# 中国大学生计算机设计大赛


大数据实践赛作品报告

作品编号：2021040625

作品名称：whatVideoSaid——

基于卷积神经网络的智能 b 站视频评论语义分析小程序

版本编号：

填写日期：2021 年 5 月 30 日

**填写说明：**

1、本文档适用于大数据实践小类；
2、正文一律用小四号宋体，1.3 倍行距；一级标题为二号黑体，其他级别标题如有需要，可根据需要设置；
3、本文档应结构清晰，突出重点，适当配合图表，描述准确，不易冗长拖沓；
4、提交文档时，以 PDF 格式提交；
5、本文档内容是正式参赛内容的组成部分，务必真实填写。如不属实，将导致奖项等级降低甚至终止本作品参加比赛。

# 1 作品概述

## 1.1 作品创意/项目特色

随着当今互联网视频行业快速发展，随着视频网站的快速发展，在大数据时代下，不管是视频的制作者、还是观看者，都越来越急迫的显现出一种需求——一种快速的获取视频背后受众群体对视频到底产生何种情感的需求。

在作品的创意、应用和功能设计方面，本项目具有发现社会热点、快速分析社会热点背后的各受众情感的功能。

在作品的开发实现、技术问题解决的方面，我们综合使用了 Web 后端开发、小程序开发、算法开发等技术。网站使用 Django 开发框架开发后台，采用微信小程序开发者工具实现前端框架实现前端页面设计。

为了提升运行效率，我们采用 CentOS 8.0 64bit 系统运行 django 框架以及配套词云与卷积神经网络。

卷积神经网络的语义分析算法是平台的核心功能，通过我们团队的一步步优化调整，最终实现了综合考虑预测准确度、稳定性以及复杂度与网站可运行性的综合。网站也为用户提供了方便的当前热榜一览表页面，供用户发现当今热点；当然，如果用户有自己感兴趣的视屏，也可以直接搜索此视频直接进行分析。

在前端的设计上，采用微信小程序开发者工具，以及其可视化开发方法，使页面整体美观度和可读性增强，并且自我绘制了大量主题色图标，使用户交互性更好。于个人视屏发布者，此便携式 app 可以让你快速掌握你的受众对你视屏的态度与看法，以加速、优化视屏制作。

于企业视屏宣传部，此便携式 app 可以让你快速掌握与企业相关的舆论风评、快速了解客户需求，以与客户打成一片，提升客户忠诚度、或及时完成危急公关。

于党和国家的主流宣传媒体，此便携式 app 可以使其快速掌握意识形态领域舆情新情况，加深、加速主流思想通过视屏网站在人民群众中间的传播，及时管控错误言论、了解社会特定事件舆论导向等等，提振互联网的主旋律之声。

## 1.2 作品效果

图表 1 小程序第一页热榜综合数据


图表 2 小程序第二页核心功能（详细播放收藏记录、语义分析、弹幕词云）


# 2 问题描述

## 2.1 项目背景

如果我是一位个人视频发布者，我怎么才能快速了解观众对视频的情感喜好？怎么了解观众对视频内容的反馈？如果我是一个公司的视屏宣传部部长，我该怎么快速进行舆论公关？怎么快速了解客户对我们品牌的需求与意见？如果我是一个党和政府的主流宣传媒体，我怎么快速了解人民意见所向，了解网络空间意识形态？怎么在互联网唱响社会主义主旋律？

可见，随着当今互联网视频行业快速发展，随着视频网站的快速发展，在大数据时代下，不管是视频的制作者、还是观看者，都越来越急迫的显现出一种需求——一种快速的获取视频背后受众群体对视频到底产生何种情感的需求。

## 2.2 功能指标/需求

基于上述背景，我们希望开发一个为有此类需求的人员设计的、无需下载的、方便的语义分析一体化工具。

当然，为了深入服务客户的个性定制化服务，我们直接开放了后端的 api 供用户使用。

# 3 技术方案

【本章对系统实现的功能、开发技术和应用环境进行介绍。】

## 3.1 系统功能

### 3.1.1 功能概述

【概述系统实现的主要功能，包括系统性能。需给出系统功能的框架结构图，样例如图 1 所示。】

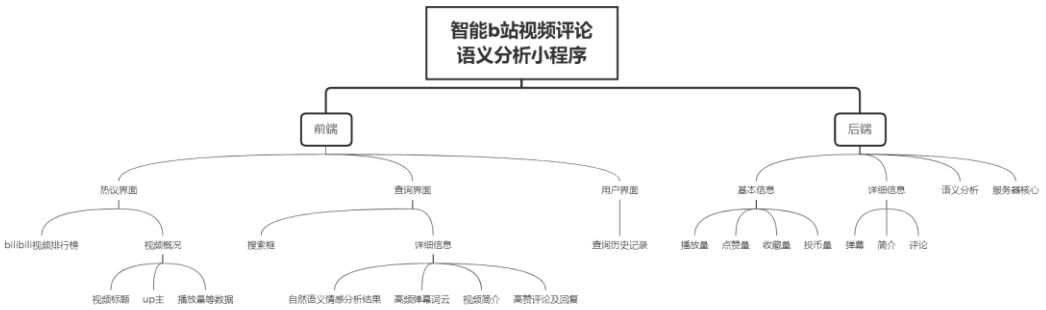



图 1 系统功能框架图

### 3.1.2 功能说明

【对系统功能分层逐一详细说明。注意此处给出的是从用户角度看到的功能。】

本项目以卷积神经网络语义分析为主，围绕其功能，前端由三大模块组成：b 站热榜信息展示、待搜索视频详细信息展示（包括评论、回复、点赞数据；以及筛选概括后的词云、以及最重要的——语义分析）、以及用户搜索的历史记录。以下分页详述：

a. 小程序**第一页**为爬虫抓取的 b 站热榜数据，
    a) 给予用户一个立体的视角以观测 b 站最近的热点（即视频说了什么），响应项目的主题。

b) 点击感兴趣的主题，界面将自动跳转到第二页，进行进一步详细分析与数据汇总输出。


这个例子就很好地显示出目前关注峰值为**新疆棉花**事件，而这也符合我们近期在生活

b. 小程序**第二页**为项目核心功能页面：

a) 获取当前视屏的详细信息

b) 当前视屏可在上方框框内直接搜索，或者直接承接首页热榜或者浏览记录

c) 返回对该视频评论的自然语义情感分析

d) 返回改视屏高频弹幕组成的词云

e) 返回视屏的简介

f) 返回高赞评论及其回复


本页面以——在**法国外长不满被制裁要召见中国大使却以没空为理由拒绝**——这件事件的舆情评论为例

从这里的卷积神经网络的预测中，我们可以看到，近乎机器预测在96%的大概率下，网友的评论都指向的是高兴。

下方的词云也突出显示了网友的评论，如“哈哈哈哈”。

两处合并起来合证，就很好的显现出，对于这一件事件，人民群众是喜闻乐见的，充满了对


本页面一另一个**新疆棉花**事件视频的舆情

从这里的高赞评论中，我们可以看到，在对新疆棉花事件的声讨与不满外，许多网友对“赚中国人钱，还恶心中国人” 的企业及其行为十分不满。

在原来的四分类情绪上，更详细的给了用户一个具体的实例

c. 小程序**第三页**利用内存技术，记录了用户过去的搜索记录同时点击历史记录将完成跳转，重回第二页分析页。


## 3.2 系统软硬件平台

### 3.2.1 系统开发平台（含开源/第三方工具）

a. 后端

本系统的后端开发采用 Django 网站开发框架，开发时本地使用的操作系统是 windows10 家庭版。后端的 python 版本是 3.9，安装的 python 库及其版本号为：

```
  Package               Version
---------------------- ---------
absl-py                0.12.0
asgiref                3.3.1
astunparse             1.6.3
cached-property        1.5.2
cachetools             4.2.1
certifi                2020.12.5
chardet                4.0.0
cycler                 0.10.0
Django                 3.1.7
flatbuffers            1.12
gast                   0.3.3
google-auth            1.27.1
google-auth-oauthlib   0.4.3
google-pasta           0.2.0
grpcio                 1.32.0
h5py                   2.10.0
idna                   2.10
importlib-metadata     3.7.3
jieba                  0.42.1
joblib                 1.0.1
Keras                  2.4.3
Keras-Preprocessing    1.1.2
kiwisolver             1.3.1
lxml                   4.6.2
Markdown               3.3.4
matplotlib             3.3.4
numpy                  1.19.5
oauthlib               3.1.0
opt-einsum             3.3.0
pandas                 1.2.3
Pillow                 8.1.2
pip                    21.0.1
```

```
protobuf                3.15.6
pyasn1                  0.4.8
pyasn1-modules          0.2.8
pyparsing               2.4.7
python-dateutil         2.8.1
pytz                    2021.1
PyYAML                  5.4.1
requests                2.25.1
requests-oauthlib       1.3.0
rsa                     4.7.2
scipy                   1.5.4
setuptools              54.1.2
six                     1.15.0
sqlparse                0.4.1
tensorboard             2.4.1
tensorboard-plugin-wit  1.8.0
tensorflow              2.4.1
tensorflow-estimator    2.4.0
termcolor               1.1.0
typing-extensions       3.7.4.3
urllib3                 1.26.3
uWSGI                   2.0.19.1
Werkzeug                1.0.1
wheel                   0.36.2
wordcloud               1.8.1
wrapt                   1.12.1
zipp                    3.4.1
```

数据库采用微信自带的内存技术，直接将一定时限内历史记录存在手机内存中。

a. 前端

系统前端使用微信小程序开发者工具开发，版本是 stable 1.05.2102010

### 3.2.2 系统运行平台

本项目使用的服务器为腾讯云服务器，使用的操作系统是 CentOS 8.0 64bit。

实例配置为:


系统前后端由 request 请求及其涉及的 json 数据相联系。

## 3.3 关键技术

本项目的算法：基于 keras 框架搭建的神经网络以实现的情感分类的功能。我们在通过对传统的 lstm 神经网络的改进下，获得了的预测准确率更高的 textcnn-lstm 神经网络。

具体的算法介绍会在后面，作为系统实现的一部分进行详细介绍。

# 4 系统实现

## 4.1 系统结构设计

### 4.1.1 技术架构

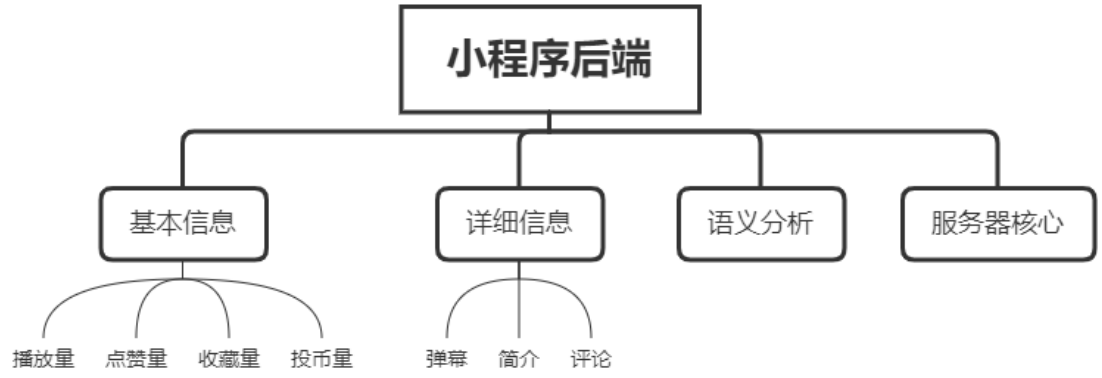
本项目采用 Django 搭建后端，该框架使用 python 语言，方便与 python 强大的人工智能库交互，以获得高效的开发效率。

本项目采用微信小程序，即移动平台作为前端载体。原因如下：

1. 微信小程序无需下载，即搜即得，增加用户的方便性，扩大其实用程度
2. 腾讯自带强大的生态配套，很好的为项目的开发提供了部分接口上的支持

### 4.1.2 功能模块设计

后端功能模块：

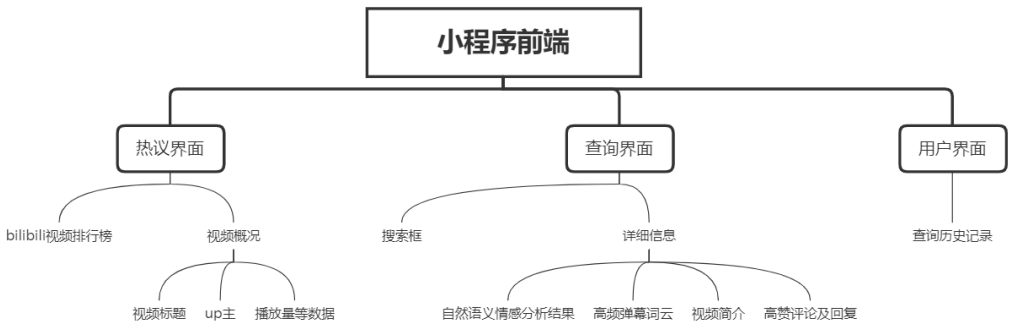



```
[root@VM-12-8-centos 121.4.163.229]# tree -L 2
.
├── 121.4.163.229_venv
│   ├── bin
│   ├── lib
│   └── pyvenv.cfg
```

```
├─── db.sqlite3
├─── FZLTCXHJW.ttf        // 词云所用字体
├─── logs
├─── manage.py
├─── myApi                    //api 功能模块（1）——基础信息
│    ├── admin.py
│    ├── apps.py
│    ├── __init__.py
│    ├── migrations
│    ├── models.py
│    ├── __pycache__
│    ├── tests.py
│    └── views.py
├─── NLPforComment       //api 功能模块（2）——评论详细信息
│    ├── admin.py
│    ├── apiForBilibiliComment.py       //评论获取
│    ├── apiForWordcloud.py             //词云生成
│    ├── apps.py
│    ├── FZLTCXHJW.ttf
│    ├── __init__.py
│    ├── migrations
│    ├── models.py
│    ├── __pycache__
│    ├── tests.py
│    └── views.py
├─── ourMind
│    ├── Lib
│    ├── pyvenv.cfg
│    └── Scripts
├─── pip
├─── preOfNpl          //api 功能模块（3）——语义分析模块
│    ├── admin.py
│    ├── apps.py
│    ├── __init__.py
```

```
│       ├── migrations
│       ├── models.py
│       ├── predict.py          //卷积神经网络算法
│       ├── __pycache__
│       ├── tests.py
│       └── views.py
├── python
├── requirements.txt
├── Tezt_cnn_2_2.h5       //卷积神经网络算法参数 1
├── tok.pkl_100           //卷积神经网络算法参数 2
├── uwsgi.ini
├── whatVedioSaid         //后端核心模块
│       ├── asgi.py
│       ├── __init__.py
│       ├── __pycache__
│       ├── settings.py
│       ├── urls.py
│       └── wsgi.py
└── WordCloud.png           //词云暂存
```

前端功能模块




E:\正式项目>tree /f

卷 DATA 的文件夹 PATH 列表

```
卷序列号为 5673-2F8A
E:.
│   .gitignore
│   app.js
│   app.json
│   app.wxss
│   project.config.json
│   sitemap.json
│
├──ec-canvas                    //图表库
│       ec-canvas.js
│       ec-canvas.json
│       ec-canvas.wxml
│       ec-canvas.wxss
│       echarts.js
│       wx-canvas.js
│
├──icon                 //图标模块——小程序下方小图标
│       0.png
│       1.png
│       11.png
│       2.png
│       22.png
│       3.png
│       33.png
│
├──img                  //图片模块——小程序中所有的演示图片
│       diluo.png
│       gaoxin.png
│       jidong.png
│       pinlun.png
│       soso.png
│       tem.png
│       TOP1.png
```

```
    │       TOP2.png
    │       TOP3.png
    │       yane.png
    │       主体标.png
    │
    └──pages                          //小程序核心页面
        ├──comment                    //页面模块 1——评论模块，返回语义分析等核心功能
        │       comment.js
        │       comment.json
        │       comment.wxml
        │       comment.wxss
        │
        ├──home                       //页面模块 2——历史记录模块，返回用户历史记录
        │       home.js
        │       home.json
        │       home.wxml
        │       home.wxss
        │
        └──main                       //页面模块 3——热榜模块,返回当前 b 站热榜
                main.js
                main.json
                main.wxml
                 main.wxss
```

### 4.1.3 关键功能/算法设计

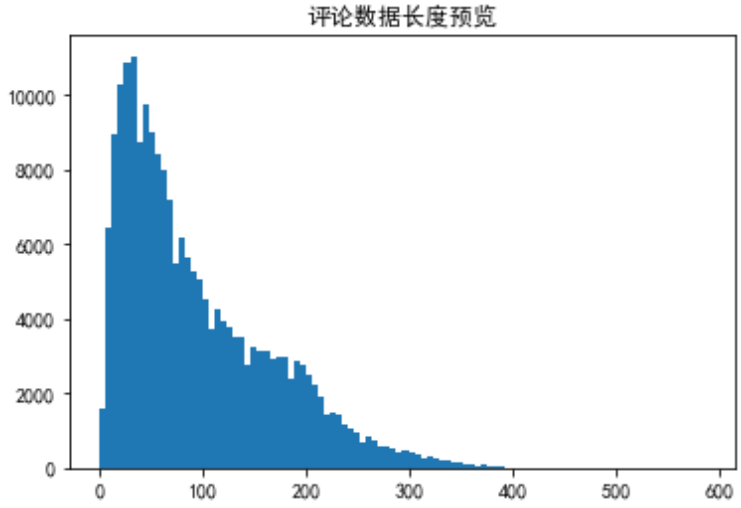
本项目的**关键功能**在于：抓取 b 站视频下方基本信息及其评论，通过卷积神经网络分析以得到这些评论的自然语义感情倾向，以达到预测视频受众感情倾向的目的。

