****

**本 科 生 课 程 论 文**

**(2020-2021学年第一学期)**

**基于卷积神经网络模型的猫狗图像识别**

**姓名：杨程宇**

**提交日期：2021年1月30日 签名：杨程宇**

|  |  |  |  |
| --- | --- | --- | --- |
| **学 号** | **201730092420** | **学 院** | **数学学院** |
| **课程编号** | **040102271** | **课程名称** | **大数据应用** |
| **课程类别** | **选修** | **任课教师** | **曾德炉** |
| **教师评语：** | | | |
| **成绩评定： 分 任课教师签名： 年 月 日** | | | |

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

**基与卷积神经网络模型的猫狗图像识别**

1. **摘要**

猫和狗在外观上的差异通过肉眼很容易识别，本文运用猫狗数据集训练出卷积神经网络模型，并使猫狗识别的准确率达到90%以上。本文同时包括卷积神经神经网络的理论理解，算法实现以及实验数据分析。

1. **动机**

在日常生活中，猫和狗在外观上的差异是比较明显的，无论是体型，四肢，脸孔和毛发等等，通过人们的肉眼就能识别出来。那么如何让机器来识别猫和狗呢？本文将运用Tensorflow搭建一个卷积神经网络模型,用自家的狗和猫来进行最后的测试。这是深度学习的典型案例。

1. **理论和算法理解**

**I.卷积神经网络**

卷积神经网络（Convolutional Neural Network,CNN)是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对大型图像处理有出色表现。卷积神经网络由一个或多个卷积层和顶端的全连通层（对应经典的神经网络）组成，同时也包括关联权重和池化层（pooling layer）。这一结构使得卷积神经网络能够利用输入数据的二维结构。与其他深度学习结构相比，卷积神经网络在图像和语音识别方面能够给出更好的结果。这一模型也可以使用反向传播算法进行训练。相比较其他深度、前馈神经网络，卷积神经网络需要考量的参数更少，使之成为一种颇具吸引力的深度学习结构。

1. 定义

“卷积神经网络”表示网络采用称为卷积的数学运算。卷积是一种特殊的线性操作。卷积网络是一种特殊的神经网络，它们在至少一个层中使用卷积代替一般矩阵乘法。

1. 结构
2. 卷积层

卷积层是一组平行的特征图（feature map），它通过在输入图像上滑动不同的卷积核并运行一定的运算而组成。此外，在每一个滑动的位置上，卷积核与输入图像之间会运行一个元素对应乘积并求和的运算以将感受野内的信息投影到特征图中的一个元素。这一滑动的过程可称为步幅 Z\_s，步幅 Z\_s 是控制输出特征图尺寸的一个因素。卷积核的尺寸要比输入图像小得多，且重叠或平行地作用于输入图像中，一张特征图中的所有元素都是通过一个卷积核计算得出的，也即一张特征图共享了相同的权重和偏置项。

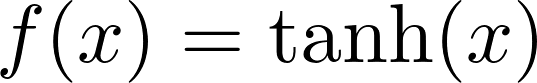
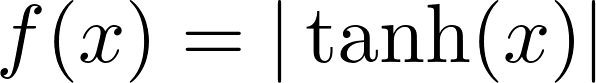
1. 线性整流层

线性整流层（Rectified Linear Units layer, ReLU layer）使用[线性整流](https://zh.wikipedia.org/wiki/%E7%BA%BF%E6%80%A7%E6%95%B4%E6%B5%81%E5%87%BD%E6%95%B0" \o "线性整流函数)（Rectified Linear Units, ReLU）

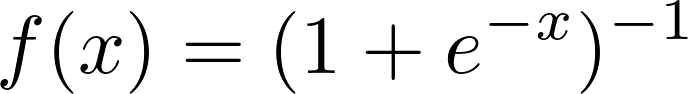


作为这一层神经的激励函数（Activation function）。它可以增强判定函数和整个神经网络的非线性特性，而本身并不会改变卷积层。

事实上，其他的一些函数也可以用于增强网络的非线性特性，如[双曲正切函数](https://zh.wikipedia.org/wiki/%E5%8F%8C%E6%9B%B2%E6%AD%A3%E5%88%87%E5%87%BD%E6%95%B0" \o "双曲正切函数)

