# 图像的读取、显示与灰度共生矩阵纹理提取实验报告

刘焕宇 后胜涛

## 一、题目描述

在图像的研究中，我们经常需要对图像的内容进行特征提取，以进行下一步的物体识别等工作。图像的纹理特征就是一种常用的图像特征，可以充分反映图像的整体，因此也成为了诸多图像后处理技术所必备的研究条件。但是，纹理的复杂多样性使得研究者们对其分析和准确识别是非常困难的。而解决这个困难的方法之一是对图像提取纹理，然后对提取的纹理进行分析研究。这也是模式识别和计算机视觉等研究领域的基础。在纹理研究的每个阶段，国内外学者研究对图像纹理的提取模型及算法的不断发展，促使着大家对这领域进行更深入的研究。

目前，人们对纹理的精确定义还没有完全统一，当前几个类别的定义基本上按不同的应用类型形成相对的定义。一般认为，纹理是图像色彩或者灰度在空间上重复或变化形成纹理，人们对纹理信息所具有的如下特性达成共识：

（1）纹理基元是纹理存在的基本元素，并一定是按照某种规律排列组合形成纹理。

（2）纹理信息具有局部显著性，通常表现为纹理基元序列在一定的局部空间重复出现

（3）纹理具有周期性、方向性、密度、强度和粗糙程度等基本特征，而与人类世界特征相一致的特征也更多地用于进行纹理分类

（4）纹理区域内大致是均匀的统一体，都有大致相同的结构。

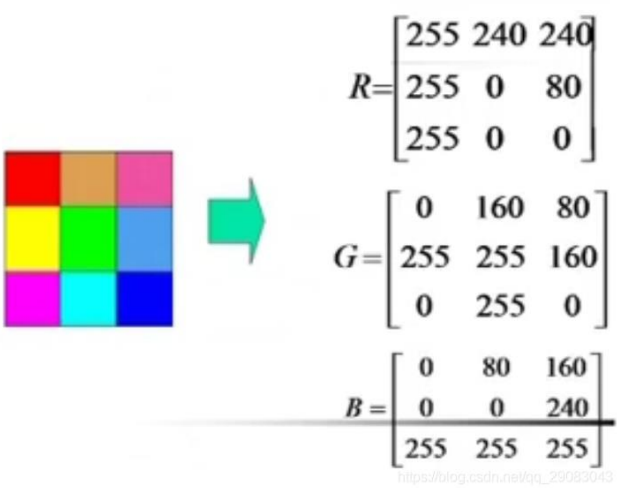
灰度共生矩阵法 $(GLCM, Gray-level\ co-occurrence\ matrix)$ 是一种经典的对图像做纹理特征提取的一种统计分析方法，是 20 世纪 70 年代初由 R. Haralick 等人提出的。它是在假定图像中各像素间的空间分布关系包含了图像纹理信息的前提下，提出的具有广泛性的纹理分析方法。灰度共生矩阵法通过计算灰度图像得到它的共生矩阵，然后再计算所得到的共生矩阵的部分特征值，来分别代表图像的某些纹理特征。灰度共生矩阵能反映图像灰度关于方向、相邻间隔、变化幅度等综合信息，它是分析图像的局部模式和它们排列规则的基础。

本实验旨在学习灰度共生矩阵算法、熟悉数字图像的基本读写、显示、保存与基本纹理提取流程。

## 二、算法原理

（1）数字图像的存取

在计算机中，数字图像有多种存储方式，最常见的是以矩阵形式。图片的宽$m$和高$n$代表了矩阵的两个维度，每一个像素$a_{ij}$表示一个颜色。一维矩阵通常为灰度图表示，三维矩阵的三个通道分别为 R，G，B 三种颜色的分量，如图所示。




