Lab6实验报告

22307130158 吴优

实验内容

卷积操作

本模块中，要求基于张量类实现卷积操作。 卷积前张量为四阶张量，大小为 "图片数 x 图片通道数 x 图片高度 x 图片宽度" 卷积核为四阶张量，大小为 "输出通道数 x 输入通道数 x 卷积核高度 x 卷积核宽度" 卷积后输出为同样是四阶张量，大小为 "图片数 x 卷积核的输出通道数 x 卷积后高度 x 卷积后宽度"。

批处理功能

在 `./figs/` 目录下有若干图片文件夹，文件名类似于 `./figs/001/01.jpg` ，按顺序读入所有图片，把同一组中的所有图片用一个张量进行储存，并将此张量作卷积后输出到`./results/` 中，保存的文件名类似于 `./results/001_result.txt` 。

代码功能分析

对于张量类的修改

在实现卷积操作的过程中，修改了张量类中的get方法，新增了泛型方法，用于返回指定的子张量，同时保证对于Object[]类和int[]类值的正常返回，防止产生get方法返回类型与调用想要返回值类型不匹配且不支持显式转换情况的出现。

```
    public <T> T get(int... indices) {
        Object current = data;
        for (int index : indices) {
            if (current instanceof Object[]) {
                current = ((Object[]) current)[index];
            } else if (current instanceof int[]) {
                current = ((int[]) current)[index];
            } else {
                throw new IllegalStateException("Unsupported data type");
            }
        }
        // 强制类型转换并返回
        return (T) current;
    }
```

附加函数

由于编译器不能显式的转换Object[]型到int[][][]型，所以定义convertToPrimitive方法用于转化Object[]型数组到int[][][]型数组，用于提取出指定张量的内部的具体信息。

```
public static int[][][] convertToPrimitive(Object[] objectArray) {
    int[][][] result = new int[objectArray.length][][];
    for (int i = 0; i < objectArray.length; i++) {
        Object[] innerArray = (Object[]) objectArray[i];
        result[i] = new int[innerArray.length][];
        for (int j = 0; j < innerArray.length; j++) {
            result[i][j] = (int[]) innerArray[j];
        }
    }
    return result;
}
```

加载卷积核

- `loadKernel(String filePath)`：从文件中加载卷积核，解析文件内容并创建相应的卷积核张量。 先读取四个数字分别代表输出通道数、输入通道数、卷积核高度和宽度。用这四个数字初始化卷积核，创建四阶张量，再读取卷积核的数据。

```
 public static void loadKernel(String filePath) {
        try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
            // 读取四个整数，分别表示输出通道数、输入通道数、卷积核高度、卷积核宽度
            String[] sizes = reader.readLine().trim().split("\\s+");
            int outputChannels = Integer.parseInt(sizes[0]);
            int inputChannels = Integer.parseInt(sizes[1]);
            int kernelHeight = Integer.parseInt(sizes[2]);
            int kernelWidth = Integer.parseInt(sizes[3]);

            // 根据大小创建四阶张量
            kernel = new Tensor(new int[]{outputChannels, inputChannels, kernelHeight,
kernelWidth});

            // 读取每个卷积核的数据
            for (int o = 0; o < outputChannels; o++) {
                for (int i = 0; i < inputChannels; i++) {
                    for (int h = 0; h < kernelHeight; h++) {
                        String[] values = reader.readLine().trim().split("\\s+");
                        for (int w = 0; w < kernelWidth && w < values.length; w++) {
                            kernel.set( Integer.parseInt(values[w]),o, i, h, w);
                        }
                    }
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

读取图像并转换成张量

- `loadImageAsTensor(String imagePath)`：读取图像文件并将其转换成张量表示，将图像的 RGB 通道分别存储在张量的三个通道中。 用于读取一张图片，分别读取RGB的值，生成一个三阶张量。

```
public static Tensor loadImageAsTensor(String imagePath) {
      try {
          BufferedImage image = ImageIO.read(new File(imagePath));
          int height = image.getHeight();
          int width = image.getWidth();
          int[][][] tensorData = new int[3][height][width]; // 假设图片的颜色通道数为 3
          for (int i = 0; i < height; i++) {
              for (int j = 0; j < width; j++) {
                  Color pixelColor = new Color(image.getRGB(j, i));
                  tensorData[0][i][j] = pixelColor.getRed(); // 红色通道
                  tensorData[1][i][j] = pixelColor.getGreen(); // 绿色通道
                  tensorData[2][i][j] = pixelColor.getBlue(); // 蓝色通道
              }
          }
          return new Tensor(tensorData);
      } catch (IOException e) {
          e.printStackTrace();
          return null;
      }
  }
```

对张量进行卷积操作

- `convolution(Tensor input)`：对输入张量进行卷积操作，使用已加载的卷积核进行滑动卷积操作，生成输出张量。 进行具体的卷积操作，输入张量为一个三阶张量，代表每一张图片，对一张图片的每一个像素点进行卷积操作，并将RGB三个通道与对应的值相乘之后相加，产生结果，最后创建输出三阶张量。

```
public static Tensor convolution(Tensor input) {
      // 获取输入张量的维度信息
      int[] inputDims = input.getDimensions();
      int inputChannels = inputDims[0];
      int inputHeight = inputDims[1];
      int inputWidth = inputDims[2];

      // 获取卷积核的维度信息
      int[] kernelDims = kernel.getDimensions();
      int outputChannels = kernelDims[0];

      // 创建输出张量
      int[][][] outputData = new int[outputChannels][inputHeight-2][inputWidth-2];

