

网络空间先进技术研究院

**机器学习与人工智能**

**课程报告**

项目名称 基于CNN的图片分类

学 院 计算机技术与教育软件学院

专业班级 计算机技术方班

学生姓名

学生学号

任课教师 仇晶

20 18 年 11 月

**课程报告评定表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学生姓名 |  | 学 号 |  | 成绩 |  |
| 专业班级 |  | | | | |
| 项目名称 |  | | | | |
| 指  导  教  师  评  语 | 指导教师：  年 月 日 | | | | |

**目 录**

[1. 项目目的 1](#_Toc531422253)

[2. 项目环境和条件 1](#_Toc531422254)

[3. 项目原理 1](#_Toc531422255)

[4. 项目内容 3](#_Toc531422256)

[5. 项目过程与内容 3](#_Toc531422257)

[5.1 任务分析 3](#_Toc531422258)

[5.2 数据分析 3](#_Toc531422259)

[5.3 开发步骤 4](#_Toc531422260)

[5.4 关键问题 14](#_Toc531422261)

[5.5 实验结果分析 15](#_Toc531422262)

[5.6 拓展 16](#_Toc531422263)

[6. 项目总结与心得体会 19](#_Toc531422264)

[7. 参考文献 20](#_Toc531422265)

|  |
| --- |
| 项目目的 深入理解CNN算法，理解深度学习模型搭建过程，在CIFAR-10数据集上进行应用，借助TensorFlow完成一个图片识别程序。 项目环境和条件 软件环境：windows10及ubuntu16.04下的python开发环境（集成有tensorflow等深度学习常用工具包）。  硬件环境：个人计算机 项目原理   CNN由输入和输出层以及多个隐藏层组成，隐藏层可分为卷积层，池化层、RELU层和全连通层。  输入层：CNN的输入一般是二维向量，可以有高度，比如，RGB图像    卷积层：卷积层是CNN的核心，层的参数由一组可学习的滤波器（filter）或内核（kernels）组成，它们具有小的感受野，延伸到输入容积的整个深度。 在前馈期间，每个滤波器对输入进行卷积，计算滤波器和输入之间的点积，并产生该滤波器的二维激活图（输入一般二维向量，但可能有高度（即RGB））。 简单来说，卷积层是用来对输入层进行卷积，提取更高层次的特征。    池化层：池化（pool）即下采样（downsamples），目的是为了减少特征图。池化操作对每个深度切片独立，规模一般为 2＊2，相对于卷积层进行卷积运算，池化层进行的运算一般有以下几种：  \* 最大池化（Max Pooling）。取4个点的最大值。这是最常用的池化方法。  \* 均值池化（Mean Pooling）。取4个点的均值。  \* 高斯池化。借鉴高斯模糊的方法。不常用。  \* 可训练池化。训练函数 ff ，接受4个点为输入，出入1个点。不常用。  池化操作将保持深度大小不变。  如果池化层的输入单元大小不是二的整数倍，一般采取边缘补零（zero-padding）的方式补成2的倍数，然后再池化。  RELU层：RELU全名Rectified Linear Units，可被翻译成线性整流函数，是神经元的激活函数，对输入值x的作用是max(0,x)，当然RELU只是一种选择，还有选Leak-RELU等等，一般都是用RELU。  全连接层：此层即是常规神经网络，它的作用是对经过多次卷积层和多次池化层所得出来的高级特征进行全连接（全连接就是常规神经网络的性质），算出最后的预测值。  输出层：对输入集合的分类预测。   项目内容 (1)CNN算法的深度理解。  (2)CNN算法在CIFAR-10数据集上的应用。  (3)python工作环境及tensorflow框架的使用及理解。 项目过程与内容任务分析 深入学习CNN算法，并基于tensorflow和CIFAR-10数据集开发图片识别应用程序，主要工作集中于算法的理解、tensorflow框架的熟悉使用、基于tensorflow和CIFAR-10数据集的训练程序的编程、应用程序界面的开发。 数据分析  * 数据介绍： 对CIFAR-10 数据集的分类是机器学习中一个公开的基准测试问题，其任务是对一组32x32RGB的图像进行分类，这些图像涵盖了10个类别：飞机， 汽车， 鸟， 猫， 鹿， 狗， 青蛙， 马， 船以及卡车。      * 数据集分为 训练：测试 = 5：1   每个batch文件包含一个字典，每个字典包含有Data：numpy数组存储像素，labels一个范围在0-9的的列表，第i个数就是第i个图像的类标，还有一个文件batches.meta，存储10个图像类别名字。   开发步骤  1. [模型输入:](http://www.tensorfly.cn/tfdoc/tutorials/deep_cnn.html#model-inputs) 包括inputs() 、 distorted\_inputs()等一些操作，分别用于读取CIFAR的图像并进行预处理，做为后续评估和训练的输入；   1）数据处理：输入数据通过 inputs() 和distorted\_inputs()函数实现，从CIFAR-10二进制文件中读取图片文件。图片文件的处理流程如下：  图片会被统一裁剪到24x24像素大小，  裁剪中央区域用于评估或随机裁剪用于训练；  图片会进行近似的白化处理，使得模型对图片的动态范围变化不敏感。  