， ，

或者[Sigmoid函数](https://zh.wikipedia.org/wiki/S%E5%87%BD%E6%95%B0" \o "S函数)

。

相比其它函数来说，ReLU函数更受青睐，这是因为它可以将神经网络的训练速度提升数倍，而并不会对模型的泛化准确度造成显著影响。

（三）池化层

池化（Pooling）是卷积神经网络中另一个重要的概念，它实际上是一种非线性形式的降采样。有多种不同形式的非线性池化函数，而其中“最大池化（Max pooling）”是最为常见的。它是将输入的图像划分为若干个矩形区域，对每个子区域输出最大值。

直觉上，这种机制能够有效地原因在于，一个特征的精确位置远不及它相对于其他特征的粗略位置重要。池化层会不断地减小数据的空间大小，因此参数的数量和计算量也会下降，这在一定程度上也控制了[过拟合](https://zh.wikipedia.org/wiki/%E8%BF%87%E6%8B%9F%E5%90%88" \o "过拟合)。通常来说，CNN的网络结构中的卷积层之间都会周期性地插入池化层。池化操作提供了另一种形式的平移不变性。因为卷积核是一种特征发现器，我们通过卷积层可以很容易地发现图像中的各种边缘。但是卷积层发现的特征往往过于精确，我们即使高速连拍拍摄一个物体，照片中的物体的边缘像素位置也不大可能完全一致，通过池化层我们可以降低卷积层对边缘的敏感性。

池化层每次在一个池化窗口（depth slice）上计算输出，然后根据步幅移动池化窗口。下图是目前最常用的池化层，步幅为2，池化窗口为2\*2的二维最大池化层。每隔2个元素从图像划分出2\*2的区块，然后对每个区块中的4个数取最大值。这将会减少75%的数据量。



图1 步幅为2， 池化窗口为2\*2的最大池化层

1. 完全连接层

最后，在经过几个卷积和最大池化层之后，神经网络中的高级推理通过完全连接层来完成。就和常规的非卷积人工神经网络中一样，完全连接层中的神经元与前一层中的所有激活都有联系。因此，它们的激活可以作为[仿射变换](https://zh.wikipedia.org/wiki/%E4%BB%BF%E5%B0%84%E5%8F%98%E6%8D%A2" \o "仿射变换)来计算，也就是先乘以一个矩阵然后加上一个偏差(bias)偏移量(向量加上一个固定的或者学习来的偏差量)。

1. 应用

卷积神经网络长期以来是图像识别领域的核心算法之一，并在学习数据充足时有稳定的表现。对于一般的大规模图像分类问题，卷积神经网络可用于构建阶层分类器（hierarchical classifier),也可以在精细分类识别（fine-grained recognition）中用于提取图像的判别特征以供其它分类器进行学习。对于后者，特征提取可以认为地将图像的不同部分分别输入卷积神经网络，也可以由卷积神经网络通过非监督学习自行提取。

对于字符检测（text detection）和字符识别（text recognition）/光学字符读取，卷积神经网络被用于判断输入的图像是否包含字符，并从中剪取有效的字符片断。其中使用多个归一化指数函数直接分类的卷积神经网络被用于谷歌街景图像的门牌号识别、包含条件随机场（Conditional Random Fields，CRF）图模型的卷积神经网络可识别图像中的单词，卷积神经网络与循环神经网络（Recurrent Neural Network，RNN）相结合可以分别从图像中提取字符特征和进行序列标注（Sequence labelling）。