而这中关键，在于训练卷积神经网络模型，对自然语义情感做出分析。下面着重介绍一下本项目训练卷积神经网络以预测的技术过程。

神经网络的算法是基于对原有的算法的改进，传统的 textcnn 算法在自然语言处理的方面已经得到了较为广泛的应用，Yoon Kim 在 Convolutional Neural Networks for Sentence Classification 中已经阐明，所以我在实验中关于卷积核的数量和池化层的大小等一些超参数的选择上受到了他的启发。本项目的神经网络的设计的基础是传统的 textcnn 神经网络，通过引入 word-embedding 层降低词向量的维度，再通过引入卷积神经网络实现词向量文本的特征的提取，最后再加上循环神经网络中的长短期记忆神经网络和全连接层实现文本的分类工作。对于多种神经网络，我分别进行了测试，绘制了训练的准确值和损失值的变化曲线，在实验中寻找最适合的神经网络，最后在阅读了 Faliang Huang 等人的 Attention-Emotion-Enhanced Convolutional LSTM for Sentiment Analysis 确定了在传统的 LSTM 神经网络的基础上，强化的 LSTM 神经网络，也获得了较好的预测的效果。

## a.数据预览与预处理阶段

数据准备的是从网络下载的微博评论数据集，数据集总共分为四类，分别对应的四种情感是喜悦，激动，厌恶与低落。每种情感选择了 50000 条评论。由于每条评论来自网络，有较多的符号语言，运用正则表达式对评论进行清洗。调用 python 的 jieba 对评论进行分词的处理，并且计算分词过后的每条评论的长度用于下面分词器的建立的工作的准备。




**（图一）**

```
count    206000.000000
mean         95.723655
std          72.731714
```

```
min          1.000000
25%         38.000000
50%         74.000000
75%        141.000000
max        585.000000
Name: num of fenci, dtype: float64
```

**（图二）**

图一展示了的是每一句话在进行分词过后的长度，图二的频数直方图可以看到平均的长度是 95，且大多数的评论都是在 100 词以内，因此在接下来的分词器的创建中所选择的最大的词语数量被定为 100。

## b.分词器，词典的创建

经过第一部分的分析，因此已经对于分词器的参数有了认识。运用 keras 的 tokennizer 对于已经分完词的数据进行处理，获得了训练数据集，稳定数据集和测试数据集的矩阵，以及一个对应的出现的词语的词频字典和词语的数量的词典。经过分词器处理过的数据的维度是训练集：200000*100，稳定集：8000*100 以及测试集：2000*100。到这里，完成了对于模型的数据的预处理的全部的工作。

其中总共选择了 5000 个词语，每一个词语的出现的数量以及出现数量的排名都已经被统计。

## c.模型的选择与训练

完成了对于数据的预处理之后，开始对于神经网络的模型的搭建以及训练。在经

过资料的阅读以及考量，考虑搭建 MLP 神经网络，TextCNN 神经网络以及 TextCNN+LSTM 神经网络并进行网络的训练。

这三种神经网络各有特点，全连接神经网络的特点是比较简单，但是缺点就是对于数据的预测的效果是比较差的，CNN 网络可以对于数据的特征进行比较好的提取，TextCNN 模型已经成为了提取文本特征的有力的工具，但是多层的 CNN 网络会导致梯度的消失，我通过模型的训练对于传统的全连接的 CNN 模型进行了改良。LSTM 网络对于传统的 RNN 神经网络有一个比较好的提升，对于多变量的问题有比较好的效果。LSTM 也已经广泛的运用到了情感分类之中，我也对于传统的 LSTM 进行了增强的处理。下面我对于这三种神经网络都分别进行了训练和测试，每一次的训练次数都是 20 次，每 1000 个训练数据分为一组，所以一次的训练数据需要训练 200 次。记录了随实验的次数变化的测试集和稳定集的准确率以及损失值的变化。

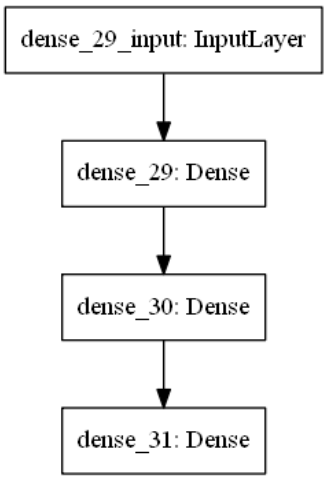
**i.全连接神经网络**

全连接层的网络的基本结构是经过一层词嵌入层更改输入的词向量的维度之后连接了三层的全连接层，最后一层是输出维度为 4 的全连接层，模型的激活函数是 adam。

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_29 (Dense)             (None, 32)                3232
_________________________________________________________________
dense_30 (Dense)             (None, 16)                528
_________________________________________________________________
dense_31 (Dense)             (None, 4)                 68
```

```
=================================================================
====
Total params: 3,828
Trainable params: 3,828
Non-trainable params: 0
```

**（图六）**

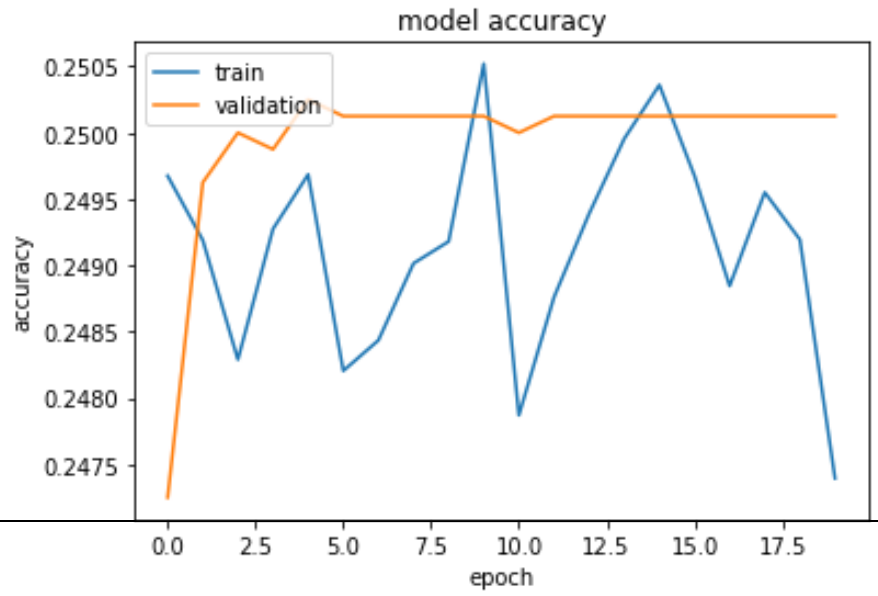



**（图七）**

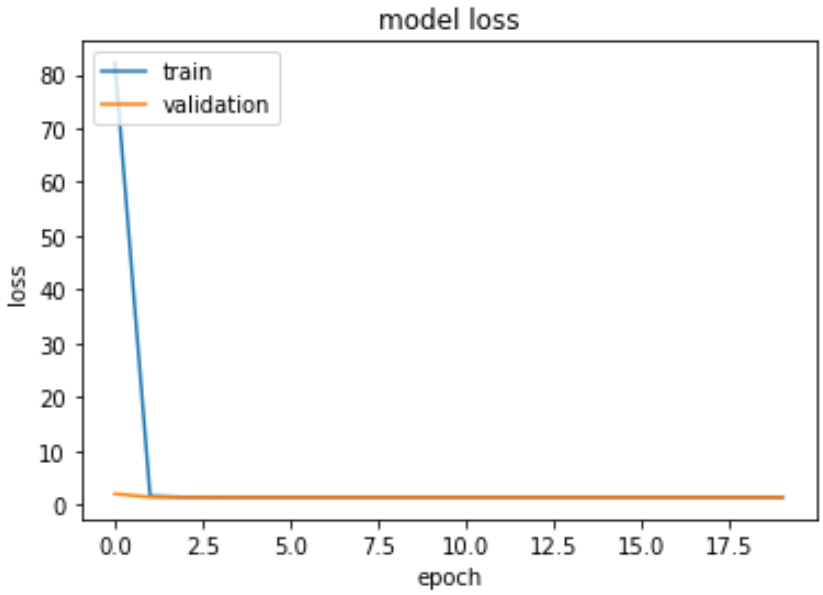
```
Epoch 1/20
200/200 [==============================] - 0s 2ms/step - loss:
82.1700 - accuracy: 0.2497 - val_loss: 2.0274 - val_accuracy: 0.2473
Epoch 2/20
```

```
200/200 [==============================] - 0s 2ms/step - loss:
1.6381 - accuracy: 0.2492 - val_loss: 1.4417 - val_accuracy: 0.2496
Epoch 3/20
200/200 [==============================] - 0s 2ms/step - loss:
1.4307 - accuracy: 0.2483 - val_loss: 1.3990 - val_accuracy: 0.2500
Epoch 4/20
200/200 [==============================] - 0s 2ms/step - loss:
1.4038 - accuracy: 0.2493 - val_loss: 1.3898 - val_accuracy: 0.2499
Epoch 5/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3950 - accuracy: 0.2497 - val_loss: 1.3872 - val_accuracy: 0.2503
Epoch 6/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3908 - accuracy: 0.2482 - val_loss: 1.3866 - val_accuracy: 0.2501
Epoch 7/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3890 - accuracy: 0.2484 - val_loss: 1.3864 - val_accuracy: 0.2501
Epoch 8/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3883 - accuracy: 0.2490 - val_loss: 1.3864 - val_accuracy: 0.2501
Epoch 9/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3878 - accuracy: 0.2492 - val_loss: 1.3863 - val_accuracy: 0.2501
Epoch 10/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3874 - accuracy: 0.2505 - val_loss: 1.3863 - val_accuracy: 0.2501
Epoch 11/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3869 - accuracy: 0.2479 - val_loss: 1.3868 - val_accuracy: 0.2500
Epoch 12/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3867 - accuracy: 0.2488 - val_loss: 1.3862 - val_accuracy: 0.2501
Epoch 13/20
```

```
200/200 [==============================] - 0s 2ms/step - loss:
1.3866 - accuracy: 0.2494 - val_loss: 1.3862 - val_accuracy: 0.2501
Epoch 14/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3865 - accuracy: 0.2500 - val_loss: 1.3862 - val_accuracy: 0.2501
Epoch 15/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3865 - accuracy: 0.2504 - val_loss: 1.3862 - val_accuracy: 0.2501
Epoch 16/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3864 - accuracy: 0.2497 - val_loss: 1.3862 - val_accuracy: 0.2501
Epoch 17/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3864 - accuracy: 0.2488 - val_loss: 1.3861 - val_accuracy: 0.2501
Epoch 18/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3864 - accuracy: 0.2495 - val_loss: 1.3861 - val_accuracy: 0.2501
Epoch 19/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3864 - accuracy: 0.2492 - val_loss: 1.3862 - val_accuracy: 0.2501
Epoch 20/20
200/200 [==============================] - 0s 2ms/step - loss:
1.3864 - accuracy: 0.2474 - val_loss: 1.3861 - val_accuracy: 0.2501
accuracy 25.006500
```



（图八）




（图九）

图六是模型的概览，图七是模型的结构示意图，图八和图九是模型的准确度和损失值随实验批次的增加而变化的折线图。此模型对于模型的预测效果尚欠佳，因此在此基础上，我们开始了下一步的训练。

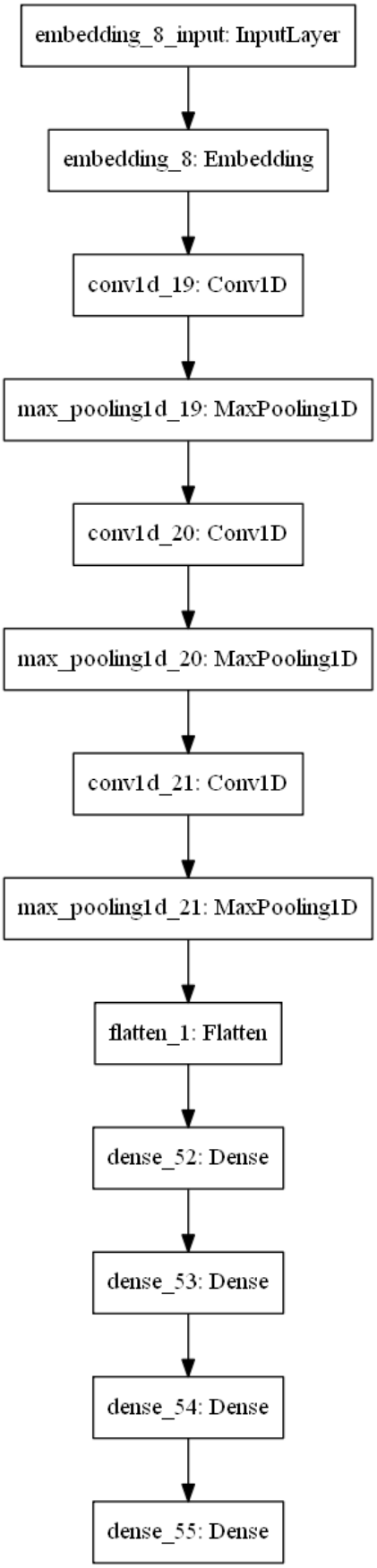
**ii.TextCNN 模型网络**

我们构建的 TextCNN 模型首先是词嵌入层，然后连接的是三层的一维的卷积神经网络与三层的全连接层神经网络，最后输出维度为 4 的全连接层，模型的激活函数是 adam。

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_8 (Embedding)      (None, 100, 256)          1280256
_________________________________________________________________
conv1d_19 (Conv1D)           (None, 100, 80)           61520
_________________________________________________________________
max_pooling1d_19 (MaxPooling (None, 50, 80)            0
_________________________________________________________________
conv1d_20 (Conv1D)           (None, 50, 80)            25680
_________________________________________________________________
max_pooling1d_20 (MaxPooling (None, 25, 80)            0
_________________________________________________________________
conv1d_21 (Conv1D)           (None, 25, 80)            32080
_________________________________________________________________
max_pooling1d_21 (MaxPooling (None, 12, 80)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 960)               0
_________________________________________________________________
dense_52 (Dense)             (None, 100)               96100
_________________________________________________________________
```

```
dense_53 (Dense)            (None, 63)                6363
_________________________________________________________________
_____
dense_54 (Dense)            (None, 32)                2048
_________________________________________________________________
_____
dense_55 (Dense)            (None, 4)                 132
=================================================================
====
Total params: 1,504,179
Trainable params: 1,504,179
Non-trainable params: 0
```

**(图十)**

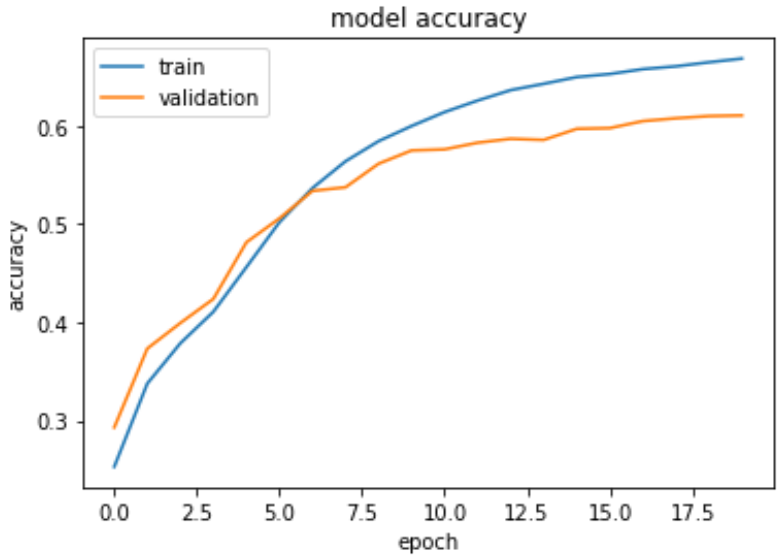

embedding_8_input: InputLayer
embedding_8: Embedding
conv1d_19: Conv1D
max_pooling1d_19: MaxPooling1D
conv1d_20: Conv1D
max_pooling1d_20: MaxPooling1D
conv1d_21: Conv1D
max_pooling1d_21: MaxPooling1D
flatten_1: Flatten
dense_52: Dense
dense_53: Dense
dense_54: Dense
dense_55: Dense

