远程项目总结报告

1. 背景介绍

随着计算机和信息技术的迅猛发展和普及应用，行业应用系统的规模迅速扩大，行业应用所产生的数据呈爆炸性增长。

动辄达到数百 TB 甚至数十至数百 PB 规模的行业，企业大数据已远远超出了现有传统的计算技术和信息系统的处理能力，“信息泛滥”、“数据爆炸”等现象不绝于耳，海量的数据信息使得人们难以做出快速的抉择。全球互联网发展速度达到每半年就增加一倍，《纽约时报》在 20 世纪 60 年代的内容版面十几二十页左右，而如今已达到一百到二百页。国内出版业的《北京青年报》也已出版到四十版。据统计全球人均日阅读时间通常为三十分钟左右，也就说如此大的的数据量仅靠人的阅读来获得有效准确的信息已经难以为继。信息冗余、信息真假、信息安全、信息处理、信息统一等问题也随着大数据给人们带来价值的同时也造成了一系列的问题。人们不仅希望能够从大数据中提取出有价值的信息，更希望发现能够有效支持生产生活中需要决策的更深层次的规律。因此，寻求有效的大数据处理技术、方法和手段已经成为现实世界的迫切需求。

在现实情况的背景下，人们意识到如何有效地解决海量数据的利用问题具有研究价值和经济利益。但是面对如此海量的数据并且维度高、数据结构复杂，如何能够有效的从中挖掘出数据价值是人们需要面对的问题。面向大数据

的数据挖掘研究就显得十分重要。面向大数据的数据挖掘的特有两个最重要的任务。一是实时性，如此海量的数据规模需要实时分析并迅速反馈结果。二是准确性，需要我们从海量的数据中精准提取出隐含在其中的用户需要的有价值信息，再将挖掘所得到的信息转化成有组织的知识以模型等方式表示出来，从而将分析模型应用到现实生活中提高生产效率、优化营销方案等。

2. 项目使用技术介绍

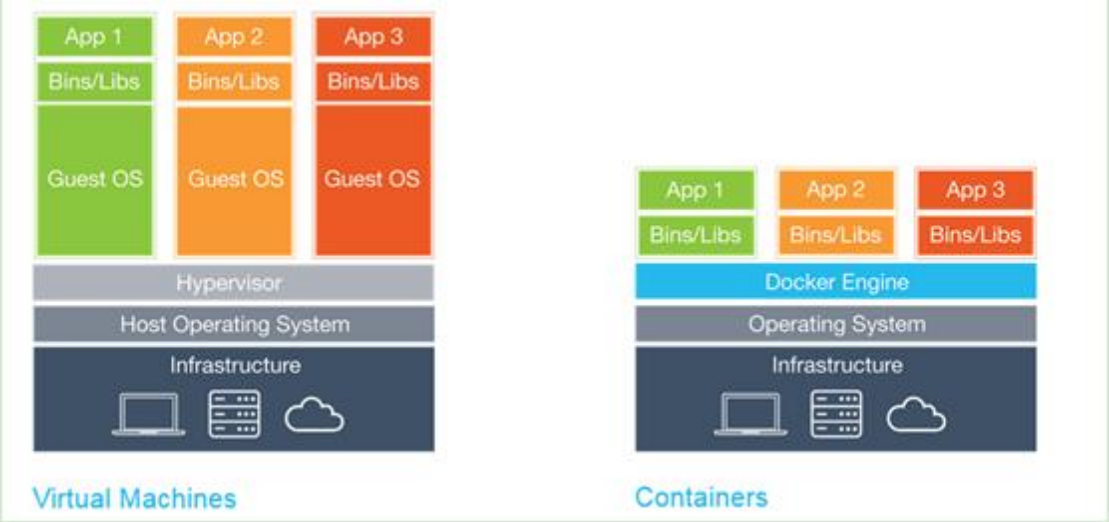
2.1 DOCKER

从各种 OS 到各种中间件到各种 app，一款产品能够成功作为开发者需要关心的东西太多，且难于管理，这个问题几乎在所有现代 IT 相关行业都需要面对，比如处理 Web 应用、后台应用、数据库应用、大数据应用比如 Hadoop 集群、消息队列等等问题时。

云计算时代的到来--AWS 的成功，引导开发者将应用转移到 cloud 上,

解决了硬件管理的问题，然而中间件相关的问题依然存在。cloud 时代采用标配硬件来降低成本，而 docker 的出现给软件开发者提供了新的思路——采用虚拟化手段来满足用户按需使用的需求以及保证可用性和隔离性。在这一点上 virtual machine 能够起到很好的作用，然而用户需要的是高效运行环境而非 OS,这就导致了传统的 vm 十分浪费资源(占用的内存过多)。面对上面的问题，docker 设想是交付运行环境如同海运，OS 如同一个货轮，每一个在 OS 基础上的软件都如同一个集装箱，用户可以通过标准化手段自由组装运行环境，同时集装箱的内容可以由用户自定义，也可以由专业人员制造。这样，交

