****

课程设计报告书

**题目：根据图片寻找相似图片**

**学 院 计算机科学与工程学院**

**专 业 计算机科学与技术**

**学生姓名 陈钊浩 陈浩鑫 洪浩强 张朝晖**

**学生学号**

**指导教师 高英**

**课程编号 145242**

**课程学分 2.0**

**起始日期 2018/04**

|  |  |
| --- | --- |
| 教  师  评  语 | 教师签名：  日期： |
| 成  绩  评  定 |  |
| 备  注 |  |

**根据图片寻找相似图片**

一**、**选题背景（总体设计）

需求规定：输入一张图片或者文字,在图片库内搜索相似的图片

软硬件环境需求：可移植，采用python语言，keras平台

软件总体结构：

1.训练过程：（分模块，并行开发）

爬虫模块：根据需求爬取数据集

目标识别模块：采用VOC格式数据集进行训练，得到权重文件

搜图模块：jpg图片输入，并行训练多个分类器，得到权重文件

1. 形成图片库：（串行开发）

目标识别：根据权重文件预处理图片库

搜图模块提取特征子模块：提取图片库特征形成特征库文件

搜图模块分类别子模块：图片库中各图片分类概率形成类别概率文件

1. 落地应用：（串行开发）

a.以图搜图

输入图片：使用图片路径输入

图片预处理：目标识别模块

图片特征生成：搜图模块形成图片的特征

查询图片库：特征比较模块在图片库特征中搜索相似图片

b.以文字搜图

输入文字属性：采用不定项选择输入

属性转换为类别：使用算法将输入转换为类别向量

查询图片库：类别比较模块在图片库特征中搜索相似图片

二**、**方案论证(接口设计)

本次需求是构建以图搜图，以文字搜图的系统。

1.流程设计

因从我们对自己定位是算法研发团队，使用我们产品的为产品开发团队。所以我们设计是一个针对特定问题的算法开发流程。

根据需求，我们确定了算法使用的流程图如下:

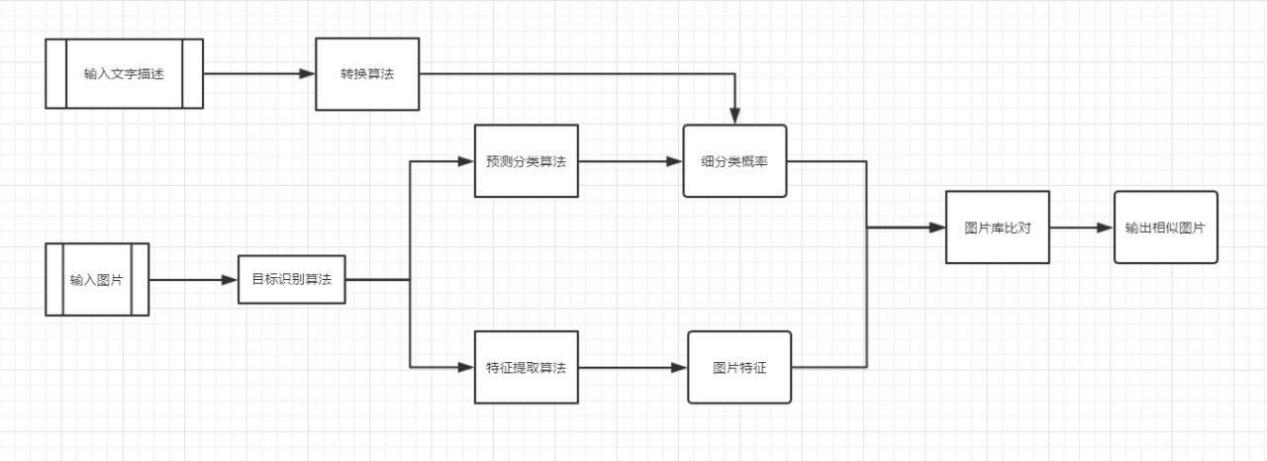


图1. 以图搜图系统算法设计流程图

如图所示， 算法的输入分为两部分， 一部分是直接输入原始图片， 另一部分是输入文字搜索。

针对输入图片部分， 需要经过目标识别(背景去除)， 特诊提取， 特征比较， 类别标记， 类别比较等。

针对输入文字部分， 需要经过文字与类别概率转换，类别比较。

算法的输出为输出搜索出来的结果图片

2.算法设计

算法的处理采用机器学习的方法，需要经过以下几个流程:

数据获取: 根据算法输入，定制数据集，使用爬虫从各大网站抓取图片资源

数据清洗: 对抓取的图片进行数据清洗， 用于筛选质量高的数据，有利于之后算法的训练

模型设计: 根据不同的需要采用不同模型， 本次目标识别采用SSD模型; 特征提取部分采用VGG预训练模型，分类模型采用简单卷积网络、VGG变形网络双模型。

训练模型: 将数据集输入到模型中，调试训练的参数，使算法的误差收敛至一个较优的结果。

模型测试：将训练出来的权重保存，用于算法落地应用中。

3.主要模块设计

3.1目标识别模块

目标识别算法现在已经有相当成熟的开源解决方案，如SSD,YOLO等均在Pascal VOC, COCO上有优秀的表现。本次使用的Single Shot MultiBox Detectors算法完全消除了提出生成和随后的像素或特征重新采样阶段，并将所有计算封装到单个网络中，相对Faster R-CNN等算法效率更高。此外，SSD采用python语言编写，相比于采用C++算法编写的YOLO，可以在训练时采用谷歌Colab云平台进行加速。因此，本次大作业选用SSD算法作为目标识别的框架，并进行裁剪整合，作为整个项目中的图片预处理模块。

