总结报告

黎袁昊

2020.8.17

目录

一、项目背景

1.1 简介

该项目由两个容器构建：一个自行搭建的 Flask 容器和一个数据库 Cassandra 容器。

该项目的主要工作是识别图片进行分类。用户在命令行中提交图片后，图片会通过特定端口传输进 Flask 容器，并使用 TensorFlow 进行分类。分类结果将通过网页或命令行显示给用户，同时也通过 Docker 中的”bridge”网络模型将日期、图片名字和图片种类数据提交给 Cassandra 数据库容器并保存。

2.2 大数据与项目的联系

大数据是指无法在可承受的时间内用软硬件进行捕捉、管理和处理的数据集合，需要新处理模式才能使数据集合称为具有更强的决策力、洞察力和流程优化等能力的海量、多样化的信息资产。它具有四个特征（4V）：量大（Volume）；样多（Variety）；快速（Velocity）；价值（Value）。

然而大数据也带来新的挑战：传统网络架构不适用大数据时代。传统数据中心，计算、存储等各个子系统相对独立，用于大数据处理的数据中心，需要更高的资源利用率、自动化，需要使用虚拟化、云计算等技术对这些子系统进行整合和拉通。

云计算之于大数据，云计算是底层平台，大数据是应用。云计算作为底层平台整合计算、存储和网络等资源，同时提供基础脚骨资源弹性伸缩的能力。大数据在云计算平台的支撑下，调度下层资源，进行数据源加载，计算和最终结果输出等动作。

二、 项目知识回顾

2.1 神经网络

神经网络是一种模仿生物神经网络的结构和功能的数学模型或计算模型，用于对函数进行估计或近似。神经网络由大量的人工神经元联结进行计算。大多数情况下人工神经网络能在外界信息的基础上改变内部结构，是一种自适应系统。

现代神经网络是一种非线性统计性数据建模工具，神经网络通常是通过一个基于数学统计学类型的学习方法（Learning Method）得以优化，所以也是数学统计学方法的一种实际应用，通过统计学的标准数学方法我们能够得到大量的可以用函数来表达的局部结构空间，

2.2 TensorFlow

TensorFlow 是一个端到端开源机器学习平台。它拥有一个全面而灵活的生态系统，其中包含各种工具、库和社区资源，可使开发者能够轻松地构建和部署由机器学习提供支持的应用。

TensorFlow 在即刻执行环境中使用 Keras 等直观的高阶 API 轻松地构建和训练机器学习模型，该环境使我们能够快速迭代模型并轻松地调试模型。

同时 TensorFlow 都可以在云端、本地、浏览器中或设备上轻松地训练和部署模型。

2.3 Flask

Flask 是一个微框架，但“微”并不代表整个应用只能塞在一个 Python 文件内，也不代表 Flask 功能不强。微框架中的“微”字表示 Flask 的目标是保持核心简单而又可扩展。Flask 不会替用户做出许多决定，比如选用何种数据库。类似的决定，如使用何种模板引擎，是非常容易改变的

Flask 可以变成用户任何想要的东西，一切恰到好处，由用户做主。缺省情况下，Flask 不包含数据库抽象层、表单验证或者其他已有的库可以处理的东西。然而，Flask 通过扩展为用户的应用添加这些功能，就如同这些功能是 Flask 生的一样。大量的扩展用以支持数据库整合、表单验证、上传处理和各种开放验证等等。Flask 可能是 “微小”的，但它已经为满足用户的各种生产需要做出了充足的准备。

2.4 Docker 和 Kubernetes

Docker 是一个开源的用于开发，交付和运行应用程序的开放平台。Docker 使用户能够将应用程序与基础架构分开，让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，从而可以快速交付软件。借助 Docker，用户可以与管理应用程序相同的方式来管理基础架构。通过利用 Docker 的方法来快速交付，测试和部署代码，用户可以大大减少编写代码和在生产环境中运行代码之间的延迟。

Kubernetes，又称为 k8s "，是一种可自动实施 Linux 容器操作的开源平台。它可以帮助用户省去应用容器化过程的许多手动部署和扩展操作。也就是说，您可以将运行 Linux 容器的多组主机聚集在一起，由 Kubernetes 帮助您轻松高效地管理这些集群。而且，这些集群可跨公共云、私有云或混合云部署主机。

2.5 NoSQL 和 Cassandre

NoSQL 数据库提供一种机制来存储和检索数据，而不是关系数据库中使用的表格关系。这些数据库是无架构的，支持简单的复制，具有简单的 API，最终一致，并且可以处理大量的数据。

Apache Cassandra 是一个开源的分布式和分散式/分布式数据库。它用于管理遍布世界各地的大量结构化数据，提供高可用性的服务，并且没有单点故障，是一种 NoSQL 类型的数据库。

