

**软件课程实验报告**

**题目：图像识别 日期：2019.12.25**

**专业：通信工程 班级：1702班**

**姓名： 熊嘉恒 学号：U201713438**

**目录**

[一、 问题描述 3](#_Toc26883)

[二、 系统分析 3](#_Toc22411)

[三、 系统设计 5](#_Toc31754)

[四、 系统实现 9](#_Toc15097)

[五、 使用说明 2](#_Toc17118)3

[六、 系统评价 2](#_Toc16197)5

[附录A源程序: 2](#_Toc10041)5

1. **问题描述**

深度学习，作为近年来人工智能最有效的方法和工具，已经被证实可以应用到图像识别，语音处理，智能控制，信息通信等各个领域。但是现在深度学习网络最流行的平台，如Tensorflow, Keras Pytorch, Caffe等深度学习平台都不需要使用者手动构建深度学习网络，虽然使用方便，但是，对底层网络的构建的完全屏蔽，加深了深度学习的黑盒化和不可解释性。深刻理解深度学习网络的机理，构建**可解释**的深度学习模型是目前研究的热点，本课程设计深度学习网络模型的多种数据结构实现是基础。本课程设计的任务是利用所学的数据结构知识，自己动手构建深度学习网络模型，如CNN, RNN, C-LSTM, GAN等。能够使用网络模型进行深度学习，训练和测试，完成其特定的图像识别分割语音处理等功能。

[基本要求]

（1）能够静态构建深度学习网络模型；

（2）能够构造低维的网络模型；

（3）能够使用网络模型进行深度学习，训练和测试；

（4）完成其特定的图像识别等简单功能；

（5）良好的人机交互界面。

[团队分工]

前端：杨一帆

后端：熊嘉恒，徐家洛

1. **系统分析**

**1）深度学习概述**

深度学习被引入机器学习使其更接近于最初的目标----人工智能（AI，Artificial Intelligence）。深度学习是学习样本数据的内在规律和表示层次，这些学习过程中获得的信息对诸如文字、图像和声音等数据的解释有很大的帮助。它的最终目标是让机器能够像人一样具有分析学习能力，能够识别文字、图像和声音等数据。

深度学习是一个复杂的机器学习算法，在语言和图像识别方面取得的效果，远远超过先前相关技术。它在搜索技术、数据挖掘、机器学习、机器翻译、自然语言处理、多媒体学习、语音、推荐和个性化技术，以及其它相关领域都取得了很多成果。深度学习使机器模仿视听和思考等人类的活动，解决了很多复杂的模式识别难题，使得人工智能相关技术取得了很大进步。

**2）深度学习与图像识别**

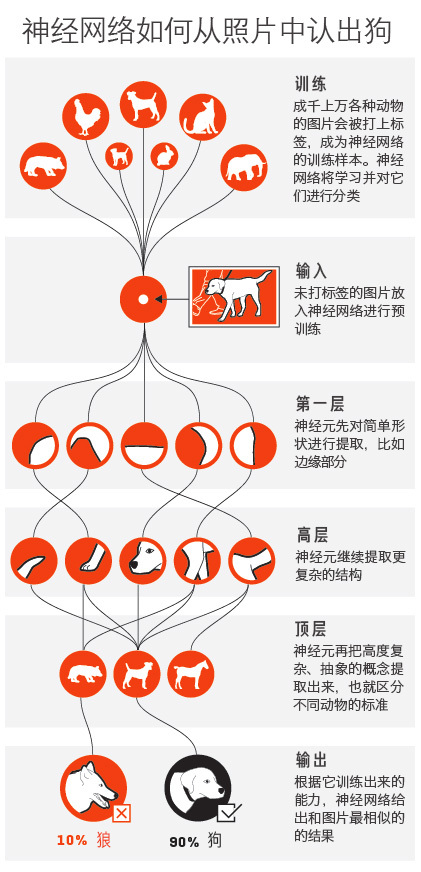
计算机的图像识别技术与人类的图像识别原理相同，它们的过程也是大同小异的。图像识别技术的过程分以下几步：信息的获取、预处理、特征抽取和选择、分类器设计和分类决策。

神经网络图像识别技术是一种比较新型的图像识别技术，是在传统的图像识别方法和基础上融合神经网络算法的一种图像识别方法。这里的神经网络是指人工神经网络，也就是说这种神经网络并不是动物本身所具有的真正的神经网络，而是人类模仿动物神经网络后人工生成的。在神经网络图像识别技术中，遗传算法与BP网络相融合的神经网络图像识别模型是非常经典的，在很多领域都有它的应用。在图像识别系统中利用神经网络系统，一般会先提取图像的特征，再利用图像所具有的特征映射到神经网络进行图像识别分类。以汽车拍照自动识别技术为例，当汽车通过的时候，汽车自身具有的检测设备会有所感应。此时检测设备就会启用图像采集装置来获取汽车正反面的图像。获取了图像后必须将图像上传到计算机进行保存以便识别。最后车牌定位模块就会提取车牌信息，对车牌上的字符进行识别并显示最终的结果。在对车牌上的字符进行识别的过程中就用到了基于模板匹配算法和基于人工神经网络算法。

计算机的图像识别技术是一个异常高维的识别技术。不管图像本身的分辨率如何，其产生的数据经常是多维性的，这给计算机的识别带来了非常大的困难。想让计算机具有高效地识别能力，最直接有效的方法就是降维。降维分为线性降维和非线性降维。例如主成分分析(PCA)和线性奇异分析(LDA)等就是常见的线性降维方法，它们的特点是简单、易于理解。但是通过线性降维处理的是整体的数据集合，所求的是整个数据集合的最优低维投影。经过验证，这种线性的降维策略计算复杂度高而且占用相对较多的时间和空间，因此就产生了基于非线性降维的图像识别技术，它是一种极其有效的非线性特征提取方法。此技术可以发现图像的非线性结构而且可以在不破坏其本征结构的基础上对其进行降维，使计算机的图像识别在尽量低的维度上进行，这样就提高了识别速率。例如人脸图像识别系统所需的维数通常很高，其复杂度之高对计算机来说无疑是巨大的“灾难”。由于在高维度空间中人脸图像的不均匀分布，使得人类可以通过非线性降维技术来得到分布紧凑的人脸图像，从而提高人脸识别技术的高效性。

说的简单一点就是，机器自动从海量数据中总结归纳物体的特征，然后进行识别和判断。

然后一张解释，机器如何识别一张图片的具体内容。



1. **系统设计**

**1、功能需求**

1. 良好的人机交互界面：采用HTML、CSS构建前端的网站界面，提供用户便于操作的方式，能够输入待识别的图片，和输出结果。采用端口的方式与后端交互；
2. 图片识别的功能：如上图，在搭建的Keras框架下，采用cifar10集进行图片的训练，然后通过搭建卷积神经网络和残差网络对输入的图片进行提取特征，最后输出概率最大的图片的识别结果。
3. **keras框架**

Keras是基于Theano的一个深度学习框架，它的设计参考了Torch，用Python语言编写，是一个高度模块化的神经网络库，支持GPU和CPU。

为什么我们要使用Keras框架呢？Keras是为了使深度学习工程师能够很快地建立和实验不同的模型的框架，正如TensorFlow是一个比Python更高级的框架，Keras是一个更高层次的框架，并提供了额外的抽象方法。最关键的是Keras能够以最短的时间让想法变为现实。

下面是一些我们的用到的Keras框架的包，以及它们的主要作用：

1） Models包：keras.models

作用：这是Keras中最主要的一个模块，用于对各个组件进行组装。

2）Layers包：keras.layers

作用：该模块主要用于生成神经网络层，其中Core layers里面包含了flatten（CNN的全连接层之前需要把二维特征图flatten成为一维的）、reshape（CNN输入时将一维的向量弄成二维的）、dense（隐藏层）。

Convolutional layers层包含Theano的Convolution2D的封装等。

3） Initializations包：keras.initializations

作用：该模块主要负责对模型参数（权重）进行初始化

4）Activations包：keras.activations、keras.layers.advanced\_activations（新激活函数）

作用：该模块主要负责为神经层附加激活函数

1. Objectives包：keras.objectives

作用：该模块主要负责为神经网络附加损失函数，即目标函数。

1. Optimizers包：keras.optimizers

作用：该模块主要负责设定神经网络的优化方法，如最基本的随机梯度下降SGD,另外还有Adagrad、Adadelta、RMSprop、Adam

1. Preprocessing包：keras.preprocessing

作用：数据预处理模块，包括序列数据的处理、文本数据的处理和图像数据的处理等。对于图像数据的处理，keras提供了ImageDataGenerator函数,实现数据集扩增，对图像做一些弹性变换，比如水平翻转，垂直翻转，旋转等。