在后续的工作中，开发者只需调用导入Cut.py并调用cut函数，即可完成预处理的工程。

3.2相似性搜索模块

实现图片的相似性搜索，我们得出两种思路：

第一：对图片进行类别标记，使用类别标记判断图片属于哪一些细分类，再从细分类中搜索同一分类下的图片。

第二：对图片进行特征提取，使用图片的特征进行距离计算，搜索距离最近的图片。

类别标记部分实现思想：

使用深度学习方法：自从2012年AlexNet为里程碑的深度学习方法在ImageNet竞赛上大放光彩以来，从多方面已经实践得出深度学习在图像分类方面有着与人类比肩，甚至超过人类水平的结论。

使用迁移学习方法：采用在ImageNet上针对大规模数据集的模型比直接采用初始化的网络在小数据集上的表现要好。神经网络的训练受到参数设置的影响很大，采用预训练网络是使用了前人在大规模数据集上的工作积累，节省自己实现时对参数设置的重复工作。

使用多分类器方法：本次针对T恤的分类器初步设置成7个，即七大类。每大类下分为几小类。这样多分类器能为之后模块化训练分类器提供更好的扩展性，即不同类别分类器彼此是独立的。

特征提取部分实现思想：

使用卷积网络进行特征提取：传统的基于颜色直方图、图像哈希算法在搜索图片相似上表达性有限；所以我们采用卷积网络，提取图像的高层特征，而不只是简单的颜色比例信息。

使用特征进行比较：图片的特征可以看成是图片映射到另一个空间的表示，在特征空间的距离计算能更加有效寻找在高层语义空间相似的图片。

三**、**过程论述（面向对象设计与数据库设计）

本部分论述过程为：

爬虫部分实现（张朝晖同学）

目标识别SSD模型实现（洪浩强同学）

搜图模块（陈浩鑫同学）

**1.爬虫部分：**

我们使用Python语言脚本作为抓取图片的爬虫工具，抓取的对象为淘宝和唯品会上使用特定关键词搜索时的商品图片。

def openUrl(*url*):

    headers = ("user-agent","Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.22 Safari/537.36 SE 2.X MetaSr 1.0")

    opener = urllib.request.build\_opener()

    opener.addheaders = [headers]

    urllib.request.install\_opener(opener)

    data = urllib.request.urlopen(url).read().decode("utf-8", "ignore")

    return data

openUrl方法接收一个url字符串，返回url对应页面的数据。

def savePic(*path*, *url*):

imgData = urllib.request.urlopen(url).read()

f = open(path, "wb")

f.write(imgData)

f.close()

savePic方法接收一个保存路径path和一个图片url，可将url的图片内容写入本地目录下的一个文件中。

KeywordsArray = ['长袖', '短袖', '老年', '童装', '学生', '条纹', '印花', '字母', '红', '黑', '白', '蓝', '运动', '韩风']

for Keyword in KeywordsArray:

inputString = 'T恤 女 ' + Keyword

keywords = urllib.request.quote(inputString)

savePath = "/home/keo/Programming/crawler/" + inputString + "/"

mkdir(savePath)

count = 0

pages = 3

for i in range(pages):

url = "https://s.taobao.com/search?q=" + keywords + "&imgfile=&commend=all&ssid=s5-e&search\_type=item&sourceId=tb.index&spm=a21bo.50862.201856-taobao-item.1&ie=utf8&bcoffset=4&ntoffset=4&p4ppushleft=1%2C48&s=" + str(i\*44)

data = openUrl(url)

img\_pat = '"pic\_url":"(//.\*?)"'

img\_link = re.compile(img\_pat).findall(data)

for j in range(len(img\_link)):

img = "http:" + img\_link[j]

print(img)

imgData = urllib.request.urlopen(img).read()

saveFile = savePath + str(count) + '.jpg'

print(saveFile)

savePic(saveFile, img)

count += 1

生成一个包含搜索关键词的url后，使用openUrl函数获取该url的数据，数据中包含搜索结果的商品图片的url，再使用正则表达式对数据中的图片url进行匹配，即可获取符合特定模式的图片url数组，最后使用savePic方法将图片保存到本地。

**2.目标识别SSD模型**

SSD概述：SSD(Single Shot MultiBox Detector)是一个使用单个深度神经网络来检测图像中的目标的方法。它将边界框的输出空间离散化为不同长宽比的一组默认框和并缩放每个特征映射的位置。SSD完全消除了提出生成和随后的像素或特征重新采样阶段，并将所有计算封装到单个网络中，相对其他算法而言效果更快。

SSD检测框架：SSD将根据不同尺度（例如图2的的8×8和4×4分块）划分为对应的的feature map cell集合,评估几个特征映射中每个位置处不同长宽比的default box（即下图1虚线方框）。对于每个default box，SSD预测所有目标类别(对于这次T恤识别来说实际上是个单分类)的在两个方向上的偏移量和置信度。在实际训练中，只需提供原始图片和对应目标的预测框。