Cassandra 存在以下几点特性：

弹性可扩展性 - Cassandra 是高度可扩展的; 它允许添加更多的硬件以适应更多的客户和更多的数据根据要求。

始终基于架构 - Cassandra 没有单点故障，它可以连续用于不能承担故障的关键业务应用程序。

快速线性性能 - Cassandra 是线性可扩展性的，即它为你增加集群中的节点数量增加你的吞吐量。因此，保持一个快速的响应时间。

灵活的数据存储 - Cassandra 适应所有可能的数据格式，包括：结构化，半结构化和非结构化。它可以根据您的需要动态地适应变化的数据结构。

便捷的数据分发 - Cassandra 通过在多个数据中心之间复制数据，可以灵活地在需要时分发数据。

快速写入 - Cassandra 被设计为在廉价的商品硬件上运行。 它执行快速写入，并可以存储数百 TB 的数据，而不牺牲读取效率。

2.6 Hadoop 和 Mapreduce

Hadoop 是一个能够对大量数据进行分布式处理的软件框架。 Hadoop 以一种可靠、高效、可伸缩的方式进行数据处理。它是可靠的，因为它假设计算元素和存储会失败，因此它维护多个工作数据副本，确保能够针对失败的节点重新分布处理；它是高效的，因为它以并行的方式工作，通过并行处理加快处理速度；它还是可伸缩的，能够处理 PB 级数据；它的成本比较低，任何人都可以使用。

MapReduce 是一个基于集群的高性能并行计算平台,允许用市场上普通的商用服务器构成一个包含数十、数百至数千个节点的分布和并行计算集群;它是一个并行计算与运行软件框架，提供了一个庞大但设计精良的并行计算软件框架，能自动完成计算任务的并行化处理，自动划分计算数据和计算任务，在集群节点上自动分配和执行任务以及收集计算结果，将数据分布存储、数据通信、容错处理等并行计算涉及到的很多系统底层的复杂细节交由系统负责处理，大大减少了软件开发人员的负担；它是一个并行程序设计模型与方法借助于函数式程序设计语言 Lisp 的设计思想，提供了一种简便的并行程序设计方法，用 Map 和 Reduce 两个函数编程实现基本的并行计算任务，提供了抽象的操作和并行编程接口，以简单方便地完成大规模数据的编程和计算处理。

Hadoop 是一个实现了 MapReduce 模式的开源的分布式并行编程框架。

2.7 Spark

Apache Spark 是一个轻量级的内存集群计算平台，通过不同的组件来支撑批、流和交互式用例。

Apache Spark 是个开源和兼容 Hadoop 的集群计算平台。由加州大学伯克利分校的 AMPLabs 开发，作为 Berkeley Data Analytics Stack（BDAS）的一部分，当下由大数据公司 Databricks 保驾护航，更是 Apache 旗下的顶级项目。

Spark 相对于 Hadoop 做了如下的改进：Spark 速度更快；其次，Spark 丰富的 API 带来了更强大的易用性；最后，Spark 不单单支持传统批处理应用，更支持交互式查询、流式计算、机器学习、图计算等各种应用，满足各种不同应用场景下的需求。

三、 实验步骤

3.1 基于 Tensorflow 框架对服装图像进行分类

导入数据

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [================================] - 0s 6us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 357s 13us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 8us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 78s 18us/step
```

部分图片展示

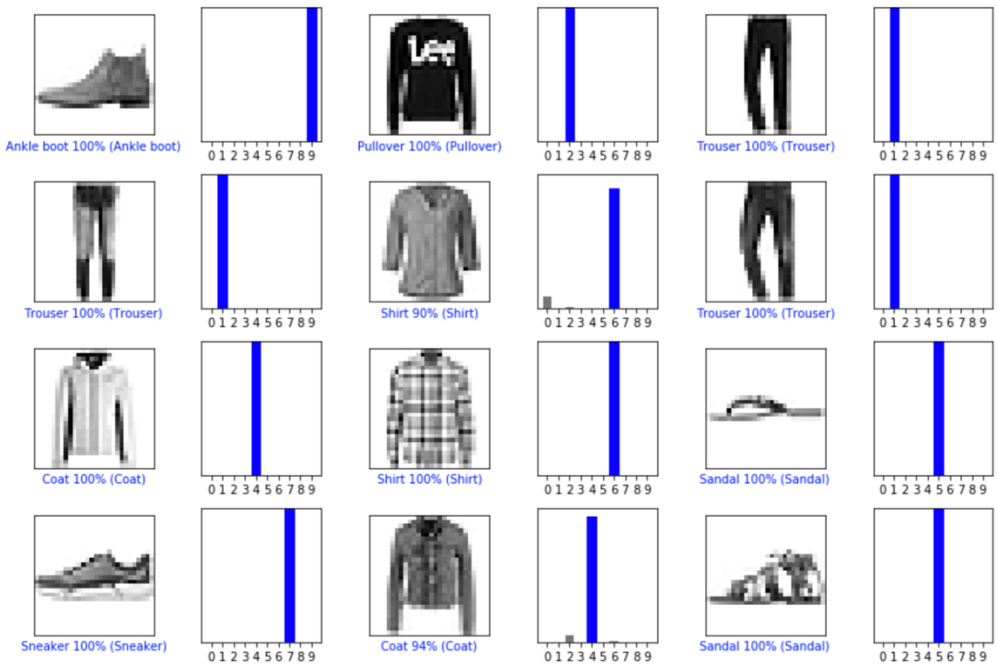
模型训练

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 4s 68us/sample - loss: 0.5030 - accuracy: 0.8239
Epoch 2/5
60000/60000 [==============================] - 3s 55us/sample - loss: 0.3799 - accuracy: 0.8631
Epoch 3/5
60000/60000 [==============================] - 3s 55us/sample - loss: 0.3408 - accuracy: 0.8773
Epoch 4/5
60000/60000 [==============================] - 3s 56us/sample - loss: 0.3158 - accuracy: 0.8839
Epoch 5/5
60000/60000 [==============================] - 3s 55us/sample - loss: 0.2999 - accuracy: 0.8906
```