1. metrics包：keras.metrics

作用：与sklearn中metrics包基本相同，主要包含一些如binary\_accuracy、mae、mse等的评价方法。

1. **卷积神经网络**

一个典型的卷积神经网络可以看做下面这样**：**

IMG_256

输入→卷积→ReLU→卷积→ReLU→池化→ReLU→卷积→ReLU→池化→全连接

1. 卷积：通过类似过滤器与输入数据卷积，提取图片的特征。卷积得到的结果的值越大，则说明该图片与该结果越符合；
2. ReLU：即修正线性单元，是一种[人工神经网络](https://baike.baidu.com/item/%E4%BA%BA%E5%B7%A5%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C" \t "https://baike.baidu.com/item/ReLU%20%E5%87%BD%E6%95%B0/_blank)中常用的激活函数。而激活函数的基本作用是将上一层的输出反馈为下一层的输入。更重要的作用是，如果我们不运用激活函数的话，则输出信号将仅仅是一个简单的线性函数。线性函数一个一级多项式。现如今，线性方程是很容易解决的，但是它们的复杂性有限，并且从数据中学习复杂函数映射的能力更小。一个没有激活函数的神经网络将只不过是一个线性回归模型（Linear regression Model）罢了，它功率有限，并且大多数情况下执行得并不好。我们希望我们的神经网络不仅仅可以学习和计算线性函数，而且还要比这复杂得多。同样是因为没有激活函数，我们的神经网络将无法学习和模拟其他复杂类型的数据，例如图像、视频、音频、语音等。这就是为什么我们要使用人工神经网络技术，诸如深度学习（Deep learning），来理解一些复杂的事情，一些相互之间具有很多隐藏层的非线性问题，而这也可以帮助我们了解复杂的数据。
3. Pool：即池化，作用是可以降低输入的维度。起到两个目的，第一降低了计算成本，第二是它可以控制过拟合。
4. **残差网络**

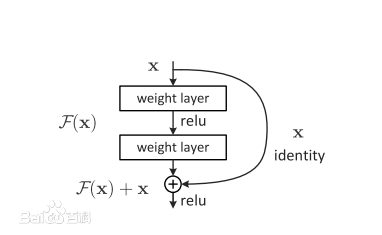
在深度学习过程中，随着深度的加深，梯度要么会呈指数将小为零，要么会呈指数爆炸，导致深度学习的效果、准确率不算太好，通过残差网络，可以随着卷及深度的增加，实现更复杂的功能。

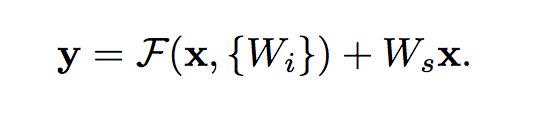
我们都知道增加网络的宽度和深度可以很好的提高网络的性能，深的网络一般都比浅的的网络效果好，比如说一个深的网络A和一个浅的网络B，那A的性能至少都能跟B一样，为什么呢？因为就算我们把B的网络参数全部迁移到A的前面几层，而A后面的层只是做一个等价的映射，就达到了B网络的一样的效果。一个比较好的例子就是VGG，该网络就是在AlexNex的基础上通过增加网络深度大幅度提高了网络性能。

对于原来的网络，如果简单地增加深度，会导致梯度弥散或梯度爆炸。对于该问题的解决方法是正则化初始化和中间的正则化层（Batch Normalization），这样的话可以训练几十层的网络。

虽然通过上述方法能够训练了，但是又会出现另一个问题，就是退化问题，网络层数增加，但是在训练集上的准确率却饱和甚至下降了。这个不能解释为overfitting，因为overfit应该表现为在训练集上表现更好才对。退化问题说明了深度网络不能很简单地被很好地优化。作者通过实验：通过浅层网络等同映射构造深层模型，结果深层模型并没有比浅层网络有等同或更低的错误率，推断退化问题可能是因为深层的网络并不是那么好训练，也就是求解器很难去利用多层网络拟合同等函数。

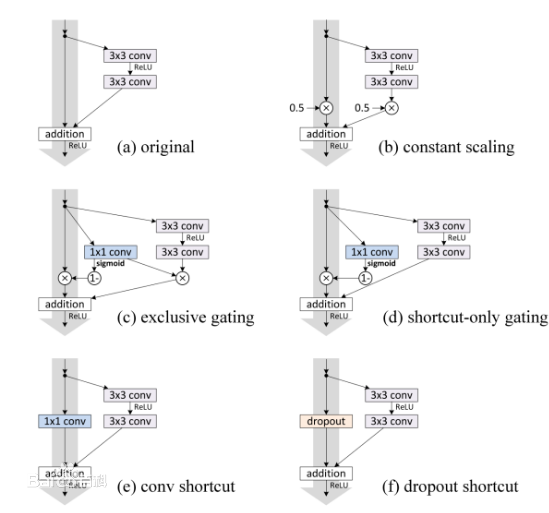
深度残差网络。如果深层网络的后面那些层是恒等映射，那么模型就退化为一个浅层网络。那当前要解决的就是学习恒等映射函数了。 但是直接让一些层去拟合一个潜在的恒等映射函数 IMG_256，比较困难，这可能就是深层网络难以训练的原因。但是，如果把网络设计为 IMG_257，我们可以转换为学习一个残差函数 IMG_258。 只要 IMG_259，就构成了一个恒等映射 IMG_260。 而且，拟合残差肯定更加容易。



一个残差单元的公式如下

F是求和前网络映射，H是从输入到求和后的网络映射。比如把5映射到5.1，那么引入残差前是 IMG_256，引入残差后是 IMG_257，IMG_258 ，

IMG_259。这里的F'和F都表示网络参数映射，引入残差后的映射对输出的变化更敏感。比如s输出从5.1变到5.2，映射 IMG_260的输出增加了2%，而对于残差结构输出从5.1到5.2，映射F是从0.1到0.2，增加了100%。明显后者输出变化对权重的调整作用更大，所以效果更好。残差的思想都是去掉相同的主体部分，从而突出微小的变化。



1. **系统实现**
2. **导入用到的库函数**

# \_\_author\_\_ = 'xiongjiaheng'  
# -\*- coding:utf-8 -\*-  
**import** numpy **as** np  
**from** keras.layers **import** Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization, Flatten, Conv2D, \  
 AveragePooling2D, MaxPooling2D  
**from** keras.models **import** Model, load\_model  
**from** keras.initializers **import** glorot\_uniform  
**import** os  
**import** keras.backend **as** K  
K.set\_image\_data\_format('channels\_last')  
K.set\_learning\_phase(1)  
  
**import** resnets\_utils

1. **构建残差网络**

**1）恒等快的构建**

恒等块实现的步骤：

每个步骤如下：

主路径的第一部分：

1、第一个CONV2D有F1​个过滤器，其大小为（1，1），步长为（1，1），使用填充方式为“valid”，命名规则为conv\_name\_base + '2a'，使用 0作为随机种子为其初始化。

2、第一个BatchNorm是通道的轴归一化，其命名规则为bn\_name\_base + '2a'。3、接着使用ReLU激活函数，它没有命名也没有超参数。

· 主路径的第二部分：

1. 第二个CONV2D有F2​个过滤器，其大小为（f，f），步长为（1，1），使用填充方式为“same”，命名规则为conv\_name\_base + '2b'，使用0作为随机种子为其初始化。
2. 第二个BatchNorm是通道的轴归一化，其命名规则为bn\_name\_base + '2b'。3、接着使用ReLU激活函数，它没有命名也没有超参数。

· 主路径的第三部分：

1. 第三个CONV2D有F3​个过滤器，其大小为（1，1），步长为（1，1），使用填充方式为“valid”，命名规则为conv\_name\_base + '2c'，使用0作为随机种子为其初始化。
2. 第三个BatchNorm是通道的轴归一化，其命名规则为bn\_name\_base + '2c'。3、注意这里没有ReLU函数

· 最后一步：

1. 将捷径与输入加在一起使用ReLU激活函数
2. 它没有命名也没有超参数。

**2）恒等快实现代码**

**def identity\_block**(X, f, filters, stage, block):  
 *"""  
 实现恒等块  
  
 参数：  
 X - 输入的tensor类型的数据，维度为( m, n\_H\_prev, n\_W\_prev, n\_H\_prev )  
 f - 整数，指定主路径中间的CONV窗口的维度  
 filters - 整数列表，定义了主路径每层的卷积层的过滤器数量  
 stage - 整数，根据每层的位置来命名每一层，与block参数一起使用。  
 block - 字符串，据每层的位置来命名每一层，与stage参数一起使用。  
  
 返回：  
 X - 恒等块的输出，tensor类型，维度为(n\_H, n\_W, n\_C)  
  
 """* # 定义命名规则  
 conv\_name\_base = "res" + str(stage) + block + "\_branch"  
 bn\_name\_base = "bn" + str(stage) + block + "\_branch"  
  
 # 获取过滤器  
 F1, F2, F3 = filters  
  
 # 保存输入数据，将会用于为主路径添加捷径  
 X\_shortcut = X  
  
 # 主路径的第一部分  
 ##卷积层  
 X = Conv2D(filters=F1, kernel\_size=(1, 1), strides=(1, 1), padding="valid",  
 name=conv\_name\_base + "2a", kernel\_initializer=glorot\_uniform(seed=0))(X)  
 ##归一化  
 X = BatchNormalization(axis=3, name=bn\_name\_base + "2a")(X)  
 ##使用ReLU激活函数  
 X = Activation("relu")(X)  
  