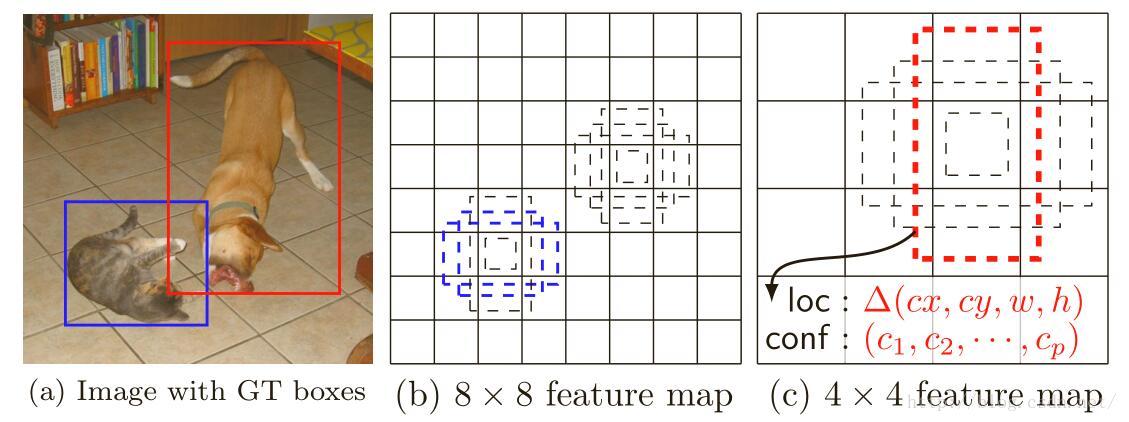
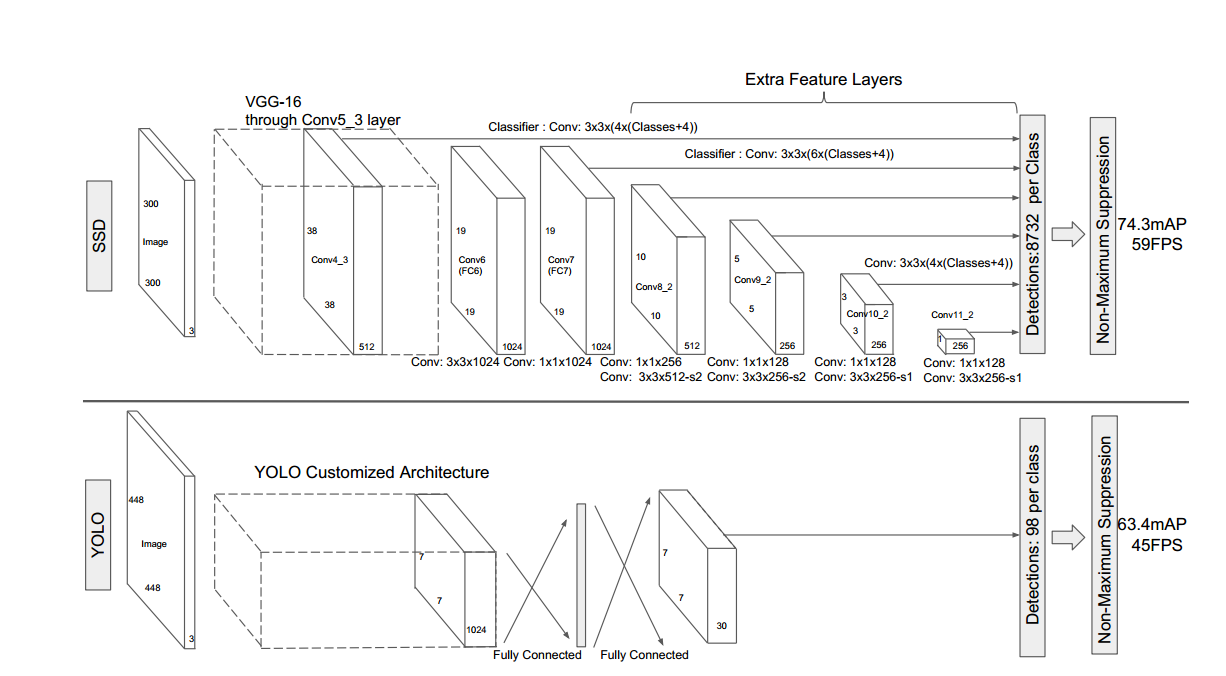
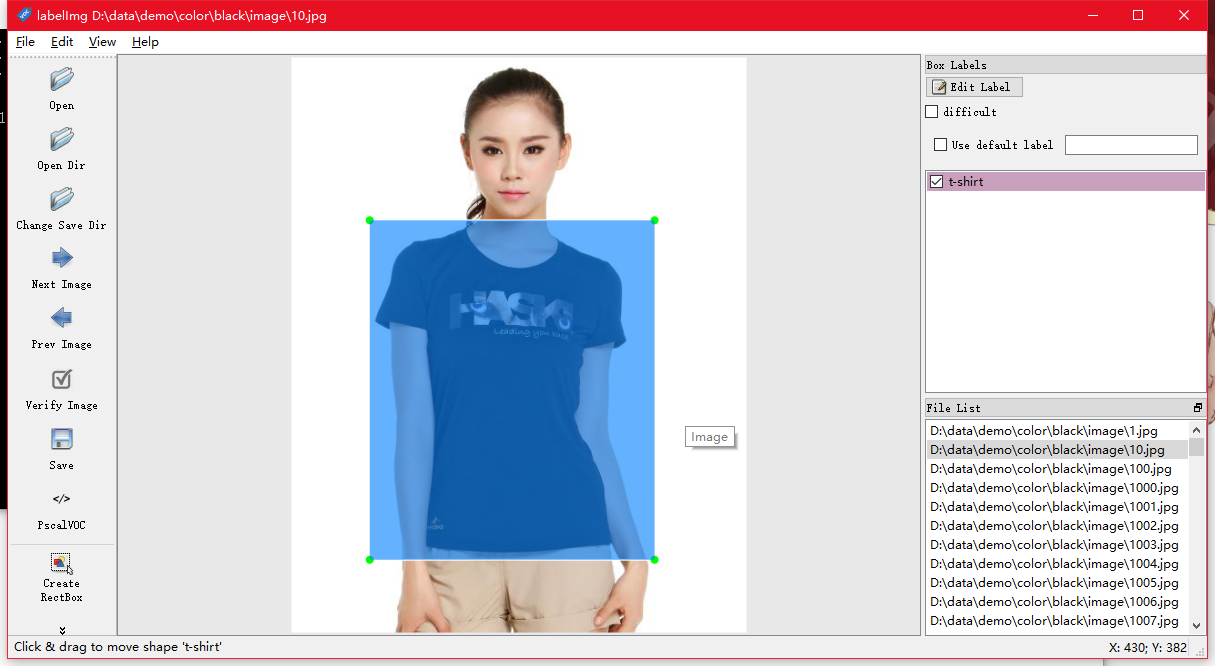


