目 录

[基于卷积神经网络的交通信号灯识别方法研究与实现 i](#_Toc515233404)

[第一章 概述 1](#_Toc515233405)

[1.1 研究背景和意义 1](#_Toc515233406)

[1.2 国内外研究现状 1](#_Toc515233407)

[1.3 软件环境介绍 2](#_Toc515233408)

[第二章 理论基础 3](#_Toc515233409)

[2.1 数学基础 3](#_Toc515233410)

[2.2 环境基础 4](#_Toc515233411)

[2.3 数据基础 5](#_Toc515233412)

[第三章 图像处理 6](#_Toc515233413)

[3.1 视频切割 6](#_Toc515233414)

[3.2 边缘处理 7](#_Toc515233415)

[3.3 图片缩放 8](#_Toc515233416)

[3.4 图片分割 8](#_Toc515233417)

[3.5 图片筛选 9](#_Toc515233418)

[3.5.1 复杂度计算 9](#_Toc515233419)

[3.5.2 模糊度计算 13](#_Toc515233420)

[3.5.3 相似性度量 15](#_Toc515233421)

[3.5.4 图片筛选程序 18](#_Toc515233422)

[第四章 图像识别 20](#_Toc515233423)

[4.1 数据集建立 20](#_Toc515233424)

[4.2 定义变量和函数 21](#_Toc515233425)

[4.3 定义模型 22](#_Toc515233426)

[4.4 运行和评估模型 26](#_Toc515233427)

[4.5 使用模型 27](#_Toc515233428)

[第五章 总结 28](#_Toc515233429)

[**附 录** 32](#_Toc515233430)

**基于卷积神经网络的交通信号灯识别方法研究与实现**

**摘 要：**信号灯识别系统可以结合导航进行实时路况分析，避免司机误闯红灯等情况，也可以结合无人驾驶技术实现无人驾驶汽车。无人驾驶汽车集自动控制、体系结构、人工智能、视觉计算等众多技术于一体，是计算机科学、模式识别和智能控制技术高度发展的产物。在未来，随着无人驾驶技术的发展，信号灯识别系统的应用也将越来越广泛。

传统的信号灯识别算法一般都是从交通信号灯的颜色特征和形状特征出发,采用基于图像处理的方法,通常对图像的质量要求非常高,不适合各种自然环境下的交通信号灯识别。基于此,本文采用基于机器学习的方法,更加能够满足各种自然环境下的交通信号灯识别。

本文介绍了在Windows10系统下使用机器学习的方式开发一个信号灯识别软件。在数据集的前期准备中，采用了OpenCV进行图像的预处理，包括视频和图片的切割，图片的筛选等。数据集建立后，采用TensorLayer搭建一个简单的卷积神经网络进行图像识别的训练，得到的训练数据用于后续的图像识别。

**关键词：**信号灯识别；卷积神经网络；图像处理；

Research and implementation of traffic signal

recognition method based on convolution neural network

**Abstract:** The signal recognition system can combine navigation to conduct real-time road condition analysis, so as to avoid the driver running the red light by mistake and other situations. Driverless cars set of automatic control, architecture, artificial intelligence, computer vision, and many other technology at an organic whole, is computer science, pattern recognition and intelligent control of the development of technology products. In the future, with the development of unmanned technology, the application of signal recognition system will be more and more extensive.

Traditional signal recognition algorithm is generally from the color feature and shape feature of traffic lights, adopts the method based on image processing, usually for image quality requirement is very high, not suitable for the identification of traffic lights under all kinds of natural environment. Based on this, the method based on machine learning is adopted in this paper to better meet traffic lights recognition in various natural environments.

This paper introduces how to develop a signal recognition software using machine learning in Windows10 system. In the preliminary preparation of data set, OpenCV was used for image preprocessing, including video, image cutting, image screening, etc. After the establishment of the data set, a simple convolution neural network was constructed using TensorLayer for image recognition training, and the training data obtained were used for subsequent image recognition.

**Key words:** Traffic Lights Recognition; Convolution Neural Network; Image Processing

# 概述

## 研究背景和意义

无人驾驶汽车经由智能传感体系感知路况信息，依托计算机体系规划行车线路，完成预定的目标。

根据目前的标准，无人驾驶汽车可以分成以下几个阶段。第—个阶段是指能完成特定功能的自动驾驶，这时汽车可以实现—些功能的自动化，如自动泊车等操作；第二个阶段是在第一个阶段的基础上进行组合；第三个阶段是受控的自动驾驶，这个阶段还需要人的干预，在限定环境下可以实现无人驾驶；第四个阶段是真正的无人驾驶，这个阶段汽车可以完全实现完全无人驾驶。

目前无人车已经来到了第三个阶段，未来即将实现真正的无人驾驶。信号灯识别系统，便是自动驾驶的一个主要的子模块。无人驾驶汽车通过信号灯识别系统辨识路上的信号灯，实现实时的规划路线。

## 国内外研究现状

在2018年，无人驾驶已构成以高级驾驶辅助系统（ADAS）和人工智能为主的两个发展方向。

高级驾驶辅助系统是指通过提高汽车的自动化和智能化程度来完善无人驾驶技术的发展，从事这一方向的主要是各种主机厂。主机厂偏向传统方向，依靠人工的算法实现各类控制系统，其车辆制造经验和服务体系也是技术重要的支撑。这项技术的核心是自动控制系统。车辆的行驶过程的决策，以人的判断为主，机器的策略为辅。

人工智能技术的是经过神经网络学习和一定的计算机决策能力来实现无人驾驶汽车的行驶。这项技术的目标是发展计算机来控制汽车，从而彻底取代人工驾驶，从事这一方向的主要是各类互联网厂商。各种互联网的公司在计算机技术上有很深的研究，能够深入发展人工智能计算和无人驾驶技术。这项技术通过计算机对行驶路况进行判断，实现自动化驾驶，减少人为的干扰来避免事故的发生。

相似的，信号灯识别也有2个方向。传统方式是使用对图像的处理和分析进行形状和颜色分割，通过模式匹配等方法来识别交通灯，而机器学习是建立一个卷积神经网络来识别交通灯。从目前的发展来看，机器学习拥有自学习的能力，能识别更加复杂的场景，是未来的发展方向。

## 软件环境介绍

**OpenCV**

OpenCV是计算机视觉库，提供了多种语言的接口。OpenCV实现了许多图像处理方面的算法[21]。

**TensorFlow**

TensorFlow是谷歌推出的人工智能学习系统，可被用于图像识别等多项机器学习领域[14]。

**Tensorlayer**

TensorLayer 是一款基于TensorFlow开发的深度学习库，它提供高级别的深度学习API，可以加快研究人员的实验速度，减少工程师在实际开发当中的重复工作[15]。

# 理论基础

## 数学基础

**卷积**

设,是R1上的两个可积函数，作积分：

函数的卷积，即。

若变量是两个序列和，则卷积可以写成：

**神经网络**

神经网络是一种运算模型，由大量的节点之间相互联接构成。每个节点代表一种特定的输出函数，称为激励函数。每两个节点间的连接都代表一个对于通过该连接信号的加权值，称之为权重，这相当于人工神经网络的记忆。网络的输出则依网络的连接方式，权重值和激励函数的不同而不同。而网络自身通常都是对自然界某种算法或者函数的逼近，也可能是对一种逻辑策略的表达。

**Sobel算子**

这个算子通过对图片求一阶导数，从而寻找图片的边缘。Sobel算子对图片的水平和垂直方向求导，从而求出梯度值。

水平方向的卷积为：

垂直方向的卷积为：

结合水平和垂直方向的结果可以求出近似梯度：

**Laplacian算子**

这个算子基于Sobel算子，区别在于对图片求的是二阶导数。

**DCT变换**

DCT(离散余弦变换)。在图像处理的应用中，可以获取图像中的低频分量，这在图片压缩中有很重要的作用。DCT变换的公式为：

其中是图像中的像素点。

## 环境基础

**硬件要求**

CPU： AMD A8-7650K Radeon R7

GPU0： AMD Radeon R7 Graphics

GPU1： Nvidia GeForce GTX 650

主板： 华硕 A88X-PLUS

内存： 16.0GB

**软件要求**

操作系统： Microsoft Windows 10

软件用到的Python库：

opencv-python;numpy;TensorFlow-GPU;TensorLayer

## 数据基础

数据的采集可以有很多方式，比如公开的数据集，网上的视频图片，也可以自己制作。自己制作数据集所用到的图片，可以从行车记录仪的视频或者其他视频来源中捕获其中关于红绿灯的图片，也可以从网上下载关于红绿灯的图片，或者自己拍摄关于红绿灯的图片和视频。