对于训练，我们另外采取了一系列随机变换的方法来人为的增加数据集的大小：   * 对图像进行[随机的左右翻转](http://www.tensorfly.cn/tfdoc/api_docs/python/image.html#random_flip_left_right)； * 随机变换[图像的亮度](http://www.tensorfly.cn/tfdoc/api_docs/python/image.html#random_brightness)； * 随机变换[图像的对比度](http://www.tensorfly.cn/tfdoc/api_docs/python/image.html#tf_image_random_contrast)；   对于每个原始图我们还附带了一个[image\_summary](http://www.tensorfly.cn/tfdoc/api_docs/python/train.html#image_summary)，以便于在TensorBoard中查看。这对于检查输入图像是否正确十分有用。  http://www.tensorfly.cn/tfdoc/images/cifar_image_summary.png  从磁盘上加载图像并进行变换需要花费不少的处理时间。为了避免这些操作减慢训练过程，我们在16个独立的线程中并行进行这些操作，这16个线程被连续的安排在一个TensorFlow[队列](http://www.tensorfly.cn/tfdoc/api_docs/python/io_ops.html#shuffle_batch)中。    2）构建模型：  conv1 实现卷积  pool1 max pooling.  norm1 局部响应归一化.  conv2 卷积  norm2 局部响应归一化.  pool2 max pooling.  local3 基于修正线性激活的全连接层.  local4 基于修正线性激活的全连接层.  softmax\_linear 进行线性变换以输出 .     1. [模型预测:](http://www.tensorfly.cn/tfdoc/tutorials/deep_cnn.html#model-prediction) 包括inference()等一些操作，用于进行统计计算，比如在提供的图像进行分类； 2. **def** inference(images): 3. """Build the CIFAR-10 model. 5. Args: 6. images: Images returned from distorted\_inputs() or inputs(). 8. Returns: 9. Logits. 10. """ 12. # conv1 13. with tf.variable\_scope('conv1') as scope: 14. kernel = \_variable\_with\_weight\_decay('weights',shape=[5, 5, 3, 64],stddev=5e-2,wd=None) 15. conv = tf.nn.conv2d(images, kernel, [1, 1, 1, 1], padding='SAME') 16. biases = \_variable\_on\_cpu('biases', [64], tf.constant\_initializer(0.0)) 17. pre\_activation = tf.nn.bias\_add(conv, biases) 18. conv1 = tf.nn.relu(pre\_activation, name=scope.name) 19. \_activation\_summary(conv1) 20. # pool1 21. pool1 = tf.nn.max\_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], 22. padding='SAME', name='pool1') 23. # norm1 24. norm1 = tf.nn.lrn(pool1, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75, 25. name='norm1') 26. """第一层卷积是由64个5×5的卷积核组成，步长为1，padding为SAME使得卷积之后的图片输入和输出的大 27. 小保持一致。将卷积之后的结果加上偏置，然后在通过RELU激活函数，再经过最大池化，需要注意的是采 28. 用的是池化的核为3×3，步长为2×2，池化的尺寸和步长不一致目的是为了增加数据的丰富性。最后再经过 29. Lrn层，LRN层模仿了生物神经系统的“侧抑制”机制，对于局部神经元的活动创建竞争环境，使得其中响应 30. 较大的值变得更大，并抑制反馈较小的神经元，提高了模型的泛化能力。Relu激活函数是没有边界的，所 31. 以lrn层对于没有边界的激活函数会比较有用，它会从多个卷积核的响应中挑选比较大的反馈。对于有固定 32. 边界且能抑制过大值的激活函数sigmoid不太适应。 33. """ 34. # conv2 35. with tf.variable\_scope('conv2') as scope: 36. kernel = \_variable\_with\_weight\_decay('weights', 37. shape=[5, 5, 64, 64], 38. stddev=5e-2, 39. wd=None) 40. conv = tf.nn.conv2d(norm1, kernel, [1, 1, 1, 1], padding='SAME') 41. biases = \_variable\_on\_cpu('biases', [64], tf.constant\_initializer(0.1)) 42. pre\_activation = tf.nn.bias\_add(conv, biases) 43. conv2 = tf.nn.relu(pre\_activation, name=scope.name) 