图2 SSD识别box示意图

模型： SSD 模型的最开始部分，作者称作 base network，是用于图像分类的标准架构。与YOLO相比，SSD将图片reshape为300\*300而不是448\*448。SSD网络主体设计的思想是特征分层提取，并依次进行边框回归和分类。可以看到在图3中每个卷积层的大小是逐层递减的，这使得SSD可以适应多尺度的目标预测。其中，第一个state使用了VGG16的前5层卷积网络。由于每个default box的位置是固定的，对于feature map中的每一个块，都需要计算最终预测框的4个偏移值，对应于c个类，需要计算每个类的得分。对于对于一张 m×n 大小的 feature map，即会产生 (c+4)×k×m×n 个输出结果。

图3 SSD模型图

训练部分：通过labellmg软件手工制作约1800张图片的VOC格式训练集，将训练得出的模型用于商品图片的预处理工程。训练集制作过程见图4。

图 4 数据集制作

整合结果（部分代码）：输入图片路径，获取SSD模型对应每个feature map cell的评估分数以及坐标偏移量，若高于阈值0.8，则视为可接受的预测结果并保存在当前文件夹cut路径下。

def plt\_bboxes(path,imgname,img, classes, scores, bboxes, figsize=(10,10), linewidth=1.5):

"""Visualize bounding boxes. Largely inspired by SSD-MXNET!

"""

#fig = plt.figure(figsize=figsize)

#plt.imshow(img)

height = img.shape[0]

width = img.shape[1]

colors = dict()

for i in range(classes.shape[0]):

cls\_id = int(classes[i])

if cls\_id >= 0:

score = scores[i]

if cls\_id not in colors:

colors[cls\_id] = (random.random(), random.random(), random.random())

ymin = int(bboxes[i, 0] \* height\*0.9)

xmin = int(bboxes[i, 1] \* width\*0.85)

ymax = int(bboxes[i, 2] \* height\*1.2)

xmax = int(bboxes[i, 3] \* width\*1.1)

print(ymin,ymax,xmin,xmax)

if(score > 0.8):

# print("processing "+ imgname +" picture")

region = img[ymin:ymax,xmin:xmax]

img\_path = path+"/cut/"+ imgname+ '-' + str(i) + ".png"

mpimg.imsave(img\_path,region)

return img\_path

return 'Fail to cut'

在实际应用中，只需导入cut.py文件，就能加载训练后得到的模型并生成相应结构。调用Cut函数并输入图片路径就能得到截取T恤的图片。

以下是cut.py 代码

import os

import math

import random

import numpy as np

import tensorflow as tf

import cv2

slim = tf.contrib.slim

#%matplotlib inline

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import sys

sys.path.append('./')

from nets import ssd\_vgg\_300, ssd\_common, np\_methods

from preprocessing import ssd\_vgg\_preprocessing

import visualization

# TensorFlow session: grow memory when needed. TF, DO NOT USE ALL MY GPU MEMORY!!!

gpu\_options = tf.GPUOptions(allow\_growth=True)

config = tf.ConfigProto(log\_device\_placement=False, gpu\_options=gpu\_options)

isess = tf.InteractiveSession(config=config)

# Input placeholder.

net\_shape = (300, 300)

data\_format = 'NHWC'

img\_input = tf.placeholder(tf.uint8, shape=(None, None, 3))

# Evaluation pre-processing: resize to SSD net shape.

image\_pre, labels\_pre, bboxes\_pre, bbox\_img = ssd\_vgg\_preprocessing.preprocess\_for\_eval(

img\_input, None, None, net\_shape, data\_format, resize=ssd\_vgg\_preprocessing.Resize.WARP\_RESIZE)

image\_4d = tf.expand\_dims(image\_pre, 0)

# Define the SSD model.

reuse = True if 'ssd\_net' in locals() else None

ssd\_net = ssd\_vgg\_300.SSDNet()

with slim.arg\_scope(ssd\_net.arg\_scope(data\_format=data\_format)):

predictions, localisations, \_, \_ = ssd\_net.net(image\_4d, is\_training=False, reuse=reuse)

# Restore SSD model.

#ckpt\_filename = '../checkpoints/ssd\_300\_vgg.ckpt'

ckpt\_filename = './weight/model.ckpt-220'

# ckpt\_filename = '../checkpoints/VGG\_VOC0712\_SSD\_300x300\_ft\_iter\_120000.ckpt'

isess.run(tf.global\_variables\_initializer())

saver = tf.train.Saver()

saver.restore(isess, ckpt\_filename)

# SSD default anchor boxes.

ssd\_anchors = ssd\_net.anchors(net\_shape)

# Main image processing routine.

def process\_image(img, select\_threshold=0.5, nms\_threshold=.45, net\_shape=(300, 300)):

# Run SSD network.

rimg, rpredictions, rlocalisations, rbbox\_img = isess.run([image\_4d, predictions, localisations, bbox\_img],

feed\_dict={img\_input: img})

# Get classes and bboxes from the net outputs.

rclasses, rscores, rbboxes = np\_methods.ssd\_bboxes\_select(

rpredictions, rlocalisations, ssd\_anchors,

select\_threshold=select\_threshold, img\_shape=net\_shape, num\_classes=21, decode=True)

rbboxes = np\_methods.bboxes\_clip(rbbox\_img, rbboxes)

rclasses, rscores, rbboxes = np\_methods.bboxes\_sort(rclasses, rscores, rbboxes, top\_k=400)

rclasses, rscores, rbboxes = np\_methods.bboxes\_nms(rclasses, rscores, rbboxes, nms\_threshold=nms\_threshold)