 # 主路径的第二部分  
 ##卷积层  
 X = Conv2D(filters=F2, kernel\_size=(f, f), strides=(1, 1), padding="same",  
 name=conv\_name\_base + "2b", kernel\_initializer=glorot\_uniform(seed=0))(X)  
 ##归一化  
 X = BatchNormalization(axis=3, name=bn\_name\_base + "2b")(X)  
 ##使用ReLU激活函数  
 X = Activation("relu")(X)  
  
  
 # 主路径的第三部分  
 ##卷积层  
 X = Conv2D(filters=F3, kernel\_size=(1, 1), strides=(1, 1), padding="valid",  
 name=conv\_name\_base + "2c", kernel\_initializer=glorot\_uniform(seed=0))(X)  
 ##归一化  
 X = BatchNormalization(axis=3, name=bn\_name\_base + "2c")(X)  
 ##没有ReLU激活函数  
  
 # 最后一步：  
 ##将捷径与输入加在一起  
 X = Add()([X, X\_shortcut])  
 ##使用ReLU激活函数  
 X = Activation("relu")(X)  
  
 **return** X

**3）卷积块的实现**

具体步骤如下：

· 主路径第一部分：

1、第一个卷积层有F1​个过滤器，其维度为（1，1），步伐为（s，s），使用“**valid**”的填充方式，命名规则为conv\_name\_base + '2a'

2、第一个规范层是通道的轴归一化，其命名规则为bn\_name\_base + '2a'

3、使用ReLU激活函数，它没有命名规则也没有超参数。

· 主路径第二部分：

1. 第二个卷积层有F2​个过滤器，其维度为（f，f），步伐为（1，1），使用“**same**”的填充方式，命名规则为conv\_name\_base + '2b'

2、第二个规范层是通道的轴归一化，其命名规则为bn\_name\_base + '2b'

3、使用ReLU激活函数，它没有命名规则也没有超参数。

· 主路径第三部分：

1. 第三个卷积层有F3​个过滤器，其维度为（1，1），步伐为（s，s），使用“**valid**”的填充方式，命名规则为conv\_name\_base + '2c'
2. 第三个规范层是通道的轴归一化，其命名规则为bn\_name\_base + '2c'

3、没有激活函数

· 捷径：

1、此卷积层有F3​个过滤器，其维度为（1，1），步伐为（s，s），使用“**valid**”的填充方式，命名规则为conv\_name\_base + '1'

2、此规范层是通道的轴归一化，其命名规则为bn\_name\_base + '1'

· 最后一步：

1、将捷径与输入加在一起

2、使用ReLU激活函数

**4）卷积块的代码实现**

**def convolutional\_block**(X, f, filters, stage, block, s=2):  
 *"""  
 实现卷积块  
  
 参数：  
 X - 输入的tensor类型的变量，维度为( m, n\_H\_prev, n\_W\_prev, n\_C\_prev)  
 f - 整数，指定主路径中间的CONV窗口的维度  
 filters - 整数列表，定义了主路径每层的卷积层的过滤器数量  
 stage - 整数，根据每层的位置来命名每一层，与block参数一起使用。  
 block - 字符串，据每层的位置来命名每一层，与stage参数一起使用。  
 s - 整数，指定要使用的步幅  
  
 返回：  
 X - 卷积块的输出，tensor类型，维度为(n\_H, n\_W, n\_C)  
 """* # 定义命名规则  
 conv\_name\_base = "res" + str(stage) + block + "\_branch"  
 bn\_name\_base = "bn" + str(stage) + block + "\_branch"  
  
 # 获取过滤器数量  
 F1, F2, F3 = filters  
  
 # 保存输入数据  
 X\_shortcut = X  
  
 # 主路径  
 ##主路径第一部分  
 X = Conv2D(filters=F1, kernel\_size=(1, 1), strides=(s, s), padding="valid",  
 name=conv\_name\_base + "2a", kernel\_initializer=glorot\_uniform(seed=0))(X)  
 X = BatchNormalization(axis=3, name=bn\_name\_base + "2a")(X)  
 X = Activation("relu")(X)  
  
 ##主路径第二部分  
 X = Conv2D(filters=F2, kernel\_size=(f, f), strides=(1, 1), padding="same",  
 name=conv\_name\_base + "2b", kernel\_initializer=glorot\_uniform(seed=0))(X)  
 X = BatchNormalization(axis=3, name=bn\_name\_base + "2b")(X)  
 X = Activation("relu")(X)  
  
 ##主路径第三部分  
 X = Conv2D(filters=F3, kernel\_size=(1, 1), strides=(1, 1), padding="valid",  
 name=conv\_name\_base + "2c", kernel\_initializer=glorot\_uniform(seed=0))(X)  
 X = BatchNormalization(axis=3, name=bn\_name\_base + "2c")(X)  
  
 # 捷径  
 X\_shortcut = Conv2D(filters=F3, kernel\_size=(1, 1), strides=(s, s), padding="valid",  
 name=conv\_name\_base + "1", kernel\_initializer=glorot\_uniform(seed=0))(X\_shortcut)  
 X\_shortcut = BatchNormalization(axis=3, name=bn\_name\_base + "1")(X\_shortcut)  
  
 # 最后一步  
 X = Add()([X, X\_shortcut])  
 X = Activation("relu")(X)  
  
 **return X**

**5）实现50层的残差网络**

这个50层的网络的细节如下:

1、对输入数据进行0填充，padding =（3,3）

2、stage1：

卷积层有64个过滤器，其维度为（7，7），步伐为（2，2），命名为“conv1”

规范层（BatchNorm）对输入数据进行通道轴归一化。

最大值池化层使用一个（3，3）的窗口和（2，2）的步伐。

3、stage2：

卷积块使用f=3个大小为[64，64，256]的过滤器，f=3，s=1,block=“a”

2个恒等块使用三个大小为[64，64，256]的过滤器，f=3，block=“b”、“c”

4、Stage3：

卷积块使用f=3个大小为[128,128,512]的过滤器，f=3，s=2,block=“a”

3个恒等块使用三个大小为[128,128,512]的过滤器，f=3，block=“b”、“c”、“d”

5、stage4：

卷积块使用f=3个大小为[256,256,1024]的过滤器，f=3，s=2,block=“a”

5个恒等块使用三个大小为[256,256,1024]的过滤器，f=3，block=“b”、“c”、“d”、“e”、“f”

6、stage5：

卷块使用f=3个大小为[512,512,2048]的过滤器，f=3，s=2,block=“a”

7、2个恒等块使用三个大小为[256,256,2048]的过滤器，f=3，block=“b”、“c”

8、均值池化层使用维度为（2,2）的窗口，命名为“avg\_pool”

9、 展开操作没有任何超参数以及命名

全连接层（密集连接）使用softmax激活函数，命名为"fc" + str(classes)

**代码如下：**

**def ResNet50**(input\_shape=(32, 32, 3), classes=10):  
 *"""  
 实现ResNet50  
 CONV2D -> BATCHNORM -> RELU -> MAXPOOL -> CONVBLOCK -> IDBLOCK\*2 -> CONVBLOCK -> IDBLOCK\*3  
 -> CONVBLOCK -> IDBLOCK\*5 -> CONVBLOCK -> IDBLOCK\*2 -> AVGPOOL -> TOPLAYER  
  
 参数：  
 input\_shape - 图像数据集的维度  
 classes - 整数，分类数  
  
 返回：  
 model - Keras框架的模型  
  
 """* # 定义tensor类型的输入数据  
 X\_input = Input(input\_shape)  
  
 # 0填充  
 X = ZeroPadding2D((3, 3))(X\_input)  
  
 # stage1  
 X = Conv2D(filters=64, kernel\_size=(7, 7), strides=(2, 2), name="conv1",  
 kernel\_initializer=glorot\_uniform(seed=0))(X)  
 X = BatchNormalization(axis=3, name="bn\_conv1")(X)  
 X = Activation("relu")(X)  
 X = MaxPooling2D(pool\_size=(3, 3), strides=(2, 2))(X)  
  
 # stage2  
 X = convolutional\_block(X, f=3, filters=[64, 64, 256], stage=2, block="a", s=1)  
 X = identity\_block(X, f=3, filters=[64, 64, 256], stage=2, block="b")  
 X = identity\_block(X, f=3, filters=[64, 64, 256], stage=2, block="c")  
  
 # stage3  
 X = convolutional\_block(X, f=3, filters=[128, 128, 512], stage=3, block="a", s=2)  
 X = identity\_block(X, f=3, filters=[128, 128, 512], stage=3, block="b")  
 X = identity\_block(X, f=3, filters=[128, 128, 512], stage=3, block="c")  
 X = identity\_block(X, f=3, filters=[128, 128, 512], stage=3, block="d")  
  
