重庆大学

数学建模校内竞赛论文

论文题目：

组号：

成员：

选题：

姓名	学院	年级	专业	学号	联系电话	数学分析	高等代数	微积分	高等数学	线性代数	概率统计	数学实验	数学模型	CET4	CET6

日期

摘 要

待补全

关键词：少样本分类；关系建模；对比学习；语义信息表示

Key words:

目 录

图目录

表目录

1 绪 论

dawd[1]

1.1 研究背景与意义

1.2 问题提出与研究内容

2 问题分析

2.1 问题重述

2.2 理论基础

3 模型建立与求解

3.1 问题 1：颜色空间转换模型

3.2 问题 2：四通道到五通道颜色转换模型

3.3 问题 3：LED 显示器颜色校正模型

4 模型实现与结果分析

4.1 数据集介绍

4.2 实验设计

4.3 结果与分析

5 模型评价

5.1 数据集介绍

5.2 实验设计

5.3 结果与分析

6 结论与展望

6.1 主要结论

6.2 不足与改进方向

参考文献

[1] Finn C, Abbeel P, Levine S. Model-agnostic meta-learning for fast adaptation of deep networks [C]//International Conference on Machine Learning. 2017: 1126-1135.

附 录

A. 问题 1 使用代码

```
def greet(name):
    print(f"Hello, {name}!")

greet("ChatGPT")
```

B. 问题 2 使用代码

```
def greet(name):
    print(f"Hello, {name}!")

greet("ChatGPT")
```

C. 问题 3 使用代码

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.optimize import minimize, differential_evolution

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei', 'DejaVu Sans']
plt.rcParams['axes.unicode_minus'] = False

class LEDColorCorrection:
    """
    基于三基色原理和CIE Lab色彩空间的颜色校正
    使用差分进化算法优化校正矩阵
    """

    def __init__(self):
        self.correction_matrix = None
        self.correction_bias = None
        self.gamma_correction = None
        self.measured_data = None
        self.target_data = None

    def load_excel_data(self, excel_path):
        """从Excel文件加载数据"""
        print(f"正在加载Excel文件: {excel_path}")

        sheets = ['R', 'G', 'B', 'target_R', 'target_G', 'target_B']
        data_dict = {}
```

```
29
30          for sheet_name in sheets:
31              df = pd.read_excel(excel_path, sheet_name=sheet_name, header=None)
                    ↪ .iloc[0:64,0:64]
32              data_dict[sheet_name] = df.values
33              print(f"已加载工作表 '{sheet_name}': {df.shape}")
34
35          # 组织数据
36          self.measured_data = np.stack([
37              data_dict['R'],
38              data_dict['G'],
39              data_dict['B']
40          ], axis=-1)
41
42          self.target_data = np.stack([
43              data_dict['target_R'],
44              data_dict['target_G'],
45              data_dict['target_B']
46          ], axis=-1)
47
48          print(f"测量数据形状: {self.measured_data.shape}")
49          print(f"目标数据形状: {self.target_data.shape}")
50
51      def rgb_to_xyz(self, rgb):
52          """RGB转XYZ色彩空间"""
53          rgb_norm = rgb / 255.0
54
55          # Gamma校正
56          rgb_linear = np.where(rgb_norm <= 0.04045,
57                                rgb_norm / 12.92,
58                                np.power((rgb_norm + 0.055) / 1.055, 2.4))
59
60          # sRGB到XYZ的转换矩阵
61          transform_matrix = np.array([
62              [0.4124564, 0.3575761, 0.1804375],
63              [0.2126729, 0.7151522, 0.0721750],
64              [0.0193339, 0.1191920, 0.9503041]
65          ])
66
67          xyz = np.dot(rgb_linear, transform_matrix.T)
68          return xyz
69
70      def xyz_to_lab(self, xyz):
71          """XYZ转CIE Lab色彩空间"""
72          # D65白点
73          Xn, Yn, Zn = 0.95047, 1.00000, 1.08883
74
75          x = xyz[..., 0] / Xn
```