44. \_activation\_summary(conv2) 46. # norm2 47. norm2 = tf.nn.lrn(conv2, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75, 48. name='norm2') 49. # pool2 50. pool2 = tf.nn.max\_pool(norm2, ksize=[1, 3, 3, 1], 51. strides=[1, 2, 2, 1], padding='SAME', name='pool2') 53. # local3 54. with tf.variable\_scope('local3') as scope: 55. reshape = tf.reshape(pool2, [images.get\_shape().as\_list()[0], -1]) 56. dim = reshape.get\_shape()[1].value 57. weights = \_variable\_with\_weight\_decay('weights', shape=[dim, 384], 58. stddev=0.04, wd=0.004) 59. biases = \_variable\_on\_cpu('biases', [384], tf.constant\_initializer(0.1)) 60. local3 = tf.nn.relu(tf.matmul(reshape, weights) + biases, name=scope.name) 61. \_activation\_summary(local3) 62. """在进行全连接操作之前，需要先将卷积后的图片进行flatten，将图片变成一个行向量， 63. 一行代表一张图片，所有有bath\_size行。在这一层全连接中对权重使用L2正则化，正则化 64. 的系数为0.004。通过relu激活函数之后输出一个384维的向量。 65. """ 66. # local4 67. with tf.variable\_scope('local4') as scope: 68. weights = \_variable\_with\_weight\_decay('weights', shape=[384, 192], 69. stddev=0.04, wd=0.004) 70. biases = \_variable\_on\_cpu('biases', [192], tf.constant\_initializer(0.1)) 71. local4 = tf.nn.relu(tf.matmul(local3, weights) + biases, name=scope.name) 72. \_activation\_summary(local4) 74. # linear layer(WX + b), 75. # We don't apply softmax here because 76. # tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits accepts the unscaled logits 77. # and performs the softmax internally for efficiency. 78. with tf.variable\_scope('softmax\_linear') as scope: 79. weights = \_variable\_with\_weight\_decay('weights', [192, NUM\_CLASSES], 80. stddev=1/192.0, wd=None) 81. biases = \_variable\_on\_cpu('biases', [NUM\_CLASSES], 82. tf.constant\_initializer(0.0)) 83. softmax\_linear = tf.add(tf.matmul(local4, weights), biases, name=scope.name) 84. \_activation\_summary(softmax\_linear) 86. **return** softmax\_linear   3.[模型训练:](http://www.tensorfly.cn/tfdoc/tutorials/deep_cnn.html#model-training) 包括loss() and train()等一些操作，用于计算损失、计算梯度、进行变量更新以及呈现最终结果。  计算损失函数   1. **def** loss(logits, labels): 2. """Add L2Loss to all the trainable variables. 4. Add summary for "Loss" and "Loss/avg". 5. Args: 6. logits: Logits from inference(). 7. labels: Labels from distorted\_inputs or inputs(). 1-D tensor 8. of shape [batch\_size] 10. Returns: 11. Loss tensor of type float. 12. """ 13. # Calculate the average cross entropy loss across the batch. 