# Resize bboxes to original image shape. Note: useless for Resize.WARP!

rbboxes = np\_methods.bboxes\_resize(rbbox\_img, rbboxes)

return rclasses, rscores, rbboxes

def cut(img\_path):

img = mpimg.imread(img\_path)

\_dir,\_name = os.path.split(img\_path)

rclasses, rscores, rbboxes = process\_image(img)

cut\_img = visualization.plt\_bboxes(\_dir,\_name[:-4],img, rclasses, rscores, rbboxes)

return cut\_img

整个预处理的流程可见图5

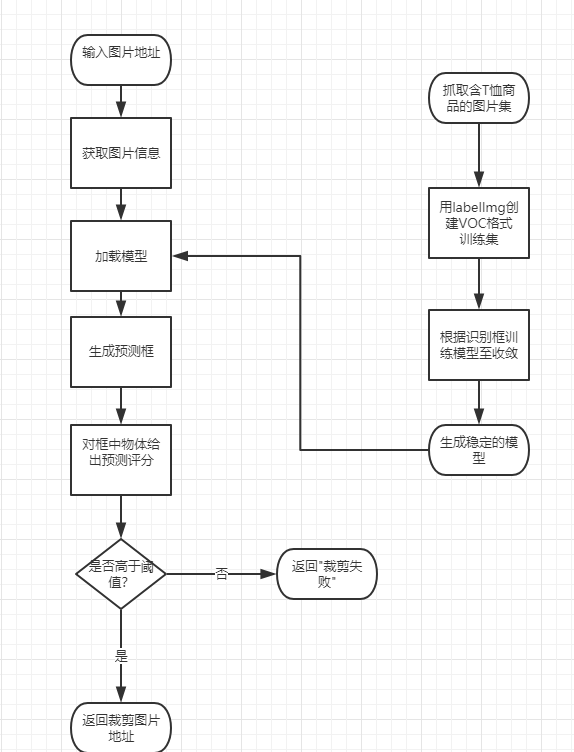


图5 目标识别算法流程图

**3. 搜图模块**

本模块开发分为**三个阶段**：

训练阶段：获取权重文件

特征库形成阶段：使用权重文件生成特征文件

在线查询阶段：输入图片或文字，生成特征与特征库比较

本模块模型包含**两个部分三个子模型**：

分类别部分：（设置七大类，每个模型有七个不同的分类器）

VGG变形网络模型

3层卷积网络模型

特征提取部分：

VGG预训练模型

**3.1 模型介绍**

3.1.1 3层卷积网络模型

卷积网络：一个简单的卷积网络由以下层组成：

a数据输入层/ Input layer

b卷积计算层/ CONV layer

c激励层 / Active layer

d池化层 / Pooling layer

e全连接层 / FC layer

数据输入层，该层要做的处理主要是对原始图像数据进行预处理。包括：去均值、归一化、PCA/白化。

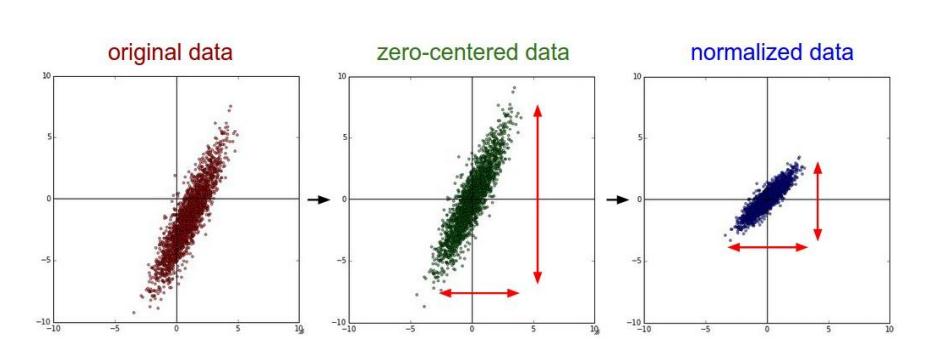
取均值是把输入数据各个维度都中心化为0，归一化是幅度归一化到同样的范围，PCA是将维度降低到方差最大的几个不相关维度。

图6 数据去均值归一化处理

卷积计算层，该层要做的处理是用一个卷积核对输入进行卷积操作。通过卷积核窗口的滑动计算出各个局部的卷积值。

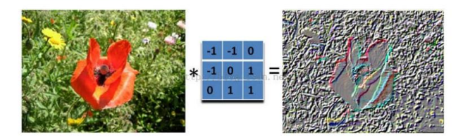
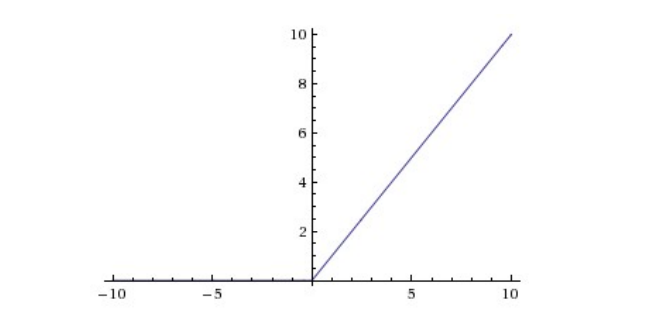