 # stage4  
 X = convolutional\_block(X, f=3, filters=[256, 256, 1024], stage=4, block="a", s=2)  
 X = identity\_block(X, f=3, filters=[256, 256, 1024], stage=4, block="b")  
 X = identity\_block(X, f=3, filters=[256, 256, 1024], stage=4, block="c")  
 X = identity\_block(X, f=3, filters=[256, 256, 1024], stage=4, block="d")  
 X = identity\_block(X, f=3, filters=[256, 256, 1024], stage=4, block="e")  
 X = identity\_block(X, f=3, filters=[256, 256, 1024], stage=4, block="f")  
  
 # stage5  
 X = convolutional\_block(X, f=3, filters=[512, 512, 2048], stage=5, block="a", s=2)  
 X = identity\_block(X, f=3, filters=[512, 512, 2048], stage=5, block="b")  
 X = identity\_block(X, f=3, filters=[512, 512, 2048], stage=5, block="c")  
  
 # 均值池化层  
 X = AveragePooling2D(pool\_size=(2, 2), padding="same")(X)  
  
 # 输出层  
 X = Flatten()(X)  
 X = Dense(classes, activation="softmax", name="fc" + str(classes),  
 kernel\_initializer=glorot\_uniform(seed=0))(X)  
  
  
 # 创建模型  
 model = Model(inputs=X\_input, outputs=X, name="ResNet50")  
  
 **return** model

1. **训练函数**

**def load\_cifar10**(path):  
 train\_data = np.ones((1, 3073), dtype=np.uint8)  
  
 # 将训练数据进行拼接  
 **for** i **in** range(1, 6):  
 data\_path = os.path.join(path, 'data\_batch\_' + str(i) + '.bin')  
  
 **with** open(data\_path, 'rb') **as** f:  
 data = np.fromfile(f, dtype=np.uint8).reshape(10000, 3073)  
 train\_data = np.concatenate((train\_data, data))  
  
 # 调整训练数据维度,原因见cifar10的数据说明  
 train\_labels = train\_data[1:, :1]  
 train\_images = train\_data[1:, 1:].reshape(50000, 3, 32, 32).transpose(0, 2, 3, 1)  
  
 # 获取训练数据  
 data\_path = os.path.join(path, 'test\_batch.bin')  
 **with** open(data\_path, 'rb') **as** f:  
 test\_data = np.fromfile(data\_path, dtype=np.uint8).reshape(10000, 3073)  
 test\_labels = test\_data[:, :1]  
 test\_images = test\_data[:, 1:].reshape(10000, 3, 32, 32).transpose(0, 2, 3, 1)  
  
 **return** (train\_images, train\_labels, test\_images, test\_labels)  
  
  
X\_train\_orig, Y\_train\_orig, X\_test\_orig, Y\_test\_orig = load\_cifar10("./datasets/")  
  
# Normalize image vectors  
X\_train = X\_train\_orig / 255.  
X\_test = X\_test\_orig / 255.  
  
# Convert training and test labels to one hot matrices  
Y\_train = resnets\_utils.convert\_to\_one\_hot(Y\_train\_orig, 10).T  
Y\_test = resnets\_utils.convert\_to\_one\_hot(Y\_test\_orig, 10).T  
  
print("number of training examples = " + str(X\_train.shape[0]))  
print("number of test examples = " + str(X\_test.shape[0]))  
print("X\_train shape: " + str(X\_train.shape))  
print("Y\_train shape: " + str(Y\_train.shape))  
print("X\_test shape: " + str(X\_test.shape))  
print("Y\_test shape: " + str(Y\_test.shape))  
  
  
model.fit(X\_train, Y\_train, epochs=120, batch\_size=1000)  
preds = model.evaluate(X\_test, Y\_test)  
print("测试集误差值 = " + str(preds[0]))  
print("测试集准确率 = " + str(preds[1]))

1. **图片格式的转换和结果的输出**

**def IsValidImage**(img\_path):  
 *"""  
 判断文件是否为有效（完整）的图片* ***:param*** *img\_path:图片路径* ***:return****:True：有效 False：无效  
 """* bValid = **True  
 try**:  
 Image.open(img\_path).verify()  
 **except**:  
 bValid = **False  
 return** bValid  
  
  
**def transimg**(img\_path):  
 *"""  
 转换图片格式* ***:param*** *img\_path:图片路径* ***:return****: True：成功 False：失败  
 """* **if** IsValidImage(img\_path):  
  
 str = img\_path.rsplit(".", 1)  
 output\_img\_path = str[0] + ".jpg"  
 print(output\_img\_path)  
 im = Image.open(img\_path)  
 im=im.convert('RGB')  
 im.save(output\_img\_path)  
 **return** output\_img\_path  
  
 **else**:  
 **return False  
  
  
  
  
  
def predict**(img\_path):  
 my\_image = image.load\_img(img\_path, target\_size=(32, 32))  
  
 my\_image = image.img\_to\_array(my\_image)  
 print("my\_image.shape = " + str(my\_image.shape))  
 my\_image = np.expand\_dims(my\_image,axis=0)/255  
  
  
  
 print("my\_image.shape = " + str(my\_image.shape))  
  
 print("class prediction vector = ")  
 model = load\_model('ResNet50.h5')  
  
 result=model.predict(my\_image)  
 classes = ['飞机', '汽车', '鸟', '猫', '鹿', '狗', '青蛙', '马', '船', '卡车']  
 index=np.argmax(result,1)  
 print(index)  
 result=classes[int(index)]  
 print("输入图像为：",classes[int(index)])  
  
  
  
 plot\_model(model, to\_file='model.png')  
 **return** result

1. **构建人机交互平台网站**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
 <style>  
 .pghead{  
 background-color: #d4d4d5;  
 height:150px;  
 }  
 .pic{  
 position: relative;  
 background-color: white;  
 margin: 10px auto;  
 width: 800px;  
 height: 600px;  
 }  
 </style>  
</head>  
  
  
<body style="margin:0 auto">  
 <div class="pghead">  
 <div style="float:left;text-indent: 350px;padding-top: 30px;font-size: 26pt">软件课设项目</div>  
 <div style="float:left;text-indent: 100px;padding-top: 100px;font-size: 16pt">——基于深度学习的CIFAR-10图片识别</div>  
 <div style="clear:both"></div>  
  
 </div>  
  
 <div class="pic">  
 <img src="" style=" height: 400px;">  
 <div>请选择图片上传</div><br>  
  
 <form id="my\_form" name="form" action="http://222.20.25.67:8210" method="POST" enctype="multipart/form-data" >  
 <input type="file" name="pic" id="my\_file" >  
 <input type="submit" value="识别">  
  
 </form>  
  
  
 <script>  
 **var** fileInput = document.querySelector('input[type=file]'),  
 previewImg = document.querySelector('img');  
 fileInput.addEventListener('change', **function** () {  
 **var** file = **this**.files[0];  
 **var** reader = **new** FileReader();  
 // 监听reader对象的的onload事件，当图片加载完成时，把base64编码賦值给预览图片  
 reader.addEventListener("load", **function** () {  
 previewImg.src = reader.result;  
 }, **false**);  
 // 调用reader.readAsDataURL()方法，把图片转成base64  
 reader.readAsDataURL(file);  
 }, **false**);  
 </script>  
  
 </div>  
  
  
  
</body>  
</html>

1. **网站输出结果**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
 <style>  
 .pghead{  
 background-color: #d4d4d5;  
 height:150px;  
 }  
 .pic{  
 position: relative;  
 background-color: white;  
 margin: 0 auto;  
 width: 800px;  
 height: 600px;  
 float:left;  
 }  
 .re{  
 position: relative;  
 background-color: white;  
 margin: 0 auto;  
 width: 200px;  
 height: 600px;  
 float:left;  
 }  
 </style>  
</head>  
  
<body style="margin:0 auto">  
 <div class="pghead">  
 <div style="float:left;text-indent: 350px;padding-top: 30px;font-size: 26pt">软件课设项目</div>  
 <div style="float:left;text-indent: 100px;padding-top: 100px;font-size: 16pt">——基于深度学习的CIFAR-10图片识别</div>  
 <div style="clear:both"></div>  
 </div>  
  
 <div class="pic">  
 <div style="margin: 0 auto;text-indent: 70px">  
 <h1>识别结果</h1><br>  
 <a style="font-size:150px;margin-top=30px">  
 {{ result }}  
 </a>  
 </div>  
 </div>  
  
 <div class="re">  
 <a href="/result" style="font-size:50px;text-indent: 700px;padding-top: 130px;margin:0 auto">继续识别</a>  
 </div>  
  
</body>  
<html>

1. **前端后端交互的构建**

# \_\_author\_\_ = 'xiongjiaheng'  
# -\*- coding:utf-8 -\*-  
**import** tornado.ioloop  
**import** tornado.web  
**import** tornado.httpserver  
**import** os  
**from** cnn\_keras\_predict **import** transimg, predict  
**import** numpy **as** np  
**from** keras.models **import** Model, load\_model  
**from** keras.preprocessing **import** image  
**from** PIL **import** Image  
**import** matplotlib.pyplot **as** plt  
**import** matplotlib.image **as** mpimg  
**from** tornado.options **import** define, options, parse\_command\_line  
  
define('port', default=8210, type=int)  
  