**II.算法实现**

**Importing the libraries**

*import tensorflow as tf*

*from keras.preprocessing.image import ImageDataGenerator*

*tf.\_\_version\_\_*

*'2.3.1'*

## **Part 1 - Data Preprocessing**

**Preprocessing the Training set**

*train\_datagen = ImageDataGenerator(rescale = 1./255,*

*shear\_range = 0.2,*

*zoom\_range = 0.2,*

*horizontal\_flip = True)*

*training\_set = train\_datagen.flow\_from\_directory('dataset/training\_set',*

*target\_size = (64, 64),*

*batch\_size = 32,*

*class\_mode = 'binary')*

*Found 20000 images belonging to 2 classes.*

## **Preprocessing the Test set**

*test\_datagen = ImageDataGenerator(rescale = 1./255)*

*test\_set = test\_datagen.flow\_from\_directory('dataset/test\_set',*

*target\_size = (64, 64),*

*batch\_size = 32,*

*class\_mode = 'binary')*

*Found 5000 images belonging to 2 classes.*

## **Part 2 - Building the CNN**

## **Initialising the CNN**

*cnn = tf.keras.models.Sequential()*

## **Step 1 - Convolution**

*cnn.add(tf.keras.layers.Conv2D(filters=32, kernel\_size=3, activation='relu', input\_shape=[64, 64, 3]))*