网络结构

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

概率预测



模型保存

```
# 将整个模型保存为 HDF5 文件
# '.h5' 扩展名指示应将模型保存到 HDF5
model.save('my_model.h5')
```

3.2 基于 Docker 搭建 Flask 容器

创建 app.py 文件，并使用 route 装饰器告诉函数以何种 url 触发。

```
@app.route('/api/tasks/<int:task_id>', methods=['GET'])
def list_image(task_id):
    rows = cqlshOperate.getData()
    row = rows[task_id]
    task = {'id': 1}
    task['time'] = row[0]
    task['image_name'] = row[1]
    task['class_name'] = row[2]
    return jsonify(task), 200
@app.route('/api/tasks', methods=['GET'])
def list_all():
    tasks = []
    rows = cqlshOperate.getData()
    id = 1
    for row in rows:
        task = {'id': id}
        task['time'] = row[0]
        task['image_name'] = row[1]
        task['class_name'] = row[2]
        tasks.append(task)
        id += 1
    return jsonify(tasks), 200
@app.route('/api/tasks/upload', methods=['POST'])
def upload_image():
    img_bin = request.files['image']
    image_name = request.files['image'].filename
    img_bin.save('img.jpg')
    if image_name:
        image = imageOperate.process(image_name)
        label = imageOperate.predict(image)
        now = datetime.now()
        cqlshOperate.insertData(image_time=str(now),
                                image_name=image_name,
                                image_label=class_names[label])
        return class_names[label]
    abort(404)
```

创建 imageOperate.py 文件，进行图片尺寸处理及图片种类分类。

```
def process(image):
    img = Image.open(image).convert('L')
    if img.size[0] != 28 or img.size[1] != 28:
        img = img.resize((28, 28))
    img = np.array(img).reshape((1, 28, 28))
    return img / 255.0
def predict(image):
    predictions = model.predict(image)
    label = np.argmax(predictions[0])
    return label
```

创建 cqlshOperate.py 文件，进行 Cassandra 数据库操作。

```
def createKeySpace():
    log.info("Creating keyspace...")
    try:
        session.execute("""
            CREATE KEYSPACE %s
            WITH replication = {
                    'class': 'SimpleStrategy',
                    'replication_factor': '1' }
            """ % KEYSPACE)
        log.info("setting keyspace...")
        session.set_keyspace(KEYSPACE)
        log.info("creating table...")
        session.execute("""
            CREATE TABLE images_info (
                image_time text PRIMARY KEY,
                image_name text,
                image_label text,
            )
            """)
    except Exception as e:
        log.error("Unable to create keyspace")
        log.error(e)
```

```
def insertData(image_time, image_name, image_label):
    session.set_keyspace(KEYSPACE)
    info = [image_time, image_name, image_label]
    session.execute(
        """
    INSERT INTO images_info (image_time, image_name, image_label)
    VALUES (%s, %s, %s)
    """, info)

def getData():
    session.set_keyspace(KEYSPACE)
    rows = session.execute('select * from Images_info')
    return rows
```

创建 Dockerfile 文件，进行 Flask 容器配置调整。

```
FROM python:3.6.3
COPY . /app
WORKDIR /app
RUN curl https://bootstrap.pypa.io/get-pip.py | python
RUN pip install -r requirements.txt --index-url https://pypi.tuna.tsinghua.edu.cn/simple/
EXPOSE 8080 9042
CMD ["python", "app.py"]
```