14. labels = tf.cast(labels, tf.int64) 15. cross\_entropy = tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits( 16. labels=labels, logits=logits, name='cross\_entropy\_per\_example') 17. cross\_entropy\_mean = tf.reduce\_mean(cross\_entropy, name='cross\_entropy') 18. tf.add\_to\_collection('losses', cross\_entropy\_mean) 20. # The total loss is defined as the cross entropy loss plus all of the weight 21. # decay terms (L2 loss). 22. **return** tf.add\_n(tf.get\_collection('losses'), name='total\_loss')   **train函数**   1. **def** train(total\_loss, global\_step): 2. """Train CIFAR-10 model. 4. Create an optimizer and apply to all trainable variables. Add moving 5. average for all trainable variables. 7. Args: 8. total\_loss: Total loss from loss(). 9. global\_step: Integer Variable counting the number of training steps 10. processed. 11. Returns: 12. train\_op: op for training. 13. """ 14. # Variables that affect learning rate. 15. num\_batches\_per\_epoch = NUM\_EXAMPLES\_PER\_EPOCH\_FOR\_TRAIN / FLAGS.batch\_size 16. decay\_steps = int(num\_batches\_per\_epoch \* NUM\_EPOCHS\_PER\_DECAY) 18. # Decay the learning rate exponentially based on the number of steps. 19. lr = tf.train.exponential\_decay(INITIAL\_LEARNING\_RATE, 20. global\_step, 21. decay\_steps, 22. LEARNING\_RATE\_DECAY\_FACTOR, 23. staircase=True) 24. tf.summary.scalar('learning\_rate', lr) 26. # Generate moving averages of all losses and associated summaries. 27. loss\_averages\_op = \_add\_loss\_summaries(total\_loss) 29. # Compute gradients. 30. with tf.control\_dependencies([loss\_averages\_op]): 31. opt = tf.train.GradientDescentOptimizer(lr) 32. grads = opt.compute\_gradients(total\_loss) 34. # Apply gradients. 35. apply\_gradient\_op = opt.apply\_gradients(grads, global\_step=global\_step) 37. # Add histograms for trainable variables. 38. **for** var **in** tf.trainable\_variables(): 39. tf.summary.histogram(var.op.name, var) 41. # Add histograms for gradients. 42. **for** grad, var **in** grads: 43. **if** grad **is** **not** None: 44. tf.summary.histogram(var.op.name + '/gradients', grad) 46. # Track the moving averages of all trainable variables. 47. variable\_averages = tf.train.ExponentialMovingAverage( 48. MOVING\_AVERAGE\_DECAY, global\_step) 49. with tf.control\_dependencies([apply\_gradient\_op]): 50. variables\_averages\_op = variable\_averages.apply(tf.trainable\_variables()) 52. **return** variables\_averages\_op   **训练过程：**    4 模型评估   1. **def** eval\_once(saver, summary\_writer, top\_k\_op, summary\_op): 2. """Run Eval once. 4. Args: 5. saver: Saver. 6. summary\_writer: Summary writer. 