**（图十一）**

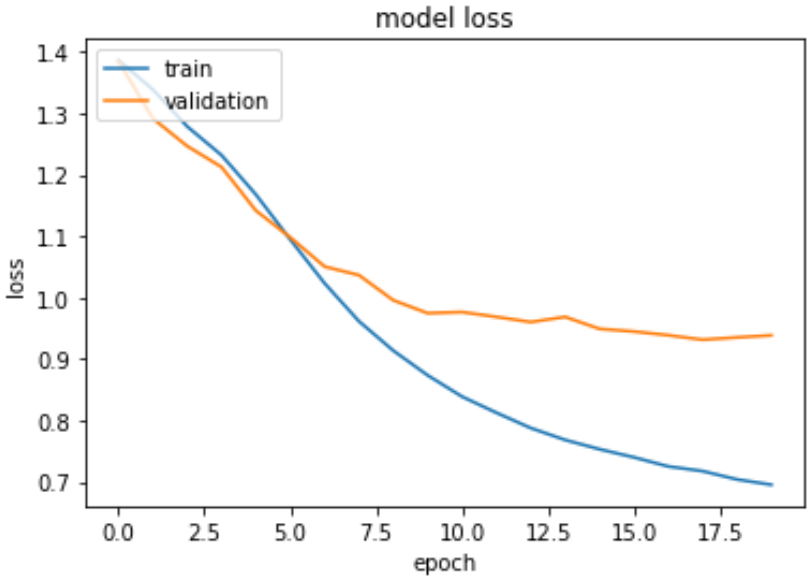
```
Epoch 1/20
200/200 [==============================] - 84s 420ms/step - loss:
1.3863 - accuracy: 0.2530 - val_loss: 1.3849 - val_accuracy: 0.2930
Epoch 2/20
200/200 [==============================] - 84s 422ms/step - loss:
1.3388 - accuracy: 0.3376 - val_loss: 1.2916 - val_accuracy: 0.3733
Epoch 3/20
200/200 [==============================] - 83s 416ms/step - loss:
1.2785 - accuracy: 0.3786 - val_loss: 1.2466 - val_accuracy: 0.3993
Epoch 4/20
200/200 [==============================] - 83s 416ms/step - loss:
1.2318 - accuracy: 0.4107 - val_loss: 1.2125 - val_accuracy: 0.4238
Epoch 5/20
200/200 [==============================] - 81s 406ms/step - loss:
1.1678 - accuracy: 0.4561 - val_loss: 1.1416 - val_accuracy: 0.4814
Epoch 6/20
200/200 [==============================] - 79s 397ms/step - loss:
1.0946 - accuracy: 0.5018 - val_loss: 1.0981 - val_accuracy: 0.5058
Epoch 7/20
200/200 [==============================] - 81s 405ms/step - loss:
1.0233 - accuracy: 0.5366 - val_loss: 1.0505 - val_accuracy: 0.5337
Epoch 8/20
200/200 [==============================] - 70s 352ms/step - loss:
0.9615 - accuracy: 0.5638 - val_loss: 1.0366 - val_accuracy: 0.5374
Epoch 9/20
200/200 [==============================] - 60s 300ms/step - loss:
0.9137 - accuracy: 0.5843 - val_loss: 0.9957 - val_accuracy: 0.5614
Epoch 10/20
200/200 [==============================] - 59s 296ms/step - loss:
0.8736 - accuracy: 0.5998 - val_loss: 0.9747 - val_accuracy: 0.5749
Epoch 11/20
200/200 [==============================] - 58s 292ms/step - loss:
0.8389 - accuracy: 0.6140 - val_loss: 0.9765 - val_accuracy: 0.5760
```

```
Epoch 12/20
200/200 [==============================] - 59s 293ms/step - loss:
0.8127 - accuracy: 0.6257 - val_loss: 0.9686 - val_accuracy: 0.5828
Epoch 13/20
200/200 [==============================] - 59s 296ms/step - loss:
0.7878 - accuracy: 0.6363 - val_loss: 0.9603 - val_accuracy: 0.5867
Epoch 14/20
200/200 [==============================] - 59s 296ms/step - loss:
0.7685 - accuracy: 0.6427 - val_loss: 0.9686 - val_accuracy: 0.5856
Epoch 15/20
200/200 [==============================] - 60s 300ms/step - loss:
0.7534 - accuracy: 0.6496 - val_loss: 0.9492 - val_accuracy: 0.5969
Epoch 16/20
200/200 [==============================] - 58s 292ms/step - loss:
0.7402 - accuracy: 0.6527 - val_loss: 0.9449 - val_accuracy: 0.5976
Epoch 17/20
200/200 [==============================] - 58s 292ms/step - loss:
0.7253 - accuracy: 0.6576 - val_loss: 0.9388 - val_accuracy: 0.6047
Epoch 18/20
200/200 [==============================] - 58s 292ms/step - loss:
0.7177 - accuracy: 0.6605 - val_loss: 0.9315 - val_accuracy: 0.6076
Epoch 19/20
200/200 [==============================] - 59s 297ms/step - loss:
0.7042 - accuracy: 0.6646 - val_loss: 0.9353 - val_accuracy: 0.6099
Epoch 20/20
200/200 [==============================] - 59s 296ms/step - loss:
0.6955 - accuracy: 0.6684 - val_loss: 0.9387 - val_accuracy: 0.6105
```

**accuracy 67.447501**


model accuracy
train
validation
accuracy
epoch
0.3
0.4
0.5
0.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

（图十二）


model loss
train
validation
loss
epoch
0.7
0.8
0.9
1.0
1.1
1.2
1.3
1.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

（图十三）

图十是模型的概览，图十一是模型的结构示意图，图十二和图十三是模型的准确度和损失值岁实验批次的增加而变化的折线图。实验结果表明使用卷积神经网络有助于文本数据特征的提取，训练集准确度大为上升。但为了追求更高的预测效率，我们开始考虑对此卷积神经网络进行加强处理。

**iii.加强的 TextCNN 模型网络**

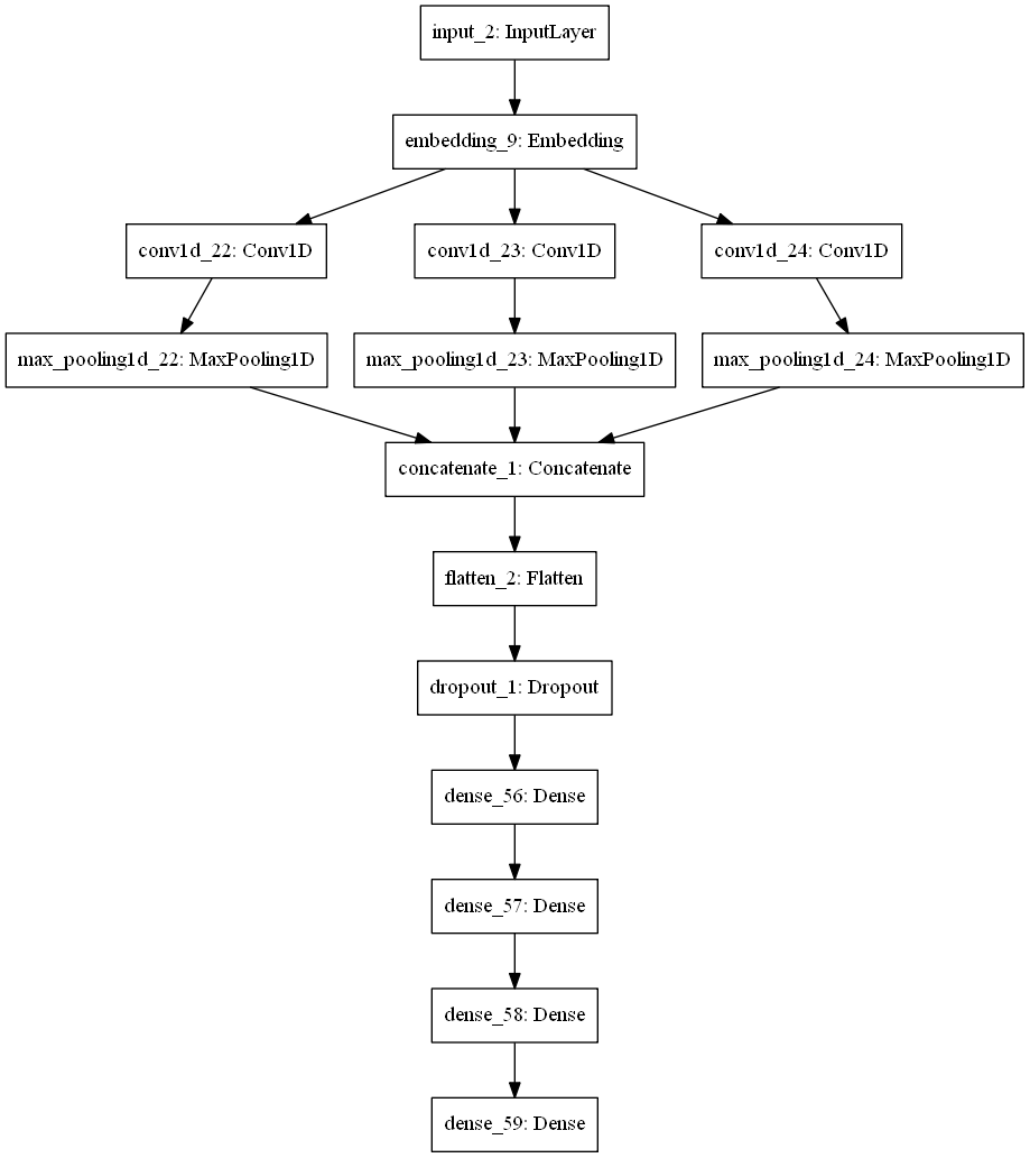
考虑运用 keras 的 concatenate 方式对于第二部分的卷积神经网络进行加强的处理，形成 Enhanced TextCNN(ETextCNN)，模型的结构是将第二部分的六层卷积神经网络放入 concatenate 层，从而减少了模型的复杂度，也减少了模型的冗余而带来的梯度消失的情况，也可以增加 CNN 网络中的输出通道对于数据的特征有更好的提取。

```
Model: "model_1"
__________________________________________________________________________
_____________________________________
Layer (type)                 Output Shape              Param #
Connected to
==========================================================================
====================================
input_2 (InputLayer)         [(None, 100)]             0
__________________________________________________________________________
_____________________________________
embedding_9 (Embedding)      (None, 100, 300)          1500300
input_2[0][0]
__________________________________________________________________________
_____________________________________
conv1d_22 (Conv1D)           (None, 100, 256)          230656
embedding_9[0][0]
__________________________________________________________________________
_____________________________________
```

```
conv1d_23 (Conv1D)              (None, 100, 256)     307456
embedding_9[0][0]
__________________________________________________________________________
_____________________________________
conv1d_24 (Conv1D)              (None, 100, 256)     384256
embedding_9[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_22 (MaxPooling1D) (None, 10, 256)      0
conv1d_22[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_23 (MaxPooling1D) (None, 10, 256)      0
conv1d_23[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_24 (MaxPooling1D) (None, 10, 256)      0
conv1d_24[0][0]
__________________________________________________________________________
_____________________________________
concatenate_1 (Concatenate)     (None, 10, 768)      0
max_pooling1d_22[0][0]
max_pooling1d_23[0][0]
max_pooling1d_24[0][0]
__________________________________________________________________________
_____________________________________
flatten_2 (Flatten)             (None, 7680)         0
concatenate_1[0][0]
__________________________________________________________________________
_____________________________________
dropout_1 (Dropout)             (None, 7680)         0
flatten_2[0][0]
__________________________________________________________________________
_____________________________________
```

```
dense_56 (Dense)                (None, 128)          983168
dropout_1[0][0]
__________________________________________________________________
_________________________________________
dense_57 (Dense)                (None, 64)           8256
dense_56[0][0]
__________________________________________________________________
_________________________________________
dense_58 (Dense)                (None, 32)           2080
dense_57[0][0]
__________________________________________________________________
_________________________________________
dense_59 (Dense)                (None, 4)            132
dense_58[0][0]
==================================================================
====================================
Total params: 3,416,304
Trainable params: 1,916,004
Non-trainable params: 1,500,300
```

**（图十四）**



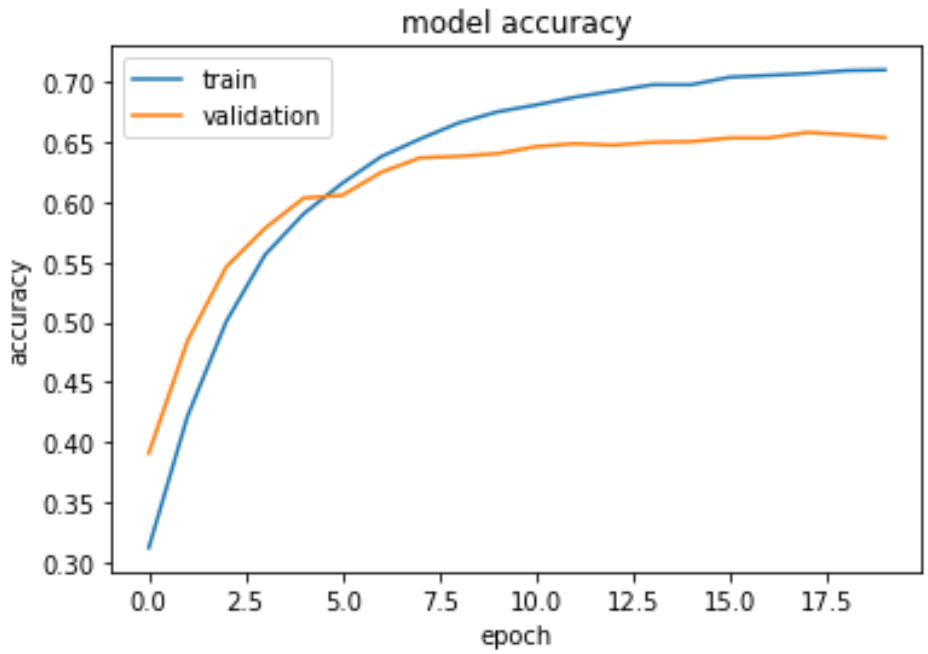

**（图十五）**

```
Epoch 1/20
200/200 [==============================] - 214s 1s/step - loss:
1.3482 - accuracy: 0.3125 - val_loss: 1.2658 - val_accuracy: 0.3911
Epoch 2/20
200/200 [==============================] - 220s 1s/step - loss:
1.2028 - accuracy: 0.4218 - val_loss: 1.0927 - val_accuracy: 0.4845
Epoch 3/20
200/200 [==============================] - 222s 1s/step - loss:
1.0259 - accuracy: 0.5005 - val_loss: 0.9545 - val_accuracy: 0.5459
```

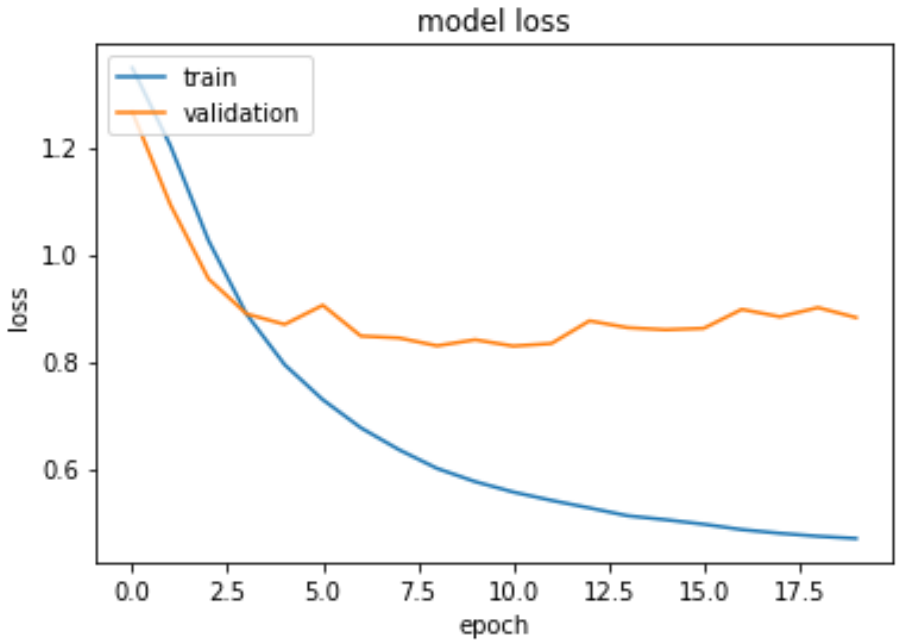
```
Epoch 4/20
200/200 [==============================] - 269s 1s/step - loss:
0.8882 - accuracy: 0.5559 - val_loss: 0.8890 - val_accuracy: 0.5780
Epoch 5/20
200/200 [==============================] - 338s 2s/step - loss:
0.7942 - accuracy: 0.5900 - val_loss: 0.8694 - val_accuracy: 0.6031
Epoch 6/20
200/200 [==============================] - 341s 2s/step - loss:
0.7290 - accuracy: 0.6155 - val_loss: 0.9052 - val_accuracy: 0.6054
Epoch 7/20
200/200 [==============================] - 342s 2s/step - loss:
0.6765 - accuracy: 0.6375 - val_loss: 0.8477 - val_accuracy: 0.6245
Epoch 8/20
200/200 [==============================] - 343s 2s/step - loss:
0.6357 - accuracy: 0.6521 - val_loss: 0.8441 - val_accuracy: 0.6365
Epoch 9/20
200/200 [==============================] - 342s 2s/step - loss:
0.6007 - accuracy: 0.6655 - val_loss: 0.8295 - val_accuracy: 0.6378
Epoch 10/20
200/200 [==============================] - 340s 2s/step - loss:
0.5760 - accuracy: 0.6747 - val_loss: 0.8407 - val_accuracy: 0.6400
Epoch 11/20
200/200 [==============================] - 341s 2s/step - loss:
0.5566 - accuracy: 0.6804 - val_loss: 0.8290 - val_accuracy: 0.6459
Epoch 12/20
200/200 [==============================] - 340s 2s/step - loss:
0.5411 - accuracy: 0.6869 - val_loss: 0.8337 - val_accuracy: 0.6482
Epoch 13/20
200/200 [==============================] - 344s 2s/step - loss:
0.5270 - accuracy: 0.6920 - val_loss: 0.8761 - val_accuracy: 0.6470
Epoch 14/20
200/200 [==============================] - 344s 2s/step - loss:
0.5124 - accuracy: 0.6972 - val_loss: 0.8633 - val_accuracy: 0.6494
Epoch 15/20
```

```
200/200 [==============================] - 353s 2s/step - loss:
0.5049 - accuracy: 0.6972 - val_loss: 0.8595 - val_accuracy: 0.6500
Epoch 16/20
200/200 [==============================] - 346s 2s/step - loss:
0.4965 - accuracy: 0.7035 - val_loss: 0.8621 - val_accuracy: 0.6530
Epoch 17/20
200/200 [==============================] - 338s 2s/step - loss:
0.4865 - accuracy: 0.7050 - val_loss: 0.8975 - val_accuracy: 0.6530
Epoch 18/20
200/200 [==============================] - 342s 2s/step - loss:
0.4796 - accuracy: 0.7065 - val_loss: 0.8837 - val_accuracy: 0.6576
Epoch 19/20
200/200 [==============================] - 339s 2s/step - loss:
0.4741 - accuracy: 0.7089 - val_loss: 0.9008 - val_accuracy: 0.6557
Epoch 20/20
200/200 [==============================] - 338s 2s/step - loss:
0.4699 - accuracy: 0.7095 - val_loss: 0.8822 - val_accuracy: 0.6532
```