视频从网上下载有很多方式，可以用爬虫，或者是一些插件，下载视频时，可以用红绿灯、交通、行车记录仪等关键字搜索相关的视频。

# 图像处理

获取图片后，需要对图像进行处理:

采集到的视频需要切割成图片

网上获取的图片有太多额外的信息，需要进行图片切割去除干扰信息。

由于采取的神经网络大小固定，因此不同大小的图片要事先统一缩放才能用于训练。

为了样本的多样性，可以对大图片进行切割，平移，旋转等操作。

样本切割会导致图片只有天空，黑暗等简单信息，因此需要对图片复杂度进行分析，去除复杂度低的照片。

夜间和自己拍的图片容易模糊，形成大量的噪点，对图像识别不利，因此需要分析图片的模糊程度，去除模糊的照片。

视频采集的图片或者图片分割后的图片容易相似，为了控制样本的大小，提升样本的丰富性，需要对图片间的相似度进行分析，去除过于相似的图片。

## 视频切割

视频的切割是指将视频分割成相同大小的图片，目录如下：

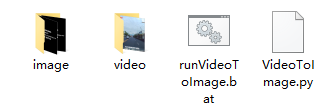


图3.1 视频切割文件夹

其中video文件夹存放要处理的视频，image文件夹存放处理后的图片，VideoToImage是视频转图片的程序，runVideoToImage是快速执行的批处理。

大体的流程是读取所有的视频，获取视频的帧数。假设帧数是20，那就按照每20张存储一次的顺序保存图片，即一秒一张，图片命名为“视频名\_图片编号.jpg”。

之后需要进行一些人工的筛选，删除广告等不合适的图片，然后进行其他的处理。

|  |  |
| --- | --- |
|  |  |

图3.2 视频中的广告

## 边缘处理

网上的视频往往带有水印，字幕等额外内容，这些内容会对图像分析形成干扰，因此需要事先去除这部分内容。



图3.3 原始图片

|  |  |
| --- | --- |
|  |  |
|  |  |

图3.4 图片中的额外内容

我采取的方法是对去除边缘部分，边缘的切割需要设定图片4个边的切割宽度，这里，我采用了两种方案，一种是按比例切割，另一种是按像素切割。

|  |  |
| --- | --- |
|  |  |
| （a） | （b） |

图3.5 图片像素切割（a），图片比例切割（b）

图片像素切割和比例切割将image1中的图片按要求切割，存在image2文件夹中，文件名与原图片相同。图片像素切割是按造指定像素切割图片，如200\*200的切割，使得图片切割后长和宽是200\*200。图片比例切割用一定比例切割去切割照片，比如说20%的比例是上下左右各取10%的总长度进行切割，切割后的图片大小是原始图片大小的0.64倍。

## 图片缩放



图3.6 图片缩放文件夹

图片缩放将image1中的图片缩放到指定大小放入image2文件夹中，文件名与原图片相同，一般用于将大图片缩小到指定尺寸。2个runImageResize快速选择缩放的大小，一般用256\*256的缩放。图片缩放使用cv2的resize函数，并广泛运用于其他图片处理的过程中。

## 图片分割

图片分割可以将大图片切割成多个小图片，增加图片的丰富程度。

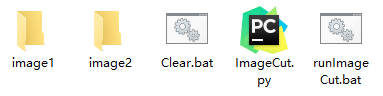


图3.7 图片分割文件夹

假设分割一张(宽\*高)的图片，分割后图片的大小为256256。将图片存入 的矩阵

第一轮，切割大小为256256，每张图片边缘坐标组成的矩阵为：

[(0,0),(0,256),(0,512),(256,0),(256,256),(256,512)]

第二轮，切割大小为512212，每张图片边缘坐标组成的矩阵为：

[(0,0)]

两轮一共切割7张图片，对每张图片缩放并输出即可。

## 图片筛选

通过图像采集，图片分割等操作后可以获得大量的图片，对这些图片做成数据集是很困难的一件事，因此需要对图片进行筛选去掉不太合适的图片。这里，我对图片的复杂度，模糊度和相似度进行了筛选，分别用到了图片熵算法，拉普拉斯算法和感知哈希算法。

### 复杂度计算

判别一个图像或者图像切割之后的结果的优劣，可以通过图片熵算法来计算图片的复杂度，通常情况下图片复杂度是评价图片的一个重要指标。

图片的复杂度代表了一张图片中传达出的信息量，如果一张图片是纯黑（夜晚）或者是纯白（天空）的，那么这张图片的复杂度就很低，信息量接近于0。

|  |  |
| --- | --- |
|  |  |
| （a） | （b） |

图3.8 一张图片切割后生成的两张图片

上图所示，图片（a）有大片的天空，图片（b）具有很多复杂的场景，因此图片（a）的信息量很明显不如图片（b）。计算得知，其中图片（a）的图片熵为1点多，图片（b）的图片熵为7点多。在实际应用中，删除图片熵小于4的图片，便可以去除大量的图片。

**信息熵**

图片熵的计算基于信息熵公式，信息熵公式定义如下：

其中P(X)为X的信息量，取值为(0,1]。式中对数一般取2为底，但也可以取其它对数底。

信息熵有以下几个特点：

1. 信息熵不为负，信息熵永远是大于0的。
2. 信息熵可以叠加，假如一句话的信息熵为2，另一句话的信息熵为10，当两句话叠加时，信息熵肯定为增大。
3. 信息熵是连续的，单独的信息变化会引起整体信息熵的变化，在信息变化较大时，信息熵的变化也会较大。
4. 信息熵还跟信息的数量有关。假如两个信息的概率相同，其中一个信息更多，那么信息熵就越大。

下面分析这条公式，因为P(X)的取值为(0,1],所以恒为正，满足条件1。

因为，所以2个信息熵可以叠加，满足条件2。

因为连续依赖于x，满足条件3。

假设有n种可能性，那么出现任意可能的p是，因为是一个增函数，满足条件4。

**一元灰度熵**

从信息熵公式，我们可以推出图片一元灰度熵的公式，一元灰度熵是图片信息量和灰度分布特征的体现。

设尺寸为M\*N的图像，灰度取值【0,255】,对应的p可表示为 ,由此可以得出灰度类,从而可以推出公式，一元灰度熵的公式为：

其中表示灰度值等于i的像素在图片中的比例，i的范围为【0,255】,共256个灰度等级，i出现的频数。

**二元灰度熵**

普通的一元灰度熵虽然可以体现出图片像素的聚集特征，却不能反映图片的空间特征。为了表示这种空间特征，可以使用二元灰度熵。二元灰度熵即二维熵，在一元灰度熵的基础上引入了能够反映空间特征的量。

选择图片的像素灰度值和相邻像素的灰度均值组成一个特征二元组，记为(i,j)，二元灰度熵的公式如下：

其中是图片在像素位置上空间分布的特征，是二元组出现的次数，N是图片的大小，如果长不等于宽也可以是，H即为图片的二维熵。

**JPEG图像压缩**

JPEG格式的图像是做常用的图像格式，采用了JPEG图像压缩算法。比起无压缩的图片格式，JPEG的图片会很小，并且在图片信息量很小时，图片的大小会更小。因此，我们可以比较两张相同尺寸的JPEG图片的大小来衡量两张图片的复杂度。