**图一 数字图像 RGB 通道存储**

（2）$RGB$图像转化灰度图像

最大值法：直接取 R，B，G 三个分量中数值最大的分量的数值。（0 视为最小，255 视为最大）。

均值法：取 R，B，G 三个分量中数值的均值

加权平均法：根据人眼对于 R，B，G 三种颜色的敏感度，按照一定的权值进行加权平均得到。

我们在转换的过程中使用的是加权平均法，利用$opencv$的 COLOR_BPG2GRAY 将原图$img$转换为灰度图$imgGray$，计算公式为$Gray = 0.1140 * B + 0.5870 * G + 0.2989 * R$

（3）灰度共生矩阵法

设原始灰度图像矩阵$A_{nm} = a_{ij} \in [0,255]$，称$gl = 255$为图片的灰度等级$(gray\ level)$。设$dx, dy$分别为矩阵横向、纵向相邻间距。则关于特定间距$(dx, dy)$的$gl$维灰度共生矩阵$G_{gl,gl} = g_{ij}$可由以下伪代码得到：

```
for i from 1 to n:
    for j from 1 to m:
        g[a_{i,j}][a_{i+dx,j+dy}] += 1;
```

即统计所有间距为$(dx, dy)$的像素点对，根据点对间的灰度值，映射到灰度矩阵中。灰度矩阵$g_{i,j}$的含义为，在原图像中，有多少个间距为$(dx, dy)$的像素点对，满足：

$$\begin{cases} a_{x,y} = i \\ a_{x+dx,y+dy} = j \end{cases} \tag{1}$$

实际纹理提取中，出于对图像局部纹理的考虑，通常是选取一个大小为$w \times w$的窗口，对这个窗口范围内的像素点对进行灰度共生矩阵的变换。再对局部矩阵进一步的**代数运算**，将运算结果作为该区域纹理的特征。之后进行滑动窗口，即可对图像的所有区域进行纹理提取。本实验选用的窗口大小为$7 \times 7$。

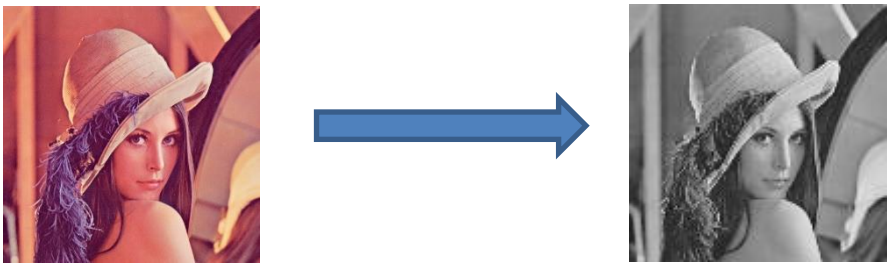
同时，出于时间复杂度的考虑，灰度共生矩阵的维度一般不会很大，算法的时间复杂度为$O(n \times m \times gl^2)$。所以进行实际操作时，通常将原图256维的灰度压缩为16维左右，压缩公式为：

$$a_{i,j} = \left\lfloor \frac{a_{i,j}}{256} \times 16 \right\rfloor \tag{2}$$

通常的**代数运算**选取的指标有：均值、标准差、同质度、能量、对比度、熵、角二阶矩、对比度等计算特征。

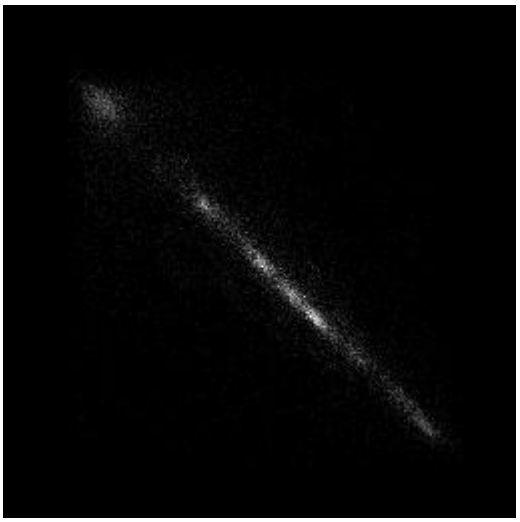
## 三、仿真实验

本实验以计算机视觉经典图像$lena$图为对象，用 $python + OpenCV$ 作为实验环境。首先读取并显示图片，转换为灰度图：



图二 灰度图的转化

之后计算灰度共生矩阵并可视化：



图三 灰度共生矩阵

灰度共生矩阵的特征直觉上来说，如果图像的是由具有相似灰度值的像素块构成，则灰度共生矩阵的对角元素会有比较大的值；如果图像像素灰度值在局部有变化，那么偏离对角线的元素会有比较大的值。结果说明$lena$图有较高的灰度像素值相似度。