付一个软件，就是一系列标准化组件的集合的交付，如同乐高积木，用户只需要选择合适的积木组合，并且在最顶端署上自己的名字（最后标准化组件是用户的 app）。如下图所示



可以很明显的看到 docker 不同于 vm 需要加载操作环境也省去了 hypervisor 层，节约了不少空间，使得整个程序的加载速度更快。

Docker 能让任何的应用和它的依赖打包成为一个轻量级的，便携的，独立的一个集装箱。集装箱有标准的运作模式，因此可以方便自动化。他们被设计成能跑在几乎所有 LINUX 服务器上的容器。一个开发者在笔记本上建立和测试的一个 Docker 容器，能跑在测试环境、生产环境、虚拟机、新买的服务器、**OpenStack** 集群、公共环境、或者是以上的组合的服务器群上。

2.2 REST API

Rest 就是用固定的 URI 和可变的参数访问某个服务，来完成一系列业务请求。其中每一个 URI 代表一种资源；客户端与服务器之间传递这种资源的某种表现层；客户端通过几个 HTTP 动词对服务器段资源进行操作，实现“表现层状态转化”。

而 REST API 的本质就是一个 HTTP 的请求，不同的动词就是不同数据的提交方式。所以，决定 REST API 的几个因素就是：URI，参数，请求方式与请求头。

对于这种 REST 的请求，常见的响应结果就是 XML 或者是 JSON 形式，往往结果中会包含请求状态和时间戳，业务系统响应结果。

REST API 有一个好处就是跨平台，因为大多数的前端编程语言如 java，php，node.js 等都支持 HTTP 请求，因此 API 只需保证提供该语言的 SDK，保证能够按照协议调用就行。

2.3 flask 框架

Flask 是 python 自己的一个 web 微框架，也是一个基于 MVC 设计模式的 web 框架。Flask 是个轻量级框架，没有默认的数据库和窗体验证工具，非常适合初学者上手，不需要花费多少就可以开发一个简单的网站，具有良好的拓展性。