  
**class MainHandler**(tornado.web.RequestHandler):  
 **def get**(self):  
 self.render('index.html ')  
  
 **def post**(self, \*args, \*\*kwargs):  
  
 file\_metas = self.request.files["pic"]  
 # print(file\_metas)  
 **for** meta **in** file\_metas:  
 file\_name = meta['filename']  
  
 **with** open('./images/' + file\_name, 'wb') **as** up:  
 up.write(meta['body'])  
  
 img\_path = './images/' + file\_name  
  
 **if** transimg(img\_path) **is not False**:  
 img\_path = str(transimg(img\_path))  
 lena = mpimg.imread(img\_path) # 读取和代码处于同一目录下的 lena.png  
 # 此时 lena 就已经是一个 np.array 了，可以对它进行任意处理  
 plt.imshow(lena) # 显示图片  
 plt.axis('off') # 不显示坐标轴  
 plt.show()  
 **else**:  
 self.write("图片格式错误")  
 result = predict(img\_path)  
 self.render('result.html', result=result)  
  
  
**class ReturnHandler**(tornado.web.RequestHandler):  
 **def get**(self, \*args, \*\*kwargs):  
 self.render('index.html ')  
  
  
**def make\_app**():  
 **return** tornado.web.Application(handlers=[  
 (r'/', MainHandler),  
 (r'/result', ReturnHandler),  
 ],  
 template\_path=os.path.join(os.path.dirname(os.path.abspath(\_\_file\_\_)), 'templates'),  
 static\_path=os.path.join(os.path.dirname(os.path.abspath(\_\_file\_\_)), 'static'),  
 cookie\_secret='agdfiuwetr9w4689rfhjdc'  
 )  
  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 parse\_command\_line()  
 app = make\_app()  
 app.listen(options.port)  
 tornado.ioloop.IOLoop.current().start()

1. **其他用到的资料代码**

此处便不一一展示，源码展示出会把资料代码写出来。

1. **使用说明**

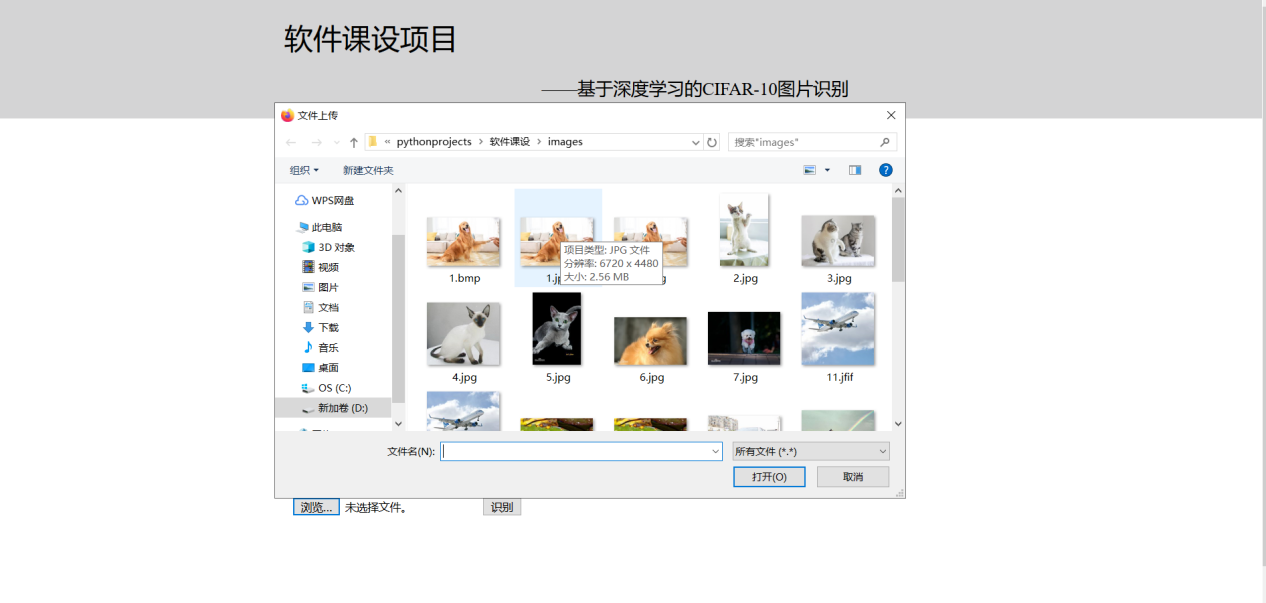
我们构建了一个网站，有一个良好的人机交互方式。网站有一个很方便输入图片的界面，点击“识别”后，在本机服务器上运行代码后，会在网站上输出结果，点击“继续识别”可退回原网站识别下一张图片。

下面是使用演示：

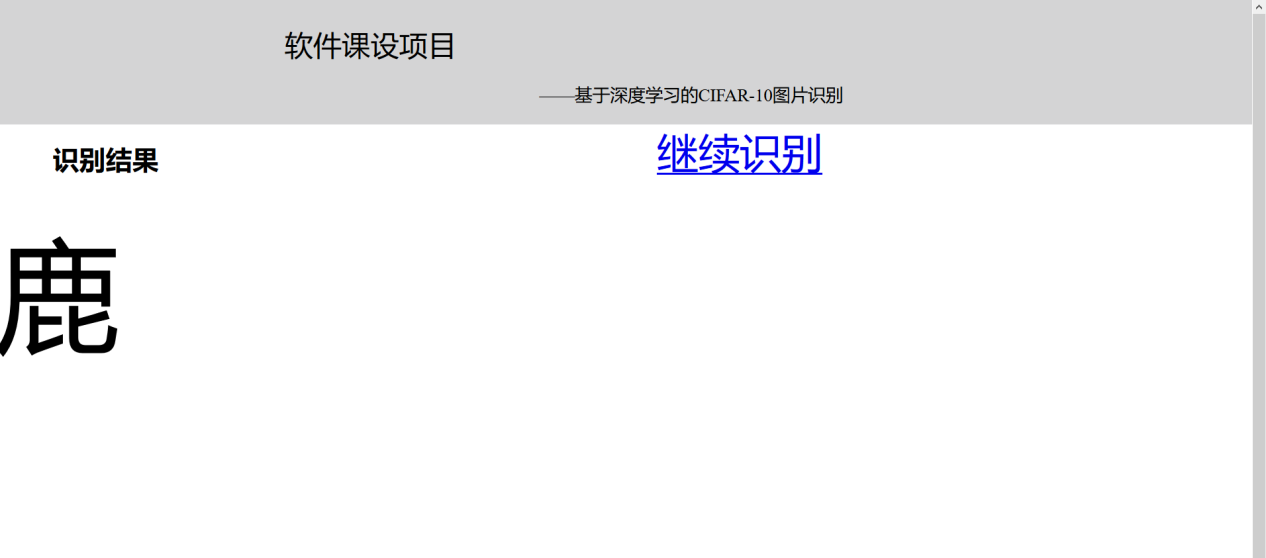
1. 网站示意图



1. 演示



1. 识别结果



1. **系统评价**

人机交互方式简单易于操作，但系统功能稍微有些单一，仅能进行图片识别，且对于某些图片的识别准确率不高。但毕竟是第一次尝试深度学习，且由于之前的考试，学习时间并没有那么长，能够在短短两个星期使用keras框架成功实现图片识别，并且能够自行搭建网站实现前端和后端的交互，实属不易。

**附录A源程序:**

1. **keras框架下的残差网络实现训练：cnn\_keras.py**

**import** numpy **as** np  
**from** keras.layers **import** Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization, Flatten, Conv2D, \  
 AveragePooling2D, MaxPooling2D  
**from** keras.models **import** Model, load\_model  
**from** keras.initializers **import** glorot\_uniform  
**import** os  
**import** keras.backend **as** K  
K.set\_image\_data\_format('channels\_last')  
K.set\_learning\_phase(1)  
  
**import** resnets\_utils  
  
  
**def identity\_block**(X, f, filters, stage, block):  
 *"""  
 实现恒等块  
  
 参数：  
 X - 输入的tensor类型的数据，维度为( m, n\_H\_prev, n\_W\_prev, n\_H\_prev )  
 f - 整数，指定主路径中间的CONV窗口的维度  
 filters - 整数列表，定义了主路径每层的卷积层的过滤器数量  
 stage - 整数，根据每层的位置来命名每一层，与block参数一起使用。  
 block - 字符串，据每层的位置来命名每一层，与stage参数一起使用。  
  
 返回：  
 X - 恒等块的输出，tensor类型，维度为(n\_H, n\_W, n\_C)  
  
 """* # 定义命名规则  
 conv\_name\_base = "res" + str(stage) + block + "\_branch"  
 bn\_name\_base = "bn" + str(stage) + block + "\_branch"  
  