1、均值

反映纹理的规则程度，纹理杂乱无章、难以描述的，值较小；规律性强的、易于描述的，值较大。

$$\text{Mean} = \frac{1}{w}\sum g_{ij} \tag{3}$$

2、标准差

反映图像元值与均值偏差的度量。

$$Std = \sqrt{\sum_i \sum_j g(i,j) * (i - Mean)^2} \tag{4}$$

3、同质度

是图像局部灰度均匀的度量，如果图像局部的灰度均匀，同质度取值较大

$$Homogeneity = \sqrt{\sum_i \sum_j g(i,j) * \frac{1}{1 + (i-j)^2}} \tag{5}$$

4、对比度

度量矩阵的值是如何分布和图像中局部变化的多少，反应了图像的清晰度和纹理的沟纹深浅。纹理的沟纹越深，反差越大，效果越清晰；反之，对比值小，则沟纹浅，效果模糊。

$$\text{Con} = \sum_i \sum_j (i-j)^2 \, g(i,j) \tag{6}$$

5、能量(角二阶矩)

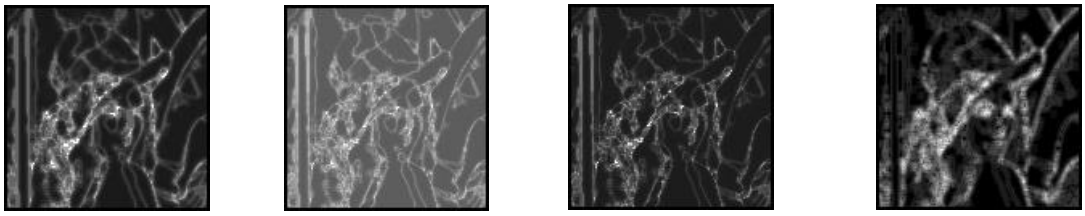
能量变换反映了图像灰度分布均匀程度和纹理粗细度。若灰度共生矩阵的元素值相近，则能量较小，表示纹理细致；若其中一些值大，而其它值小，则能量值较大。能量值大表明一种较均一和规则变化的纹理模式。