**accuracy 71.01**



（图十六）




（图十七）

图十四是模型的概览，图十五是模型的结构示意图，图十六和图十七是模型的准确度和损失值随实验批次的增加而变化的折线图。实验结果表明加强的卷积神经网络也有助于文本数据的特征的提取，预测准确度相较以前，提高了整整 5%。

### iv LSTM-TextCNN 模型网络

为了进一步提高准确度，在加强的卷积神经网络的基础上我们增加了 LSTM 模型，在情感分析的领域 LSTM 模型已经广为使用，因此在我们的第三部分的模型的基础上，又在加强卷积神经网络与全连接层神经网络之间又增加了一层 LSTM 网络。

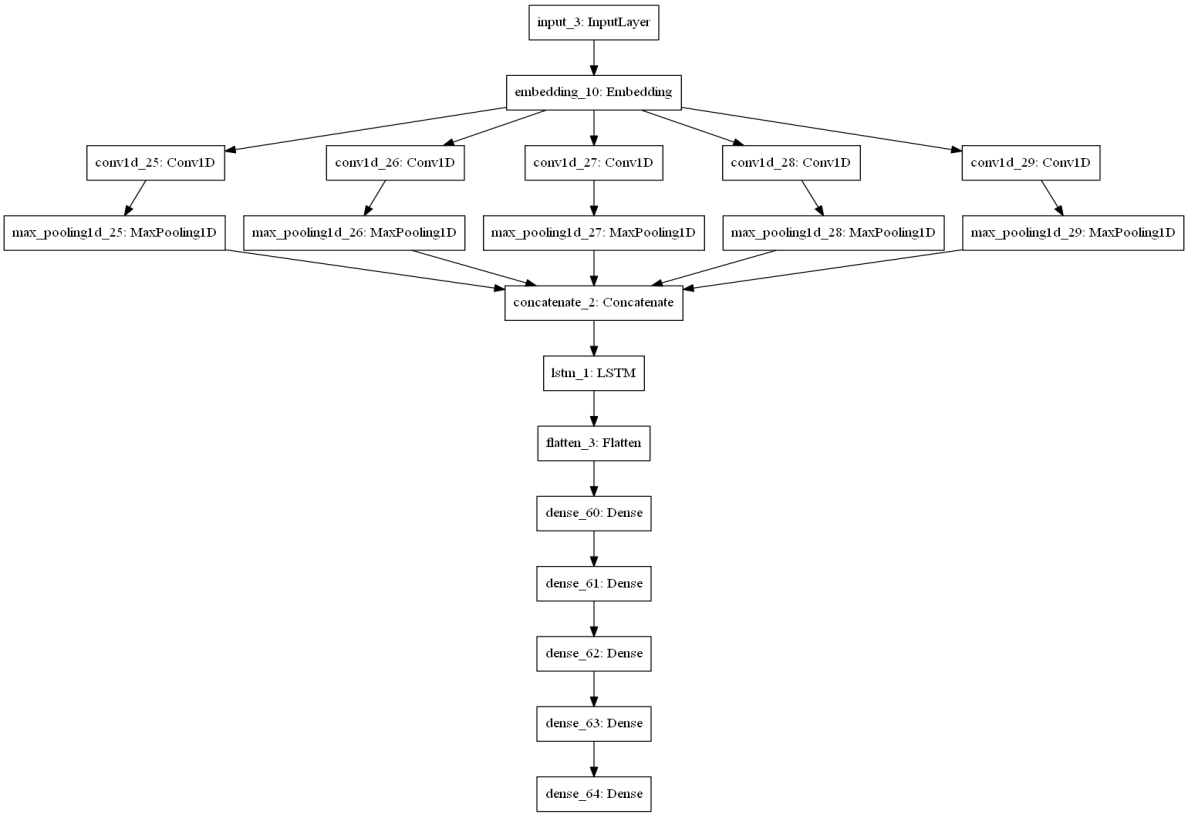
```
Model: "model_2"
_________________________________________________________________
_________________________________________________
```

```
Layer (type)                 Output Shape              Param #
Connected to
==================================================================
==================================
input_3 (InputLayer)         [(None, 100)]             0
__________________________________________________________________
_________________________________
embedding_10 (Embedding)     (None, 100, 256)          1280256
input_3[0][0]
__________________________________________________________________
_________________________________
conv1d_25 (Conv1D)           (None, 100, 256)          196864
embedding_10[0][0]
__________________________________________________________________
_________________________________
conv1d_26 (Conv1D)           (None, 100, 256)          262400
embedding_10[0][0]
__________________________________________________________________
_________________________________
conv1d_27 (Conv1D)           (None, 100, 256)          327936
embedding_10[0][0]
__________________________________________________________________
_________________________________
conv1d_28 (Conv1D)           (None, 100, 256)          393472
embedding_10[0][0]
__________________________________________________________________
_________________________________
conv1d_29 (Conv1D)           (None, 100, 256)          459008
embedding_10[0][0]
__________________________________________________________________
_________________________________
max_pooling1d_25 (MaxPooling1D) (None, 10, 256)        0
conv1d_25[0][0]
```

```
__________________________________________________________________________
_____________________________________
max_pooling1d_26 (MaxPooling1D) (None, 10, 256)      0
conv1d_26[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_27 (MaxPooling1D) (None, 10, 256)      0
conv1d_27[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_28 (MaxPooling1D) (None, 10, 256)      0
conv1d_28[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_29 (MaxPooling1D) (None, 10, 256)      0
conv1d_29[0][0]
__________________________________________________________________________
_____________________________________
concatenate_2 (Concatenate)     (None, 10, 1280)     0
max_pooling1d_25[0][0]
max_pooling1d_26[0][0]
max_pooling1d_27[0][0]
max_pooling1d_28[0][0]
max_pooling1d_29[0][0]
__________________________________________________________________________
_____________________________________
lstm_1 (LSTM)                   (None, 256)          1573888
concatenate_2[0][0]
__________________________________________________________________________
_____________________________________
flatten_3 (Flatten)             (None, 256)          0
lstm_1[0][0]
__________________________________________________________________________
_____________________________________
```

```
dense_60 (Dense)                (None, 256)          65792
flatten_3[0][0]
__________________________________________________________________________
_____________________________________
dense_61 (Dense)                (None, 128)          32896
dense_60[0][0]
__________________________________________________________________________
_____________________________________
dense_62 (Dense)                (None, 64)           8256
dense_61[0][0]
__________________________________________________________________________
_____________________________________
dense_63 (Dense)                (None, 32)           2080
dense_62[0][0]
__________________________________________________________________________
_____________________________________
dense_64 (Dense)                (None, 4)            132
dense_63[0][0]
==========================================================================
==================================
Total params: 4,602,980
Trainable params: 3,322,724
Non-trainable params: 1,280,256
```

**（图十八）**



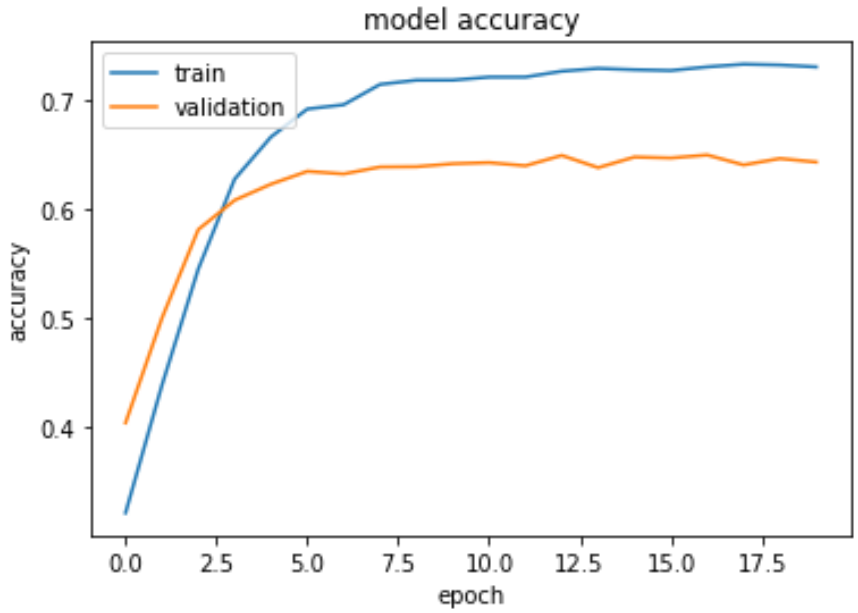

**（图十九）**

```
Epoch 1/20
200/200 [==============================] - 696s 3s/step - loss:
1.3433 - accuracy: 0.3219 - val_loss: 1.2505 - val_accuracy: 0.4046
Epoch 2/20
200/200 [==============================] - 676s 3s/step - loss:
1.1748 - accuracy: 0.4390 - val_loss: 1.0810 - val_accuracy: 0.5006
Epoch 3/20
200/200 [==============================] - 674s 3s/step - loss:
0.9278 - accuracy: 0.5458 - val_loss: 0.8899 - val_accuracy: 0.5820
Epoch 4/20
200/200 [==============================] - 677s 3s/step - loss:
0.7078 - accuracy: 0.6284 - val_loss: 0.9237 - val_accuracy: 0.6090
Epoch 5/20
```

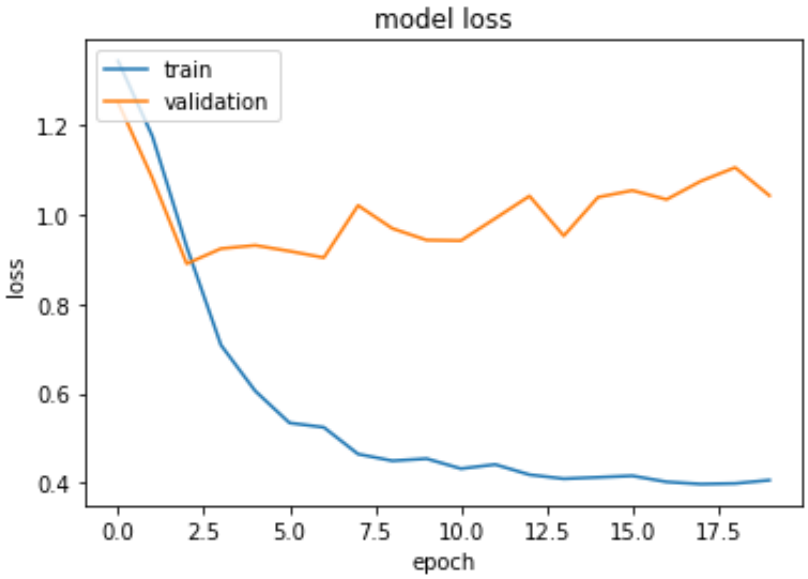
```
200/200 [==============================] - 675s 3s/step - loss:
0.6052 - accuracy: 0.6671 - val_loss: 0.9310 - val_accuracy: 0.6237
Epoch 6/20
200/200 [==============================] - 675s 3s/step - loss:
0.5338 - accuracy: 0.6927 - val_loss: 0.9180 - val_accuracy: 0.6355
Epoch 7/20
200/200 [==============================] - 680s 3s/step - loss:
0.5240 - accuracy: 0.6967 - val_loss: 0.9035 - val_accuracy: 0.6331
Epoch 8/20
200/200 [==============================] - 675s 3s/step - loss:
0.4641 - accuracy: 0.7153 - val_loss: 1.0202 - val_accuracy: 0.6395
Epoch 9/20
200/200 [==============================] - 676s 3s/step - loss:
0.4491 - accuracy: 0.7192 - val_loss: 0.9688 - val_accuracy: 0.6398
Epoch 10/20
200/200 [==============================] - 674s 3s/step - loss:
0.4538 - accuracy: 0.7192 - val_loss: 0.9428 - val_accuracy: 0.6425
Epoch 11/20
200/200 [==============================] - 676s 3s/step - loss:
0.4315 - accuracy: 0.7220 - val_loss: 0.9416 - val_accuracy: 0.6435
Epoch 12/20
200/200 [==============================] - 674s 3s/step - loss:
0.4408 - accuracy: 0.7221 - val_loss: 0.9912 - val_accuracy: 0.6406
Epoch 13/20
200/200 [==============================] - 672s 3s/step - loss:
0.4180 - accuracy: 0.7274 - val_loss: 1.0413 - val_accuracy: 0.6501
Epoch 14/20
200/200 [==============================] - 673s 3s/step - loss:
0.4090 - accuracy: 0.7300 - val_loss: 0.9522 - val_accuracy: 0.6389
Epoch 15/20
200/200 [==============================] - 674s 3s/step - loss:
0.4122 - accuracy: 0.7288 - val_loss: 1.0387 - val_accuracy: 0.6486
Epoch 16/20
```

```
200/200 [==============================] - 669s 3s/step - loss:
0.4155 - accuracy: 0.7281 - val_loss: 1.0538 - val_accuracy: 0.6478
Epoch 17/20
200/200 [==============================] - 671s 3s/step - loss:
0.4019 - accuracy: 0.7314 - val_loss: 1.0336 - val_accuracy: 0.6505
Epoch 18/20
200/200 [==============================] - 673s 3s/step - loss:
0.3970 - accuracy: 0.7339 - val_loss: 1.0747 - val_accuracy: 0.6414
Epoch 19/20
200/200 [==============================] - 672s 3s/step - loss:
0.3984 - accuracy: 0.7331 - val_loss: 1.1051 - val_accuracy: 0.6472
Epoch 20/20
200/200 [==============================] - 671s 3s/step - loss:
0.4058 - accuracy: 0.7314 - val_loss: 1.0423 - val_accuracy: 0.6440
```

**accuracy 73.445499**



（图二十）




（图二十一）

图十八是模型的概览，图十九是模型的结构示意图，图二十和图二十一是模型的准确度和损失值随实验批次的增加而变化的折线图。

实验结果表明增加了一层的 LSTM 的模型更进一步提高了模型预测的准确度。

**v 对 LSTM-TextCNN 模型网络的改进**

第五部分是在通过前四部分的对于整个模型的主要框架进行了确定的基础上，对于现有的模型进行改进。首先考虑的是增加 concatenate 层增加卷积神经网络的数量，其次是增加 LSTM 的复杂度，可以考虑在 LSTM 的基础上增加一层全连接层和一层 Dropout 层，实现数据特征的更好的提取。

基于这样的考量，我们构造了这样的模型，首先是一层词嵌入层，接下来的一层

是由 6 个卷积神经网络组成的 concatenate 层，接下来的是一层 LSTM 层，我增加了 LSTM 层的隐藏神经元的个数增加到了 1000 个。再下面是全连接层的神经网络和上面的模型是一样的。