```
76          y = xyz[..., 1] / Yn
77          z = xyz[..., 2] / Zn
78
79          # 立方根变换
80          fx = np.where(x > 0.008856, np.power(x, 1/3), (7.787 * x + 16/116))
81          fy = np.where(y > 0.008856, np.power(y, 1/3), (7.787 * y + 16/116))
82          fz = np.where(z > 0.008856, np.power(z, 1/3), (7.787 * z + 16/116))
83
84          L = 116 * fy - 16
85          a = 500 * (fx - fy)
86          b = 200 * (fy - fz)
87
88          return np.stack([L, a, b], axis=-1)
89
90      def calculate_color_difference(self, lab1, lab2):
91          """计算CIE Delta E 2000色差"""
92          L1, a1, b1 = lab1[..., 0], lab1[..., 1], lab1[..., 2]
93          L2, a2, b2 = lab2[..., 0], lab2[..., 1], lab2[..., 2]
94
95          C1 = np.sqrt(a1**2 + b1**2)
96          C2 = np.sqrt(a2**2 + b2**2)
97          C_bar = 0.5 * (C1 + C2)
98
99          G = 0.5 * (1 - np.sqrt(C_bar**7 / (C_bar**7 + 25**7)))
100         a1p = (1 + G) * a1
101         a2p = (1 + G) * a2
102
103         C1p = np.sqrt(a1p**2 + b1**2)
104         C2p = np.sqrt(a2p**2 + b2**2)
105
106         h1p = np.degrees(np.arctan2(b1, a1p)) % 360
107         h2p = np.degrees(np.arctan2(b2, a2p)) % 360
108
109         dLp = L2 - L1
110         dCp = C2p - C1p
111
112         dhp = h2p - h1p
113         dhp = dhp - 360 * (dhp > 180) + 360 * (dhp < -180)
114         dHp = 2 * np.sqrt(C1p * C2p) * np.sin(np.radians(dhp / 2))
115
116         L_bar = 0.5 * (L1 + L2)
117         C_bar_p = 0.5 * (C1p + C2p)
118
119         h_bar_p = (h1p + h2p + 360 * (np.abs(h1p - h2p) > 180)) / 2
120         h_bar_p %= 360
121
122         T = (1
123             - 0.17 * np.cos(np.radians(h_bar_p - 30))
```

```
            + 0.24 * np.cos(np.radians(2 * h_bar_p))
            + 0.32 * np.cos(np.radians(3 * h_bar_p + 6))
            - 0.20 * np.cos(np.radians(4 * h_bar_p - 63)))

        Sl = 1 + (0.015 * (L_bar - 50)**2) / np.sqrt(20 + (L_bar - 50)**2)
        Sc = 1 + 0.045 * C_bar_p
        Sh = 1 + 0.015 * C_bar_p * T

        delta_theta = 30 * np.exp(-((h_bar_p - 275)/25)**2)
        Rc = 2 * np.sqrt(C_bar_p**7 / (C_bar_p**7 + 25**7))
        Rt = -np.sin(np.radians(2 * delta_theta)) * Rc

        dE = np.sqrt(
            (dLp / Sl)**2 +
            (dCp / Sc)**2 +
            (dHp / Sh)**2 +
            Rt * (dCp / Sc) * (dHp / Sh)
        )

        return dE

    def estimate_gamma_parameters(self):
        """估计LED的Gamma参数（保留线性比例偏移）"""
        print("正在估计Gamma参数...")
        gamma_params = {}
        for i, channel in enumerate(['R', 'G', 'B']):
            meas = self.measured_data[..., i].flatten() / 255.0
            targ = self.target_data[..., i].flatten() / 255.0
            mask = (targ > 0.04) & (targ < 0.96) & (meas > 0)
            m = meas[mask]
            t = targ[mask]
            if len(m) > 0:
                # 拟合 log(m) = gamma * log(t) + offset
                A = np.vstack([np.log(t + 1e-8), np.ones_like(t)]).T
                gamma, offset = np.linalg.lstsq(A, np.log(m + 1e-8), rcond=
                    ↪ None)[0]
                gamma = float(np.clip(gamma, 0.1, 3.0))
                scale = float(np.exp(offset))
            else:
                gamma, scale = 1.0, 1.0
            gamma_params[channel] = {'gamma': gamma, 'scale': scale}
            print(f"{channel}通道 Gamma: {gamma:.3f}, Scale: {scale:.3f}")
        self.gamma_correction = gamma_params
        return gamma_params

    def apply_gamma_correction(self, rgb_data, inverse=False):
        """应用Gamma校正：在归一化 [0,1] 空间先应用线性比例，再做幂运算"""
        if self.gamma_correction is None:
```