$$\text{Asm} = \sum_i \sum_j g(i,j)^2 \tag{7}$$

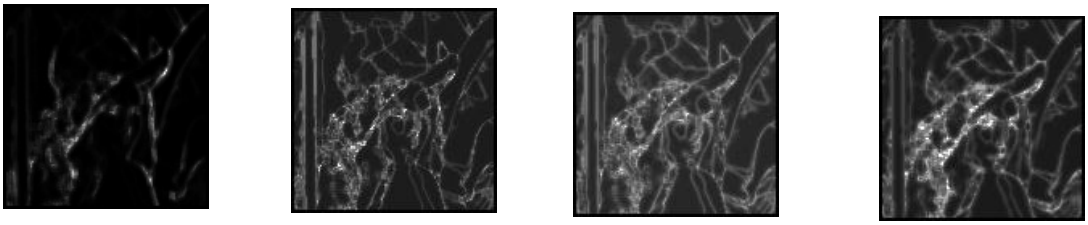
6、熵

图像包含信息量的随机性度量。当共生矩阵中所有值均相等或者像素值表现出最大的随机性时，熵最大；因此熵值表明了图像灰度分布的复杂程度，熵值越大，图像越复杂。

$$\text{Entropy} = \sum_i \sum_i g(i,j) log(i,j) \tag{8}$$

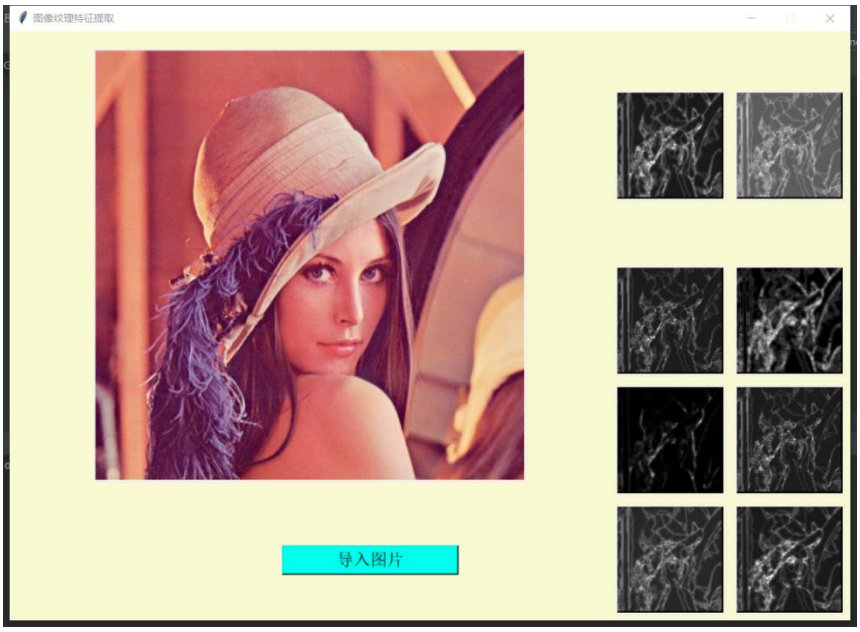


均值　　标准差　　角二阶矩　　熵



对比度　　能量　　同质性　　最大概率

图四 八种纹理提取示意图




图五 输出结果展示

## 四、心得体会

在研究本课题的过程中，我们从图像的存储到图像的纹理特征提取及输出。通过这一过程，我们对于图像在计算机中的存在形式有了深刻的认识。其间我们也遇到了一些问题并且尝试去解决了它们，同时也学到了一些新的知识。

1、图像由彩色图转为灰度图的方式：在这个过程中，我们学习了几种不同的转换为灰度图的方式，最后选择按照一定的权值进行加权平均得到，使用了 opencv 的 CV_BGR2GRAY。

2、在展示界面的制作过程中，我们使用了 Python 自带的 tkinter 库，通过这次的界面制作，我们对于前端有了一定的了解，特别是在界面布局上，放弃了原始的 pack()布局，使用了更为灵活的 grid()布局。最后在图片展示上，我们使用了 Carvas 控件。

3、在设计灰度共生矩阵算法过程中，我们使用了 Python 中的 numpy 和 openCV 库，对数字图像的处理有了更深的认识。应采用逐步求精的思想，先建立内容的框架并完成重要部分的内容，再去讨论细枝末节的处理。

4、在算法到具体实验中，需要考虑时间复杂度、可行性等因素，滑动窗口大小的选择、步长的选择、灰度矩阵维度的选择，要保证细粒度、精度的同时，尽量降低算法时间复杂度，从计划与分析开始初步体会工程实践的意义。