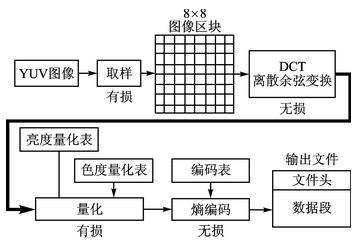


图3.9 JPEG压缩流程

下面，选择多张256\*256JPEG格式的图片，统计图片的图片熵跟图片大小，得出下图，可以看出图片熵跟图片大小大致呈正比关系。

图3.10 图片大小跟图片熵之间的关系

下面选择两张大小相差很大，而图片熵差不多的图片，其中左边图片的图片熵为7.360，大小为19.40KB，右边图片的图片熵为7.361，大小为35.33KB。

|  |  |
| --- | --- |
|  |  |

图3.11 两张图片熵相近的图片

从直观上来看右边的信息量更大，个人感觉尺寸相同的图片，用图片大小来比较图片的复杂度更加方便。不过如果是不同尺寸的图片，或者不保存只在程序内比较的图片，还是要用图片熵进行比较。

### 模糊度计算

在图像处理的过程中，图像的模糊度是衡量图像质量好坏的重要指标。其算法有很多，下面对目前几种较为常用的、具有代表性模糊度算法进行分析。

**Brenner函数**

Brenner函数通过计算相邻两个像素灰度差的平方来评价图片的模糊程度，该函数的公式为：

其中 表示图像对应像素点(m,n)的灰度值。

**Tenengrad函数**

Tenengrad 函数采用Sobel算子来计算梯度值，该函数的公式为：

其中和分别是像素点(m,n)处Sobel算子在水平和垂直方向边的卷积，T是边缘检测阈值。

**Laplacian函数**

Laplacian函数与Tenengrad梯度函数基本一致，用Laplacian算子替代Sobel算子即可。将图片用拉普拉斯算子做卷积运算，得到的矩阵计算方差，结果可以衡量图片的模糊程度。

在程序用到了cv2中的拉普拉斯函数，其定义是：



其中ksize是高斯内核的大小，取值可以是[1,3,5,7]。默认为1。

为了选择ksize，xiamian1对一张图片进行拉普拉斯变换

|  |  |
| --- | --- |
|  |  |
| 原始图片 | ksize=1 |
|  |  |
| ksize=3 | ksize=5 |
|  | |
| ksize=7 | |

图3.12 不同ksize的图片

从上面可以看出当取值为3或5是，边缘比较清晰。计算图片模糊度，可以调用cv2中的Laplacian函数的var函数来衡量图片的模糊度。不同的ksize，计算出的模糊度是不同的，以上图为例：ksize=1时，var= 314.09；ksize=3时，var= 3515.08；ksize=5时，var= 278203.71；ksize=7时，var= 36214609.37，实际我选择ksize为3来计算模糊度。

### 相似性度量

图片分类和大量图片筛选时往往需要估算不同图片是否相似，去掉过于相似的图片。图片相似性度量通常采用的方法就是计算样本的特征，根据两个样本特征间的距离来评估图片的相似程度。

计算样本的特征一般用到以下几种算法：

**SIFT,SURF,FAST**

SIFT, SURF, FAST这类算法是检测图片局部特征的算法，最大的特点是具有尺度不变性。

其中SIFT通过求图片中的特征点及其有关缩放和旋转的因子得到图片的特征，通过图片的特征点来计算图片的相似度。

SURF对SIFT进行改进。论文《A comparison of SIFT, PCA-SIFT and SURF》对这两者进行比较，发现SIFT在尺度和旋转变换的情况下效果最好，SURF在亮度变化下匹配效果最好，在模糊方面优于SIFT，而尺度和旋转的变化不及SIFT，旋转不变上比SIFT差很多，从速度上来看，SURF差不多是SIFT的3倍[22]。

以上两个算法都能在改变图片的旋转角度，亮度或拍摄视角的情况下获得较好的检测效果。

FAST是一种角点检测算法。是3个算法中最快的，但没有旋转不变性并且健壮性并不好，噪点很多时会产生很多的错误特征点。

**哈希算法**

上述算法通过图片特征点等信息，提取出图片的特征空间，比较2个图像在特征空间中的欧式距离，可以计算图片的相似程度。但是，当图像数量非常庞大时，这类算法的缺点也很明显，需要的存储空间庞大，而且检索的速度较慢。

为了解决上述问题，哈希算法是一个不错的选择。这类算法在对时间和空间需求较低的情况下，还能够得到不错的结果，因此成为了一种不错的选择。

**aHash和pHash**

平均哈希算法(aHash)，通过比较图片的像素和平均值来实现的。大致步骤如下：

1. 将图片缩放成8\*8像素的图片并转成灰度图
2. 计算图片灰度的平均值
3. 将每个像素与平均值比较，大于平均值的为1，小于或等于平均值的为0，统计这些结果，得到64位的hash码
4. 计算汉明距离比较2张图片的hash码，通常认为汉明距离大于10的两张图片是完全不同的

aHash算法有一个缺点，那就是不够精确，更适合比较缩略图。为了更精确的比较图片的相似程度，可以选择pHash算法，算法的大致步骤如下：

将图片缩放成3232像素的图片并转成灰度图

1. 计算图片DCT矩阵，并保留左上角8\*8的矩阵
2. 计算缩小DCT后的所有像素点的平均值
3. 将每个像素与平均值比较，大于平均值的为1，小于或等于平均值的为0，得到64个比特位的hash码
4. 计算汉明距离比较2张图片的hash码，通常认为汉明距离大于10的两张图片是完全不同的

由下图可以看出，DCT矩阵的低频分量在左上角，在图像处理的过程中，可以通过提取左上角8\*8的矩阵，来反映出图像特征情况。

|  |  |
| --- | --- |
|  |  |
|  |  |

图3.13 左上：8\*8像素矩阵 右上：像素灰度矩阵

左下： DCT系数矩阵 右下：系数的排列方式

因为人眼对图像的亮度信息和低频分量具有较高的灵敏度，对图像的颜色信息不够敏感，而DCT矩阵左上角和右下角的数值分布代表了图像和主体部分和细节部分，所以进行DCT通常用来提取左上角的部分。

在获得样本的特征后，需要计算样本间的“距离”，常用的计算方法有以下几种：

**欧几里得距离**

欧几里得距离是经常用到的距离计算公式，对于平面上的两个点,，欧氏距离的公式如下：

公式还可以表示成向量运算的形式，即：

**汉明距离**

汉明距离表示两个相同长度的字符串，两个字符串中不相同的对应位的总和。公式为：

对于2个64位的hash码，对hash码相减得出2个hash码不同的位，使用cv2中的countNonZero函数即可统计不同位的和。

### 图片筛选程序

实际使用中，图片筛选程序的文件结构如下：

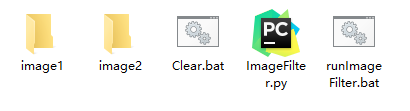


图3.14 图片筛选程序文件结构

其中程序运行的大致流程如下：

* + - 1. 读取image1中的一张图片，转成灰度图
      2. 计算图片的复杂度，模糊度和pHash值
      3. 将图片pHash值与Hash值列表中的所有pHash值计算汉明距离比较相似度，并将pHash值存入Hash值列表中
      4. 如果图片在复杂度，模糊度和相似度上有不符合要求的，则删除这张图片

下面，选择400张照片，经过图片分割的处理之后，可以获得22916张图片。这些靠人工筛的话还是有点艰难的，如果原始图片量上万的话，分割出的图片可能过百万了。经过图片筛选后，得到4332张图片，删除了81.1%的图片。筛选速度大概一秒钟10张图片，越往后图片相似性的计算量越大，速度越慢。

在图片筛选中，主要还是靠复杂度和模糊度算法筛选图片，相似度的pHash算法在相似度比较上是不如SIFT之类的算法的，不过这么大的数据量还是pHash快一些。

# 图像识别

图像识别方式采用机器学习的方式，搭建一个简单的卷积神经网络（CNN）来识别交通信号灯。

## 数据集建立

数据集分为训练集，验证集和测试集。其中验证集是为了在训练过程中方便看到结果而设置的，一般可以不用。测试集是模型训练结束后用于验证模型性能的。对于模型训练来说，只有训练集是必须的。一般而言，训练集，验证集和测试集的分配比例是8:1:1。