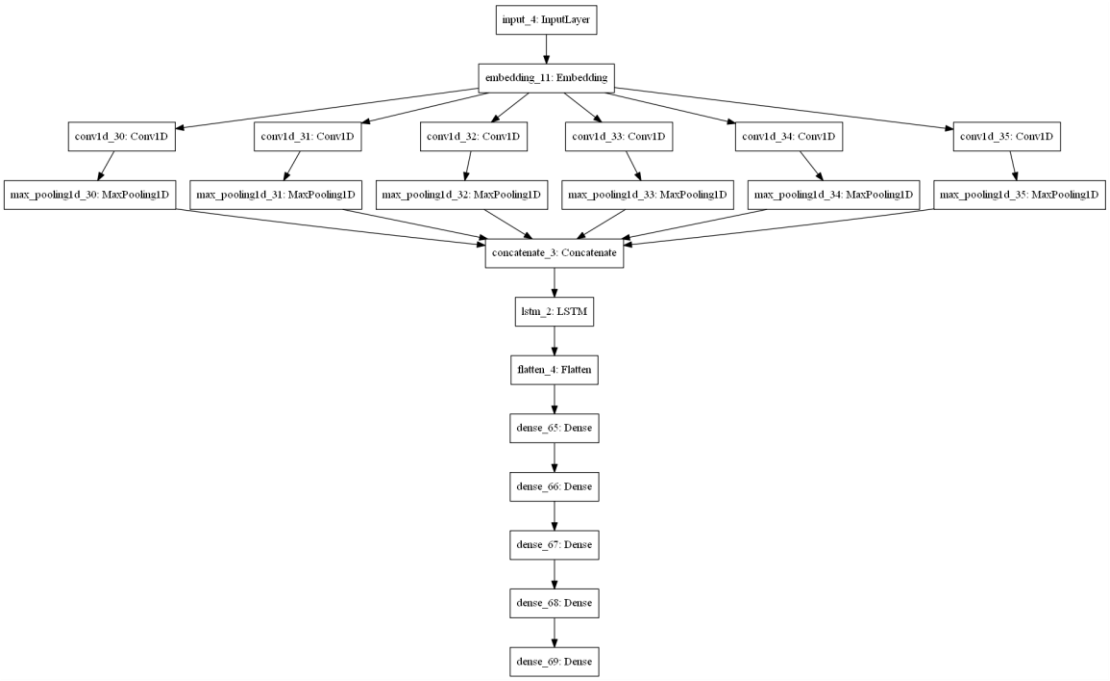
```
Model: "model_3"
__________________________________________________________________________
_____________________________________
Layer (type)                 Output Shape              Param #
Connected to
==========================================================================
====================================
input_4 (InputLayer)         [(None, 100)]             0
__________________________________________________________________________
_____________________________________
embedding_11 (Embedding)     (None, 100, 256)          1280256
input_4[0][0]
__________________________________________________________________________
_____________________________________
conv1d_30 (Conv1D)           (None, 50, 256)           196864
embedding_11[0][0]
__________________________________________________________________________
_____________________________________
conv1d_31 (Conv1D)           (None, 50, 256)           262400
embedding_11[0][0]
__________________________________________________________________________
_____________________________________
conv1d_32 (Conv1D)           (None, 50, 256)           327936
embedding_11[0][0]
__________________________________________________________________________
_____________________________________
conv1d_33 (Conv1D)           (None, 50, 256)           393472
embedding_11[0][0]
```

```
__________________________________________________________________________
_____________________________________
conv1d_34 (Conv1D)              (None, 50, 256)      459008
embedding_11[0][0]
__________________________________________________________________________
_____________________________________
conv1d_35 (Conv1D)              (None, 50, 256)      524544
embedding_11[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_30 (MaxPooling1D) (None, 5, 256)       0
conv1d_30[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_31 (MaxPooling1D) (None, 5, 256)       0
conv1d_31[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_32 (MaxPooling1D) (None, 5, 256)       0
conv1d_32[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_33 (MaxPooling1D) (None, 5, 256)       0
conv1d_33[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_34 (MaxPooling1D) (None, 5, 256)       0
conv1d_34[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_35 (MaxPooling1D) (None, 5, 256)       0
conv1d_35[0][0]
__________________________________________________________________________
_____________________________________
```

```
concatenate_3 (Concatenate)     (None, 5, 1536)      0
max_pooling1d_30[0][0]
max_pooling1d_31[0][0]
max_pooling1d_32[0][0]
max_pooling1d_33[0][0]
max_pooling1d_34[0][0]
max_pooling1d_35[0][0]
__________________________________________________________________________
_____________________________________
lstm_2 (LSTM)                   (None, 1000)         10148000
concatenate_3[0][0]
__________________________________________________________________________
_____________________________________
flatten_4 (Flatten)             (None, 1000)         0
lstm_2[0][0]
__________________________________________________________________________
_____________________________________
dense_65 (Dense)                (None, 256)          256256
flatten_4[0][0]
__________________________________________________________________________
_____________________________________
dense_66 (Dense)                (None, 128)          32896
dense_65[0][0]
__________________________________________________________________________
_____________________________________
dense_67 (Dense)                (None, 64)           8256
dense_66[0][0]
__________________________________________________________________________
_____________________________________
dense_68 (Dense)                (None, 32)           2080
dense_67[0][0]
__________________________________________________________________________
_____________________________________
```

```
dense_69 (Dense)                (None, 4)           132
dense_68[0][0]
==================================================================
==================================
Total params: 13,892,100
Trainable params: 12,611,844
Non-trainable params: 1,280,256
```

（图二十二）

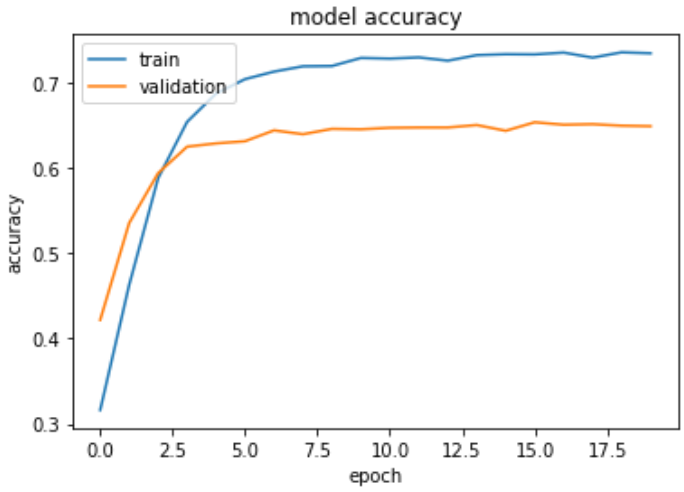



（图二十三）

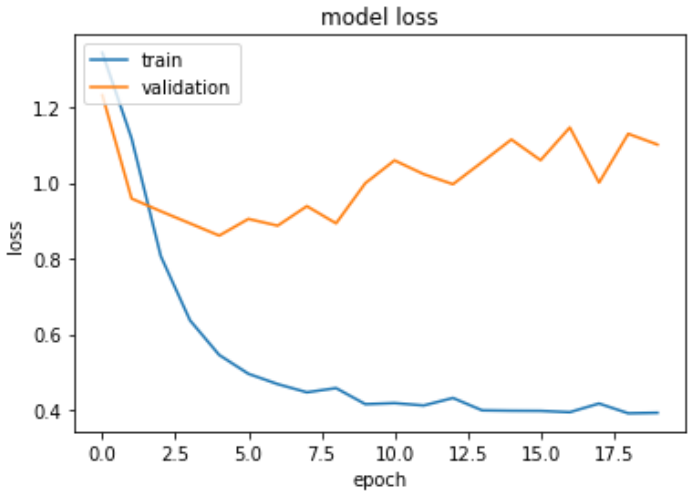
```
Epoch 1/20
200/200 [==============================] - 629s 3s/step - loss:
1.3432 - accuracy: 0.3161 - val_loss: 1.2313 - val_accuracy: 0.4218
Epoch 2/20
200/200 [==============================] - 654s 3s/step - loss:
1.1175 - accuracy: 0.4627 - val_loss: 0.9585 - val_accuracy: 0.5353
Epoch 3/20
200/200 [==============================] - 629s 3s/step - loss:
0.8065 - accuracy: 0.5879 - val_loss: 0.9250 - val_accuracy: 0.5938
Epoch 4/20
```

```
200/200 [==============================] - 628s 3s/step - loss:
0.6370 - accuracy: 0.6539 - val_loss: 0.8929 - val_accuracy: 0.6248
Epoch 5/20
200/200 [==============================] - 633s 3s/step - loss:
0.5456 - accuracy: 0.6871 - val_loss: 0.8607 - val_accuracy: 0.6286
Epoch 6/20
200/200 [==============================] - 629s 3s/step - loss:
0.4959 - accuracy: 0.7039 - val_loss: 0.9047 - val_accuracy: 0.6311
Epoch 7/20
200/200 [==============================] - 631s 3s/step - loss:
0.4688 - accuracy: 0.7127 - val_loss: 0.8869 - val_accuracy: 0.6439
Epoch 8/20
200/200 [==============================] - 629s 3s/step - loss:
0.4471 - accuracy: 0.7190 - val_loss: 0.9384 - val_accuracy: 0.6394
Epoch 9/20
200/200 [==============================] - 629s 3s/step - loss:
0.4582 - accuracy: 0.7193 - val_loss: 0.8930 - val_accuracy: 0.6456
Epoch 10/20
200/200 [==============================] - 628s 3s/step - loss:
0.4158 - accuracy: 0.7287 - val_loss: 0.9990 - val_accuracy: 0.6451
Epoch 11/20
200/200 [==============================] - 627s 3s/step - loss:
0.4186 - accuracy: 0.7280 - val_loss: 1.0592 - val_accuracy: 0.6469
Epoch 12/20
200/200 [==============================] - 627s 3s/step - loss:
0.4124 - accuracy: 0.7294 - val_loss: 1.0227 - val_accuracy: 0.6472
Epoch 13/20
200/200 [==============================] - 626s 3s/step - loss:
0.4321 - accuracy: 0.7255 - val_loss: 0.9963 - val_accuracy: 0.6472
Epoch 14/20
200/200 [==============================] - 627s 3s/step - loss:
0.3994 - accuracy: 0.7321 - val_loss: 1.0549 - val_accuracy: 0.6501
Epoch 15/20
```

```
200/200 [==============================] - 626s 3s/step - loss:
0.3984 - accuracy: 0.7331 - val_loss: 1.1143 - val_accuracy: 0.6434
Epoch 16/20
200/200 [==============================] - 628s 3s/step - loss:
0.3980 - accuracy: 0.7329 - val_loss: 1.0597 - val_accuracy: 0.6534
Epoch 17/20
200/200 [==============================] - 628s 3s/step - loss:
0.3946 - accuracy: 0.7350 - val_loss: 1.1460 - val_accuracy: 0.6505
Epoch 18/20
200/200 [==============================] - 632s 3s/step - loss:
0.4175 - accuracy: 0.7291 - val_loss: 1.0004 - val_accuracy: 0.6511
Epoch 19/20
200/200 [==============================] - 627s 3s/step - loss:
0.3917 - accuracy: 0.7353 - val_loss: 1.1294 - val_accuracy: 0.6494
Epoch 20/20
200/200 [==============================] - 643s 3s/step - loss:
0.3931 - accuracy: 0.7342 - val_loss: 1.1009 - val_accuracy: 0.6488
accuracy 73.684502
```



（图二十四）

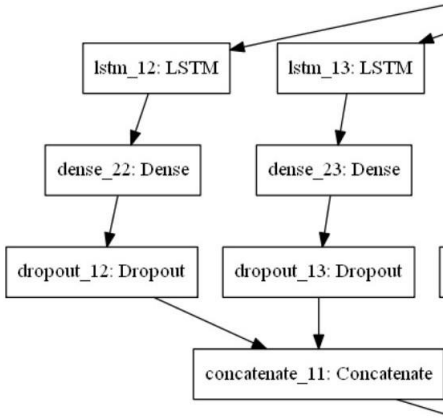



（图二十五）

图二十二是模型的概览，图二十三是模型的结构示意图，图二十四和图二十五是模型的准确度和损失值随实验批次的增加而变化的折线图。

我们可以发现，虽然在此操作下准确度有所提高，但可以看到稳定集的误差在训练的过程中波动起伏较大，因此我们继续改进了模型。

打算采用这样的改进方式，改进传统的 LSTM 神经网络，首先将传统的 LSTM 神经网络后面再接上一层全连接层网络和一层 Dropout 神经网络防止模型在训练过程中出现过拟合的现象，然后将这样的两个神经网络放入 concatenate 层形成加强的 LSTM 神经网络，再用这三个加强的神经网络放入 concatenate 层实现 LSTM 神经网络的加强。由于神经网络的复杂性，我减少了一次训练的 batch_size 到 100，减少了训练的次数到 10 次防止模型出现过拟合的现象。




**（图二十六）**

```
_________________________________________________________________
_____________________________________
Layer (type)                 Output Shape              Param #
Connected to
==================================================================
==================================
input_4 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
_____________________________________
embedding_3 (Embedding)      (None, 100, 256)          1280256
input_4[0][0]
_________________________________________________________________
_____________________________________
conv1d_18 (Conv1D)           (None, 50, 256)           196864
embedding_3[0][0]
_________________________________________________________________
_____________________________________
```

```
conv1d_19 (Conv1D)              (None, 50, 256)      262400
embedding_3[0][0]
__________________________________________________________________________
_____________________________________
conv1d_20 (Conv1D)              (None, 50, 256)      327936
embedding_3[0][0]
__________________________________________________________________________
_____________________________________
conv1d_21 (Conv1D)              (None, 50, 256)      393472
embedding_3[0][0]
__________________________________________________________________________
_____________________________________
conv1d_22 (Conv1D)              (None, 50, 256)      459008
embedding_3[0][0]
__________________________________________________________________________
_____________________________________
conv1d_23 (Conv1D)              (None, 50, 256)      524544
embedding_3[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_18 (MaxPooling1D) (None, 5, 256)       0
conv1d_18[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_19 (MaxPooling1D) (None, 5, 256)       0
conv1d_19[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_20 (MaxPooling1D) (None, 5, 256)       0
conv1d_20[0][0]
__________________________________________________________________________
_____________________________________
max_pooling1d_21 (MaxPooling1D) (None, 5, 256)       0
conv1d_21[0][0]
```

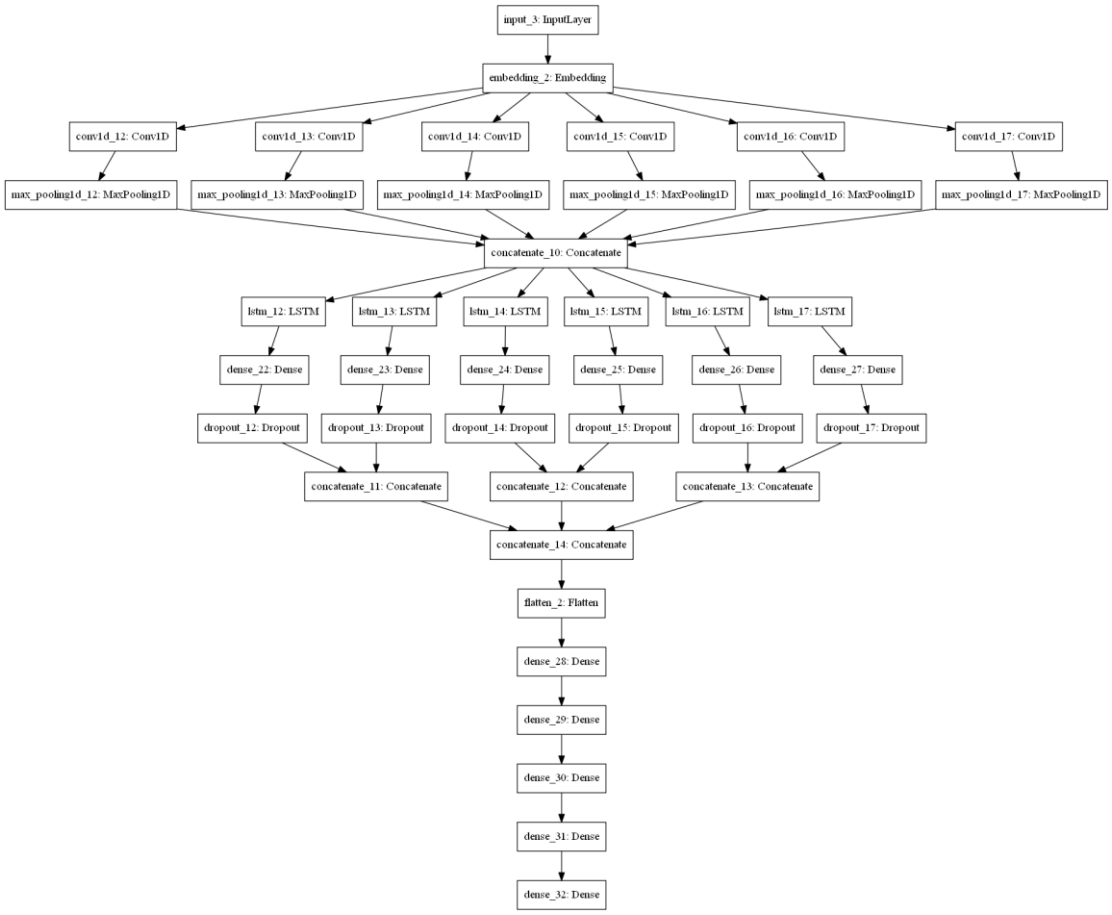
```
__________________________________________________________________________
__________________________________________
max_pooling1d_22 (MaxPooling1D) (None, 5, 256)      0
conv1d_22[0][0]
__________________________________________________________________________
__________________________________________
max_pooling1d_23 (MaxPooling1D) (None, 5, 256)      0
conv1d_23[0][0]
__________________________________________________________________________
__________________________________________
concatenate_15 (Concatenate)    (None, 5, 1536)     0
max_pooling1d_18[0][0]
max_pooling1d_19[0][0]
max_pooling1d_20[0][0]
max_pooling1d_21[0][0]
max_pooling1d_22[0][0]
max_pooling1d_23[0][0]
__________________________________________________________________________
__________________________________________
lstm_18 (LSTM)                  (None, 1000)        10148000
concatenate_15[0][0]
__________________________________________________________________________
__________________________________________
lstm_19 (LSTM)                  (None, 1000)        10148000
concatenate_15[0][0]
__________________________________________________________________________
__________________________________________
lstm_20 (LSTM)                  (None, 1000)        10148000
concatenate_15[0][0]
__________________________________________________________________________
__________________________________________
lstm_21 (LSTM)                  (None, 1000)        10148000
concatenate_15[0][0]
```

```
_________________________________________________________________
_________________________________________
lstm_22 (LSTM)                  (None, 1000)         10148000
concatenate_15[0][0]
_________________________________________________________________
_________________________________________
lstm_23 (LSTM)                  (None, 1000)         10148000
concatenate_15[0][0]
_________________________________________________________________
_________________________________________
dense_33 (Dense)                (None, 16)           16016
lstm_18[0][0]
_________________________________________________________________
_________________________________________
dense_34 (Dense)                (None, 16)           16016
lstm_19[0][0]
_________________________________________________________________
_________________________________________
dense_35 (Dense)                (None, 16)           16016
lstm_20[0][0]
_________________________________________________________________
_________________________________________
dense_36 (Dense)                (None, 16)           16016
lstm_21[0][0]
_________________________________________________________________
_________________________________________
dense_37 (Dense)                (None, 16)           16016
lstm_22[0][0]
_________________________________________________________________
_________________________________________
dense_38 (Dense)                (None, 16)           16016
lstm_23[0][0]
_________________________________________________________________
_________________________________________
```

```
dropout_18 (Dropout)            (None, 16)           0
dense_33[0][0]
__________________________________________________________________________
_____________________________________
dropout_19 (Dropout)            (None, 16)           0
dense_34[0][0]
__________________________________________________________________________
_____________________________________
dropout_20 (Dropout)            (None, 16)           0
dense_35[0][0]
__________________________________________________________________________
_____________________________________
dropout_21 (Dropout)            (None, 16)           0
dense_36[0][0]
__________________________________________________________________________
_____________________________________
dropout_22 (Dropout)            (None, 16)           0
dense_37[0][0]
__________________________________________________________________________
_____________________________________
dropout_23 (Dropout)            (None, 16)           0
dense_38[0][0]
__________________________________________________________________________
_____________________________________
concatenate_16 (Concatenate)    (None, 32)           0
dropout_18[0][0]
dropout_19[0][0]
__________________________________________________________________________
_____________________________________
concatenate_17 (Concatenate)    (None, 32)           0
dropout_20[0][0]
dropout_21[0][0]
__________________________________________________________________________
_____________________________________
```

```
concatenate_18 (Concatenate)    (None, 32)          0
dropout_22[0][0]
dropout_23[0][0]
__________________________________________________________________________
_____________________________________
concatenate_19 (Concatenate)    (None, 96)          0
concatenate_16[0][0]
concatenate_17[0][0]
concatenate_18[0][0]
__________________________________________________________________________
_____________________________________
flatten_3 (Flatten)             (None, 96)          0
concatenate_19[0][0]
__________________________________________________________________________
_____________________________________
dense_39 (Dense)                (None, 256)         24832
flatten_3[0][0]
__________________________________________________________________________
_____________________________________
dense_40 (Dense)                (None, 128)         32896
dense_39[0][0]
__________________________________________________________________________
_____________________________________
dense_41 (Dense)                (None, 64)          8256
dense_40[0][0]
__________________________________________________________________________
_____________________________________
dense_42 (Dense)                (None, 32)          2080
dense_41[0][0]
__________________________________________________________________________
_____________________________________
dense_43 (Dense)                (None, 4)           132
dense_42[0][0]
```

```
==================================================================
===================================
Total params: 64,496,772
Trainable params: 63,216,516
Non-trainable params: 1,280,256
```