      // 对每个输出通道进行卷积操作
      for (int o = 0; o < outputChannels; o++) {
          // 遍历输入张量的每个像素，进行卷积
          for (int i = 0; i < inputHeight-2; i++) {
```

```
                for (int j = 0; j < inputWidth-2; j++) {
                    // 初始化卷积结果为 0
                    int result = 0;
                    // 对每个输入通道进行卷积操作
                    for (int c = 0; c < inputChannels; c++) {
                        // 对每个卷积核元素进行遍历
                        for (int m = 0; m < kernelDims[2]; m++) {
                            for (int n = 0; n < kernelDims[3]; n++) {
                                // 计算卷积核在输入张量上的索引位置
                                int inputI = i  + m;
                                int inputJ = j  + n;
                                    // 执行卷积操作
                                    result += (int)input.get(c, inputI, inputJ) *
(int)kernel.get(o, c, m, n);
                                }
                            }
                        }
                        outputData[o][i][j] = result;
                    }
                }
            }
        }
        // 使用输出数据创建输出张量
        return new Tensor(outputData);
    }
```

保存张量到文件

- `saveTensorToFile(Tensor tensor, String filePath)`：将张量数据保存到文件中，便于后续分析和处理。

```
    public static void saveTensorToFile1(Tensor tensor, String filePath,int imageLength) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath,true))) {
            int[][][] data = (int[][][])tensor.getData();
            for (int i = 0; i < data.length; i++) {
                for (int j = 0; j < data[i].length; j++) {
                    for (int k = 0; k < data[i][j].length; k++) {
                            writer.write(data[i][j][k] + " ");
                    }
                    writer.newLine();
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public static void saveTensorToFile2(Tensor tensor, String filePath,int imageLength)
    {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
            writer.write(imageLength + " ");
            int[]dimensions=tensor.getDimensions();
            for (int dim : dimensions) {
```

```
                writer.write(dim + " ");
            }
            writer.newLine();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

主函数

- 主函数加载卷积核，遍历图像文件夹，将每组图片转换成张量进行卷积操作，并保存结果到文件中。

```
    public static void main(String[] args) {
        // 加载卷积核
        loadKernel("./conv_kernel.txt");
        // 遍历图像文件夹
        File figsDir = new File("./figs");
        File[] groups = figsDir.listFiles(File::isDirectory);
        if (groups != null) {
            for (File group : groups) {
                File[] images = group.listFiles();
                if (images != null && images.length > 0) {
                    // 创建一个张量，存储当前组中的所有图片
                    Tensor tensor = new Tensor(images.length, 3,
loadImageAsTensor(images[0].getPath()).getDimensions()[1],
loadImageAsTensor(images[0].getPath()).getDimensions()[2]);
                    for (int i = 0; i < images.length; i++) {
                        // 读取图像并转换成张量
                        Tensor imageTensor = loadImageAsTensor(images[i].getPath());
                        tensor.set(imageTensor.getData(), i);
                    }
                    Tensor padedTensor=tensor.pad(2,2);
                    // 对张量进行填充
                    for(int i=0;i<images.length;i++)
                    {
                        int [][][]mm=convertToPrimitive(tensor.get(i));
                        Tensor temp=new Tensor(mm);
                        Tensor resultTensor = convolution(temp);//temp为三阶张量-输入*长宽,
resultTensor为三阶张量-输出*长宽
                        String resultFilePath = String.format("./results/%03d_result.txt",
Integer.parseInt(group.getName()));
                        saveTensorToFile(resultTensor, resultFilePath);
                    }
                }
            }
        }
    }
```

实验过程分析

1. **加载卷积核**：程序首先加载了一个卷积核，这个卷积核是从文件中读取的。卷积核的参数包括输出通道数、输入通道数、卷积核高度和宽度等。

2. **读取图像并转换成张量**：接下来，程序遍历了图像文件夹，读取每个图像文件，并将图像转换成张量表示。在这个过程中，每个图像的 RGB 通道分别被提取出来，并存储在张量的三个通道中。

3. **卷积操作**：对于每组图像，程序首先对图像进行填充操作，然后将填充后的张量作为输入，使用加载的卷积核进行卷积操作。卷积操作完成后，将结果张量保存到文件中。

算法改进策略（未实现）

观察到，整个程序最复杂的地方在于卷积函数convolution，对于卷积核和原张量的卷积操作达到了惊人的六重循环嵌套，这样的循环嵌套势必会导致时间复杂度的过于复杂，从而导致整个程序的运行效率低下，所以这一部分的算法需要改进。 在网上查阅了关于快速卷积的方法，发现了一个可以很大程度上提高效率的方法，核心思想是将卷积操作转换成为矩阵的乘法。 在进行传统卷积的时，我们会移动滑块，然后对滑块进行加权求和。现在我们要做的是,将每个滑块形成的子矩阵给拉直，形成一个行向量，然后移动滑块，生成一个一个的行向量，然后将这些行向量堆叠起来形成一个矩阵，将卷积核也对应的拉直成为一个列向量，卷积操作也就等价的转化成了一个矩阵和一个列向量的乘法操作。由于对于矩阵的乘法在线性代数领域已经得到了很大程度上的优化，在这里直接调用库函数就能得到很大的效率提升，从而简化时间复杂度，提高程序运行效率。 参考资料：https://blog.csdn.net/qq_43409114/article/details/105426806

图像实际操作

对处理后的张量，负数置为0，超过255的置为255，并生成rgb表示，保存为图像文件，实际图像处理效果如下
（生成图像的操作代码由于篇幅限制不在此列出，以注释的形式附加在Bach_test.java末）