一般来说,数据集的制作是将数据和标签封装成tfrecords文件，在实际使用中再读取。不过为了方便起见，这里不对数据集进行制作，而是直接读取图片来训练。具体操作如下：

1. 将图片进行分类，分成[无红绿灯,有绿灯，有红灯，有红灯和绿灯]四组，文件夹命名为[None,Green,Red,Both]。
2. 对所有文件夹内的图片进行图片分割、缩放和筛选。对[Green,Red,Both]中的文件进行人工筛选，将其中有红绿灯的图片放在原文件夹中，其他图片放到None文件夹中。
3. 一般来说，None文件夹中的文件会远多于其他文件夹中的文件，这时除了简单筛选以外还可以对照片进行随机丢弃。
4. 最后时，控制4个文件夹中图片的比例，不要相差太大。

实际文件夹如下图所示：

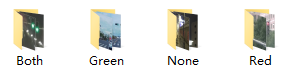


图4.1 数据集文件夹

本次训练共使用645张图片，其中红绿灯都有的101张，绿灯60张，红灯114张，其他图片370张。

## 定义变量和函数

**Session**

Session在TensorFlow里是一个会话，可以管理程序运行时的全部资源。

**数据集**

机器学习用到的数据集分为为训练集和测试集。并具有相应数据中的数据部分和标签部分。数据部分在图像识别中就是像素矩阵;标签部分是一个int型的数组。

**占位符**

占位符(placeholder)表示数据和标签的数据格式，为了统一起见，所有数据的格式都是一致的，比如说都是256\*256的图片，有3个颜色通道。

**激活函数**

激活函数决定了给定输入节点的输出，非线性激活函数允许网络复制复杂的非线性行为。激活函数有很多种，最基础的激活函数有ReLU、identity和sigmoid，基于这三个函数有很多其他的变体，下面对用到的函数进行介绍：

ReLU是最常用的激活函数,公式如下：

函数在时，不为0，从而可以用于基于梯度的机器学习。因为函数和导数都不复杂，所以可以很快的计算，缺点是时，函数会变得很慢，可能会使神经元直接失效。

identity函数可以说是最简单的激活函数了，函数的输入等于输出，适合用于线性的任务，因此该函数可以用于最终输出节点上，缺点是不适合用于非线性的任务。

**损失函数**

损失函数(cost)用来衡量网络的权重，机器学习的目标是让损失函数尽可能的小。这里采用交叉熵（cross entropy）来衡量，调用tl中的entropy函数来实现。值得注意的是，由于输出层用到的激活函数是identity，因此softmax函数在该函数中实现。

**分类准确度**

分类准确度 (Accuracy)体现了分类器能否对样本进行正确的分类，设，分别是样本的预测和真实结果，则n个样本上的分类准确度为：

这里，我们使用argmax函数对预测值和真实值进行评估，最后将结果转成浮点数即可。

**优化器**

把机器学习的目地想象成对一个目标函数的求解，优化器(Optimization)便是一个求解目标函数中参数的重要手段。常见的优化器有RMSprop，SGD，Adadelta，Adagrad，Adam，这里采用Adam。

## 定义模型

神经网络包括输入输出层和隐藏层，在图像识别的神经网络中，隐藏层用到了腐蚀层，卷积层，池化层，扁平化层和全连接层。下面是论文中的神经网络结构：

表4.1 信号灯识别软件的网络结构

|  |  |  |
| --- | --- | --- |
| 类别 | 名称 | 参数 |
| 输入层 | input | shape:(?, 256, 256, 3) |
| 卷积层1 | cnn\_layer1\_1 | shape:[3, 3, 3, 8] |
| cnn\_layer1\_2 | shape:[3, 3, 8, 8] |
| pool\_layer1 | ksize:[1, 2, 2, 1] |
| drop\_layer1 | keep:0.8 |
| 卷积层2 | cnn\_layer2\_1 | shape:[3, 3, 8, 16] |
| cnn\_layer2\_2 | shape:[3, 3, 16, 16] |
| pool\_layer2 | ksize:[1, 2, 2, 1] |
| drop\_layer2 | keep:0.8 |
| 卷积层3 | cnn\_layer3\_1 | shape:[3, 3, 16, 32] |
| cnn\_layer3\_2 | shape:[3, 3, 32, 32] |
| pool\_layer3 | ksize:[1, 2, 2, 1] |
| drop\_layer3 | keep:0.8 |
| 卷积层4 | cnn\_layer4\_1 | shape:[3, 3, 32, 64] |
| cnn\_layer4\_2 | shape:[3, 3, 64, 64] |
| pool\_layer4 | ksize:[1, 2, 2, 1] |
| drop\_layer4 | keep:0.8 |
| 卷积层5 | cnn\_layer5\_1 | shape:[3, 3, 64, 64] |
| cnn\_layer5\_2 | shape:[3, 3, 64, 64] |
| pool\_layer5 | ksize:[1, 2, 2, 1] |
| drop\_layer5 | keep:0.8 |
| 全连接层 | flatten\_layer |  |
| dense\_layer1 | units:256 act:relu |
| dense\_layer1 | units:256 act:relu |
| 输出层 | output\_layer | units:4 act:identity |

下面，简单介绍各层的作用。

**InputLayer**

InputLayer是输入层，是神经网络的输入项。

**DropoutLayer**

DropoutLayer是腐蚀层，会随机抛弃一些网络参数，目标是为了避免网络的过拟合。

神经网络训练多次后，可能会出现对训练集拟合程度很好，但对验证集拟合程度很差的情况。腐蚀层可以让每一次迭代随机去更新网络中的参数，从而增加神经网络的泛化能力，提升网络的健壮性。



神经网络层 加入腐蚀层

图4.2 Dropout层效果

腐蚀层按照一定的概率来对网络参数进行随机采样，将这个子网络作为此次更新的目标网络。可以想象，如果整个网络有个参数，那么可用的子网络为 个。 所以，当很大时，每次迭代时使用的子网络基本上不会重复，从而避免神经网络的过拟合。

腐蚀层一般在训练时使用，在实际使用中舍弃。

**Conv2dLayer**

Conv2dLayer是卷积层，卷积层通过卷积来进行特征区分，通过卷积权值的共享及池化等方法，来降低网络参数的数量级，最后通过传统神经网络如全连接来完成分类等任务。其主要思想是模拟的是人对物体的认知过程，人的认知过程大体可以分成获取事物信息（眼睛观察），初步处理（双目视觉合成，分析事物的景深、边缘等），抽象（对事物进行判定，形状，特征等），进一步抽象（如根据人脸判断是哪个人，人的心情等）。

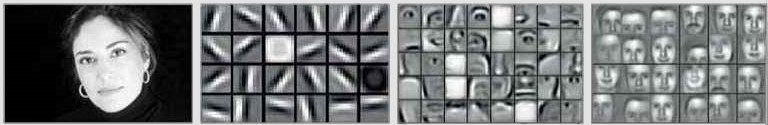


图4.3 卷积层识别过程

第三章用到的拉普拉斯函数也是一种卷积操作，这个函数同样可以提取出一个图片的特征，机器学习中卷积层的主要目标就是通过不同的卷积核获得不同的特征，通过训练不断地调整卷积核，从而获得较优解。

**PoolLayer**

PoolLayer是池化层，操作跟卷积层差不多，只不过池化层的卷积核只取一个值。

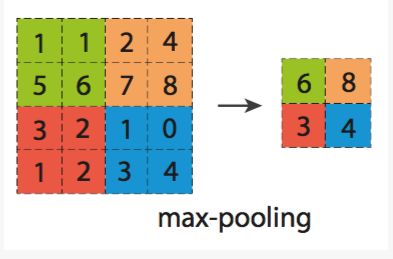


图4.4 最大池化层识别过程

上图表示对的图片采用最大池化的结果，池化层的卷积核的大小为。

池化层不进行反向传播的修改，其主要作用是降低数据量，因为卷积后图像还是很大的。同时，下采样减少了数据的维度，可以避免过拟合的发生。

**FlattenLayer**

FlattenLayer是平面化层，作用是把高维的数据展开成一维的数据，作为全连接层的输入。

**DenseLayer**

DenseLayer是全连接层，具有分类器的功能。全连接层的参数非常的多，因为全连接层的每一个节点都跟上一层所有的节点相连接。全连接层虽然可以用于提取特征，但是我们一般使用卷积层进行这项工作，在网络的最后使用全连接层进行分类。