图7 卷积示意图

主要功能：对数据的特征学习与选取。

激励层的计算，该层要做的处理是对卷积层的输出进行非线性变换。将线性空间的组合映射到非线性变换中。

图8 有效且主流的Relu函数。

池化层是为卷积网络提供了对图像平移，伸缩变换带来影响的一个层，将一个局部的输出取均值或取最大值。

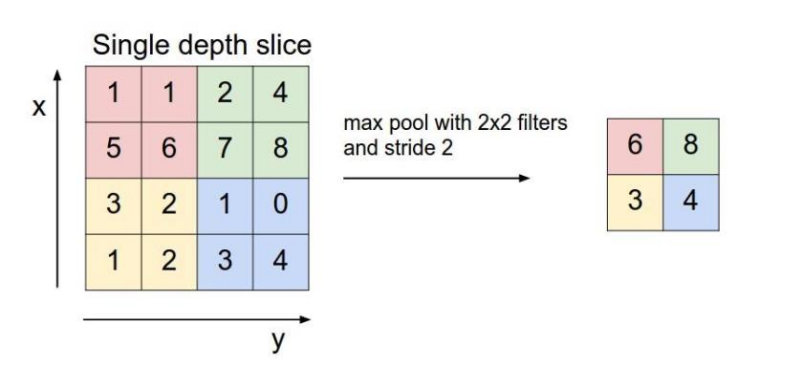
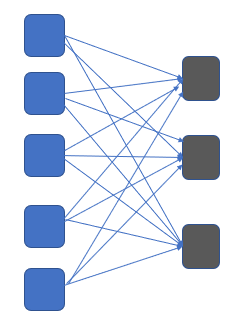


图9 池化层

全连接层的计算是将最后的卷积输出用权重进行组合，匹配到分类或者预测问题。

图10 全连接层

三层卷积网络使用以下的网络：

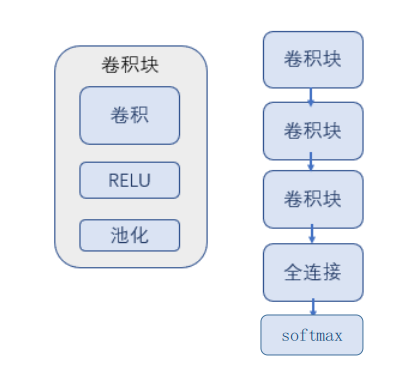


图11 3层卷积网络示意图

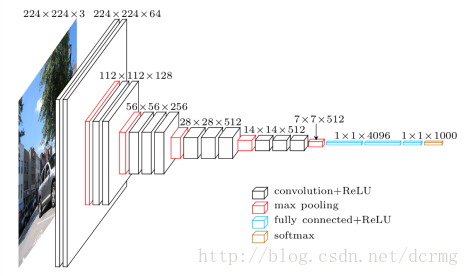
3层卷积能很好地对图片进行分类，将图片的类别设置为离散的one-shot类型，采用softmax损失。

3.1.2 VGG变形网络

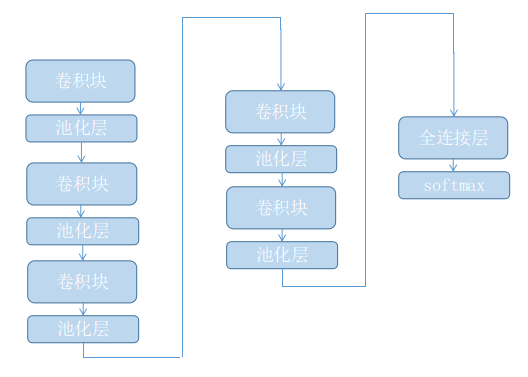
VGG是牛津大学 Visual Geometry Group（视觉几何组）的缩写，曾于2014的ImageNet上获得冠军，成为后来许多模型的预训练模型。

VGG一共有五段卷积，每段卷积之后紧接着最大池化层。

使用多个较小卷积核的卷积层代替一个卷积核较大的卷积层，一方面可以减少参数，另一方面相当于进行了更多的非线性映射，增加了网络的拟合表达能力。

图12 VGG网络示意图

根据VGG原始网络，未来适配本次的分类，将VGG的全连接层去除。加上针对不同类别数的全连接层，使用softmax计算损失。

图13 VGG变形网络

3.1.3 VGG特征提取

将训练好分类器的VGG分类器模型，只加载全连接层之前的层。VGG作为一个特征提取器，很好地将图片高层信息提取。

**3.2 阶段介绍：**

3.2.1训练阶段：

我们设计将T恤的类别分为七类，分别是：

第一类：年龄Age

（'child':0,'girl':1,'orded':2）

第二类：颜色Color

（'black':0,'blue':1,'brown':2,'green':3,'orange':4,'pink':5,'purple':6,'red':7,'white':8,'yellow':9）

第三类：设计Design

（'floral':0,'printing':1,'pure':2,'stripe':3）

第四类：领子lingzi

（'fangling':0,'vling':1,'yuanling':2）

第五类：衣长length

（'long':0,'short':1）

第六类：风格style

（'catoon':0,'korea':1,'sport':2）

第七类：类型type

（'loose':0,'tight':1）

数据集描述：

我们从百度爬去以上所有细分类的图片，每细类图片为train集1000+张，validation集300+张。

数据加载：使用加载器加载：目标大小（150\*150\*3）

# prepare data augmentation configuration  
  
train\_datagen = ImageDataGenerator(  
 rescale=1. / 255,  
 shear\_range=0.2,  
 zoom\_range=0.2,  
 horizontal\_flip=True)  
  
test\_datagen = ImageDataGenerator(rescale=1. / 255)  
  
train\_generator = train\_datagen.flow\_from\_directory(  
 train\_data\_dir,  
 target\_size=(img\_height, img\_width),  
 batch\_size=batch\_size,  
 class\_mode='categorical')  
  
validation\_generator = test\_datagen.flow\_from\_directory(  
 validation\_data\_dir,  
 target\_size=(img\_height, img\_width),  
 batch\_size=batch\_size,  
 class\_mode='categorical')

模型设置：三层卷积网络：

model = Sequential()  
model.add(Conv2D(32, (3, 3), input\_shape=input\_shape))  
model.add(Activation('relu'))  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
  
model.add(Conv2D(32, (3, 3)))  
model.add(Activation('relu'))  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
  
model.add(Conv2D(64, (3, 3)))  
model.add(Activation('relu'))  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
  
model.add(Flatten())  
#model.add(Dense(64))  
model.add(Activation('relu'))  
#model.add(Dropout(0.5))  
model.add(Dense(10))  
model.add(Activation('softmax'))