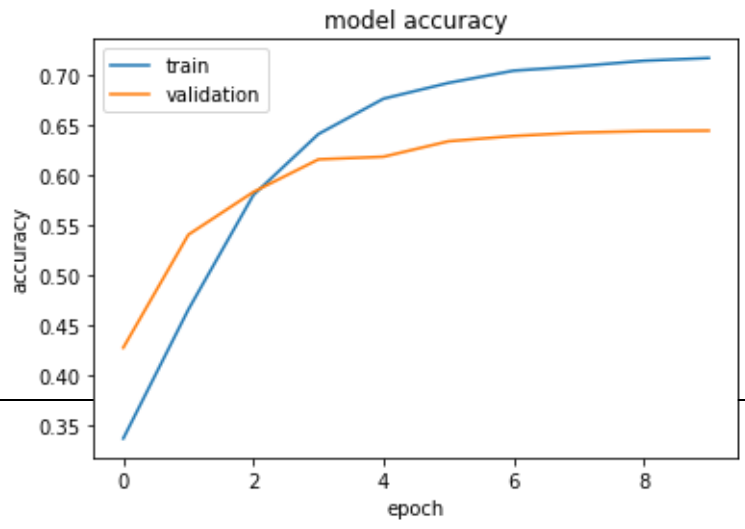



**（图二十七）**

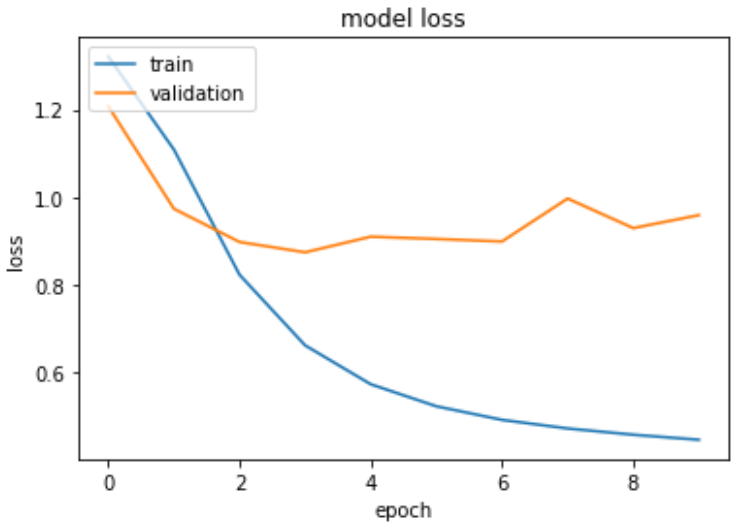
**（图二十八）**

```
Epoch 1/10
2000/2000 [==============================] - 4814s 2s/step - loss:
1.3205 - accuracy: 0.3362 - val_loss: 1.2070 - val_accuracy: 0.4270
Epoch 2/10
2000/2000 [==============================] - 4990s 2s/step - loss:
1.1085 - accuracy: 0.4651 - val_loss: 0.9736 - val_accuracy: 0.5400
Epoch 3/10
```

```
2000/2000 [==============================] - 4737s 2s/step - loss:
0.8232 - accuracy: 0.5797 - val_loss: 0.8979 - val_accuracy: 0.5828
Epoch 4/10
2000/2000 [==============================] - 5231s 3s/step - loss:
0.6624 - accuracy: 0.6407 - val_loss: 0.8744 - val_accuracy: 0.6154
Epoch 5/10
2000/2000 [==============================] - 4366s 2s/step - loss:
0.5738 - accuracy: 0.6761 - val_loss: 0.9100 - val_accuracy: 0.6180
Epoch 6/10
2000/2000 [==============================] - 4361s 2s/step - loss:
0.5236 - accuracy: 0.6919 - val_loss: 0.9048 - val_accuracy: 0.6335
Epoch 7/10
2000/2000 [==============================] - 4381s 2s/step - loss:
0.4926 - accuracy: 0.7041 - val_loss: 0.8991 - val_accuracy: 0.6388
Epoch 8/10
2000/2000 [==============================] - 4638s 2s/step - loss:
0.4731 - accuracy: 0.7085 - val_loss: 0.9970 - val_accuracy: 0.6421
Epoch 9/10
2000/2000 [==============================] - 4606s 2s/step - loss:
0.4588 - accuracy: 0.7140 - val_loss: 0.9295 - val_accuracy: 0.6436
Epoch 10/10
2000/2000 [==============================] - 4625s 2s/step - loss:
0.4473 - accuracy: 0.7167 - val_loss: 0.9591 - val_accuracy: 0.6441
accuracy 72.961003
```



**（图二十九）**




**（图三十）**

图二十六是加强的 LSTM 的模型的结构，图二十七是模型的概览，图二十八是模型的结构示意图，图二十九和图三十是模型的准确度和损失值随实验批次的增加而变化的折线图，

我们发现，在保证高准确度的同时。我们可以看到模型的稳定集的损失值是有稳定下降的趋势；且均比之前的模型要低。

**在各项性能都俱佳的情况下，我们选择了此模型为我们的预测模型。**

## 4.2 数据结构设计

### 4.2.1 存储数据

1. 数据库

本项目主旨为一个泛语义分析类小程序，是一种效率工具，所以需要储存的数据比较少。

但是为了用户的方便，我们还是使用了手机内存技术，以键值存储数据库为用户储存了一定时间内的浏览记录，以完善用户的体验。

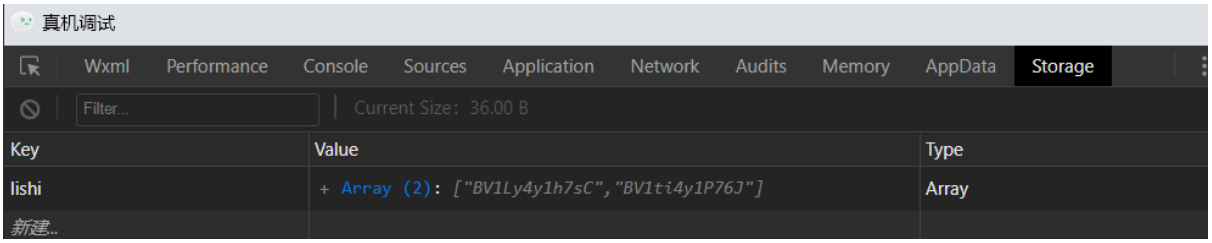



表 1　小程序内存

| 数据表名:storage | 中文描述：用户历史记录 |
|---|---|
| Key | Value(Type) |
| Lishi | Array(n) |

即{key:[“bv1l4y4hzsc”, “bv1l4y4p76g”… … ]}格式

2. 文件存储

为了调用方便此类文件都存储于与虚拟环境文件夹同级的文件夹内。具体分为两类：

1. Wordcloud+{{bv 号码}}.png 存储当前搜索视频生成的词云,并返回给前端，附加的 bv 号码可保证在并发访问时内容不会出错。
2. 为了系统的可维护性，Tezt_cnn_2_2.h5 与 tok.pkl_100 存储训练的卷积神经网络的独立训练数据，一旦新的网络训练成功，即可马上更换完成版本更新；FZLTCXHJW.ttf 存储词云中的字体文件，当需要更换词云字体时，也可马上更

换完成更新。

### 4.2.2 接口（模块接口、系统间接口）

因为源网站对微信本身的爬虫请求持拒绝态度，且微信请求头不可修改；同时更重要的，需要对数据加以精简与处理（语义分析），以提高小程序运行效率与服务效果，我们的项目采用两层请求模式。

项目前端向后端发送 request 请求，后端根据请求的不同，向数据源网站发出相对应的新 request 请求指令。提取所需原 json 数据后，后端再处理加工，返回精炼后的 json 给前端。

即：项目后端服务器提供 api，供前端微信小程序调用，前后端采用 json 数据格式传输数据。

### 4.2.3 关键数据结构

在后端中，将 json 数据转化为 python 的列表与字典，进行遍历或键值对查找，将处理过的数据，按逻辑分隔包装后，重新打包成 json 返回前端；同理，前端先将 json 解开，转换为 JavaScript 中对应的对象数组，再查找键值对或者遍历以取得所需值，最后交给 wxml(微信版 html)渲染输出。

在历史记录数据上，使用微信小程序 js 的 append 与 slice 构建了一个队列来完成对历史记录页面数据的处理。

## 4.3 系统界面设计

### 4.3.1 界面设计风格

总体而言，在前端的设计上，采用微信小程序开发者工具可视化开发方法，核心原则为简约明快清新的视觉效果和交互逻辑，使页面整体美观度和可读性增强，也便于用户快速上手使用。

配色整体采用微信的主题色，使得小程序与微信本身的 UI 设计更加融洽，并使用大量主题色图标，使整体图像化、用户交互性更好。

每个页面下方均有延续的导航栏，便于用户在三个主要页面间快速切换。

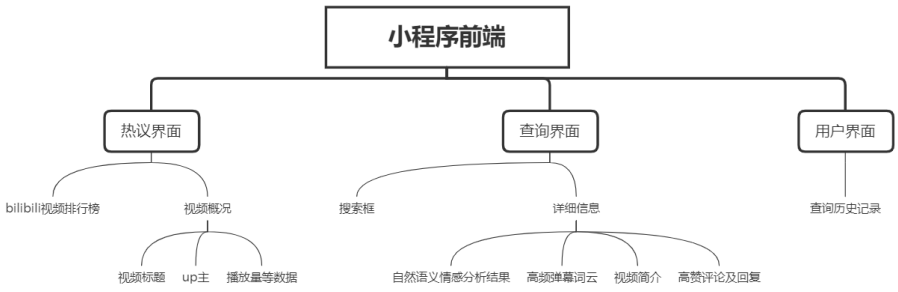

### 4.3.2 主要功能页面

**【介绍主要功能页面，并简要介绍这些页面的设计特色、操作方法。】**

首页上方为水平滑动的轮播图，展示了 b 站热榜前三名的视频，下方则以列表形式展示了热榜的更多视频内容，并显示了“温度”信息（播放点赞量等数据加和结果），更加直观显示了视频热度。


任意点击一个视频即可进入查询页面获取当前视频的详细信息，这也是小程序的核心页面。上方的搜索框可以方便用户直接搜索其他需要查询的视频，或者承接首页热榜或浏览记录。在视频封面图和标题下方，是该视频的播放量、点赞量、收藏量、投币量等信息。再往下即是对该视频评论的自然语义情感分析结果，以四个情感图标和量化数据展示。下方是视频高频弹幕组成的词云，生动形象地展现了视频的热点讨论。最下方是视频的简介，帮助用户快速了解视频的具体内容或者了解更多评论情况。


### 4.3.3 Web 网站页面结构设计



# 5 系统测评

## 5.1 实验简介/构架

在上述系统界面、功能等介绍中，我们已经展示了小程序前端的样貌，及其舆论测评与应用。在第一章就提到，我们的程序作为一个工具，不仅提供模块化的小程序服务，更能提供个性化 api 服务。那么，在这里的实验中，我们进一步，以后台 api 角度完整一次个性化舆情——时序分析。

即面对有存量需求的客户——即想了解一件事情舆论始末的客户——由于其需求更为定制化，且是要求一段时间的舆论变化流量，而不是上述提到的即时存量，作为工具定位的我们的程序为其提供了方便的后台 api，供其自由调用我们的舆论语义分析功能接口。

由此，通过调用我们的 api,就可以满足客户对一件其关心的热点事件舆情变化的持续追踪。这是一次运用后端 api 的个性化实验，主要用于探究某一件要研究事情始末的舆论变化趋势及其影响因素。

## 5.2 实验步骤及案例选取

这里，我们将以舆论反转多次，造成巨大社会震荡的成都 49 中学生跳楼事件的舆论为例，进行一次演示实验。

实验步骤如下：

A）选择数据源

此次实验选择了微博、b 站等多源大数据，在恰当的时间点，选择热度最高的热搜、或者事情，进行舆论评论的搜集。

B）爬取数据

在确定所要搜集的热搜或者视频对象之后，运用微博爬虫以及 b 站爬虫（b 站爬虫即调用我们小程序 api）完成数据的采集。

C）分析情感

将获取的源评论，按照获取的时间，分词后分批导入我们的舆论语义分析卷积网络 api 中，进行分析，并逐一记录结果。

D）分析数据并得出结论

结合原始数据，加之刚刚获得的情感分析的数据，分析情感数据的变化，以及其影响因子，以获得最后的结论。

## 5.3 案例背景与研究起点

### 5.3.1 事件爆发

5 月 10 日早晨 6:35，微博用户@四十九中林同学妈妈（原 id：@人生就像泡沫）发布微博，称 5 月 9 日晚上九点接到学校通知，得知其就读于成都四十九中的儿子林同学于楼道坠亡。

中午 12:32，当事人母亲再次发布微博，称救护车 8 点半到学校，林同学直接被运往殡仪馆。

中午 14:33，当事人母亲再次发布微博求助，并称“有媒体表示背后的水太深”“唯独事发那一段没有监控”“学校已经想好了对策舆论不怕法律程序更不怕随便我们怎么闹”。

该起案件在当天中午得到了网友的热烈回应，在其发布的第一条微博的评论区，网友们热心地艾特各大媒体的微博账号，并在评论出谋划策，如给相关领导打电话、花钱上热搜。然而，当事人母亲发布第二条微博后（微博中指出救护车八点半到达学校，当事人被直接拉去殡仪馆），评论区逐渐转变风向，一时之间，对学校、媒体的质疑纷涌而至，@成都 49 中的微博评论区充满愤怒、乃至谩骂之声。在其称“唯独事发那一段没有监控”后，众怒彻底被点燃。由于事件关键真相不明，舆论不断发酵。

5 月 10 日晚，微博话题#成都 49 中#下的实时广场充斥着控诉、不满、愤怒、悲伤的微博，他们控诉社会的黑暗，不满官方的敷衍，愤怒地截下匿名回答保留证据，为世间缺席的正义感到悲伤。舆论不断发酵。一时之间，官媒新发布的任何一条微博，不管有关无关，评论区都刷满了“成都 49 中”，激动的网民们向上面讨要证据，要求公布监控。