7. top\_k\_op: Top K op. 8. summary\_op: Summary op. 9. """ 10. with tf.Session() as sess: 11. ckpt = tf.train.get\_checkpoint\_state(FLAGS.checkpoint\_dir) 12. **if** ckpt **and** ckpt.model\_checkpoint\_path: 13. # Restores from checkpoint 14. saver.restore(sess, ckpt.model\_checkpoint\_path) 15. # Assuming model\_checkpoint\_path looks something like: 16. #   /my-favorite-path/cifar10\_train/model.ckpt-0, 17. # extract global\_step from it. 18. global\_step = ckpt.model\_checkpoint\_path.split('/')[-1].split('-')[-1] 19. **else**: 20. **print**('No checkpoint file found') 21. **return** 23. # Start the queue runners. 24. coord = tf.train.Coordinator() 25. **try**: 26. threads = [] 27. **for** qr **in** tf.get\_collection(tf.GraphKeys.QUEUE\_RUNNERS): 28. threads.extend(qr.create\_threads(sess, coord=coord, daemon=True, 29. start=True)) 31. num\_iter = int(math.ceil(FLAGS.num\_examples / FLAGS.batch\_size)) 32. true\_count = 0  # Counts the number of correct predictions. 33. total\_sample\_count = num\_iter \* FLAGS.batch\_size 34. step = 0 35. **while** step < num\_iter **and** **not** coord.should\_stop(): 36. predictions = sess.run([top\_k\_op]) 37. true\_count += np.sum(predictions) 38. step += 1 40. # Compute precision @ 1. 41. precision = true\_count / total\_sample\_count 42. **print**('%s: precision @ 1 = %.3f' % (datetime.now(), precision)) 44. summary = tf.Summary() 45. summary.ParseFromString(sess.run(summary\_op)) 46. summary.value.add(tag='Precision @ 1', simple\_value=precision) 47. summary\_writer.add\_summary(summary, global\_step) 48. **except** Exception as e:  # pylint: disable=broad-except 49. coord.request\_stop(e) 51. coord.request\_stop() 52. coord.join(threads, stop\_grace\_period\_secs=10)   5 界面开发  使用Qtcreator和pyqt进行界面开发   1. **from** PyQt5 **import** QtWidgets 2. **from** PyQt5.QtWidgets **import** QFileDialog 3. **from** PyQt5 **import** QtCore, QtGui 4. **import** classify 5. **class** MyWindow(QtWidgets.QWidget): 6. **def** \_\_init\_\_(self): 7. super(MyWindow, self).\_\_init\_\_() 8. self.setObjectName("widget") 9. self.resize(490, 506) 10. self.setMinimumSize(QtCore.QSize(100, 100)) 11. self.setCursor(QtGui.QCursor(QtCore.Qt.ArrowCursor)) 12. self.gridLayoutWidget = QtWidgets.QWidget(self) 13. self.gridLayoutWidget.setGeometry(QtCore.QRect(60, 120, 381, 301)) 14. self.gridLayoutWidget.setObjectName("gridLayoutWidget") 15. self.gridLayout = QtWidgets.QGridLayout(self.gridLayoutWidget) 16. self.gridLayout.setContentsMargins(0, 0, 0, 0) 17. self.gridLayout.setObjectName("gridLayout") 18. self.label = QtWidgets.QLabel(self) 19. self.label.setGeometry(QtCore.QRect(70, 70, 54, 12)) 20. self.label.setObjectName("label") 