口述音频转换为文本。语音识别以开放 API（Application Programming Interface，应用程序编程接口）的方式提供给用户，用户通过实时访问和调用 API 获取语音识别结果。当前语音识别提供了短语音识别和长语音识别功能，短语音识别对时长较短的语音识别速度更快，长语音识别对时长较长的录音文件转写效果更好。

一句话识别服务：可以实现 1 分钟以内、不超过 4MB 的音频到文字的转换。对于用户上传的完整的录音文件，系统通过处理，生成语音对应文字内容。

本服务可以通过两种方式发布 POST 的 Restful HTTP 请求服务，一种是通过调用 SDK 封装的底层接口进行 Restful 服务发布，另一种是模拟前端浏览器访问的 Restful 服务发布方式。前者需要使用用户的 AK\SK 进行用户身份认证，后者需要获取用户 Token 进行用户身份认证。本实验将使用 AK\SK 认证方式来发布请求服务。

5.2 实验目的

本实验主要介绍了使用华为云服务中语音合成与识别的功能，通过本实验，学员将了解如何结合华为云进行语音合成与语音识别的功能。本实验将指导学员理解和掌握如何使用 Python 进行该业务的开发。

5.3 实验准备

5.3.1 环境准备

1. 注册并登录华为云管理控制台。
2. 了解语音合成与语音识别相关文档，详见 https://support.huaweicloud.com/api-sis/sis_03_0111.html 和 https://support.huaweicloud.com/api-sis/sis_03_0040.html。
3. 开通语音合成与语音识别服务：登录语音交互服务管理控制台（<https://console.huaweicloud.com/speech/?region=cn-north-4>），选择左侧的“定制语音识别”和“定制语音合成”，然后在“定制语音识别”项，选择“一句话识别”，然后在界面单击“开通服务”，另外在“定制语音合成”项也单击“开通服务”。服务开通一次即可，后续使用时无需再开通。
4. 准备华为云账号的AK/SK。如果之前可以获取过，可以继续使用之前的AK/SK。如果之前没有生成过AK/SK，可登录华为云，在用户名处点击“我的凭证”，在“我的凭证”界面，选择“管理访问密钥 > 新增访问密钥”来获取，下载认证账号的AK/SK，请妥善保管AK/SK信息。之后的实验不用再新增，可以直接使用此AK/SK信息。
5. 准备project_id。如果之前已经获取过，还可以继续使用之前的project_id。如果没有获取过，可在“我的凭证”界面的项目列表中查看项目ID，复制所属区域的项目ID为自己的project_id。

项目ID	项目	所属区域
07693c7eba000fe22f9	cn-north-4	华北-北京四
07691909f080266a2f4	cn-east-2	华东-上海二

6. 已经安装好 Python 环境，Python SDK 适用于 Python3，推荐使用 Python3.6。

5.3.2 SDK 获取和配置

1. 下载语音交互服务的 Python SDK（<https://mirrors.huaweicloud.com/sis-sdk/python/huaweicloud-python-sdk-sis-1.0.0.rar>）并解压。data 文件夹中的数据我们可以用，代码与 data 文件夹同级别即可，我们也可以使用自己的数据放在 data 文件夹中。
2. 请确认已安装 Python 包管理工具 setuptools，请确认已安装 requests 和 websocket-client，可通过“pip list”命令查看已安装列表。如果没有安装，请使用以下命令安装：

```
pip install setuptools
```

```
pip install requests
pip install websocket-client
```

3. 命令行切换到 Python SDK 解压目录。
4. 在 SDK 目录中，执行 `python setup.py install` 命令安装 Python SDK 到开发环境，或者将.py 文件直接引入项目。

5.4 实验步骤

该实验需要在华为公有云服务上下载语音交互服务的 SDK，通过 AK\SK 信息进行身份认证从而调用 SDK 底层接口服务进行 Restful 服务请求的提交，本实验就是通过 SDK 来调用语音合成和语音识别服务的，并在 Jupyter Notebook 中实验。以下的实验我们先进行语音合成生成语音数据，然后我们再用语音识别对语音数据进行识别。具体步骤如下：

5.4.1 语音合成

定制语音合成，是一种将文本转换成逼真语音的服务。用户通过实时访问和调用 API 获取语音合成结果，将用户输入的文字合成为音频。通过音色选择、自定义音量、语速，为企业和个人提供个性化的发音服务。

步骤 1 导入所需要的包

```
# -*- coding: utf-8 -*-
from huaweicloud_sis.client.tts_client import TtsCustomizationClient
from huaweicloud_sis.bean.tts_request import TtsCustomRequest
from huaweicloud_sis.bean.sis_config import SisConfig
from huaweicloud_sis.exception.exceptions import ClientException
from huaweicloud_sis.exception.exceptions import ServerException
import json
```

步骤 2 配置相关参数

```
ak = "****" #配置自己的 ak
sk = "****" #配置自己的 sk
project_id = "****" #配置自己的 project_id
region = "cn-north-4" #默认使用北京-4 区，对应的区域代码即为 cn-north-4
```

步骤 3 配置数据和保存路径

```
text = 'I like you, do you like me?'          # 待合成文本, 不超过 500 字
path = 'data/test.wav'                        # 保存路径, 可在设置中选择不保存本地
```

步骤 4 初始化客户端

```
config = SisConfig()
config.set_connect_timeout(5)                # 设置连接超时, 单位 s
config.set_read_timeout(10)                  # 设置读取超时, 单位 s
ttsc_client = TtsCustomizationClient(ak, sk, region, project_id, sis_config=config)
```