### 5.3.2 舆论发酵

面对民众的呼声，5 月 11 日，@央视新闻评论，通报需要以更多事实回应，“有一分证据说一分话才可能消弭质疑”，学生、家长、公众都想要一个“在逻辑上形成闭环”的说法。

5 月 11 日凌晨 3:54，@成都成华教育对该事件作出通报，排除刑事案件，基本确定为个人轻生。随后四川网警转发。

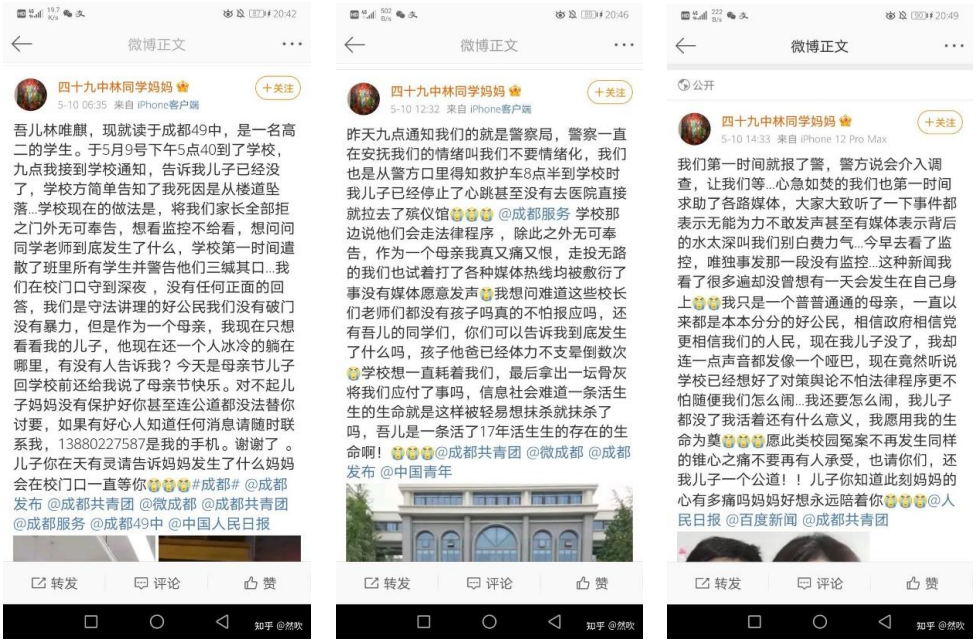
当日晚 19:43，警方@平安成华通报该案，指出“经现场勘查、走访调查、调阅监控、电子数据勘验，提取书证、尸体检验”，排除刑事案件，家属无异议。真相已出，公众情绪略有平息，只是仍然要求得到对其中疑点的解答，如监控情况如何，为何案发两小时后才通知家属（注：此时已知尸体并未火化，当事人已查看监控且监控并未

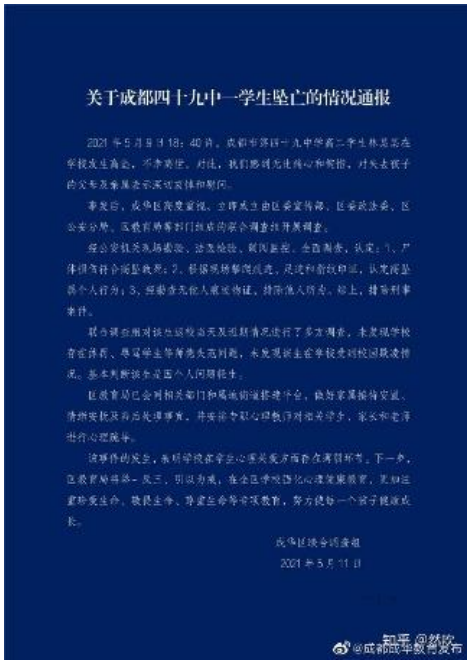
丢失）。

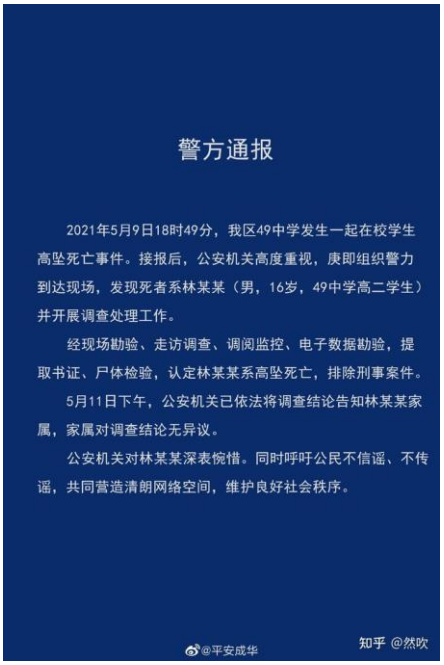
### 5.3.3 水落石出

5 月 13 日凌晨 2:11，新华社还原成都 49 中学生坠亡事件，对该案中学校通知当事人父母时间、救护车到达时间等大众最为关心的问题进行了解答。

当天 6:40，媒体发布监控还原当事人坠亡前轨迹，事件才终于水落石出，这时微博的评论热度达才逐渐平息下来。



### 5.3.4 新的波折与研究起点

可是之后——即 5 月 14 日以后，按理说新华社已经官方辟谣，随着时间流逝，舆论应该逐渐式微，趋于平静。但是，此时微博微博舆论却又开始了不正常的躁动。

这种反常的舆情状况吸引了我们的关注，而这就是我们实验开始的缘由。

## 5.4 实验内容

### 5.4.1 确定研究词条，爬取数据

根据 514 这个特殊的时间点，我们搜集了此时评论量、热度均处于高位的几条热搜词条，比如#小林所在班级化学老师为留学名额将其推下楼#等等，并以此作为我们爬虫的爬取对象。

经过爬虫的爬取，我们获得的评论原数据数据结构（以微博为例，因为 b 站数据通过我们的小程序 api 可以直接获得）抽样展示的直接结果如下表：

| 获赞 | 评论 | 转发 | time | review |
|---|---|---|---|---|
| 81 | 8 | 3 | Fri May 14 00:41:13 +0800 | <a href="https://m.weibo.cn |
| 146 | 28 | 33 | Fri May 14 03:41:20 +0800 | 为什么说爱国成了臭流氓们的最 |
| 23 | 1 | 6 | Fri May 14 04:00:07 +0800 | #成都49中回应事发后通知家长# |
| 179 | 27 | 28 | Fri May 14 06:41:39 +0800 | 转发评论了两条成都49中事件的 |
| 369 | 93 | 13 | Fri May 14 06:46:55 +0800 | <a href="https://m.weibo.cn |
| 7073 | 485 | 623 | Fri May 14 08:13:22 +0800 | <a href="https://m.weibo.cn |

### 5.4.2 源数据处理与情感分析

我们明显可以看到，此时的评论数据是包含于 html 字符串中的，所以我们首先对评论进行正则，将评论中文本体从 html 语言中提取出来。

在此基础上，利用 jieba 对其进行分词处理，将其分解为词向量以便于后续卷积神经网络的预测与计算。

下表抽样显示了从原数据，经过正则、分词分析后得到的结果。

| review | fenci |
|---|---|
| <a href="https://m.weibo.cn | 成都 中以 一个 过来 者 |
| 为什么说爱国成了臭流氓们的最 | 为什么 说 爱国 成 了 臭 流氓 们 |
| #成都49中回应事发后通知家长# | 成都 中 回应 事发 后 通知 家长 |
| 转发评论了两条成都49中事件的 | 转发 评论 了 两条 成都 中 事件 |
| <a href="https://m.weibo.cn | 中 林 同学 妈妈 |
| <a href="https://m.weibo.cn | 光 计划 嘻嘻 |

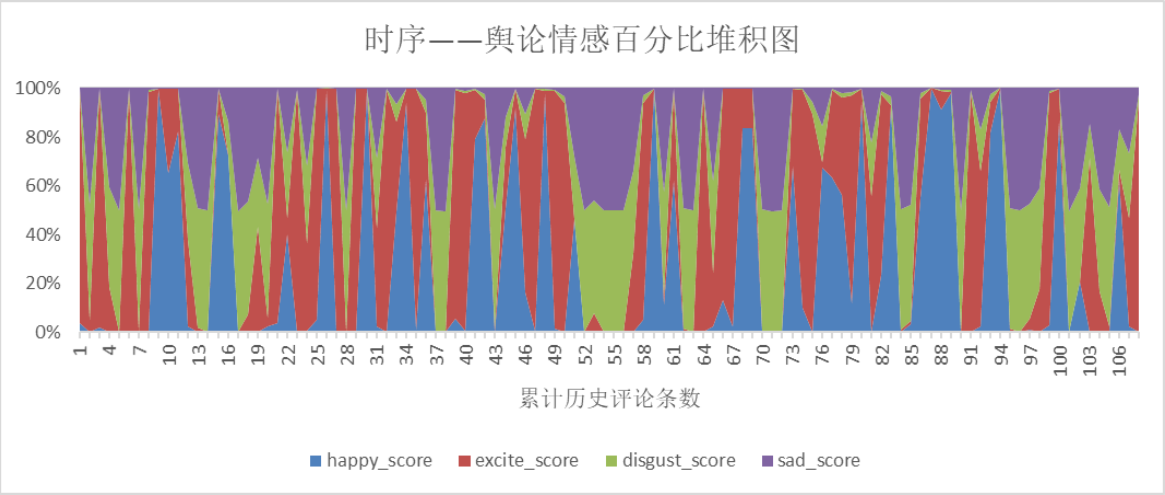
在将原始评论分解为分词向量后，就以这些词向量为基点，送进卷积神经网络进行语义预测。

预测结果数据结构显示如下

| review | fenci | ppy_sco | cite_sco | gust_sc | ad_score |
|---|---|---|---|---|---|
| <a href="https://m.weibo.cn | 成都 中以 一个 过来 者 | 0.037278 | 0.960651 | 0.000962 | 0.001109 |
| 为什么说爱国成了臭流氓们的最 | 为什么 说 爱国 成 了 臭 流氓 们 | 0.002583 | 0.046699 | 0.476029 | 0.474689 |
| #成都49中回应事发后通知家长# | 成都 中 回应 事发 后 通知 家长 | 0.021399 | 0.977814 | 0.000361 | 0.000427 |
| 转发评论了两条成都49中事件的 | 转发 评论 了 两条 成都 中 事件 | 0.001479 | 0.183952 | 0.408119 | 0.40645 |
| <a href="https://m.weibo.cn | 中 林 同学 妈妈 | 0.003303 | 0.000981 | 0.495645 | 0.500071 |
| <a href="https://m.weibo.cn | 光 计划 嘻嘻 _ | 0.002685 | 0.996863 | 0.000198 | 0.000254 |

由此，即可获得最终的对于每一句的情感语义分析结果。

按时间可视化后画图如下：




可以很明显看到，单条评论的时序情感波动太大，数据可观察性较差，需要进一步处理。

### 5.4.3 对情感数据的进一步处理

由此，为了获取稳定的舆情数据，我们将获得的单个舆情点打包成组，对其进行了时间序列平滑，通过将 100 个点为一组的采样平滑窗口，对曲线进行了整理，整理后得到数据结构结果如下：

| happy | excite | disgust | sad | time |
|---|---|---|---|---|
| 0.274875 | 0.338429 | 0.192936 | 0.193759 | Fri May 14 22:25:46 +0800 2021 |
| 0.283532 | 0.339753 | 0.18797 | 0.188745 | Fri May 14 22:26:22 +0800 2021 |
| 0.283161 | 0.330147 | 0.192935 | 0.193757 | Fri May 14 22:26:28 +0800 2021 |
| 0.285233 | 0.329686 | 0.191988 | 0.193093 | Fri May 14 22:27:17 +0800 2021 |

在这里，为了能充分分析数据中的信息，以更好地对舆情进行分析，我们首先对获得的平滑数据作一个方差分析，分析其每一个因子是否存在显著的特异性。

将四个情感因子分别记录为 0、1、2、3，到方差检验数据如下：

ANOVA

value

| | Sum of Squares | df | Mean Square | F | Sig. |
|---|---|---|---|---|---|
| Between Groups | 3.971 | 3 | 1.324 | 1188.900 | .000 |
| Within Groups | 1.332 | 1196 | .001 | | |
| Total | 5.303 | 1199 | | | |

得到各个因子对数据的检验结果 p=0.000（极其显著），说明各个参数间以极其显著的 p 值存在不同质，情感分析有效。

接下来进一步分析各个因子内部情况。

Multiple Comparisons

Dependent Variable: value

LSD

| (I) type | (J) type | Mean Difference (I-J) | Std. Error | Sig. | 95% Confidence Interval | |
|---|---|---|---|---|---|---|
| | | | | | Lower Bound | Upper Bound |
| 0 | 1 | -.10159490350* | .00272453758 | .000 | -.1069403085 | -.0962494985 |
| | 2 | .03972819984* | .00272453758 | .000 | .0343827948 | .0450736049 |
| | 3 | .03841413650* | .00272453758 | .000 | .0330687315 | .0437595415 |
| 1 | 0 | .10159490350* | .00272453758 | .000 | .0962494985 | .1069403085 |
| | 2 | .14132310333* | .00272453758 | .000 | .1359776983 | .1466685084 |
| | 3 | .14000904000* | .00272453758 | .000 | .1346636350 | .1453544450 |
| 2 | 0 | -.03972819984* | .00272453758 | .000 | -.0450736049 | -.0343827948 |
| | 1 | -.14132310333* | .00272453758 | .000 | -.1466685084 | -.1359776983 |
| | 3 | -.00131406333 | .00272453758 | .630 | -.0066594684 | .0040313417 |
| 3 | 0 | -.03841413650* | .00272453758 | .000 | -.0437595415 | -.0330687315 |
| | 1 | -.14000904000* | .00272453758 | .000 | -.1453544450 | -.1346636350 |
| | 2 | .00131406333 | .00272453758 | .630 | -.0040313417 | .0066594684 |

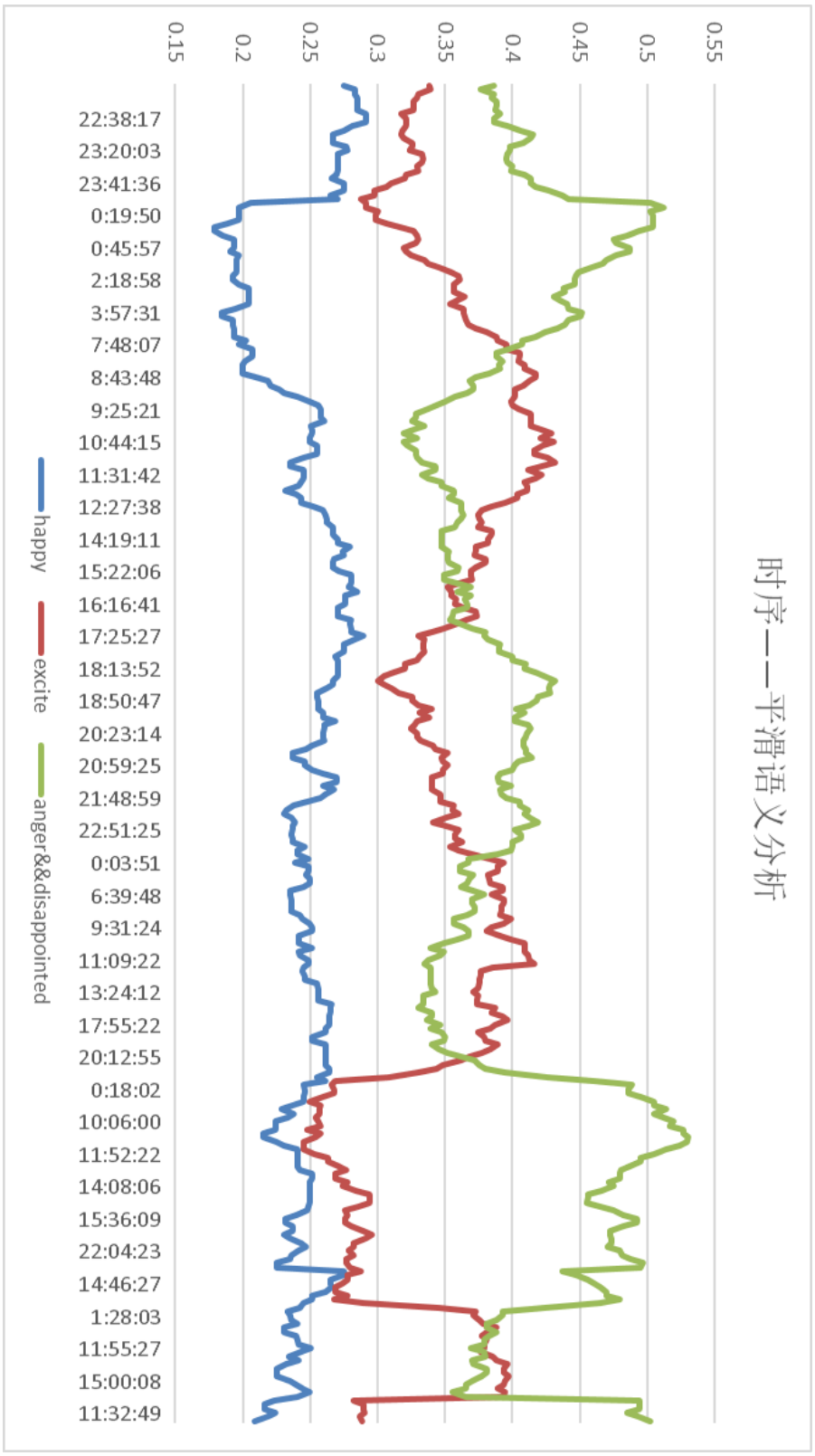
*. The mean difference is significant at the 0.05 level.