21. self.textEdit = QtWidgets.QTextEdit(self) 22. self.textEdit.setGeometry(QtCore.QRect(120, 45, 261, 25)) 23. self.textEdit.setObjectName("textEdit") 24. self.toolButton = QtWidgets.QToolButton(self) 25. self.toolButton.setGeometry(QtCore.QRect(379, 43, 50, 28)) 26. self.toolButton.setObjectName("toolButton") 27. self.toolButton.clicked.connect(self.msg) 28. self.pushButton = QtWidgets.QPushButton(self) 29. self.pushButton.setGeometry(QtCore.QRect(200, 80, 81, 31)) 30. self.pushButton.setObjectName("pushButton") 31. self.pushButton.clicked.connect(self.sbing) 32. #  放图片的label 33. self.label2 = QtWidgets.QLabel(self) 34. self.label2.setGeometry(QtCore.QRect(72, 150, 360, 300)) 35. #  参数分别是左上点距左边框宽度，距顶高度，长度，高度 36. self.label2.setObjectName("label2") 38. self.retranslateUi(self) 39. QtCore.QMetaObject.connectSlotsByName(self) 40. **def** retranslateUi(self, widget): 41. \_translate = QtCore.QCoreApplication.translate 42. widget.setWindowTitle(\_translate("widget", "CNN图片识别 by White-Lotus")) 43. self.label.setText(\_translate("widget", "源图片:")) 44. self.toolButton.setText(\_translate("widget", "浏览")) 45. self.pushButton.setText(\_translate("widget", "开始识别")) 46. **def** msg(self): 48. fileName1, filetype = QFileDialog.getOpenFileName(self, 49. "选取文件", 50. "/", 51. "All Files (\*);;image Files (\*.jpg)")  # 设置文件扩展名过滤,注意用双分号间隔 52. #  print(fileName1, filetype) 53. #  print(fileName1) 55. png = QtGui.QPixmap(fileName1).scaled(self.label2.width(), self.label2.height()) 56. self.label2.setPixmap(png) 57. self.textEdit.setText(fileName1) 58. classify.imgf=fileName1 59. **def** sbing(self): 60. self.pushButton.setText("识别中") 61. classify.sjsy() 62. self.pushButton.setText("开始识别") 63. **if** \_\_name\_\_ == "\_\_main\_\_": 64. **import** sys 65. app = QtWidgets.QApplication(sys.argv) 66. myshow = MyWindow() 67. myshow.show() 68. sys.exit(app.exec\_()) 69. exit()        关键问题 1） 对于数据的处理，返回数据和对应标签的方法  2） 卷积，池化的参数的设置  3） 损失函数的计算  4） 模型的训练以及评估  5） 保存模型并使用，界面的开发 实验结果分析   训练一个可进行N维分类的网络的常用方法是使用[多项式逻辑回归](https://en.wikipedia.org/wiki/Multinomial_logistic_regression" \t "_blank),又被叫做softmax 回归。Softmax 回归在网络的输出层上附加了一个[softmax](http://www.tensorfly.cn/tfdoc/api_docs/python/nn.html" \l "softmax) nonlinearity，并且计算归一化的预测值和label的[1-hot encoding](http://www.tensorfly.cn/tfdoc/api_docs/python/sparse_ops.html" \l "sparse_to_dense)的[交叉熵](http://www.tensorfly.cn/tfdoc/api_docs/python/nn.html" \l "softmax_cross_entropy_with_logits)。在正则化过程中，我们会对所有学习变量应用[权重衰减损失](http://www.tensorfly.cn/tfdoc/api_docs/python/nn.html#l2_loss)。