```
            return rgb_data
        data = rgb_data.astype(np.float32) / 255.0
        out = np.zeros_like(data)
        for i, channel in enumerate(['R', 'G', 'B']):
            gamma = self.gamma_correction[channel]['gamma']
            scale = self.gamma_correction[channel]['scale']
            ch = data[..., i]
            if not inverse:
                # 前向：先比例，再幂
                tmp = ch * scale
                tmp = np.clip(tmp, 0.0, 1.0)
                out_ch = np.power(tmp, gamma)
            else:
                # 反向：开幂，再去比例
                tmp = np.power(ch, 1.0 / gamma)
                out_ch = tmp / np.maximum(scale, 1e-8)
            out[..., i] = np.clip(out_ch, 0.0, 1.0)
        # 恢复到 [0,255]
        return (out * 255.0).astype(rgb_data.dtype)

    def correction_function(self, params, measured_lin, target_lin):
        """
        优化函数：线性校正矩阵 M 和偏置 b，params 长度 12。
        corrected = clip(M @ measured + b, [0,1])
        计算 ΔE  + 正则化。
        """
        M = params[:9].reshape(3,3)
        b = params[9:].reshape(1,3)

        # 应用矩阵和偏置
        corr = np.dot(measured_lin, M.T) + b
        corr = np.clip(corr, 0.0, 1.0)

        # 转到 XYZ → Lab
        transform = np.array([[0.4124564,0.3575761,0.1804375],
                              [0.2126729,0.7151522,0.0721750],
                              [0.0193339,0.1191920,0.9503041]])
        tgt_xyz = np.dot(target_lin, transform.T)
        corr_xyz = np.dot(corr, transform.T)
        tgt_lab = self.xyz_to_lab(tgt_xyz.reshape(-1,3)).reshape(corr.shape)
        corr_lab = self.xyz_to_lab(corr_xyz.reshape(-1,3)).reshape(corr.shape)

        # 色差
        deltaE = self.calculate_color_difference(tgt_lab, corr_lab)
        loss = np.mean(deltaE)

        # 矩阵正则 + 偏置正则
        loss += 0.001 * (np.sum((M - np.eye(3))**2) + np.sum(b**2))
```

```
219         det = np.linalg.det(M)
220         if det <= 0 or abs(det) < 0.1:
221             loss += 1000.0
222         return loss
223
224
225     def calibrate_correction_matrix(self):
226         print("开始校正: 矩阵 + 偏置...")
227         self.estimate_gamma_parameters()
228         # 预处理: 线性化
229         meas = self.apply_gamma_correction(self.measured_data.astype(np.
                ↪ float32), inverse=True)/255.0
230         targ = self.apply_gamma_correction(self.target_data.astype(np.float32)
                ↪ , inverse=True)/255.0
231         meas_flat = meas.reshape(-1,3)
232         targ_flat = targ.reshape(-1,3)
233         # 差分进化优化 12 参数
234         bounds = [(-2,2)]*9 + [(-0.1,0.1)]*3
235         res = differential_evolution(
236             self.correction_function, bounds,
237             args=(meas_flat, targ_flat), maxiter=200, popsize=15, seed=42
238         )
239         x0 = res.x
240         # 局部 L-BFGS-B
241         local = minimize(
242             self.correction_function, x0, args=(meas_flat,targ_flat),
243             method='L-BFGS-B', options={'maxiter':500}
244         )
245         M_opt = local.x[:9].reshape(3,3)
246         b_opt = local.x[9:].reshape(3)
247         self.correction_matrix = M_opt
248         self.correction_bias = b_opt
249         print("校正完成; 矩阵行列式: ", np.linalg.det(M_opt))
250         print("偏置: ", b_opt)
251         return M_opt, b_opt
252
253
254     def apply_correction(self, input_rgb):
255         """应用带偏置的线性校正"""
256         lin = self.apply_gamma_correction(input_rgb.astype(np.float32),
                ↪ inverse=True)/255.0
257         flat = lin.reshape(-1,3)
258         corr = np.dot(flat, self.correction_matrix.T) + self.correction_bias
259         corr = np.clip(corr, 0.0, 1.0).reshape(input_rgb.shape)
260         out = (corr * 255.0).astype(np.float32)
261         final = self.apply_gamma_correction(out, inverse=False)
262         return final.astype(np.uint8)
263
```