 # 获取过滤器  
 F1, F2, F3 = filters  
  
 # 保存输入数据，将会用于为主路径添加捷径  
 X\_shortcut = X  
  
 # 主路径的第一部分  
 ##卷积层  
 X = Conv2D(filters=F1, kernel\_size=(1, 1), strides=(1, 1), padding="valid",  
 name=conv\_name\_base + "2a", kernel\_initializer=glorot\_uniform(seed=0))(X)  
 ##归一化  
 X = BatchNormalization(axis=3, name=bn\_name\_base + "2a")(X)  
 ##使用ReLU激活函数  
 X = Activation("relu")(X)  
  
 # 主路径的第二部分  
 ##卷积层  
 X = Conv2D(filters=F2, kernel\_size=(f, f), strides=(1, 1), padding="same",  
 name=conv\_name\_base + "2b", kernel\_initializer=glorot\_uniform(seed=0))(X)  
 ##归一化  
 X = BatchNormalization(axis=3, name=bn\_name\_base + "2b")(X)  
 ##使用ReLU激活函数  
 X = Activation("relu")(X)  
  
  
 # 主路径的第三部分  
 ##卷积层  
 X = Conv2D(filters=F3, kernel\_size=(1, 1), strides=(1, 1), padding="valid",  
 name=conv\_name\_base + "2c", kernel\_initializer=glorot\_uniform(seed=0))(X)  
 ##归一化  
 X = BatchNormalization(axis=3, name=bn\_name\_base + "2c")(X)  
 ##没有ReLU激活函数  
  
 # 最后一步：  
 ##将捷径与输入加在一起  
 X = Add()([X, X\_shortcut])  
 ##使用ReLU激活函数  
 X = Activation("relu")(X)  
  
 **return** X  
  
  
**def convolutional\_block**(X, f, filters, stage, block, s=2):  
 *"""  
 实现卷积块  
  
 参数：  
 X - 输入的tensor类型的变量，维度为( m, n\_H\_prev, n\_W\_prev, n\_C\_prev)  
 f - 整数，指定主路径中间的CONV窗口的维度  
 filters - 整数列表，定义了主路径每层的卷积层的过滤器数量  
 stage - 整数，根据每层的位置来命名每一层，与block参数一起使用。  
 block - 字符串，据每层的位置来命名每一层，与stage参数一起使用。  
 s - 整数，指定要使用的步幅  
  
 返回：  
 X - 卷积块的输出，tensor类型，维度为(n\_H, n\_W, n\_C)  
 """* # 定义命名规则  
 conv\_name\_base = "res" + str(stage) + block + "\_branch"  
 bn\_name\_base = "bn" + str(stage) + block + "\_branch"  
  
 # 获取过滤器数量  
 F1, F2, F3 = filters  
  
 # 保存输入数据  
 X\_shortcut = X  
  
 # 主路径  
 ##主路径第一部分  
 X = Conv2D(filters=F1, kernel\_size=(1, 1), strides=(s, s), padding="valid",  
 name=conv\_name\_base + "2a", kernel\_initializer=glorot\_uniform(seed=0))(X)  
 X = BatchNormalization(axis=3, name=bn\_name\_base + "2a")(X)  
 X = Activation("relu")(X)  
  
 ##主路径第二部分  
 X = Conv2D(filters=F2, kernel\_size=(f, f), strides=(1, 1), padding="same",  
 name=conv\_name\_base + "2b", kernel\_initializer=glorot\_uniform(seed=0))(X)  
 X = BatchNormalization(axis=3, name=bn\_name\_base + "2b")(X)  
 X = Activation("relu")(X)  
  
 ##主路径第三部分  
 X = Conv2D(filters=F3, kernel\_size=(1, 1), strides=(1, 1), padding="valid",  
 name=conv\_name\_base + "2c", kernel\_initializer=glorot\_uniform(seed=0))(X)  
 X = BatchNormalization(axis=3, name=bn\_name\_base + "2c")(X)  
  
 # 捷径  
 X\_shortcut = Conv2D(filters=F3, kernel\_size=(1, 1), strides=(s, s), padding="valid",  
 name=conv\_name\_base + "1", kernel\_initializer=glorot\_uniform(seed=0))(X\_shortcut)  
 X\_shortcut = BatchNormalization(axis=3, name=bn\_name\_base + "1")(X\_shortcut)  
  
 # 最后一步  
 X = Add()([X, X\_shortcut])  
 X = Activation("relu")(X)  
  
 **return** X  
  
  
**def ResNet50**(input\_shape=(32, 32, 3), classes=10):  
 *"""  
 实现ResNet50  
 CONV2D -> BATCHNORM -> RELU -> MAXPOOL -> CONVBLOCK -> IDBLOCK\*2 -> CONVBLOCK -> IDBLOCK\*3  
 -> CONVBLOCK -> IDBLOCK\*5 -> CONVBLOCK -> IDBLOCK\*2 -> AVGPOOL -> TOPLAYER  
  
 参数：  
 input\_shape - 图像数据集的维度  
 classes - 整数，分类数  
  
 返回：  
 model - Keras框架的模型  
  
 """* # 定义tensor类型的输入数据  
 X\_input = Input(input\_shape)  
  
 # 0填充  
 X = ZeroPadding2D((3, 3))(X\_input)  
  
 # stage1  
 X = Conv2D(filters=64, kernel\_size=(7, 7), strides=(2, 2), name="conv1",  
 kernel\_initializer=glorot\_uniform(seed=0))(X)  
 X = BatchNormalization(axis=3, name="bn\_conv1")(X)  
 X = Activation("relu")(X)  
 X = MaxPooling2D(pool\_size=(3, 3), strides=(2, 2))(X)  
  
 # stage2  
 X = convolutional\_block(X, f=3, filters=[64, 64, 256], stage=2, block="a", s=1)  
 X = identity\_block(X, f=3, filters=[64, 64, 256], stage=2, block="b")  
 X = identity\_block(X, f=3, filters=[64, 64, 256], stage=2, block="c")  
  
 # stage3  
 X = convolutional\_block(X, f=3, filters=[128, 128, 512], stage=3, block="a", s=2)  
 X = identity\_block(X, f=3, filters=[128, 128, 512], stage=3, block="b")  
 X = identity\_block(X, f=3, filters=[128, 128, 512], stage=3, block="c")  
 X = identity\_block(X, f=3, filters=[128, 128, 512], stage=3, block="d")  
  
 # stage4  
 X = convolutional\_block(X, f=3, filters=[256, 256, 1024], stage=4, block="a", s=2)  
 X = identity\_block(X, f=3, filters=[256, 256, 1024], stage=4, block="b")  
 X = identity\_block(X, f=3, filters=[256, 256, 1024], stage=4, block="c")  
 X = identity\_block(X, f=3, filters=[256, 256, 1024], stage=4, block="d")  
 X = identity\_block(X, f=3, filters=[256, 256, 1024], stage=4, block="e")  
 X = identity\_block(X, f=3, filters=[256, 256, 1024], stage=4, block="f")  
  
 # stage5  
 X = convolutional\_block(X, f=3, filters=[512, 512, 2048], stage=5, block="a", s=2)  
 X = identity\_block(X, f=3, filters=[512, 512, 2048], stage=5, block="b")  
 X = identity\_block(X, f=3, filters=[512, 512, 2048], stage=5, block="c")  
  
 # 均值池化层  
 X = AveragePooling2D(pool\_size=(2, 2), padding="same")(X)  
  
 # 输出层  
 X = Flatten()(X)  
 X = Dense(classes, activation="softmax", name="fc" + str(classes),  
 kernel\_initializer=glorot\_uniform(seed=0))(X)  
  
  
 # 创建模型  
 model = Model(inputs=X\_input, outputs=X, name="ResNet50")  
  
 **return** model  
  
  
  
model = ResNet50(input\_shape=(32, 32, 3), classes=10)  
  
model.compile(optimizer="adam", loss="categorical\_crossentropy", metrics=["accuracy"])  
  
  
**def load\_cifar10**(path):  
 train\_data = np.ones((1, 3073), dtype=np.uint8)  
  
 # 将训练数据进行拼接  
 **for** i **in** range(1, 6):  
 data\_path = os.path.join(path, 'data\_batch\_' + str(i) + '.bin')  
  
 **with** open(data\_path, 'rb') **as** f:  
 data = np.fromfile(f, dtype=np.uint8).reshape(10000, 3073)  
 train\_data = np.concatenate((train\_data, data))  
  
 # 调整训练数据维度,原因见cifar10的数据说明  
 train\_labels = train\_data[1:, :1]  
 train\_images = train\_data[1:, 1:].reshape(50000, 3, 32, 32).transpose(0, 2, 3, 1)  
  
 # 获取训练数据  
 data\_path = os.path.join(path, 'test\_batch.bin')  
 **with** open(data\_path, 'rb') **as** f:  
 test\_data = np.fromfile(data\_path, dtype=np.uint8).reshape(10000, 3073)  
 test\_labels = test\_data[:, :1]  
 test\_images = test\_data[:, 1:].reshape(10000, 3, 32, 32).transpose(0, 2, 3, 1)  
  