模型的目标函数是求交叉熵损失和所有权重衰减项的和，loss()函数的返回值就是这个值。  在TensorBoard中使用[scalar\_summary](http://www.tensorfly.cn/tfdoc/api_docs/python/train.html" \l "scalar_summary)来查看该值的变化情况：  CIFAR-10 Loss  我们使用标准的梯度下降算法来训练模型（也可以在[Training](http://www.tensorfly.cn/tfdoc/api_docs/python/train.html)中看看其他方法），其学习率随时间以指数形式衰减。  CIFAR-10 Learning Rate Decay  评估脚本只是周期性的返回precision@1 (The script merely returns the precision @ 1 periodically)--在该例中返回的准确率是86.5%。 拓展 1. 使用Resnet实现图片分类    上图为残差神经网络的基本模块（专业术语叫残差学习单元），输入为x，输出为F(x)+x，F(x)代表网络中数据的一系列乘、加操作，假设神经网络最优的拟合结果输出为H(x)=F(x)+x，那么神经网络最优的F(x)即为H(x)与x的残差，通过拟合残差来提升网络效果。为什么转变为拟合残差就比传统卷积网络要好呢？因为训练的时候至少可以保证残差为0，保证增加残差学习单元不会降低网络性能，假设一个浅层网络达到了饱和的准确率，后面再加上这个残差学习单元，起码误差不会增加。  通过不断堆叠这个基本模块，就可以得到最终的ResNet模型，理论上可以无限堆叠而不改变网络的性能。  特点：  使得训练超级深的神经网络成为可能，避免了不断加深神经网络，准确率达到饱和的现象（后来将层数增加到1000层）  输入可以直接连接到输出，使得整个网络只需要学习残差，简化学习目标和难度。  ResNet是一个推广性非常好的网络结构，容易和其他网络结合    **图像流分析**  下面将要实现的是resnet-50。下面是网络模型的整体模型图。其中的CONV表示卷积层，Batch Norm表示Batch 归一化层，ID BLOCK表示Identity块，由多个层构成，具体见第二个图。Conv BLOCK表示卷积块，由多个层构成。为了使得model个结构更加清晰，才提取出了conv block 和id block两个‘块’，分别把它们封装成函数。  cifar10 model  上图表示Resnet-50的整体结构图  cifar10 model  上图表示ID block  cifar10 model  上图表示conv block  下载imagenet2012数据集  <http://www.image-net.org/challenges/LSVRC/2012/nnoupb/ILSVRC2012_img_test.tar>  <http://www.image-net.org/challenges/LSVRC/2012/nnoupb/ILSVRC2012_img_val.tar>  <http://www.image-net.org/challenges/LSVRC/2012/nnoupb/ILSVRC2012_img_train.tar>  <http://www.image-net.org/challenges/LSVRC/2012/nnoupb/ILSVRC2012_devkit_t12.tar>  <http://www.image-net.org/challenges/LSVRC/2012/nnoupb/ILSVRC2012_bbox_train_v2.tar>  tensorflow-resnet模型： <https://github.com/ry/tensorflow-resnet>  使用了Tensorflow，可识别物体量从10类增加到1001类，可为：狗熊 椅子 汽车 键盘 箱子 婴儿床 旗杆iPod播放器 轮船 面包车 项链 降落伞 桌子 钱包 球拍 步枪等等接着导入ResNet50网络模型进行处理，主要图像数据处理函数如下：  （1）将PIL格式的图像转换为numpy数组。  （2）将我们的(3，224，224)大小的图像转换为(1，3，224，224)。因为model.predict函数需要4维数组作为输入，其中第4维为每批预测图像的数量。这也就是说，我们可以一次性分类多个图像。  （3）使用训练数据集中的平均通道值对图像数据进行零值处理，即使得图像所有点的和为0。这是非常重要的步骤，如果跳过，将大大影响实际预测效果。这个步骤称为数据归一化。  （4）tensorflow中训练差不多需要两个小时的时间，训练好之后实际识别的速率快达1~2分钟一张图，准确率高达90%以上   1. 基于CNN的VGG模型应用：     IMG_256+IMG_256  =IMG_256 项目总结与心得体会 深入理解了CNN和图片分类问题，从模型算法到具体代码实践以及应用，我觉得对于进一步学习人工智能和机器学习有了更大的帮助。看似高大上的人工智能，其实包含了很多前期的人为工作，对于机器学习的具体应用有了更好的把握和认知。在项目当中还学习了使用tensorflow工具的使用，这是学习深度学习的基本工具，在这一工具使用之前还接触过其他例如paddlepaddle,ntlk的使用，感觉性能还是ts使用的好，学习了很多教程，结合项目进行调整参数，训练，测试和评估，还学习了qt界面的开发，结合具体场景进行实践，对于理解机器学习算法理论大有裨益。在参与项目的过程中查阅了很多文献，同时和其他组员进行了深入探讨和学习，对于以后项目合作和学习奠定了基础。 参考文献 [1] 李欣怡. 基于深度网络的图像特征提取及应用[D].江南大学,2018.  [2] 童浩然. 基于深度学习的图像分类的研究[D].安徽理工大学,2018.  [3] 苏越. 基于卷积神经网络的多标签图像分类[D].吉林大学,2018. |