```
264     def evaluate_correction(self):
265         """评估校正效果"""
266         corrected = self.apply_correction(self.measured_data.astype(np.float32
                ↪ ))
267 
268         measured_xyz = self.rgb_to_xyz(self.measured_data.astype(np.float32))
269         corrected_xyz = self.rgb_to_xyz(corrected.astype(np.float32))
270         target_xyz = self.rgb_to_xyz(self.target_data.astype(np.float32))
271 
272         measured_lab = self.xyz_to_lab(measured_xyz)
273         corrected_lab = self.xyz_to_lab(corrected_xyz)
274         target_lab = self.xyz_to_lab(target_xyz)
275 
276         diff_before = self.calculate_color_difference(measured_lab, target_lab
                ↪ )
277         diff_after = self.calculate_color_difference(corrected_lab, target_lab
                ↪ )
278 
279         print("="*50)
280         print("校正效果评估报告")
281         print("="*50)
282         print(f"校正前平均色差: {np.mean(diff_before):.3f}")
283         print(f"校正后平均色差: {np.mean(diff_after):.3f}")
284         print(f"色差改善: {np.mean(diff_before) - np.mean(diff_after):.3f}")
285         print(f"改善百分比: {((np.mean(diff_before) - np.mean(diff_after)) /
                ↪ np.mean(diff_before) * 100):.1f}%")
286         print(f"校正前最大色差: {np.max(diff_before):.3f}")
287         print(f"校正后最大色差: {np.max(diff_after):.3f}")
288         print(f"色差<1.0的像素比例: 校正前{np.mean(diff_before < 1.0)*100:.1f
                ↪ }%, 校正后{np.mean(diff_after < 1.0)*100:.1f}%")
289         print("="*50)
290 
291         return corrected, diff_before, diff_after
292 
293     def visualize_results(self):
294         """可视化校正结果"""
295         corrected_data = self.apply_correction(self.measured_data.astype(np.
                ↪ float32))
296 
297         fig, axes = plt.subplots(3, 4, figsize=(20, 15))
298 
299         # 第一行: 测量数据
300         for i, (channel, color) in enumerate(zip(['R', 'G', 'B'], ['Reds', '
                ↪ Greens', 'Blues'])):
301             im = axes[0, i].imshow(self.measured_data[:, :, i], cmap=color,
                    ↪ vmin=0, vmax=255)
302             axes[0, i].set_title(f'测量值 - {channel} 通道')
303             axes[0, i].axis('off')
```

```
304             plt.colorbar(im, ax=axes[0, i], fraction=0.046, pad=0.04)
305
306         measured_rgb = np.clip(self.measured_data / 255.0, 0, 1)
307         axes[0, 3].imshow(measured_rgb)
308         axes[0, 3].set_title('测量值 - RGB合成')
309         axes[0, 3].axis('off')
310
311         # 第二行: 目标数据
312         for i, (channel, color) in enumerate(zip(['R', 'G', 'B'], ['Reds', '
                ↪ Greens', 'Blues'])):
313             im = axes[1, i].imshow(self.target_data[:, :, i], cmap=color, vmin
                    ↪ =0, vmax=255)
314             axes[1, i].set_title(f'目标值 - {channel} 通道')
315             axes[1, i].axis('off')
316             plt.colorbar(im, ax=axes[1, i], fraction=0.046, pad=0.04)
317
318         target_rgb = np.clip(self.target_data / 255.0, 0, 1)
319         axes[1, 3].imshow(target_rgb)
320         axes[1, 3].set_title('目标值 - RGB合成')
321         axes[1, 3].axis('off')
322
323         # 第三行: 校正后数据
324         for i, (channel, color) in enumerate(zip(['R', 'G', 'B'], ['Reds', '
                ↪ Greens', 'Blues'])):
325             im = axes[2, i].imshow(corrected_data[:, :, i], cmap=color, vmin
                    ↪ =0, vmax=255)
326             axes[2, i].set_title(f'校正后 - {channel} 通道')
327             axes[2, i].axis('off')
328             plt.colorbar(im, ax=axes[2, i], fraction=0.046, pad=0.04)
329
330         corrected_rgb = np.clip(corrected_data / 255.0, 0, 1)
331         axes[2, 3].imshow(corrected_rgb)
332         axes[2, 3].set_title('校正后 - RGB合成')
333         axes[2, 3].axis('off')
334
335         plt.tight_layout()
336         plt.show()
337
338
339 # 主函数
340 if __name__ == "__main__":
341     files = ["MathModel_Code\\data\\preprocess\\p3\\RedPicture.xlsx", "
            ↪ MathModel_Code\\data\\preprocess\\p3\\GreenPicture.xlsx", "
            ↪ MathModel_Code\\data\\preprocess\\p3\\BluePicture.xlsx"]
342
343     corrector = LEDColorCorrection()
344
345     for filepath in files:
```

```
346         corrector.load_excel_data(filepath)
347         correction_matrix = corrector.calibrate_correction_matrix()
348
349         print("\n评估校正效果:")
350         corrected_display, diff_before, diff_after = corrector.
                ↪ evaluate_correction()
351
352         corrector.visualize_results()
353
354         print("\n校正完成!")
```

D. 像素数据集