3. 项目研究过程

本次的研究项目是开发一个简单的 MNIST 手写数据集的识别网络并将其布置在 DOCKER 容器中，发布到平台上。

3.1 神经网络的搭建

综合神经网络的性能以及识别物体的属性，我决定采用卷积神经网络来进行手写体识别。

首先将神经网络初步搭建出来，我使用了 TENSORFLOW 的第三方包来进行搭建，设置好神经网络的权重，偏置项，卷积层与池化层的数据。最后按照卷积层-池化层-全链接层-输出层链接好。下面贴出神经网络的部分代码以供参考：

```
def weight_variable(slef,shape):
    initial = tf.truncated_normal(shape,stddev = 0.1)
    return tf.Variable(initial)

def bias_variable(slef,shape):
    initial = tf.constant(0.1,shape = shape)
    return tf.Variable(initial)

def conv2d(self,x,W):
    return tf.nn.conv2d(x, W, strides = [1,1,1,1], padding = 'SAME')

def max_pool_2x2(self,x):
    return tf.nn.max_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME'

def work(self):
    W_conv1 = self.weight_variable([5, 5, 1, 32])
    b_conv1 = self.bias_variable([32])
```

```
def work(self):
    W_conv1 = self.weight_variable([5, 5, 1, 32])
    b_conv1 = self.bias_variable([32])

    x_image = tf.reshape(self.x,[-1,28,28,1])

    h_conv1 = tf.nn.relu(self.conv2d(x_image,W_conv1) + b_conv1)
    h_pool1 = self.max_pool_2x2(h_conv1)

    W_conv2 = self.weight_variable([5, 5, 32, 64])
    b_conv2 = self.bias_variable([64])

    h_conv2 = tf.nn.relu(self.conv2d(h_pool1, W_conv2) + b_conv2)
    h_pool2 = self.max_pool_2x2(h_conv2)

    W_fc1 = self.weight_variable([7 * 7 * 64, 1024])
    b_fc1 = self.bias_variable([1024])

    h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

在布置好神经网络后初步运行一遍，使用网上 MNIST 官网给出的数据集进行测试，并使用 TESNAOFLOW 中自带的 SAVER 将识别调整后的神经网络参数保存下来，方便后面使用。随后编写读取图片的步骤，这步采用了 PILLOW 包。将图片读入程序后对其进行预处理方便传入神经网络。最后将整个程序分装进一个类中方便后续调用。

3.2 CASSANDRA 数据库的搭建。

本次采用 CASSANDRA 数据库来对用户访问的信息进行储存，主要储存的是上传文件名，上传时间以及识别结果。在链接数据库时使用了 CASSANDRA-DRIVER 这个拓展包来进行链接。

首先先链接数据库，并创建制定名字的键空间，并在该空间内创建表存储信息。这样就完成了数据库的初始化。

随后编写数据插入的代码，使用%替代符来进行参数传入，避免注入攻击。数据库的基本功能完成了。

3.3 WEB 框架的搭建

本次采用了 FLASK 框架来进行搭建，因为本程序较为简单因此只需要编写主要的两个页面就可以完成初步要求：1.程序主页面，2.完成操作后的信息返回页面。

程序主页面的地址设定为主机访问地址，端口号使用 5000，信息跳转页面地址为主机地址后加上‘/UP’来表示上传的动作。根据识别的结果不同设计了两个页面，一为正常返回页面，另一个为错误发生页面，两者都设置了返回主页面的链接按钮。

程序主页面

识别成功页面

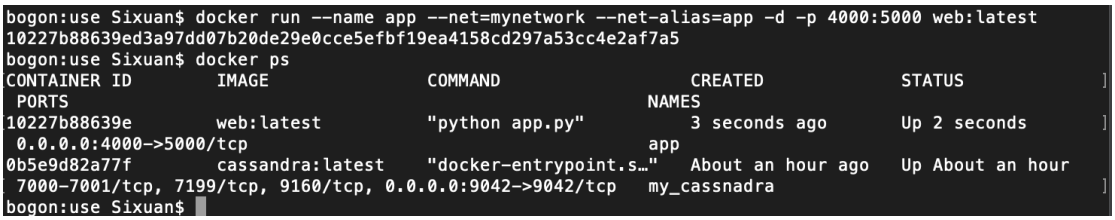
错误发生页面

在框架启动的同时启动数据库，在用户传入正确的图片后启动神经网络进行识别，完成识别进入正确跳转页面后捕捉用户信息完成数据导入数据库，如果用户输入信息错误或发生意外错误就跳转到报错页面，这就是整个项目工作的大体流程。

3．4 部署 DOCKER IMAGE

在整个项目在本地运转成功后，要把项目发布到平台上实现全平台使用。在本地的项目文件夹中首先创建 REQURIEMENTS.TXT 文件，将本次编写使用的第三方拓展包名字写入进去。随后进行 DOCKERFILE 的编写，这里至关重要，DOCKER IMAGE 能否创建成功完全取决于 DOCKERFILE 编写是否正确。注意创建好编程环境（本次项目在 PYTHON3.7 版本中完成），工作目录，导入第三方包，暴露本机端口，最后在 CMD 中运行 APP。这样就完成了 DOCKERFILE 的编写。

最后有一点需要注意，这个 IMAGE 需要提前启动 CASSANDRA 服务器否则无法正常运行，因此，在拉下来容器使用时应当提前在 DOCKER 中启动 CASSANDRA 服务并将两个容器链接在同一个 NETWORK 中才能正常运行。



DOCKER IMAGE 运行状况示意图

4. 项目总结

在为期一个月的学习中，老师在课堂上给我们讲解了不少先进的技术如用于存储大数据的 NOSQL 数据库 MONGODB，CASSANDRA 等，用于处理运算大数据的 HADOOP ，SPARK，将大数据可视化的图表工具 ARMCHARTS，JAVA-SCRIPT 库 D3 以及一系列的大数据工具。在课下我根据老师留下的任务一点点熟练相关的知识，最后我掌握了 FLASK 框架与 CASSANDRA 数据库的初步使用，并了解了 DOCKER 的工作流程与搭建流程，掌握了不少 DOCKER 与 LINUX 的终端指令，了解了主机之间对大数据进行处理的基本工作流程，大数据的几个应用方向。在初步接触到这种技术后我感到十分好奇，原来

从来没有接触过这些比较前沿的技术，也只是从课上老师和家长嘴里听到的模糊概念，在这里通过学习知识我对这些概念有了初步的意向与映像，激发了自己对于这方面的学习兴趣，特别是对于 DOCKER 技术的接触，让我十分兴奋，DOCKER 技术给我提供了不少崭新的思路，尤其是在搭建与布置服务器和引入新服务的地方。之前所采用的虚拟机搭建方法如今看来过于浪费，或许我可以通过 DOCKER 来对其进行改进，达到三个服务器在 DOCKER 上实现联通。感谢老师这一个月来的授课，从这里学到的东西十分有用且新鲜，我甚至可以把这些在老师这里学到知识应用到我的将来的毕业论文中

因为我之前在学校学习过相关的基础知识，所以在神经网络方面和 PYTHON 的编程方面我并没有感到十分吃力，但是在接触到之前从没接触过的新型数据库时还是碰到了些许困难，尽管 NOSQL 与传统的关系型数据库在后端的搭建链接上并没有太多的区别，但是由于网上资料稀少且有效资料几乎为零，我还是碰到了好几个 ERROR，所以我私下问了张老师不少问题，感谢老师一一耐心解答我的问题，偶尔在课上我又好奇疑问的地方询问老师，老师也讲解的很详细，让我对概念中产生模糊理解的地方理解的更清楚。这次的远程课程我的收获非常大，谢谢老师一个月来的指导，帮助我发现了一个新的发展与研究方向，领我进入新世界的大门。