## **Step 2 - Pooling**

*cnn.add(tf.keras.layers.MaxPool2D(pool\_size=2, strides=2))*

## **Adding a second convolutional layer**

*cnn.add(tf.keras.layers.Conv2D(filters=32, kernel\_size=3, activation='relu'))*

*cnn.add(tf.keras.layers.MaxPool2D(pool\_size=2, strides=2))*

## **Step 3 - Flattening**

*cnn.add(tf.keras.layers.Flatten())*

## **Step 4 - Full Connection**

*cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))*

## **Step 5 - Output Layer**

*cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))*

## **Part 3 - Training the CNN**

## **Compiling the CNN**

*cnn.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])*

## **Training the CNN on the Training set and evaluating it on the Test set**

*cnn.fit(x = training\_set, validation\_data = test\_set, epochs = 25)*

## **Part 4 - Making a single prediction**

*import numpy as np*

*from keras.preprocessing import image*

*import glob as glob*

*import os*

*test\_images = glob.glob('dataset/single\_prediction/\*.jpg')*

*for image\_src in test\_images:*

*test\_image = image.load\_img(image\_src, target\_size = (64, 64))*

*test\_image = image.img\_to\_array(test\_image)*

*test\_image = np.expand\_dims(test\_image, axis = 0)*

*result = cnn.predict(test\_image)*

*training\_set.class\_indices*

*if result[0][0] == 1:*

*prediction = 'dog'*

*else:*

*prediction = 'cat'*

*img\_dir,img\_fn = os.path.split(image\_src)*

*print(img\_fn+":"+prediction)*

1. **实验数据分析与总结**

实验数据：

训练集包含有10000张猫的图片和10000张狗的图片：

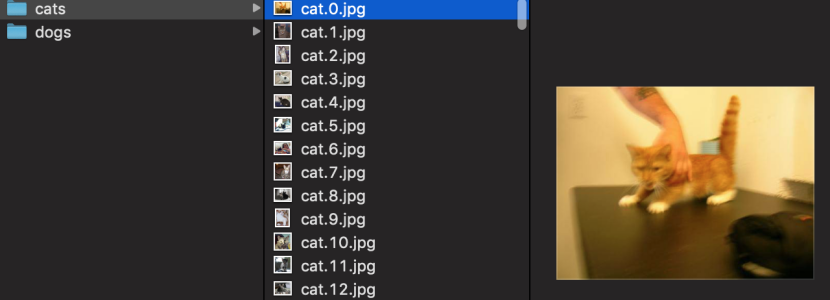


图2 猫的训练数据集

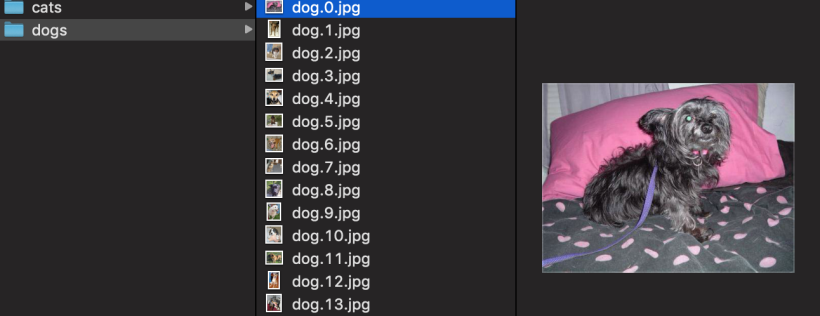


图3 狗的训练数据集

测试集包含有2500张猫的图片和2500张狗的图片：

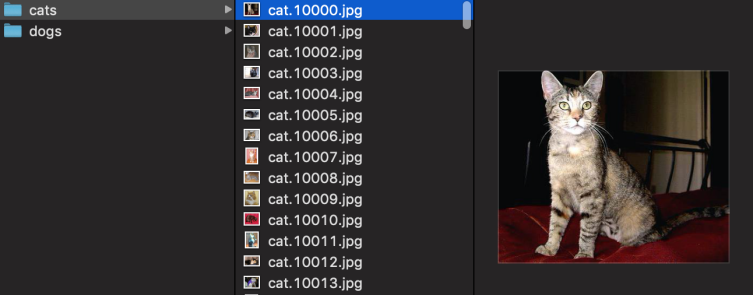


图4 猫的测试数据集

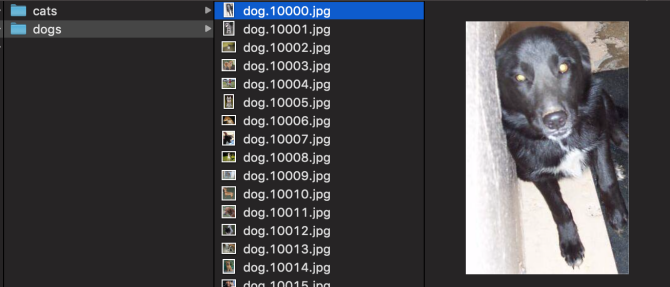


图5 狗的测试数据集

该模型一共经历了25次epoch，每次得到的结果如下：

Epoch 1/25 - loss: 0.6097 - accuracy: 0.6568 - val\_loss: 0.5094 - val\_accuracy: 0.7444

Epoch 2/25 - loss: 0.5243 - accuracy: 0.7383 - val\_loss: 0.4784 - val\_accuracy: 0.7710

Epoch 3/25 - loss: 0.4809 - accuracy: 0.7675 - val\_loss: 0.4444 - val\_accuracy: 0.7876

Epoch 4/25 - loss: 0.4526 - accuracy: 0.7847 - val\_loss: 0.4318 - val\_accuracy: 0.7958

Epoch 5/25 - loss: 0.4347 - accuracy: 0.7994 - val\_loss: 0.4432 - val\_accuracy: 0.7972

Epoch 6/25 - loss: 0.4157 - accuracy: 0.8087 - val\_loss: 0.4037 - val\_accuracy: 0.8120

Epoch 7/25 - loss: 0.4074 - accuracy: 0.8113 - val\_loss: 0.4402 - val\_accuracy: 0.8060

Epoch 8/25 - loss: 0.3919 - accuracy: 0.8202 - val\_loss: 0.4139 - val\_accuracy: 0.8114

Epoch 9/25 - loss: 0.3787 - accuracy: 0.8265 - val\_loss: 0.4091 - val\_accuracy: 0.8152

Epoch 10/25 - loss: 0.3652 - accuracy: 0.8327 - val\_loss: 0.4318 - val\_accuracy: 0.8130

Epoch 11/25 - loss: 0.3536 - accuracy: 0.8438 - val\_loss: 0.4175 - val\_accuracy: 0.8210

Epoch 12/25 - loss: 0.3400 - accuracy: 0.8501 - val\_loss: 0.3987 - val\_accuracy: 0.8286

Epoch 13/25 - loss: 0.3304 - accuracy: 0.8554 - val\_loss: 0.3908 - val\_accuracy: 0.8318

Epoch 14/25 - loss: 0.3181 - accuracy: 0.8583 - val\_loss: 0.4384 - val\_accuracy: 0.8132