## 五、分工合作

刘焕宇：负责核心算法灰度共生矩阵的实现

完成情况：实现了图片各种纹理特征的提取及输出。

后胜涛：负责图片的存取和界面展示

完成情况：实现了目标图片的选取已经将提取的特征图片进行存储和展示。

## 附录

**GLCM 算法部分**

```
import cv2 as cv
import numpy as np
```

```
import math
import common


class GLCM:
    # 读入图片转灰度图

    def __init__(self):
        cv.imwrite("d:/img.jpg",common.img)
        img = cv.resize(common.img, (128, 128))
        gry = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        self.n, self.m = gry.shape   # 图片长宽
        self.gray_level = 256
        # 灰度等级，整张图的为256
        glcm = self.calc_glcm(gry, 0)
        cv.imwrite("d:/imgGray.jpg", glcm * 255)
        self.gray_level = 8   # 灰度等级，默认为8
        rec = 256 / self.gray_level
        for i in range(self.n):
            for j in range(self.m):
                gry[i][j] /= rec

        for i in range(0,8):
            ans = self.work(gry, i)
            cv.imwrite("d:/img" + str(i+1) + ".jpg", ans * 255)
        cv.waitKey(1)
        print('1')

    def calc_glcm(self, img, opt, d=1):   # Gray Level Co-occurrence Matrix 灰度共生矩阵
        # 默认步长 d = 1，水平计算
        ret = np.zeros([self.gray_level, self.gray_level])
        # print(img)
        h = img.shape[0]
        w = img.shape[1]
        if opt != 7:
            for i in range(h):
                for j in range(w - d):
                    x = img[i][j]
                    y = img[i][j + d]
                    ret[x][y] += 1
        else:
            for i in range(h - d):
                for j in range(w):
```

```
                    x = img[i][j]
                    y = img[i + d][j]
                    ret[x][y] += 1
        ret /= ret.max()
        return ret

    def work(self, a, opt, win=5):
        pad = int(win / 2)
        ret = np.zeros([self.n, self.m])
        for i in range(pad, self.n - pad):
            for j in range(pad, self.m - pad):
                ag = self.calc_glcm(a[i - pad: i + pad + 1, j - pad: j + pad + 1],
opt, )
                ret[i][j] = self.feature_computer(ag, opt)
        ret /= ret.max()
        return ret

    def feature_computer(self, a, opt):
        if opt == 0 or opt == 7:   # 均值
            return a.mean()
        if opt == 1:   # 标准差
            return a.std()
        Asm = 0.0   # 角二阶矩 Angular Second Moment (Energy)
        Ent = 0.0   # 熵
        Con = 0.0   # 对比度 Contrast
        Idm = 0.0   # 同质性
        for i in range(0,self.gray_level):
            for j in range(0,self.gray_level):
                Asm += a[i][j] ** 2
                Con += (i - j) ** 2 * a[i][j]
                if a[i][j] > 0:
                    Ent += a[i][j] * math.log(a[i][j])
                Idm += a[i][j] / (1 + (i - j) ** 2)

        if opt == 2 or opt == 5:   # 角二阶矩/ 能量
            return Asm
        if opt == 3:   # 熵
            return -Ent
        if opt == 4:   # 对比度
            return Con
        if opt == 6:   # 同质性
            return Idm
```

