AI 追色算法技术报告

一、技术概述

AI 追色是一种基于颜色迁移的图像处理技术，核心目标是将参考图像的颜色风格（包括影调和色彩）迁移到目标图像上，实现快速且精准的色彩风格统一。该技术在影楼修图、图像编辑等领域有广泛应用，例如像素蛋糕软件中的 AI 追色功能，支持全图和局部区域的精细化调整，可针对人像的不同部位（如皮肤、头发、衣物等）进行独立的影调和色彩控制。

二、核心原理与技术实现

（一）颜色空间转换

算法基于 Lab 颜色空间实现影调和色彩的解耦处理。Lab 颜色空间将颜色分为三个通道：

- **L 通道（影调 / 亮度）**：表示图像的明暗程度，范围 0-100（0 为黑，100 为白）。
- **a 通道（色彩）**：表示从绿色到红色的色彩变化。
- **b 通道（色彩）**：表示从蓝色到黄色的色彩变化。

通过将图像从 RGB 空间转换为 Lab 空间，可独立调节亮度（影调）和色彩（a、b 通道），避免传统 RGB 空间中亮度与色彩强耦合的问题。

2. **分区域迁移**
   对不同区域分别应用颜色迁移算法，例如：
   - 背景区域：全局迁移参考图像的影调和色彩。
   - 皮肤区域：优先保留自然肤色，仅微调影调或轻度色彩匹配。
   - 头发 / 衣物区域：允许更激进的色彩迁移，以匹配参考图的风格。

   通过分区域处理，避免全局迁移导致的细节失真（如肤色异常），同时实现复杂场景下的精准风格统一。
3. **边界融合**
   对相邻分割区域的边界进行高斯模糊或线性插值，消除因分割不精确导致的颜色断层，提升整体过渡的自然度。

三、算法流程与代码实现（简化示例）

（一）核心步骤

1. **图像预处理**
   - 将目标图像和参考图像从 RGB 转换为 Lab 颜色空间。
   - 生成各区域的掩码（Mask），如背景掩码、皮肤掩码等。
2. **统计特征计算**
   - 对每个分割区域，分别计算 Lab 均值和标准差。
3. **分区域颜色迁移**
   - 根据掩码遍历目标图像的像素，仅对掩码覆盖区域执行颜色迁移。
   - 应用强度比例和影调 / 色彩参数，调整迁移效果。
4. **后处理**
   - 将图像从 Lab 转换回 RGB 空间。
   - 全局融合原图与效果图（通过参数k控制融合比[illegible]），避免过度处理。

（二）颜色迁移算法

1. **基础原理**
   基于经典颜色迁移算法《Color Transfer Between Images》，通过统计参考图像和目标图像在 Lab 空间的均值与标准差，实现颜色风格的匹配。具体步骤如下：
   - 计算目标图像的 Lab 均值（$\mu_{tgt-L}, \mu_{tgt-a}, \mu_{tgt-b}$）和标准差（$\sigma_{tgt-L}, \sigma_{tgt-a}, \sigma_{tgt-b}$）。
   - 计算参考图像的 Lab 均值（$\mu_{ref-L}, \mu_{ref-a}, \mu_{ref-b}$）和标准差（$\sigma_{ref-L}, \sigma_{ref-a}, \sigma_{ref-b}$）。
   - 对目标图像的每个像素，通过以下公式完成颜色迁移：$\begin{cases} L' = \mu_{ref-L} + \frac{\sigma_{ref-L}}{\sigma_{tgt-L}}(L - \mu_{tgt-L}) \\ a' = \mu_{ref-a} + \frac{\sigma_{ref-a}}{\sigma_{tgt-a}}(a - \mu_{tgt-a}) \\ b' = \mu_{ref-b} + \frac{\sigma_{ref-b}}{\sigma_{tgt-b}}(b - \mu_{tgt-b}) \end{cases}$
     其中，L, a, b为目标图像像素的原始 Lab 值，L', a', b'为迁移后的 Lab 值。
2. **参数调节**
   - **强度比例（$ratio$）**：通过参数$ratio$控制迁移强度（0-100%），公式为：
     $\begin{cases} L'' = L' \times \frac{ratio}{100} + L \times (1 - \frac{ratio}{100}) \\ a'' = a' \times \frac{ratio}{100} + a \times (1 - \frac{ratio}{100}) \\ b'' = b' \times \frac{ratio}{100} + b \times (1 - \frac{ratio}{100}) \end{cases}$
     实现从完全保留原图（$ratio = 0$）到完全迁移（$ratio = 100$）的平滑过渡。
   - **影调与色彩独立控制**：引入`luminance_ratio`（影调比例）和`chrominance_ratio`（色彩比例），分别控制亮度（L 通道）和色彩（a、b 通道）的迁移强度，实现更精细的风格调整。

（三）区域分割与精细化处理

1. **图像分割**
   通过人像分割技术（如语义分割模型）将图像分为主体（人像）和背景，主体进一步细分为面部皮肤、身体皮肤、头发、衣物等 11 个区域。每个区域生成独立的掩码（Mask），用于限定颜色迁移的作用范围。