可以看出，各个整体因子间显著性都为 0，效果非常良好。但是第二、三号因子间显著性稍有逊色。

所以为了更好的分析数据，我们这里根据二者数据具体的均值、方差对此指标进

行了降维归一化，将其合并成愤怒失望因子。

最后，我们得到的时序舆情数据如下：




上表的时间原点是 5 月 14 号，每次时间经过 24 点说明达到了第二天。

观察数据，峨眉明显可以发现，在官方公布真相之后，随着时间的推移，仍然整

体舆论起伏严重。

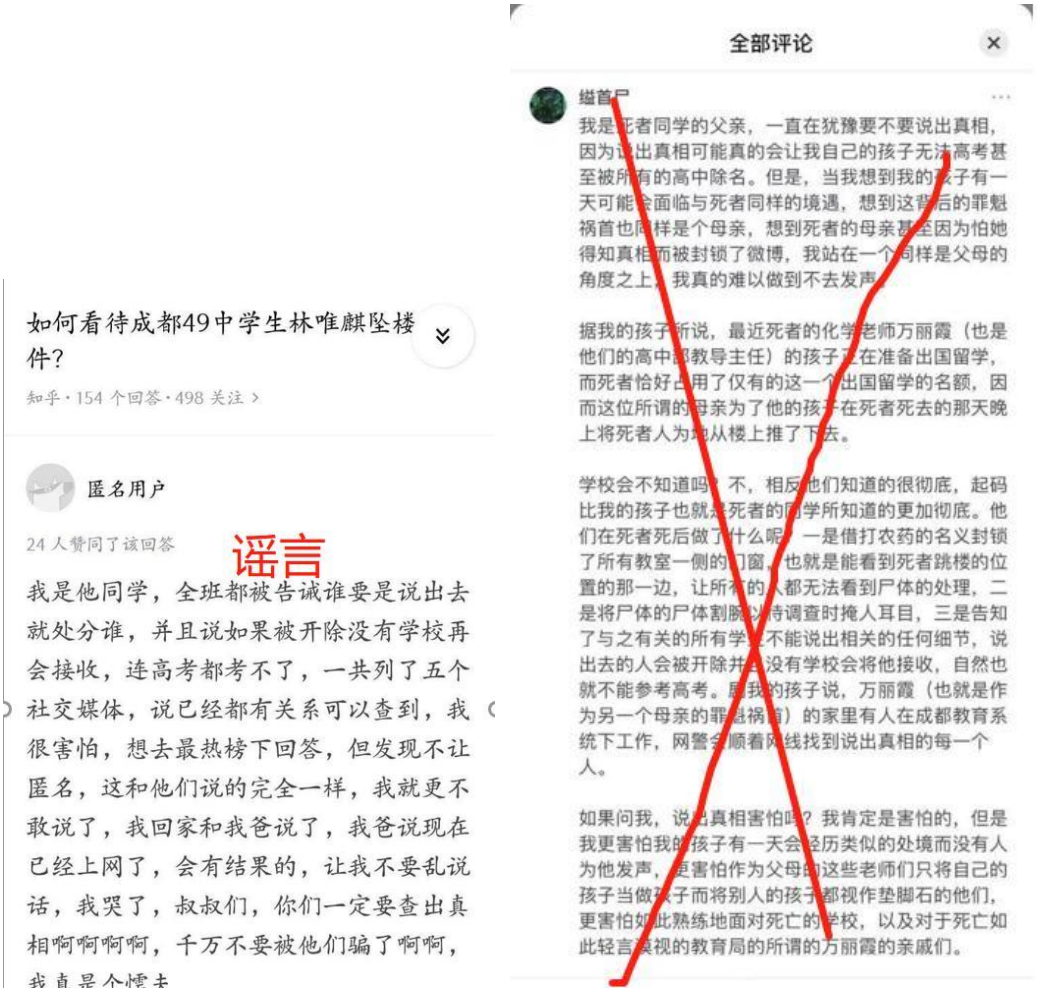
### 5.4.4 结果数据分析与舆情变动原因

在检查我们自己计算出的舆情数据后，我们发现在官方定性之后舆情仍然沸腾其主要特点在于负面情绪的快速上升，顺着厌恶曲线的上升节点，几乎所有主要情感数据都指向了 “小林所在班级化学老师为留学名额将其推下楼”“坠楼学生非自杀”这两个热搜。

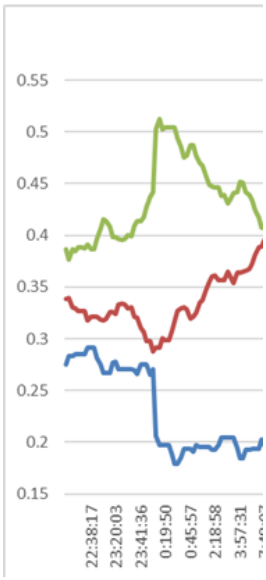
而经过查阅资料，这两个热搜均为造谣。

这两个帖子均来自于知乎上的两个用户，其中一个匿名用户自称为死者小林的同学，并其贴文中写道“全班被告诫谁要说出去就处分谁……甚至连高考都不让参加”，其还在文后呼吁更多的人“查出真相”。

另外一名为“缢首尸”的知乎用户自称为死者小林同学的父亲，其在知乎上发表了“小林是因为占了化学老师孩子出国名额被害”，并在知乎上呼吁更多人来关注“事情真相”。

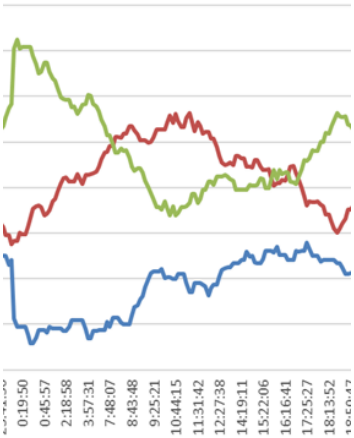



这就是微博舆情在事件官方定性后，依然沸腾的罪魁祸首——谣言。

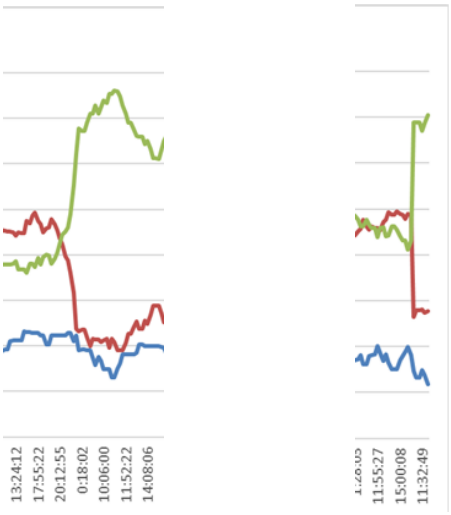



这就是为什么会出现上图代表愤怒与失望的绿色曲线陡然上升，而代表乐观的曲线快速下降的原因。这使得原来政府的披露报告瞬间受到许多网民的怀疑，一石激起千层浪，网上阴谋论不断传出，大量的不实负面情绪充斥于互联网。


之后，随着造谣帖子的被查明，造谣者的自首，理论上舆情应该恢复稳定了，而观察图线，发现民意确实有所缓和。说明即时的辟谣和坚决打击造谣是绝对必要的。

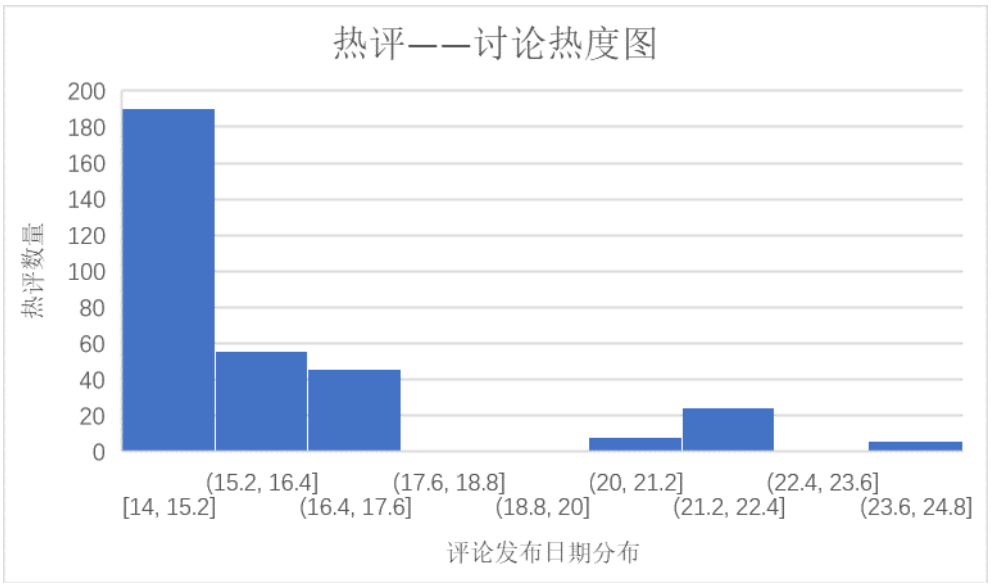



但是继续观察曲线，我们并没有发现这一点。




观察上图，我们可以发现，每过一段时间，舆论就会暴起一次。

为了研究这个情况，我们又处理的原数据，绘制了热评——讨论分布图（横坐标的值代表发帖日期）




这里我们就明显可以看到，舆情具有的独特的特性：反复性与聚集性。对比上述的两个负面情绪暴涨的时段，我们可以发现其正好对应于 20-22 日以及 23-24 日的两个发帖讨论高潮。因此，舆情曲线会反复震荡也就不足为怪了。

## 5.5 案例分析与结论

从鲍毓明事件到如今的成都 49 中学生坠亡事件，近年来，我们好像经历了太多次网络舆论和真相的反转。

而往往造成巨大反转、引发舆论灾难的背后动因，很大一部分归咎于谣言，“造谣一张嘴，辟谣跑断腿。” 说的就是此。

这次的实验在官方定性后对舆论进行监测，说明了即使有官方定性，在受到谣言

的影响之下，在互联网上依旧会产生巨大的舆论振动之余波。试问，如果造谣时，官方仍未定性，将会造成多么惨烈的舆论风暴？

同时，我们通过时序图以及发言分布，找到了舆论的反复性与聚集性。反复性在于过一阵子，舆论就会死灰复燃，重新肆虐；聚集性在于一旦某条舆论由死灰复燃，就带来的舆情就会立马集中爆发。

所以，我们的实验结论是：

a)网络不是法外之地，政府一定要严打造谣；即时的辟谣和坚决打击造谣是绝对必要的。

b)作为新时代的网民，一定要有思辨能力，拒绝消极发言（“不相信正义了”），拒绝听风就是雨。理性面对舆情，独立作出判断。

c)由于舆情的反复性，官方消息对真相的披露一定要快速、彻底，杜绝舆论由于自然传播的死灰复燃，或者被“别有用心”的人主动提起。

d)由于舆论的集中性，在打击谣言时，一定要快准狠，精准敲掉谣言源头，防止不明真相的群众由于信息不对称而跟风转发，造成舆论的蝴蝶效应。

## 5.6 展望

反转、造谣、再反转，不禁令我们思考，下一次再有类似事件，情况是否还会相同，会不会发生“狼来了”这样的可悲情形？

事实证明，公众的同情心还是非常强，社会永远不会缺少热忱的人。但在感性的笼罩下，理性似乎总是迟到一步。固然，有疑点可以质疑，政府的处理能力也可以在与民众的交流中得到提升。但不能以违法的方式表示“质疑”，比如“人肉”、网暴当事人；更不能为达到某种目的，裹挟民意造谣。对网传消息可以留一份心眼，不必总是盲目相信，不要被浑水摸鱼之人利用。曲解了事实，就要接受自己的曲解，正视事实，拥抱真相。反转本身不可怕，可怕的是舆论被造假、被操纵、被利用。

因此，我们希望不会有“狼来了”的悲剧，也希望感性的背后，能有理性支撑。

这就是我们开发这项语义分析工具的**初心**。

# 6 安装使用

## 6.1 后端 api 调用方法

http://121.4.163.229/info/ 首页热榜信息 api

http://121.4.163.229/detail/?id=BV1yh411S7au 视频详细信息 api

http://121.4.163.229/comment/?id=BV1yh411S7au 评论信息 api

http://121.4.163.229/danmaku/?id=BV1yh411S7au 弹幕信息 api

http://121.4.163.229/pre/?id=BV1yh411S7au 语义分析信息 api

http://121.4.163.229/wordcloud/?id=BV1yh411S7au 词云 api

#这里的 id=BV1yh411S7au 后面的编码为 b 站视频对应 bv 号，可以在 b 站任意视频旁直接获取，只要输入不同的 bv 号，就可实现对特定视频的具体分析了。这里给出一例子供评委老师使用

## 6.2 前端小程序使用方法

微信扫一扫 扫码即可

# 7 作品总结

## 7.1 项目背景与创意

如果我是一位个人视频发布者，我怎么才能快速了解观众对视频的情感喜好？怎么了解观众对视频内容的反馈？如果我是一个公司的视屏宣传部部长，我该怎么快速进行舆论公关？怎么快速了解客户对我们品牌的需求与意见？如果我是一个党和政府的主流宣传媒体，我怎么快速了解人民意见所向，了解网络空间意识形态？怎么在互联网唱响社会主义主旋律？

可见，随着当今互联网视频行业快速发展，随着视频网站的快速发展，在大数据时代下，不管是视频的制作者、还是观看者，都越来越急迫的显现出一种需求——一种快速的获取视频背后受众群体对视频到底产生何种情感的需求。

基于上述背景，我们希望开发一个为有此类需求的人员设计的、无需下载的、方便的语义分析一体化工具。

对于个人视频发布者，我们希望这个小程序可以帮助其快速掌握受众对视频的态度与看法，从而加速、优化视频制作。

对于企业视频宣传部工作者，我们希望这个小程序可以让其快速掌握与企业相关的舆论风评、快速了解客户需求，提升客户忠诚度、或及时完成危急公关。

对于党和国家的主流宣传媒体，我们这个小程序可以帮助其快速掌握社会舆情新情况，加深、加速主流思想通过各类视频媒介在人民群众中间的传播，及时管控错误言论、了解社会特定事件舆论导向等等，提振互联网的主旋律之声。

## 7.2 项目模块与功能

本项目以卷积神经网络语义分析为主，围绕其功能，前端由三大模块组成：b 站热榜信息展示、待搜索视频详细信息展示（包括评论、回复、点赞数据；以及筛选概括后的词云、以及最重要的——语义分析结果）、以及用户搜索的历史记录。

小程序**第一页**为爬虫抓取的 b 站热榜数据，

a) 给予用户一个立体的视角以观测 b 站最近的热点（即视频说了什么），响应项目的主题。
b) 点击感兴趣的主题，界面将自动跳转到第二页，进行进一步详细分析与数据汇总输出。

小程序**第二页**为项目核心功能页面：

c) 获取当前视屏的详细信息
d) 当前视屏可在上方框框内直接搜索，或者直接承接首页热榜或者浏览记录
e) 返回对该视频评论的自然语义情感分析
f) 返回改视屏高频弹幕组成的词云
g) 返回视屏的简介
h) 返回高赞评论及其回复

小程序**第三页**利用内存技术，记录了用户过去的搜索记录；同时点击历史记录将完成跳转，重回第二页分析页。

## 7.3 技术路线运用与特色

此项目的主要功能是使用 Django 框架搭建后端，并部署于腾讯云服务器（CentOS 8.0 64bit），使用微信小程序开发者工具开发前端。以快速语义分析为核心，并为了进一步优化用户使用方便性相继开发了热榜搜索、历史记录等附加内容。

即时的评论抓取与即时的卷积自然语义分析是我们项目最大的特色。

## 7.4 工作量

本项目从初步定方向，到搭建中转后端，从设计前端微信小程序，到部署服务器正是提供 api 上线，单单在技术构架实现方面，就已经付出了极大的工作量。

在训练卷积神经网络时，我们运用了各种方法进化我们的模型，尝试了各种的参数设置与模型优化，这也是需要极大的工作量的。

最后的舆情分析案例，需要大量搜集网络上的相关信息，并反复运行我们的 api，在数据中挖掘结论，这也需要很大的工作量。

但是，系统在我们手中一步步走向使用，这种成就感早已盖过了那巨大的工作量了。

## 7.5 进一步应用/展望

这套微信智能语义评价系统，作为一个效率工具，以互联网视频为抓手，集发现当下热点、快速分析群众背后具体情感等功能为一体。

项目内容紧密结合生活实际需求，如果获得规模化应用，能够有效解决个人、企业以及主流宣传媒体快速分析视频受众情感与相关热点舆情的需求，完成了热门搜索，评论语义，弹幕词云，历史记录等功能，体验良好，具有较强的实用价值和应用前景；同时也提供了 api 定制服务，将极大的提升运用的灵活性与定制性。

# 8 参考文献