VGG变形网络：

#fix  
input\_tensor = Input(shape=(150,150,3))  
model = Sequential()  
# build the VGG16 network  
base\_model = applications.VGG16(weights='imagenet', include\_top=False, input\_tensor = input\_tensor)  
print('Model loaded.')  
model.add(base\_model)  
# build a classifier model to put on top of the convolutional model  
top\_model = Sequential()  
top\_model.add(Flatten(input\_shape=base\_model.output\_shape[1:]))  
#top\_model.add(Dense(256, activation='relu'))  
#top\_model.add(Dropout(0.5))  
top\_model.add(Dense(3, activation='softmax'))  
model.add(top\_model)  
model.layers[0].trainable = True  
print (model.summary())

设置优化器：

model.compile(loss='categorical\_crossentropy',  
 optimizer=optimizers.SGD(lr=1e-4, momentum=0.9),  
 #optimizer=optimizers.Adam(),  
 metrics=['accuracy'])

设置迭代：

model.fit\_generator(  
 train\_generator,  
 steps\_per\_epoch=nb\_train\_samples // batch\_size,  
 epochs=epochs,  
 validation\_data=validation\_generator,  
 validation\_steps=nb\_validation\_samples // batch\_size)  
model.save\_weights(weights\_path)

3.2.2特征库形成

分类器网络（使用面向对象方式）

特征提取网络

# 直接使用VGG16提取出的特征  
class Myfeature:  
 def \_\_init\_\_(self,classname):  
 self.input\_shape = (150, 150, 3)  
 self.classname = classname  
 self.weight = rootdir+'VGG'+classname+'.h5'  
 self.pooling = 'max'  
 self.model = VGG16(weights = self.weight, input\_shape = (self.input\_shape[0], self.input\_shape[1], self.input\_shape[2]), pooling = self.pooling, include\_top = False)  
 self.model.predict(np.zeros((1, 150, 150 , 3)))  
 print('load\_done: model\_feats\_'+classname)  
   
 def extract\_feat(self, img\_path):  
 img = image.load\_img(img\_path, target\_size=(self.input\_shape[0], self.input\_shape[1]))  
 img = image.img\_to\_array(img)  
 img = img / 255  
 img = np.array(img)  
 img = np.expand\_dims(img, axis=0)  
 #img = preprocess\_input(img)  
 feat = self.model.predict(img)  
 norm\_feat = feat[0]/LA.norm(feat[0])  
 return norm\_feat

特征提取过程：

for i, img\_path in enumerate(img\_list):  
 img\_name = os.path.split(img\_path)[1]  
  
 for j in range(all\_class):  
 norm\_classifier = model\_Classifiers[j].extract\_feat(img\_path)  
 norm\_feats = model\_feats[j].extract\_feat(img\_path)  
 norm\_sim = model\_sim[j].extract\_feat(img\_path)  
 classicifier\_result[j].append(norm\_classifier)  
 feats[j].append(norm\_feats)  
 sims[j].append(norm\_sim)  
  
 names.append(img\_name.encode())  
 print ("extracting feature from image No. %d , %d images in total" %((i+1), len(img\_list)))

特征写入过程：

h5f = h5py.File(output, 'w')  
h5f.create\_dataset('dataset\_0', data = names)  
for i in range(all\_class):  
 sims\_dataset = 'dataset\_' + str(i + 1) + '\_sims'  
 h5f.create\_dataset(sims\_dataset, data=sims[i])  
 feats\_dataset = 'dataset\_'+str(i+1)+'\_feats'  
 h5f.create\_dataset(feats\_dataset,data = feats[i])  
 classi\_dataset = 'dataset\_'+str(i+1)+'\_classi'  
 h5f.create\_dataset(classi\_dataset,data = classicifier\_result[i])  
h5f.close()

3.2.3 在线查询

输入图片：

A目标识别

path = testdir + x + '.jpg'  
path = Cut.cut(path)  
if path == 'Fail to cut':  
 continue  
image = mpimg.imread(path)  
plt.imshow(image)  
plt.show()  
query(path)

B 使用余弦相似度

for i in range(all\_class):  
 if i == 0:  
 scores\_feats = np.dot(queryVec\_feats[i],feats[i].T) \* weight[i]  
 scores\_classi = cal\_cos(classifier\_result[i], queryVec\_classi[i])\*weight[i]  
 scores\_sims = cal\_cos(sims[i],queryVec\_sims[i])\*weight[i]  
 else:  
 scores\_feats += np.dot(queryVec\_feats[i], feats[i].T) \* weight[i]  
 scores\_classi += cal\_cos(classifier\_result[i], queryVec\_classi[i]) \* weight[i]  
 scores\_sims += cal\_cos(sims[i], queryVec\_sims[i]) \* weight[i]