 **return** (train\_images, train\_labels, test\_images, test\_labels)  
  
  
X\_train\_orig, Y\_train\_orig, X\_test\_orig, Y\_test\_orig = load\_cifar10("./datasets/")  
  
# Normalize image vectors  
X\_train = X\_train\_orig / 255.  
X\_test = X\_test\_orig / 255.  
  
# Convert training and test labels to one hot matrices  
Y\_train = resnets\_utils.convert\_to\_one\_hot(Y\_train\_orig, 10).T  
Y\_test = resnets\_utils.convert\_to\_one\_hot(Y\_test\_orig, 10).T  
  
print("number of training examples = " + str(X\_train.shape[0]))  
print("number of test examples = " + str(X\_test.shape[0]))  
print("X\_train shape: " + str(X\_train.shape))  
print("Y\_train shape: " + str(Y\_train.shape))  
print("X\_test shape: " + str(X\_test.shape))  
print("Y\_test shape: " + str(Y\_test.shape))  
  
  
model.fit(X\_train, Y\_train, epochs=120, batch\_size=1000)  
preds = model.evaluate(X\_test, Y\_test)  
print("测试集误差值 = " + str(preds[0]))  
print("测试集准确率 = " + str(preds[1]))  
  
model.save('ResNet50.h5')

1. **图片的导入和结果输出：cnn\_keras\_predict.py**

**from** PIL **import** Image  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt # plt 用于显示图片  
**import** numpy **as** np  
**import** tensorflow **as** tf  
  
**from** keras **import** layers  
**from** keras.layers **import** Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization, Flatten, Conv2D, \  
 AveragePooling2D, MaxPooling2D, GlobalMaxPooling2D  
**from** keras.models **import** Model, load\_model  
**from** keras.preprocessing **import** image  
**from** keras.utils **import** layer\_utils  
**from** keras.utils.data\_utils **import** get\_file  
**from** keras.applications.imagenet\_utils **import** preprocess\_input  
**from** keras.utils.vis\_utils **import** model\_to\_dot  
**from** keras.utils **import** plot\_model  
**from** keras.initializers **import** glorot\_uniform  
**import** os  
**from** PIL **import** Image  
**import** matplotlib.image **as** mpimg  
**import** scipy  
  
  
  
**def IsValidImage**(img\_path):  
 *"""  
 判断文件是否为有效（完整）的图片* ***:param*** *img\_path:图片路径* ***:return****:True：有效 False：无效  
 """* bValid = **True  
 try**:  
 Image.open(img\_path).verify()  
 **except**:  
 bValid = **False  
 return** bValid  
  
  
**def transimg**(img\_path):  
 *"""  
 转换图片格式* ***:param*** *img\_path:图片路径* ***:return****: True：成功 False：失败  
 """* **if** IsValidImage(img\_path):  
  
 str = img\_path.rsplit(".", 1)  
 output\_img\_path = str[0] + ".jpg"  
 print(output\_img\_path)  
 im = Image.open(img\_path)  
 im=im.convert('RGB')  
 im.save(output\_img\_path)  
 **return** output\_img\_path  
  
 **else**:  
 **return False  
  
  
  
  
  
def predict**(img\_path):  
 my\_image = image.load\_img(img\_path, target\_size=(32, 32))  
  
 my\_image = image.img\_to\_array(my\_image)  
 print("my\_image.shape = " + str(my\_image.shape))  
 my\_image = np.expand\_dims(my\_image,axis=0)/255  
  
  
  
 print("my\_image.shape = " + str(my\_image.shape))  
  
 print("class prediction vector = ")  
 model = load\_model('ResNet50.h5')  
  
 result=model.predict(my\_image)  
 classes = ['飞机', '汽车', '鸟', '猫', '鹿', '狗', '青蛙', '马', '船', '卡车']  
 index=np.argmax(result,1)  
 print(index)  
 result=classes[int(index)]  
 print("输入图像为：",classes[int(index)])  
  
  
  
 plot\_model(model, to\_file='model.png')  
 **return** result

1. **两个资料包:实现卷积的方式:1）resnets\_utils.py**

**import** os  
**import** numpy **as** np  
**import** tensorflow **as** tf  
**import** h5py  
**import** math  
**import** pickle  
  
  
**def load\_file**(filename):  
 **with** open(filename, 'rb') **as** fo:  
 data = pickle.load(fo, encoding='latin1')  
 **return** data  
  
  
**def load\_cifar10**(path):  
  
 train\_data=np.ones((1,3073),dtype=np.uint8)  
  
 #将训练数据进行拼接  
 **for** i **in** range(1,6):  
 data\_path=os.path.join(path,'data\_batch\_'+str(i)+'.bin')  
  
 **with** open(data\_path,'rb') **as** f:  
 data=np.fromfile(f,dtype=np.uint8).reshape(10000,3073)  
 train\_data=np.concatenate((train\_data,data))  
  
 #调整训练数据维度,原因见cifar10的数据说明  
 train\_labels=train\_data[1:,:1]  
 train\_images=train\_data[1:,1:].reshape(50000,3,32,32).transpose(0,2,3,1)  
  
 #获取训练数据  
 data\_path = os.path.join(path, 'test\_batch.bin')  
 **with** open(data\_path,'rb') **as** f:  
 test\_data=np.fromfile(data\_path,dtype=np.uint8).reshape(10000,3073)  
 test\_labels=test\_data[:,:1]  
 test\_images=test\_data[:,1:].reshape(10000,3,32,32).transpose(0,2,3,1)  
  
 **return** (train\_images,train\_labels,test\_images,test\_labels)  
  
'''  
def load\_dataset():  
 train\_dataset = h5py.File('datasets/train\_signs.h5', "r")  
 train\_set\_x\_orig = np.array(train\_dataset["train\_set\_x"][:]) # your train set features  
 train\_set\_y\_orig = np.array(train\_dataset["train\_set\_y"][:]) # your train set labels  
  
 test\_dataset = h5py.File('datasets/test\_signs.h5', "r")  
 test\_set\_x\_orig = np.array(test\_dataset["test\_set\_x"][:]) # your test set features  
 test\_set\_y\_orig = np.array(test\_dataset["test\_set\_y"][:]) # your test set labels  
  
 classes = np.array(test\_dataset["list\_classes"][:]) # the list of classes  
  
 train\_set\_y\_orig = train\_set\_y\_orig.reshape((1, train\_set\_y\_orig.shape[0]))  
 test\_set\_y\_orig = test\_set\_y\_orig.reshape((1, test\_set\_y\_orig.shape[0]))  
   
 return train\_set\_x\_orig, train\_set\_y\_orig, test\_set\_x\_orig, test\_set\_y\_orig, classes  
'''  
  
  
**def random\_mini\_batches**(X, Y, mini\_batch\_size = 64, seed = 0):  
 *"""  
 Creates a list of random minibatches from (X, Y)  
   
 Arguments:  
 X -- input data, of shape (input size, number of examples) (m, Hi, Wi, Ci)  
 Y -- true "label" vector (containing 0 if cat, 1 if non-cat), of shape (1, number of examples) (m, n\_y)  
 mini\_batch\_size - size of the mini-batches, integer  
 seed -- this is only for the purpose of grading, so that you're "random minibatches are the same as ours.  
   
 Returns:  
 mini\_batches -- list of synchronous (mini\_batch\_X, mini\_batch\_Y)  
 """* m = X.shape[0] # number of training examples  
 mini\_batches = []  
 np.random.seed(seed)  
   
 # Step 1: Shuffle (X, Y)  
 permutation = list(np.random.permutation(m))  
 shuffled\_X = X[permutation,:,:,:]  
 shuffled\_Y = Y[permutation,:]  
  
 # Step 2: Partition (shuffled\_X, shuffled\_Y). Minus the end case.  
 num\_complete\_minibatches = math.floor(m/mini\_batch\_size) # number of mini batches of size mini\_batch\_size in your partitionning  
 **for** k **in** range(0, num\_complete\_minibatches):  
 mini\_batch\_X = shuffled\_X[k \* mini\_batch\_size : k \* mini\_batch\_size + mini\_batch\_size,:,:,:]  
 mini\_batch\_Y = shuffled\_Y[k \* mini\_batch\_size : k \* mini\_batch\_size + mini\_batch\_size,:]  
 mini\_batch = (mini\_batch\_X, mini\_batch\_Y)  
 mini\_batches.append(mini\_batch)  
   
 # Handling the end case (last mini-batch < mini\_batch\_size)  
 **if** m % mini\_batch\_size != 0:  
 mini\_batch\_X = shuffled\_X[num\_complete\_minibatches \* mini\_batch\_size : m,:,:,:]  
 mini\_batch\_Y = shuffled\_Y[num\_complete\_minibatches \* mini\_batch\_size : m,:]  
 mini\_batch = (mini\_batch\_X, mini\_batch\_Y)  
 mini\_batches.append(mini\_batch)  
   
 **return** mini\_batches  
  
  
**def convert\_to\_one\_hot**(Y, C):  
 Y = np.eye(C)[Y.reshape(-1)].T  
 **return** Y  
  
  
**def forward\_propagation\_for\_predict**(X, parameters):  
 *"""  
 Implements the forward propagation for the model: LINEAR -> RELU -> LINEAR -> RELU -> LINEAR -> SOFTMAX  
   