创建 requirements.txt 文件，对容器中 Python 库进行约束下载。

```
numpy==1.16.0
tensorflow
flask
pillow
cassandra-driver
setuptools==41.0.0
```

打开终端，进入文件所在目录，输入以下指令，创建 flask 镜像。

```
# Build flask image
docker build -t flask-app .
```

3.3 基于 Docker 搭建 Cassandra 镜像

打开终端，输入以下指令，创建 Cassandra 镜像。

```
# Pull cassandra image from Docker Hub
docker pull cassandra:latest
```

3.4 运行 Flask 容器和 Cassandra 容器，并进行通信

打开终端，输入以下指令，创建 Docker 网络模型。

```
# Construct a network
docker network create my-net --driver bridge
```

输入以下指令，创建 Cassandra 容器。

```
# Run the cassandra container
docker run -d --name my-cassandra \
-v ~/docker/data/cassandra:/app/data \
-p 9042:9042 \
--network my-net \
cassandra:latest
```

输入以下指令，创建 Flask 容器。

```
# Run the flask server
docker run -d --name my-flask \
-v ~/docker/data/flask:/app/data \
-p 8080:8080 \
--network my-net \
flask-app
```

由于选择“bridge”网络模型，所以两容器可以在 my-net 中通信。

3.5 通过 Volume 加载模型

由于通过-v 指令将本地~/docker/data/flask 与/app/data 进行映射，故只需将模型文件移动至本地~/docker/data/flask 即可成功加载。

四、 实验结果

打开终端，进入图片文件所在目录，输入以下指令，获得运行结果。

```
curl -X POST -F 'image=@7.jpeg' -v http://127.0.0.1:8080/api/tasks/upload
* Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/tasks/upload HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.64.1
> Accept: */*
> Content-Length: 4680
> Content-Type: multipart/form-data; boundary=------------------------2cbef56cc74f7008
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
* We are completely uploaded and fine
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 3
< Server: Werkzeug/1.0.0 Python/3.7.1
< Date: Mon, 17 Aug 2020 08:08:01 GMT
<
* Closing connection 0
Bag
```

```
curl -X POST -F 'image=@8.jpeg' -v http://127.0.0.1:8080/api/tasks/upload

* Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/tasks/upload HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.64.1
> Accept: */*
> Content-Length: 33271
> Content-Type: multipart/form-data; boundary=------------------------80c2cc488c41feca
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
* We are completely uploaded and fine
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 3
< Server: Werkzeug/1.0.0 Python/3.7.1
< Date: Mon, 17 Aug 2020 08:08:09 GMT
<
* Closing connection 0
Bag
```

```
curl -X POST -F 'image=@9.jpeg' -v http://127.0.0.1:8080/api/tasks/upload
```

```
* Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/tasks/upload HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.64.1
> Accept: */*
> Content-Length: 7524
> Content-Type: multipart/form-data; boundary=------------------------23476bbe2a3aaec6
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
* We are completely uploaded and fine
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 6
< Server: Werkzeug/1.0.0 Python/3.7.1
< Date: Mon, 17 Aug 2020 08:08:16 GMT
<
* Closing connection 0
Sandal
```

```
curl -X GET http://127.0.0.1:8080/api/tasks
[
    {
        "class_name": "8.jpeg",
        "id": 1,
        "image_name": "Bag",
        "time": "2020-08-17 16:08:09.816061"
    },
    {
        "class_name": "9.jpeg",
        "id": 2,
        "image_name": "Sandal",
        "time": "2020-08-17 16:08:16.675968"
    },
    {
        "class_name": "7.jpeg",
        "id": 3,
        "image_name": "Bag",
        "time": "2020-08-17 16:08:01.655165"
    }
]
```

```
curl -X GET http://127.0.0.1:8080/api/tasks/2
{
    "class_name": "7.jpeg",
    "id": 1,
    "image_name": "Bag",
    "time": "2020-08-17 16:08:01.655165"
}
```

五、 讨论与展望

在本次项目中出现了各种各样的问题，比如由于某些原因导致使用 Docker 的速度比较慢、Python 访问 Cassandra 数据库失败和获取本地文件权限等等。每一个问题都花费了一天甚至数天的时间去思考解决。虽然所耗费的时间比较长，但是所收获到的东西却是无比珍贵的，比如已经习惯看纯英文的文档、经常查阅 Github 和 Stackoverflow 和对每个 bug 需要尽可能的耐心，逐步调试等等，这些都是在平时的校园生活中难以学习到的。

我希望通过这一次的项目，能够获得勇于探索未知技术的态度。在以后的学习生活，甚至之后的工作生活中，我能始终保持一种学习未知技术的热情，面对困难能够披荆斩棘，勇攀人生高峰。

六、 实验资源索引

项目资源下载： https://github.com/liyuanhao6/Flask-Cassandra

清华开源镜像： https://mirrors.tuna.tsinghua.edu.cn/