C 分数排序

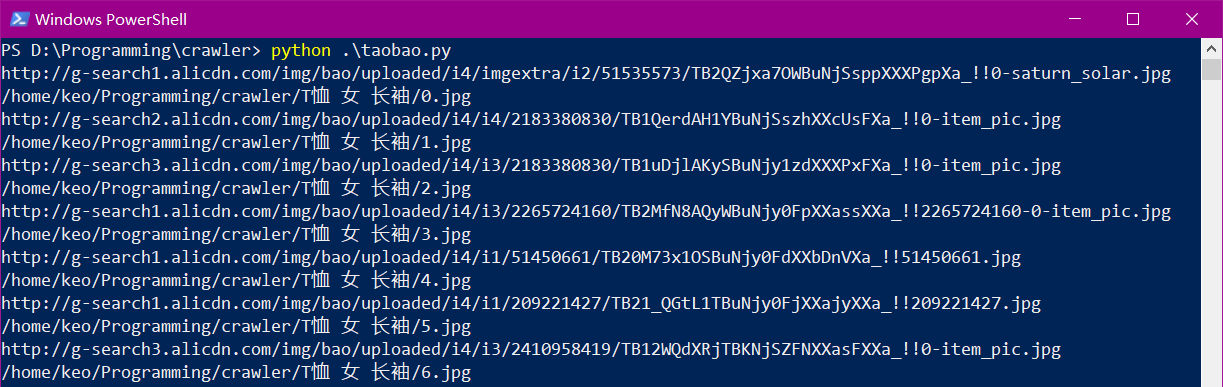
rank\_ID\_classi = np.argsort(scores\_classi)[::-1]  
rank\_ID\_feats = np.argsort(scores\_feats)[::-1]  
rank\_ID\_sims = np.argsort(scores\_sims)[::-1]  
rank\_ID\_all = np.argsort(scores\_all)[::-1]

D 结果写入

for i, im in enumerate(imlist\_all):  
 im = im.decode()  
  
 srcpath = args["result"] + "/" + str(im)  
 fpath, fname = os.path.split(queryDir)  
 fname = fname.split('.')  
 desdir = args["copydir"] + "/"+fname[0]+"/all"  
 despath = desdir+'/'+str(im)  
 mycopyfile(srcpath,despath)  
  
  
 image = mpimg.imread(args["result"] + "/" + str(im))  
 plt.title("search output %d" % (i + 1))  
 plt.imshow(image)  
 plt.show()

四**、**结果分析

1. 爬虫结果：

图14 运行脚本

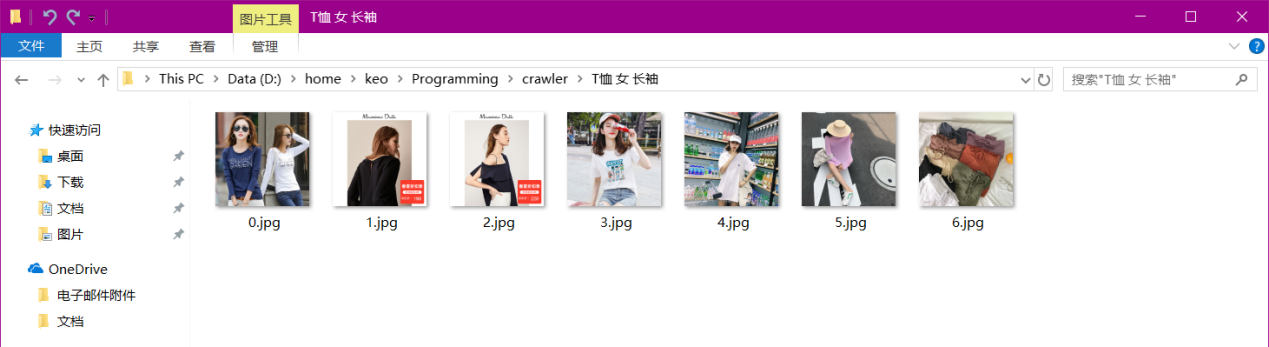
可以正常获取数据并保存到本地

图15 图片爬取

检查本地目录，图片已被保存。

2. 目标识别：





图16 目标识别结果

1. 以图搜图：

搜索图片：

搜索结果：

搜索图片：

搜索结果：



4. 以文字搜图：

输入特征：

童装，白色，印花，短袖

输出结果：



五**、**课程设计总结

陈浩鑫：这是一次富有趣味的课程设计。

最开始，感谢谷歌云的免费深度学习平台的实验项目，使得我们的模型得以顺利地训练。GPU资源很是重要。

首先回顾在本次课程设计中的收获：更多是一种对深度学习的实践。

从实践角度上，这次的课程设计，让我接触了数据集获取，数据集输入，模型的搭建，模型训练，模型预测，结果可视化，特征抽取等算法实现流程。

数据集获取：目标识别部分是通过自己打标签获得。

数据集输入：尝试过使用图片库函数，读取样本-标签后，将样本路径的图片读取并转化为数组；最后使用迭代器生成。

模型的搭建：在不同平台搭建的方式是不同，keras上使用序列化的搭建，依次堆叠网络层，加入损失函数后，自动求导，后向传播。

模型训练，batch训练能加快训练速度，且有效地防止结果到较差的局部最优。Epochs是训练周期，对于大数据下，一个epochs可能需要几个小时，所以训练时，为了防止后续出错。需要用小数据进行训练，再正式训练。

模型预测，将模型的权重导出之后便可以随时加载，用于预测。

结果可视化，使用中间层输出和结果展示，可以显示中间层的激活图与结果的直方图。

特征抽取，将网络中的任意层抽取出输出，可以实现对图片的特征描述。

最后，深度学习的易于上手，使得对于现实的一些图像处理问题都能简单去实践。

不足之处：也是使得这次课设遗憾的是，即使在编写程序时使用了规范化的面向对象开发思想。模型开发使用类封装。但没有去实践MVC等架构，不过这也是算法设计与前后端开发的区别吧。

洪浩强：

对我而言，本次软工大作业是一个将课本上的知识付诸于实现的过程。在这次课程设计中，我学会了寻找开源项目的支持，了解了主流的目标检测技术的区别并将其用于自己的项目中。这次课程设计我感受到了python平台科学库的强大与作为解释性语言的优势，以及GPU在机器学习中作为强大的计算能力支撑的重要性。

张朝晖：

总的来说对淘宝和唯品会的数据进行抓取的爬虫实现得比较顺利，两家网站搜索url的构成都比较友好，分页的逻辑也很好理解，网站对于我们这种量级的数据抓取没有加以限制，使我们得以顺利从两家网站获取对应搜索关键词的商品图片，从而利用这些数据进行下一步的研究和功能实现。

参考文献