 Arguments:  
 X -- input dataset placeholder, of shape (input size, number of examples)  
 parameters -- python dictionary containing your parameters "W1", "b1", "W2", "b2", "W3", "b3"  
 the shapes are given in initialize\_parameters  
 Returns:  
 Z3 -- the output of the last LINEAR unit  
 """* # Retrieve the parameters from the dictionary "parameters"   
 W1 = parameters['W1']  
 b1 = parameters['b1']  
 W2 = parameters['W2']  
 b2 = parameters['b2']  
 W3 = parameters['W3']  
 b3 = parameters['b3']   
 # Numpy Equivalents:  
 Z1 = tf.add(tf.matmul(W1, X), b1) # Z1 = np.dot(W1, X) + b1  
 A1 = tf.nn.relu(Z1) # A1 = relu(Z1)  
 Z2 = tf.add(tf.matmul(W2, A1), b2) # Z2 = np.dot(W2, a1) + b2  
 A2 = tf.nn.relu(Z2) # A2 = relu(Z2)  
 Z3 = tf.add(tf.matmul(W3, A2), b3) # Z3 = np.dot(W3,Z2) + b3  
   
 **return** Z3  
  
**def predict**(X, parameters):  
   
 W1 = tf.convert\_to\_tensor(parameters["W1"])  
 b1 = tf.convert\_to\_tensor(parameters["b1"])  
 W2 = tf.convert\_to\_tensor(parameters["W2"])  
 b2 = tf.convert\_to\_tensor(parameters["b2"])  
 W3 = tf.convert\_to\_tensor(parameters["W3"])  
 b3 = tf.convert\_to\_tensor(parameters["b3"])  
   
 params = {"W1": W1,  
 "b1": b1,  
 "W2": W2,  
 "b2": b2,  
 "W3": W3,  
 "b3": b3}  
   
 x = tf.placeholder("float", [12288, 1])  
   
 z3 = forward\_propagation\_for\_predict(x, params)  
 p = tf.argmax(z3)  
   
 sess = tf.Session()  
 prediction = sess.run(p, feed\_dict = {x: X})  
   
 **return** prediction

**2）kt\_utils.py：**

**import** keras.backend **as** K  
**import** math  
**import** numpy **as** np  
**import** h5py  
**import** matplotlib.pyplot **as** plt  
  
  
**def mean\_pred**(y\_true, y\_pred):  
 **return** K.mean(y\_pred)  
  
**def load\_dataset**():  
 train\_dataset = h5py.File('datasets/train\_happy.h5', "r")  
 train\_set\_x\_orig = np.array(train\_dataset["train\_set\_x"][:]) # your train set features  
 train\_set\_y\_orig = np.array(train\_dataset["train\_set\_y"][:]) # your train set labels  
  
 test\_dataset = h5py.File('datasets/test\_happy.h5', "r")  
 test\_set\_x\_orig = np.array(test\_dataset["test\_set\_x"][:]) # your test set features  
 test\_set\_y\_orig = np.array(test\_dataset["test\_set\_y"][:]) # your test set labels  
  
 classes = np.array(test\_dataset["list\_classes"][:]) # the list of classes  
   
 train\_set\_y\_orig = train\_set\_y\_orig.reshape((1, train\_set\_y\_orig.shape[0]))  
 test\_set\_y\_orig = test\_set\_y\_orig.reshape((1, test\_set\_y\_orig.shape[0]))  
   
 **return** train\_set\_x\_orig, train\_set\_y\_orig, test\_set\_x\_orig, test\_set\_y\_orig, classes

1. **前端后端交互：view.py**

**import** tornado.ioloop  
**import** tornado.web  
**import** tornado.httpserver  
**import** os  
**from** cnn\_keras\_predict **import** transimg, predict  
**import** numpy **as** np  
**from** keras.models **import** Model, load\_model  
**from** keras.preprocessing **import** image  
**from** PIL **import** Image  
**import** matplotlib.pyplot **as** plt  
**import** matplotlib.image **as** mpimg  
**from** tornado.options **import** define, options, parse\_command\_line  
  
define('port', default=8210, type=int)  
  
  
**class MainHandler**(tornado.web.RequestHandler):  
 **def get**(self):  
 self.render('index.html ')  
  
 **def post**(self, \*args, \*\*kwargs):  
  
 file\_metas = self.request.files["pic"]  
 # print(file\_metas)  
 **for** meta **in** file\_metas:  
 file\_name = meta['filename']  
  
 **with** open('./images/' + file\_name, 'wb') **as** up:  
 up.write(meta['body'])  
  
 img\_path = './images/' + file\_name  
  
 **if** transimg(img\_path) **is not False**:  
 img\_path = str(transimg(img\_path))  
 lena = mpimg.imread(img\_path) # 读取和代码处于同一目录下的 lena.png  
 # 此时 lena 就已经是一个 np.array 了，可以对它进行任意处理  
 plt.imshow(lena) # 显示图片  
 plt.axis('off') # 不显示坐标轴  
 plt.show()  
 **else**:  
 self.write("图片格式错误")  
 result = predict(img\_path)  
 self.render('result.html', result=result)  
  
  
**class ReturnHandler**(tornado.web.RequestHandler):  
 **def get**(self, \*args, \*\*kwargs):  
 self.render('index.html ')  
  
  
**def make\_app**():  
 **return** tornado.web.Application(handlers=[  
 (r'/', MainHandler),  
 (r'/result', ReturnHandler),  
 ],  
 template\_path=os.path.join(os.path.dirname(os.path.abspath(\_\_file\_\_)), 'templates'),  
 static\_path=os.path.join(os.path.dirname(os.path.abspath(\_\_file\_\_)), 'static'),  
 cookie\_secret='agdfiuwetr9w4689rfhjdc'  
 )  
  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 parse\_command\_line()  
 app = make\_app()  
 app.listen(options.port)  
 tornado.ioloop.IOLoop.current().start()

1. **网站的实现：index.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
 <style>  
 .pghead{  
 background-color: #d4d4d5;  
 height:150px;  
 }  
 .pic{  
 position: relative;  
 background-color: white;  
 margin: 10px auto;  
 width: 800px;  
 height: 600px;  
 }  
 </style>  
</head>  
  
  
<body style="margin:0 auto">  
 <div class="pghead">  
 <div style="float:left;text-indent: 350px;padding-top: 30px;font-size: 26pt">软件课设项目</div>  
 <div style="float:left;text-indent: 100px;padding-top: 100px;font-size: 16pt">——基于深度学习的CIFAR-10图片识别</div>  
 <div style="clear:both"></div>  
  
 </div>  
  
 <div class="pic">  
 <img src="" style=" height: 400px;">  
 <div>请选择图片上传</div><br>  
  
 <form id="my\_form" name="form" action="http://localhost:8210" method="POST" enctype="multipart/form-data" >  
 <input type="file" name="pic" id="my\_file" >  
 <input type="submit" value="识别">  
  
 </form>  
  
  
 <script>  
 **var** fileInput = document.querySelector('input[type=file]'),  
 previewImg = document.querySelector('img');  
 fileInput.addEventListener('change', **function** () {  
 **var** file = **this**.files[0];  
 **var** reader = **new** FileReader();  
 // 监听reader对象的的onload事件，当图片加载完成时，把base64编码賦值给预览图片  
 reader.addEventListener("load", **function** () {  
 previewImg.src = reader.result;  
 }, **false**);  
 // 调用reader.readAsDataURL()方法，把图片转成base64  
 reader.readAsDataURL(file);  
 }, **false**);  
 </script>  
  
 </div>  
  
  
  
</body>  
</html>

1. **网站输出结果：result.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
 <style>  
 .pghead{  
 background-color: #d4d4d5;  
 height:150px;  
 }  
 .pic{  
 position: relative;  
 background-color: white;  
 margin: 0 auto;  
 width: 800px;  
 height: 600px;  
 float:left;  
 }  
 .re{  
 position: relative;  
 background-color: white;  
 margin: 0 auto;  
 width: 200px;  
 height: 600px;  
 float:left;  
 }  
 </style>  
</head>  
  
<body style="margin:0 auto">  
 <div class="pghead">  
 <div style="float:left;text-indent: 350px;padding-top: 30px;font-size: 26pt">软件课设项目</div>  
 <div style="float:left;text-indent: 100px;padding-top: 100px;font-size: 16pt">——基于深度学习的CIFAR-10图片识别</div>  
 <div style="clear:both"></div>  
 </div>  
  
 <div class="pic">  
 <div style="margin: 0 auto;text-indent: 70px">  
 <h1>识别结果</h1><br>  
 <a style="font-size:150px;margin-top=30px">  
 {{ result }}  
 </a>  
 </div>  
 </div>  
  
 <div class="re">  
 <a href="/result" style="font-size:50px;text-indent: 700px;padding-top: 130px;margin:0 auto">继续识别</a>  
 </div>  
  
</body>  
<html>