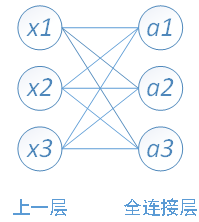


图4.5 全连接层

上图中x1、x2、x3是全连接层的输入，a1、a2、a3是全连接层的输出，W和b是全连接层的参数，因此有：

也可以写成如下矩阵形式：

跟卷积层相比，卷积层是将原始数据映射到隐层特征空间，而全连接层是将数据映射到样本的标记空间。

**OutputLayer**

在分类任务中，一般用全连接层作为输出层。当然，与一般的全连接层也有区别，在全连接层中，激活函数选择的是relu，输出层选择的是identity。

最后，对神经网络层做个大概的分类，卷积层和全连接层是用于提取特征和分类，这是神经网络的重点部分，反向传播也运用于其中。腐蚀层和平面化层算是辅助层，为整个神经网络提供各自的功能，不进行反向传播。

## 运行和评估模型

在简单的配置之后，就可以成功的运行模型了，训练时一共使用579张图片，每次训练20张，共训练150次，共用时2477s，最后的loss结果为0.0068，完整的记录详见附录，训练得到的loss曲线如下：

表4.2 训练时的loss曲线

选择65张图片用于测试，测试通过率93.3%。可以看出模型还是不太完善，就目前来看，有以下几个问题：

* + - 1. 图片数量还是太少了，图片数量太少易造成欠拟合
      2. 种类不丰富，只收集了一点点红绿灯的数据，没有复杂的多个的红绿灯图片，红绿灯种类也过于单一，导致模型无法很好的应用到实际中
      3. 夜晚图片过于模糊是很致命的，过多的噪点光晕连人都不一定能够分辨
      4. 缺乏与现实相结合，如果在路口附近再进行识别成功率会高很多

## 使用模型

下面，我选择4张图片来验证模型能否正常运行。

|  |  |
| --- | --- |
|  |  |
| 1.jpg | 2.jpg |
|  |  |
| 3.jpg | 4.jpg |

图4.6 四张不同的图片

上面四张图片分别是有红灯和绿灯，只有绿灯，没有红绿灯和只有红灯4种情况。下面运行程序，得到结果：

1.jpg : Both

2.jpg : Green

3.jpg : None

4.jpg : Red

与预期相符，证明程序是可也以识别交通信号灯的。

# 总结

这次的题目从17年12月份开始进行，前期主要是进行环境的配置。像tensorlayer，CUDA之类的安装并不容易，需要不断的去解决问题。最难解决的莫过于硬件问题了，windows系统自带的策略会阻止GPU过长的运行，这需要修改注册表和将显示换成另一个显卡（比如集显）来解决。要使用CUDA，对GPU的版本也有要求，我的电脑因为版本不够，就在家里组了台电脑，通过远程访问进行训练。

整个论文其实是以第三章图像处理为主，重点在于图片筛选那一块。一方面原因是做着做着发现图像处理方面内容非常丰富，可以写很多东西。另一方面是机器学习其实做的不好，深入研究比较困难，毕竟在缺乏专业指导的情况下想去研究如何优化还是很难的。在对图像处理的算法进行些研究之后，发现很多算法其实是相通的，比如复杂度计算中的JPEG算法和相似性度量中的pHash算法都用到了离散余弦变换

论文的过程虽然坎坷，但写完之后的成就感也是很强的。特别是完整的运用了图像处理和机器学习方面的知识，也算是开拓了自己的视野吧。

**参考文献**

1. Ron Patton.软件测试[M].北京：机械工业出版社，2006.
2. Dong Hellmann.Python标准库[M].北京：机器工业出版社，2012.
3. Mark Luts.Python编程[M].北京：中国电力出版社，2015.
4. Magnus Lie Hetland.Python基础教程（第2版•修订版）[M].北京：人民邮电出版社，2014.
5. Magnus Lie Hetland.Python算法教程[M].北京：人民邮电出版社，2016.
6. 张若愚著.Python科学计算[M].北京：清华大学出版社，2016.
7. Sebastian Raschka.Python机器学习[M].北京：机器工业出版社，2017.
8. Ethem Alpaydin.机器学习导论[M].北京：机械工业出版社，2009.
9. 周志华著.机器学习[M].北京：清华大学出版社，2016
10. Firooz Sadjadi, Bahram Javidi.自动目标识别的物理学原理[M].北京：国防工业出版社，2014.
11. Millan Sonka,Vaclav Hlavac,Roger Boyle.图像处理、分析与机器视觉[M].北京：清华大学出版社，2016.
12. 郑泽宇，顾思宇著.TensorFlow 实战Google深度学习框架[M].北京：电子工业出版社，2017.
13. Sam Abrahams,Daniiar Hefner,Enk Enwitt,Ariel Scerpinelli.面向机器智能的TensorFlow实践[M].北京：机械工业出版社，2017.
14. 王琛，胡振邦，高杰著.深度学习原理与TensorFlow实践[M].北京：电子工业出版社，2017.
15. 董豪，郭毅可，杨光.深度学习：一起玩转TensorLayer[M].北京：电子工业出版社，2017.
16. David G.Lowe Distinctive Image Features from Scale-Invariant Keypoints[J]. International Journal of Computer Vision,January 5, 2004.
17. David G.Lowe Object Recognition from Local Scale-Invariant Features[J]. Computer Science Department University of British Columbia, 1999.
18. Tomas Milokov et al. Distributed Representations of Words and Phrases and their Compositionality[J].NIPS, 2013.
19. Hinton G.E. et al.Dropout: a simple way to prevent neural networks from overfitting[J].Journal of Machine Learning Research, 2014.
20. Fei-Fei L. et al.Dropout: Large-Scale Video Classification with Convolutional Neural Networks[J]. IEEE Conference on Computer Vision and Pattern Recognition, 2014.
21. OpenCV 2.4.13.6 documentation[DB/OL] . Open Source Computer Vision Library, 2018.
22. Juan L.and Gwun,A Comparison of SIFT, PCA-SIFT and SURF[J].International Journal of Image Processing,2009

**谢 辞**

论文工作接近尾声，本科学习也到了最后的阶段。回首大学四年来的点点滴滴，我对大学的生活还是非常不舍的。在这半年来论文书写的过程中，我感受到了想写出一篇好的论文是非常困难的。在此，我向那些曾经帮助过我的老师和同学表示最衷心的感谢，感谢你们给予我的支持和关怀，使我能够全身心地投入到学习中。

**附 录**

**附录1 视频转图片的代码**

#!/usr/bin/env python

from \_\_future\_\_ import print\_function#使用括号

import cv2

import os

import sys

filepath = 'video/'

pathDir = os.listdir(filepath)

for allDir in pathDir:

b = 1

c = 1

timeF = 24

allDir = os.path.join(filepath, allDir)

(filepath, tempfilename) = os.path.split(allDir)

(shotname, extension) = os.path.splitext(tempfilename)

print('FileName=' + str(tempfilename), end='\t')

vc = cv2.VideoCapture(allDir)

if vc.isOpened():

rval, frame = vc.read()

timeF = int(vc.get(cv2.CAP\_PROP\_FPS))

else:

rval = False

print('Fps=' + str(timeF), end='\t')

while rval:

rval, frame = vc.read()

if (b % timeF == 0):

cv2.imwrite('image/' + shotname + '\_' + str(c) + '.jpg', frame)

c = c + 1

b = b + 1

cv2.waitKey(1)

print('ImgNum=' + str(c - 1))

vc.release()