**前端界面代码部分**

```
from tkinter import END
import cv2
import tkinter as tk
import numpy as np
from tkinter import *
from PIL import Image, ImageTk
import os
import tkinter.filedialog as tkf
import common
import GLCM


class frame:
    window = tk.Tk()
    selectbt = tk.Button()
    canvas = tk.Canvas()
    canvas1 = tk.Canvas()
    canvas2 = tk.Canvas()
    canvas3 = tk.Canvas()
    canvas4 = tk.Canvas()
    canvas5 = tk.Canvas()
    canvas6 = tk.Canvas()
    canvas7 = tk.Canvas()
    canvas8 = tk.Canvas()
    canvas9 = tk.Canvas()

    def __init__(self):
        self.window_set()
        self.button_set()
        self.canvas_set()
        self.window.mainloop()

    def window_set(self):
        self.window.title("图像纹理特征提取")
        self.window.geometry("800x500")
        width = 1000
        height = 700
        ws = self.window.winfo_screenwidth()
        hs = self.window.winfo_screenheight()
        x = (ws / 2) - (width / 2)
        y = (hs / 2) - (height / 2)
        self.window.geometry('%dx%d+%d+%d' % (width, height, x, y))
```

```
        self.window.resizable(False, False)
        self.window.configure(bg='#FAFAD2')
        self.window.rowconfigure(1, weight=1)
        self.window.columnconfigure(0, weight=1)

    def button_set(self):
        self.selectbt = tk.Button(self.window, text='导入图片', font=('中文简体',
15), fg='black', bg='#00FFEE', height=1,
                                  width=20, command=self.PictureSelect)
        self.selectbt.grid(padx=0, pady=0, row=4, column=0, columnspan=2)

    def canvas_set(self):
        self.canvas = Canvas(self.window, bg='#FAFAD2', width=512,
height=512)
        self.canvas.grid(padx=5, pady=5, row=1, column=0, rowspan=3)
        self.canvas1 = Canvas(self.window, bg='#FAFAD2', width=128,
height=128)
        self.canvas1.grid(padx=5, pady=5, row=1, column=1)
        self.canvas2 = Canvas(self.window, bg='#FAFAD2', width=128,
height=128)
        self.canvas2.grid(padx=5, pady=5, row=1, column=2)
        self.canvas3 = Canvas(self.window, bg='#FAFAD2', width=128,
height=128)
        self.canvas3.grid(padx=5, pady=5, row=2, column=1)
        self.canvas4 = Canvas(self.window, bg='#FAFAD2', width=128,
height=128)
        self.canvas4.grid(padx=5, pady=5, row=2, column=2)
        self.canvas5 = Canvas(self.window, bg='#FAFAD2', width=128,
height=128)
        self.canvas5.grid(padx=5, pady=5, row=3, column=1)
        self.canvas6 = Canvas(self.window, bg='#FAFAD2', width=128,
height=128)
        self.canvas6.grid(padx=5, pady=5, row=3, column=2)
        self.canvas7 = Canvas(self.window, bg='#FAFAD2', width=128,
height=128)
        self.canvas7.grid(padx=5, pady=5, row=4, column=1)
        self.canvas8 = Canvas(self.window, bg='#FAFAD2', width=128,
height=128)
        self.canvas8.grid(padx=5, pady=5, row=4, column=2)

    def PictureSelect(self):
        filepath = tkf.askopenfilename()
        common.img = cv2.imread(filepath)
        glcm = GLCM.GLCM()
```

```
        img = Image.open("d:/img.jpg")
        bm = ImageTk.PhotoImage(img)
        self.canvas.create_image(256, 256, anchor=CENTER, image=bm)
        self.GLCMshow()
        self.window.mainloop()

    def GLCMshow(self):
        img1 = Image.open("d:/img1.jpg")
        bm1 = ImageTk.PhotoImage(img1)
        self.canvas1.create_image(64, 64, anchor=CENTER, image=bm1)

        img2 = Image.open("d:/img2.jpg")
        bm2 = ImageTk.PhotoImage(img2)
        self.canvas2.create_image(64, 64, anchor=CENTER, image=bm2)

        img3 = Image.open("d:/img3.jpg")
        bm3 = ImageTk.PhotoImage(img3)
        self.canvas3.create_image(64, 64, anchor=CENTER, image=bm3)

        img4 = Image.open("d:/img4.jpg")
        bm4 = ImageTk.PhotoImage(img4)
        self.canvas4.create_image(64, 64, anchor=CENTER, image=bm4)

        img5 = Image.open("d:/img5.jpg")
        bm5 = ImageTk.PhotoImage(img5)
        self.canvas5.create_image(64, 64, anchor=CENTER, image=bm5)

        img6 = Image.open("d:/img6.jpg")
        bm6 = ImageTk.PhotoImage(img6)
        self.canvas6.create_image(64, 64, anchor=CENTER, image=bm6)

        img7 = Image.open("d:/img7.jpg")
        bm7 = ImageTk.PhotoImage(img7)
        self.canvas7.create_image(64, 64, anchor=CENTER, image=bm7)

        img8 = Image.open("d:/img8.jpg")
        bm8 = ImageTk.PhotoImage(img8)
        self.canvas8.create_image(64, 64, anchor=CENTER, image=bm8)

        self.window.mainloop()


if __name__ == '__main__':
    myframe = frame()
```