步骤 5 构造请求

```
ttsc_request = TtsCustomRequest(text)
# 设置请求, 所有参数均可不设置, 使用默认参数
# 设置属性字符串, language_speaker_domain, 默认 chinese_xiaoyan_common, 参考 api 文档
ttsc_request.set_property('chinese_xiaoyan_common')
# 设置音频格式, 默认 wav, 可选 mp3 和 pcm
ttsc_request.set_audio_format('wav')
# 设置采样率, 8000 or 16000, 默认 8000
ttsc_request.set_sample_rate('8000')
# 设置音量, [0, 100], 默认 50
ttsc_request.set_volume(50)
# 设置音高, [-500, 500], 默认 0
ttsc_request.set_pitch(0)
# 设置音速, [-500, 500], 默认 0
ttsc_request.set_speed(0)
# 设置是否保存, 默认 False
ttsc_request.set_saved(True)
# 设置保存路径, 只有设置保存, 此参数才生效
ttsc_request.set_saved_path(path)
```

步骤 6 语音合成测试

```
#发送请求, 返回结果。如果设置保存, 可在指定路径里查看保存的音频。
result = ttsc_client.get_ttsc_response(ttsc_request)
print(json.dumps(result, indent=2, ensure_ascii=False))
```

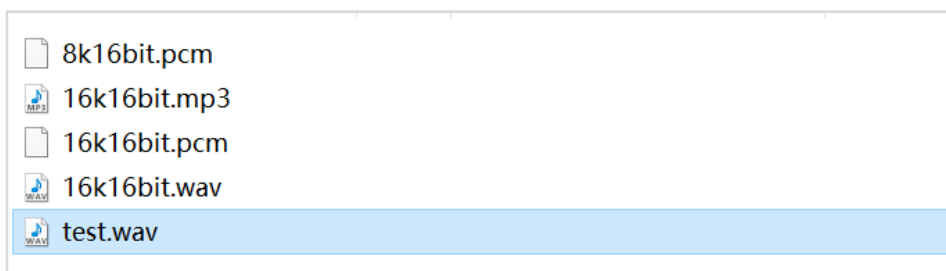
输出结果:

```
{
```

```
"result": {
"data": "UklGRuT..."
...
},
"trace_id": "b9295ebb-1c9c-4d00-b2e9-7d9f3dd63727",
"is_saved": true,
"saved_path": "data/test.wav"
}
```

trace_id 代表服务内部的令牌，可用于在日志中追溯具体流程，调用失败无此字段，在某些错误情况下可能没有此令牌字符串；result 代表调用成功表示识别结果，调用失败时无此字段。data 代表语音数据，base64 编码格式返回。

保存的语音数据如下：



5.4.2 语音识别

一句话识别接口，用于短语音的同步识别。一次性上传整个音频，响应中即返回识别结果。

步骤 1 导入所需要的包

```
# -*- coding: utf-8 -*-
from huaweicloud_sis.client.asr_client import AsrCustomizationClient
from huaweicloud_sis.bean.asr_request import AsrCustomShortRequest
from huaweicloud_sis.bean.asr_request import AsrCustomLongRequest
from huaweicloud_sis.exception.exceptions import ClientException
from huaweicloud_sis.exception.exceptions import ServerException
from huaweicloud_sis.utils import io_utils
from huaweicloud_sis.bean.sis_config import SisConfig
import json
```

步骤 2 配置相关参数

```
ak = "****" #配置自己的 ak
```

```
sk = "****" #配置自己的 sk
project_id = "****" #配置自己的 project_id
region = "cn-north-4" #默认使用北京-4 区，对应的区域代码即为 cn-north-4
```

步骤 3 配置数据和属性

```
# 一句话识别参数，我们使用语音合成的语音数据，1min 以内的音频
path = 'data/test.wav'
path_audio_format = 'wav' # 音频格式，详见 api 文档
path_property = 'chinese_8k_common' # language_sampleRate_domain, 如 chinese_8k_common, 详见 api 文档
```

步骤 4 初始化客户端

```
config = SisConfig()
config.set_connect_timeout(5) # 设置连接超时
config.set_read_timeout(10) # 设置读取超时
asr_client = AsrCustomizationClient(ak, sk, region, project_id, sis_config=config)#初始化客户端
```

步骤 5 构造请求

```
data = io_utils.encode_file(path)
asr_request = AsrCustomShortRequest(path_audio_format, path_property, data)
# 所有参数均可不设置，使用默认值
# 设置是否添加标点，yes or no，默认 no
asr_request.set_add_punc('yes')
```

步骤 6 语音识别测试

```
#发送请求，返回结果,返回结果为 json 格式
result = asr_client.get_short_response(asr_request)
print(json.dumps(result, indent=2, ensure_ascii=False))
```

输出结果：

```
{
  "trace_id": "e560a346-b1ce-4505-adf2-622fc7c2ec76",
  "result": {
    "text": "i like you do you like me。",
    "score": 0.0976272220950637
  }
}
```

trace_id 代表服务内部的令牌，可用于在日志中追溯具体流程，调用失败无此字段，在某些错误情况下可能没有此令牌字符串；result 代表调用成功表示识别结果，调用失败时无此字段；text 代表调用成功表示识别出的内容；score 代表调用成功表示识别出的置信度（0-1 之间），目前该值无参考意义。

5.5 实验小结

本章主要介绍了应用华为公有云上的语音交互服务中（语音合成和语音识别）进行实验的具体操作，主要是通过 SDK 发布 RestFul 请求进行相关功能的实现，而在使用 SDK 发布 RestFul 请求时，需要借助进行必要用户认证信息的配置，在本章中主要针对 AK\SK 的方式进行了系统的介绍和说明，帮助学员使用语音合成和语音识别提供了实际的操作指导。