**附录2 边缘处理（比例）的代码**

filepath = "image1/"

pathDir = os.listdir(filepath)

num = 0.0

if len(sys.argv) > 1:

num = float(sys.argv[1])

if num > 1.0:

num = num / 100

for allDir in pathDir:

allDir = os.path.join(filepath, allDir)

(filepath, tempfilename) = os.path.split(allDir)

(shotname, extension) = os.path.splitext(tempfilename)

img = cv2.imread(allDir)

width = len(img)

height = len(img[0])

x = int(width \* num \* 0.5)

y = int(height \* num \* 0.5)

xl = int(width \* (1 - num \* 0.5))

yl = int(height \* (1 - num \* 0.5))

img2 = img[x:xl, -yl:-y]

cv2.imwrite('image2/' + shotname + '.jpg', img2)

**附录3 边缘处理（像素）的代码**

filepath = "image1/"

pathDir = os.listdir(filepath)

num1 = 256

num2 = 256

if len(sys.argv) > 1:

num1 = int(sys.argv[1])

num2 = int(sys.argv[2])

for allDir in pathDir:

allDir = os.path.join(filepath, allDir)

(filepath, tempfilename) = os.path.split(allDir)

(shotname, extension) = os.path.splitext(tempfilename)

img = cv2.imread(allDir)

width = len(img)

height = len(img[0])

if num1 > width | num2 > height:

print('Error:num1>width||num2>height')

exit()

x = int((width - num1) \* 0.5)

y = int((height - num2) \* 0.5)

xl = num1 + x

yl = num2 + y

img2 = img[x:xl, -yl:-y]

cv2.imwrite('image2/' + shotname + '.jpg', img2)

**附录4 图片缩放的代码**

filepath = "image1/"

pathDir = os.listdir(filepath)

width = 32

height = 32

if len(sys.argv) > 1:

width = int(sys.argv[1])

height = int(sys.argv[2])

for allDir in pathDir:

allDir = os.path.join(filepath, allDir)

(filepath, tempfilename) = os.path.split(allDir)

(shotname, extension) = os.path.splitext(tempfilename)

img = cv2.imread(allDir)

img2 = cv2.resize(img, (width, height))

cv2.imwrite('image2/' + shotname + '.jpg', img2)

**附录5 图片分割的代码**

import os

import cv2

import sys

import math

movestep = 1

resizeshape = (256, 256)

finalnum1 = 0

finalnum2 = 0

def calShannonEnt(img):

img = img.copy()

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

temp = [0] \* 256

sum = len(img) \* len(img[0])

shannonEnt = 0.0

for i in img:

for j in i:

temp[j] += 1

for i in temp:

prob = i / sum

if prob != 0.0:

shannonEnt -= prob \* math.log(prob, 2)

return shannonEnt

def ImgCut(img, num):

img = img.copy()

width0 = len(img[0])

height0 = len(img)

x = int(width0 \* num \* 0.5)

y = int(height0 \* num \* 0.5)

xl = int(width0 \* (1 - num \* 0.5))

yl = int(height0 \* (1 - num \* 0.5))

img = img[x:xl, -yl:-y]

return img

def WriteImg(img, name, shape=(256, 256)):

global finalnum1

global finalnum2

imgnum = 0

img = img.copy()

width = len(img[0])

height = len(img)

for i in range(0, height - shape[0], shape[0]):

for j in range(0, width - shape[1], shape[1]):

imgnum += 1

img2 = img[i:i + shape[0], j:j + shape[1]]

# 对图片进行缩放

img2 = cv2.resize(img2, resizeshape)

imgShannonEnt = calShannonEnt(img2)

# 显示每张图片的图片熵

# print("ShannonEnt: "+name + '\_' + str(shape[0]) + '\_' + str(imgnum)+ ":" + str(imgShannonEnt))

if (imgShannonEnt < 4):

finalnum2 -= 1

continue

filepathw = 'image2/' + name + '\_' + str(shape[0]) + '\_' + str(imgnum) + '.jpg'

cv2.imwrite(filepathw, img2)

# 显示文件大小

# print(name + '\_' + str(shape[0]) + '\_' + str(imgnum) + " " + str(imgShannonEnt),end=' ')

# fsize = os.path.getsize(filepathw)

# fsize = fsize / float(1024)

# print(fsize)

print(name + ' Shape:', shape, '\tNum:', imgnum)

finalnum1 += imgnum

finalnum2 += imgnum

return imgnum

filepath = "image1/"

pathDir = os.listdir(filepath)

if len(sys.argv) > 1:

movestep = int(sys.argv[1])

resizeshape[0] = int(sys.argv[2])

resizeshape[1] = int(sys.argv[3])

for allDir in pathDir:

allDir = os.path.join(filepath, allDir)

(filepath, tempfilename) = os.path.split(allDir)

(shotname, extension) = os.path.splitext(tempfilename)

imgnum = 0

img = cv2.imread(allDir)

width = len(img[0])

height = len(img)

minnum = min(width, height)

colnum = int(math.log(minnum / 256, 2))

for i in range(colnum + 1):

shape = (resizeshape[0] \* pow(2, i), resizeshape[1] \* pow(2, i))

WriteImg(img, shotname, shape)

print("Final CutImgNum:" + str(finalnum1) + " CreatImgNum:" + str(finalnum2))

**附录6 图片筛选的代码**

import os

import cv2

import sys

import math

from numpy import \*

class ImgFilter:

hashlist = []

def \_\_cal\_ShannonEnt(self, img):

temp = [0] \* 256

sum = len(img) \* len(img[0])

shannonEnt = 0.0

for i in img:

for j in i:

temp[j] += 1

for i in temp:

prob = i / sum

if prob != 0.0:

shannonEnt -= prob \* math.log(prob, 2)

return shannonEnt

def \_\_cal\_Laplacian(self, img):

imageVar = cv2.Laplacian(img, cv2.CV\_16S, ksize=3).var()

return imageVar

def \_\_cal\_phash\_code1(self, img):

# 缩小至32\*32

img = img.copy()

m\_img = cv2.resize(img, dsize=(32, 32))

# 浮点型用于计算

m\_img = np.float32(m\_img)

# 离散余弦变换，得到dct系数矩阵

img\_dct = cv2.dct(m\_img)

img\_mean = cv2.mean(img\_dct[0:8, 0:8])

# 返回一个8\*8bool矩阵

return img\_dct[0:8, 0:8] > img\_mean[0]

def \_\_cal\_phash\_code(self, img):

img = img.copy()

img = np.float32(img)

img\_dct = cv2.dct(img)

img\_dct = img\_dct[0:8, 0:8]

img\_mean = cv2.mean(img\_dct)[0]

phash = ""

for i in img\_dct:

for j in i:

phash += str(int(j > img\_mean))

phash = int(phash, 2)

return phash

def \_\_add\_img(self, hash):

self.hashlist.append(hash)

def \_\_cal\_hamming\_distance(self, model\_hash\_code, search\_hash\_code):

# 返回不相同的个数

# diff = model\_hash\_code - search\_hash\_code

diff = str(bin(model\_hash\_code ^ search\_hash\_code)[2:])

count = 0

for i in diff:

count += int(i)

return count

def cal\_img\_filt(self, img):

isdif = False

imgEnt = self.\_\_cal\_ShannonEnt(img.copy())

imageVar = self.\_\_cal\_Laplacian(img.copy())

imageDif = 0

search\_hash\_code = self.\_\_cal\_phash\_code(img.copy())

for model\_hash\_code in self.hashlist:

imageDif = self.\_\_cal\_hamming\_distance(model\_hash\_code, search\_hash\_code)

if imageDif < 5:

isdif = True

break

self.\_\_add\_img(search\_hash\_code)

# print('Ent:' + str(imgEnt) + 'Var:' + str(imageVar) + 'Dif:' + str(imageDif))

return imgEnt, imageVar, isdif

filepath = "image1/"

pathDir = os.listdir(filepath)

imgflit = ImgFilter()

for allDir in pathDir:

allDir = os.path.join(filepath, allDir)

(filepath, tempfilename) = os.path.split(allDir)