Epoch 15/25 - loss: 0.3166 - accuracy: 0.8605 - val\_loss: 0.3779 - val\_accuracy: 0.8330

Epoch 16/25 - loss: 0.3029 - accuracy: 0.8702 - val\_loss: 0.4222 - val\_accuracy: 0.8168

Epoch 17/25 - loss: 0.2907 - accuracy: 0.8762 - val\_loss: 0.4169 - val\_accuracy: 0.8308

Epoch 18/25 - loss: 0.2814 - accuracy: 0.8779 - val\_loss: 0.4146 - val\_accuracy: 0.8228

Epoch 19/25 - loss: 0.2749 - accuracy: 0.8838 - val\_loss: 0.4410 - val\_accuracy: 0.8132

Epoch 20/25 - loss: 0.2649 - accuracy: 0.8892 - val\_loss: 0.5012 - val\_accuracy: 0.7988

Epoch 21/25 - loss: 0.2552 - accuracy: 0.8914 - val\_loss: 0.4090 - val\_accuracy: 0.8352

Epoch 22/25 - loss: 0.2474 - accuracy: 0.8954 - val\_loss: 0.4118 - val\_accuracy: 0.8306

Epoch 23/25 - loss: 0.2368 - accuracy: 0.9017 - val\_loss: 0.4550 - val\_accuracy: 0.8294

Epoch 24/25 - loss: 0.2274 - accuracy: 0.9051 - val\_loss: 0.4698 - val\_accuracy: 0.8172

Epoch 25/25 - loss: 0.2216 - accuracy: 0.9096 - val\_loss: 0.4271 - val\_accuracy: 0.8382

准确率随次数的变化图像为：

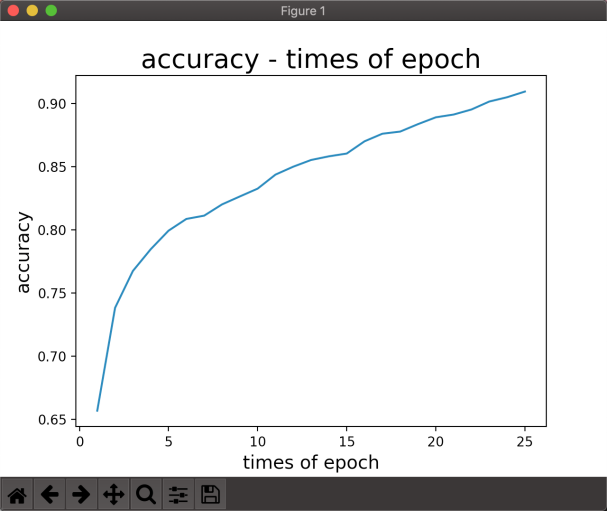


图5 准确率随epoch次数的变化规律

单点测试数据：

自家的猫和狗：

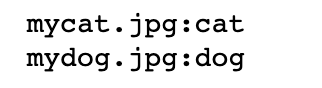


图6 mydog.jpg

****

图7 mycat.jpg

测试结果：准确。



1. **心得体会**

通过这次大数据应用的课程设计，我初步了解了大数据处理和分析的方法。并选择了CNN模型来完成此次的课程设计。

在CNN之前，图像对于人工智能来说是一个难题，主要有2个原因：

1. 图像需要处理的数据量太大，导致成本很高。
2. 图像在数字化的过程中很难保留原有的特征，导致图像处理的准确率不高。

而CNN受到人类视觉神经系统的启发，有两大特点：

1. 能够有效地将大数据量的图片降维成小数据量。
2. 能够有效地保留图片特征，符合图片处理的原则。

但通过本次课程偶也发现仍然存在很多不足之处。虽然实现了用卷积神经网络模型来进行猫狗的图像识别，但更多的只是利用开源代码库tensorflow提供的API，并对参数进行一些调整。对算法具体的原理却是一知半解。希望在下学期的统计学习以及计算智能这两门课上，对机器学习和深度学习模型的具体原理和数学推导有更加深刻的理解。