(shotname, extension) = os.path.splitext(tempfilename)

img = cv2.imread(allDir)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

img = cv2.resize(img, dsize=(32, 32))

print(shotname, end=' ')

imgEnt, imageVar, isdif = imgflit.cal\_img\_filt(img)

delfalg = False

if imgEnt < 5:

print("Delete " + shotname + ' ShannonEnt:', imgEnt, end=' ')

delfalg = True

elif imageVar < 30000:

print("Delete " + shotname + ' LaplacianVar:', imageVar, end=' ')

delfalg = True

elif isdif:

print("Delete " + shotname + ' same image', end=' ')

delfalg = True

if delfalg:

os.remove(allDir)

print('Finish.')

cv2.waitKey(0)

print('success')

**附录7 图像识别的代码**

import tensorflow as tf

import tensorlayer as tl

import numpy as np

from numpy.random import randint

import os

from PIL import Image

w\_init = tf.truncated\_normal\_initializer(stddev=5e-2)

b\_init = tf.constant\_initializer(value=0.0)

sess = tf.InteractiveSession()

img\_size\_cropped = 128

num\_channels = 3

width = 256

height = 256

path = ''

istrain = True

def conv2d(x, W, name=''):

# padding 方式有两种，SAME与VALID，SAME直接size除以步长，外面部分以0填充;VALID是size-filter+1再除以步长。

# strides是[1,x\_movement,y\_movement,1]

return tl.layers.Conv2dLayer(x, act=tf.nn.relu,

shape=W, # 32 features for each 5x5 patch

strides=[1, 1, 1, 1],

padding='SAME',

W\_init=w\_init,

W\_init\_args={},

b\_init=b\_init,

b\_init\_args={},

name=name)

def max\_pool\_2x2(x, name=''):

return tl.layers.PoolLayer(x,

ksize=[1, 2, 2, 1],

strides=[1, 2, 2, 1],

padding='SAME',

pool=tf.nn.max\_pool,

name=name)

def GetRPGList(img):

RPGList = []

for i in range(0, width):

for j in range(0, height):

cl = img.getpixel((i, j))

RPGList.append(cl[0])

RPGList.append(cl[1])

RPGList.append(cl[2])

return RPGList

def ReadImg():

classes = {'None', 'Green', 'Red', 'Both'}

classes\_test = [3, 0, 1, 2]

img\_list = []

label\_list = []

for index, name in enumerate(classes):

class\_path = os.getcwd() + '/' + name

for img\_name in os.listdir(class\_path):

img\_path = class\_path + '/' + img\_name

img = Image.open(img\_path)

img = img.resize((width, height))

index0 = randint(len(img\_list) + 1) # 获得数组大小内的随机数

img\_list.insert(index0, GetRPGList(img)) # image to bytes

label\_list.insert(index0, index)

img\_list0 = np.array(img\_list)

img\_list0 = img\_list0.reshape(-1, width, width, 3)

label\_list0 = np.array(label\_list)

img\_list0 = np.asarray(img\_list0, dtype=np.float32)

label\_list0 = np.asarray(label\_list0, dtype=np.int32)

train\_num = int(len(img\_list0) \* 0.9)

X\_train = img\_list0[:train\_num - 1]

y\_train = label\_list0[:train\_num - 1]

X\_test = img\_list0[train\_num:]

y\_test = label\_list0[train\_num:]

print("before：", y\_test)

for i, label in enumerate(y\_test):

y\_test[i] = classes\_test[label]

print("after：", y\_test)

return X\_train, y\_train, X\_test, y\_test

# 读取数据

X\_train, y\_train, X\_test, y\_test = ReadImg()

print('X\_train shape:', X\_train.shape)

print('y\_train shape:', y\_train.shape)

print('X\_test shape:', X\_test.shape)

print('y\_test shape:', y\_test.shape)

# placeholder

x = tf.placeholder(tf.float32, shape=[None, width, height, 3], name='x')

y\_ = tf.placeholder(tf.int64, shape=[None, ], name='y\_')

# network

inputs = tl.layers.InputLayer(x, name='input\_layer')

# 卷积层1

network = conv2d(inputs, [3, 3, 3, 8], 'cnn\_layer1\_1')

network = conv2d(network, [3, 3, 8, 8], 'cnn\_layer1\_2')

network = max\_pool\_2x2(network, name='pool\_layer1')

if istrain:

network = tl.layers.DropoutLayer(network, keep=0.8, name='drop1')

# 卷积层2

network = conv2d(network, [3, 3, 8, 16], 'cnn\_layer2\_1')

network = conv2d(network, [3, 3, 16, 16], 'cnn\_layer2\_2')

network = max\_pool\_2x2(network, name='pool\_layer2')

if istrain:

network = tl.layers.DropoutLayer(network, keep=0.8, name='drop2')

# 卷积层3

network = conv2d(network, [3, 3, 16, 32], 'cnn\_layer3\_1')

network = conv2d(network, [3, 3, 32, 32], 'cnn\_layer3\_2')

network = max\_pool\_2x2(network, name='pool\_layer3')

if istrain:

network = tl.layers.DropoutLayer(network, keep=0.8, name='drop3')

# 卷积层4

network = conv2d(network, [3, 3, 32, 64], 'cnn\_layer4\_1')

network = conv2d(network, [3, 3, 64, 64], 'cnn\_layer4\_2')

network = max\_pool\_2x2(network, name='pool\_layer4')

if istrain:

network = tl.layers.DropoutLayer(network, keep=0.8, name='drop4')

# 卷积层5

network = conv2d(network, [3, 3, 64, 64], 'cnn\_layer5\_1')

network = conv2d(network, [3, 3, 64, 64], 'cnn\_layer5\_2')

network = max\_pool\_2x2(network, name='pool\_layer5')

if istrain:

network = tl.layers.DropoutLayer(network, keep=0.8, name='drop5')

# 全连接层

network = tl.layers.FlattenLayer(network, name='flatten\_layer')

network = tl.layers.DenseLayer(network, n\_units=256, act=tf.nn.relu, name='relu1')

network = tl.layers.DenseLayer(network, n\_units=256, act=tf.nn.relu, name='relu2')

# 输出层

network = tl.layers.DenseLayer(network, n\_units=4, act=tf.identity, name='output')

y = network.outputs

# 定义损失函数和衡量指标

cost = tl.cost.cross\_entropy(y, y\_, name='cost')

correct\_prediction = tf.equal(tf.argmax(y, 1), y\_)

acc = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

y\_op = tf.argmax(tf.nn.softmax(y), 1)

# 定义 optimizer

train\_params = network.all\_params

train\_op = tf.train.AdamOptimizer(learning\_rate=0.0001, beta1=0.9, beta2=0.999,

epsilon=1e-08, use\_locking=False).minimize(cost, var\_list=train\_params)

# 初始化 session 中的所有参数

tl.layers.initialize\_global\_variables(sess)

'''

训练模型

batch\_size :训练量

n\_epoch : 训练次数

print\_freq :训练显示间隔

'''

sess.graph.finalize()

writer = tf.summary.FileWriter("E://TensorBoard//train\_2018\_5", sess.graph)

tl.utils.fit(sess, network, train\_op, cost, X\_train, y\_train, x, y\_,

acc=acc, batch\_size=20, n\_epoch=150, print\_freq=5,

)

writer.close()

print('trsin finish')

# 评估模型

tl.utils.test(sess, network, acc, X\_test, y\_test, x, y\_, batch\_size=20, cost=cost)

# 保存模型

tl.files.save\_npz(network.all\_params, name='model4.npz')

os.system('pause')

**附录8 图像识别训练结果**

2018-05-11 19:52:14.287386: I C:\tf\_jenkins\home\workspace\rel-win\M\windows-gpu\PY\36\tensorflow\core\platform\cpu\_feature\_guard.cc:137] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX

2018-05-11 19:52:14.896639: I C:\tf\_jenkins\home\workspace\rel-win\M\windows-gpu\PY\36\tensorflow\core\common\_runtime\gpu\gpu\_device.cc:1030] Found device 0 with properties:

name: GeForce GTX 650 major: 3 minor: 0 memoryClockRate(GHz): 1.0585

pciBusID: 0000:01:00.0

totalMemory: 1.00GiB freeMemory: 825.18MiB

2018-05-11 19:52:14.904998: I C:\tf\_jenkins\home\workspace\rel-win\M\windows-gpu\PY\36\tensorflow\core\common\_runtime\gpu\gpu\_device.cc:1120] Creating TensorFlow device (/device:GPU:0) -> (device: 0, name: GeForce GTX 650, pci bus id: 0000:01:00.0, compute capability: 3.0)

X\_train shape: (579, 256, 256, 3)

y\_train shape: (579,)

X\_test shape: (65, 256, 256, 3)

y\_test shape: (65,)

[TL] InputLayer input\_layer: (?, 256, 256, 3)

[TL] Conv2dLayer cnn\_layer1\_1: shape:[3, 3, 3, 8] strides:[1, 1, 1, 1] pad:SAME act:relu

[TL] Conv2dLayer cnn\_layer1\_2: shape:[3, 3, 8, 8] strides:[1, 1, 1, 1] pad:SAME act:relu

[TL] PoolLayer pool\_layer1: ksize:[1, 2, 2, 1] strides:[1, 2, 2, 1] padding:SAME pool:max\_pool

[TL] DropoutLayer drop1: keep:0.800000 is\_fix:False

[TL] Conv2dLayer cnn\_layer2\_1: shape:[3, 3, 8, 16] strides:[1, 1, 1, 1] pad:SAME act:relu

[TL] Conv2dLayer cnn\_layer2\_2: shape:[3, 3, 16, 16] strides:[1, 1, 1, 1] pad:SAME act:relu

[TL] PoolLayer pool\_layer2: ksize:[1, 2, 2, 1] strides:[1, 2, 2, 1] padding:SAME pool:max\_pool

[TL] DropoutLayer drop2: keep:0.800000 is\_fix:False

[TL] Conv2dLayer cnn\_layer3\_1: shape:[3, 3, 16, 32] strides:[1, 1, 1, 1] pad:SAME act:relu

[TL] Conv2dLayer cnn\_layer3\_2: shape:[3, 3, 32, 32] strides:[1, 1, 1, 1] pad:SAME act:relu

[TL] PoolLayer pool\_layer3: ksize:[1, 2, 2, 1] strides:[1, 2, 2, 1] padding:SAME pool:max\_pool

[TL] DropoutLayer drop3: keep:0.800000 is\_fix:False

[TL] Conv2dLayer cnn\_layer4\_1: shape:[3, 3, 32, 64] strides:[1, 1, 1, 1] pad:SAME act:relu

[TL] Conv2dLayer cnn\_layer4\_2: shape:[3, 3, 64, 64] strides:[1, 1, 1, 1] pad:SAME act:relu

[TL] PoolLayer pool\_layer4: ksize:[1, 2, 2, 1] strides:[1, 2, 2, 1] padding:SAME pool:max\_pool

[TL] DropoutLayer drop4: keep:0.800000 is\_fix:False

[TL] Conv2dLayer cnn\_layer5\_1: shape:[3, 3, 64, 64] strides:[1, 1, 1, 1] pad:SAME act:relu

[TL] Conv2dLayer cnn\_layer5\_2: shape:[3, 3, 64, 64] strides:[1, 1, 1, 1] pad:SAME act:relu

[TL] PoolLayer pool\_layer5: ksize:[1, 2, 2, 1] strides:[1, 2, 2, 1] padding:SAME pool:max\_pool

[TL] DropoutLayer drop5: keep:0.800000 is\_fix:False

[TL] FlattenLayer flatten\_layer: 4096

[TL] DenseLayer relu1: 256 relu

[TL] DenseLayer relu2: 256 relu

[TL] DenseLayer output: 4 identity

Start training the network ...

2018-05-11 19:54:07.279042: W C:\tf\_jenkins\home\workspace\rel-win\M\windows-gpu\PY\36\tensorflow\core\common\_runtime\bfc\_allocator.cc:217] Allocator (GPU\_0\_bfc) 2018-05-11 19:54:07.763581: W C:\tf\_jenkins\home\workspace\rel-win\M\windows-gpu\PY\36\tensorflow\core\common\_runtime\bfc\_allocator.cc:217] Allocator (GPU\_0\_bfc) ran

out of memory trying to allocate 291.75MiB. The caller 2018-05-11 19:54:07.991152: W C:\tf\_jenkins\home\workspace\rel-win\M\windows-gpu\PY\36\tensorflow\core\common\_runtime\bfc\_allocator.cc:217] Allocator (GPU\_0\_bfc) ran out of memory trying to allocate 211.63MiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is available.

2018-05-11 19:54:08.803696: W C:\tf\_jenkins\home\workspace\rel-win\M\windows-gpu\PY\36\tensorflow\core\common\_runtime\bfc\_allocator.cc:217] Allocator (GPU\_0\_bfc) ran out of memory trying to allocate 211.63MiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is

available.

2018-05-11 19:54:09.233302: W C:\tf\_jenkins\home\workspace\rel-win\M\windows-gpu\PY\36\tensorflow\core\common\_runtime\bfc\_allocator.cc:217] Allocator (GPU\_0\_bfc) ran out of memory trying to allocate 291.75MiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is

available.

2018-05-11 19:54:09.429723: W C:\tf\_jenkins\home\workspace\rel-win\M\windows-gpu\PY\36\tensorflow\core\common\_runtime\bfc\_allocator.cc:217] Allocator (GPU\_0\_bfc) ran out of memory trying to allocate 342.50MiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is

available.

2018-05-11 19:54:09.728625: W C:\tf\_jenkins\home\workspace\rel-win\M\windows-gpu\PY\36\tensorflow\core\common\_runtime\bfc\_allocator.cc:217] Allocator (GPU\_0\_bfc) ran out of memory trying to allocate 261.88MiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is

available.

Epoch 1 of 150 took 27.265677s, loss 1.267250

Epoch 5 of 150 took 16.565827s, loss 1.066999

Epoch 10 of 150 took 16.420237s, loss 0.985250

Epoch 15 of 150 took 16.451963s, loss 0.808793

Epoch 20 of 150 took 16.469332s, loss 0.689360

Epoch 25 of 150 took 16.435371s, loss 0.575539

Epoch 30 of 150 took 16.468132s, loss 0.429089

Epoch 35 of 150 took 16.466798s, loss 0.279772

Epoch 40 of 150 took 16.429645s, loss 0.244443

Epoch 45 of 150 took 16.468636s, loss 0.160199

Epoch 50 of 150 took 16.436998s, loss 0.083015

Epoch 55 of 150 took 16.425074s, loss 0.110175

Epoch 60 of 150 took 16.437185s, loss 0.089150

Epoch 65 of 150 took 16.430717s, loss 0.050067

Epoch 70 of 150 took 16.405490s, loss 0.088758

Epoch 75 of 150 took 16.421892s, loss 0.028070

Epoch 80 of 150 took 16.454625s, loss 0.011048

Epoch 85 of 150 took 16.423402s, loss 0.077526

Epoch 90 of 150 took 16.438122s, loss 0.038494

Epoch 95 of 150 took 16.405425s, loss 0.017849

Epoch 100 of 150 took 16.430488s, loss 0.009993

Epoch 105 of 150 took 16.438745s, loss 0.019449

Epoch 110 of 150 took 16.437704s, loss 0.019028

Epoch 115 of 150 took 16.422076s, loss 0.005781

Epoch 120 of 150 took 16.422211s, loss 0.066044

Epoch 125 of 150 took 16.452475s, loss 0.026337

Epoch 130 of 150 took 16.407365s, loss 0.013262

Epoch 135 of 150 took 16.439872s, loss 0.016061

Epoch 140 of 150 took 16.451490s, loss 0.008581

Epoch 145 of 150 took 16.434382s, loss 0.006438

Epoch 150 of 150 took 16.404382s, loss 0.006843

Total training time: 2477.412773s

trsin finish

[TL] Start testing the network ...

[TL] test acc: 0.950000