C 伪代码：

```
// 颜色迁移主函数（带掩码）
void color_transfer_mask(
    unsigned char* target_img, // 目标图像（BGRA 格式）
    unsigned char* ref_img, // 参考图像（BGRA 格式）
    unsigned char* mask, // 区域掩码（0-255，非零表示有效区域）
    int width, int height, int stride,
    int luminance_ratio, // 影调迁移比例（0-100）
    int chrominance_ratio // 色彩迁移比例（0-100）
) {
    // 计算目标区域与参考区域的 Lab 统计量（均值、标准差）
    compute_lab_stats(target_img, width, height, stride, mask, &tgt_mean, &tgt_std);
    compute_lab_stats(ref_img, width, height, stride, mask, &ref_mean, &ref_std);
// 遍历像素并迁移
    for (int y=0; y<height; y++) {
        for (int x=0; x<width; x++) {
            if (mask[y*width+x] == 0) continue; // 跳过非目标区域
// 转换为 Lab 颜色空间
            rgb_to_lab(target_img, &L, &a, &b);
            // 应用影调迁移（L 通道）
```

```
            double new_L = ref_mean.L + (ref_std.L / tgt_std.L) * (L - tgt_mean.L);
            new_L = new_L * (luminance_ratio/100.0) + L * (1 - luminance_ratio/100.0);
              // 应用色彩迁移（a、b 通道）
            double new_a = ref_mean.a + (ref_std.a / tgt_std.a) * (a - tgt_mean.a);
            new_a = new_a * (chrominance_ratio/100.0) + a * (1 - chrominance_ratio/100.0);
            // 同理处理 b 通道...
// 转换回 RGB 并更新像素
            lab_to_rgb(new_L, new_a, new_b, &r, &g, &b);
            update_pixel(target_img, r, g, b);
        }
    }
}
```

py 代码（该算法仅色彩迁移，不带图像分割，需自行选用合适的分割计算，这里运行的结果不是最终效果）：

```
import tkinter as tk

from tkinter import filedialog, messagebox, ttk

import numpy as np

from PIL import Image, ImageTk

import cv2

import os

import sys


# 颜色迁移核心算法（基于原 C 代码的 Python 实现）

class ColorTransfer:

def __init__(self):

# 参考白点(D65)

self.REF_X = 0.95047

self.REF_Y = 1.00000

self.REF_Z = 1.08883

def rgb_to_lab(self, r, g, b):

"""将 RGB 颜色转换为 Lab 颜色空间"""

# 转换 sRGB 到线性 RGB

r_lin = pow((r + 0.055) / 1.055, 2.4) if r > 0.04045 else r / 12.92

g_lin = pow((g + 0.055) / 1.055, 2.4) if g > 0.04045 else g / 12.92

b_lin = pow((b + 0.055) / 1.055, 2.4) if b > 0.04045 else b / 12.92

# RGB 到 XYZ 转换

X = r_lin * 0.412453 + g_lin * 0.357580 + b_lin * 0.180423

Y = r_lin * 0.212671 + g_lin * 0.715160 + b_lin * 0.072169

Z = r_lin * 0.019334 + g_lin * 0.119193 + b_lin * 0.950227

# XYZ 归一化

X /= self.REF_X

Y /= self.REF_Y

Z /= self.REF_Z

# XYZ 到 Lab 转换

fX = pow(X, 1.0/3.0) if X > 0.008856 else (7.787 * X) + (16.0/116.0)
```

```
fY = pow(Y, 1.0/3.0) if Y > 0.008856 else (7.787 * Y) + (16.0/116.0)

fZ = pow(Z, 1.0/3.0) if Z > 0.008856 else (7.787 * Z) + (16.0/116.0)

L = (116.0 * fY) - 16.0

a = 500.0 * (fX - fY)

b_ = 200.0 * (fY - fZ)

return L, a, b_

def lab_to_rgb(self, L, a, b_):

"""将 Lab 颜色转换为 RGB 颜色空间"""

# Lab 到 XYZ 转换

fY = (L + 16.0) / 116.0

fX = a / 500.0 + fY

fZ = fY - b_ / 200.0

Y = pow(fY, 3.0) if pow(fY, 3.0) > 0.008856 else (fY - 16.0/116.0) / 7.787

X = pow(fX, 3.0) if pow(fX, 3.0) > 0.008856 else (fX - 16.0/116.0) / 7.787

Z = pow(fZ, 3.0) if pow(fZ, 3.0) > 0.008856 else (fZ - 16.0/116.0) / 7.787

# 归一化

X *= self.REF_X

Y *= self.REF_Y

Z *= self.REF_Z

# XYZ 到 RGB 转换

r_lin = X * 3.240479 + Y * -1.537150 + Z * -0.498535

g_lin = X * -0.969256 + Y * 1.875992 + Z * 0.041556

b_lin = X * 0.055648 + Y * -0.204043 + Z * 1.057311

# 线性 RGB 到 sRGB 转换

r = 1.055 * pow(r_lin, 1.0/2.4) - 0.055 if r_lin > 0.0031308 else 12.92 * r_lin

g = 1.055 * pow(g_lin, 1.0/2.4) - 0.055 if g_lin > 0.0031308 else 12.92 * g_lin

b = 1.055 * pow(b_lin, 1.0/2.4) - 0.055 if b_lin > 0.0031308 else 12.92 * b_lin

# 限制在 0-1 范围内

r = max(0.0, min(1.0, r))

g = max(0.0, min(1.0, g))

b = max(0.0, min(1.0, b))

return r, g, b

def compute_lab_stats(self, img):

"""计算图像在 Lab 颜色空间的均值和标准差"""

height, width = img.shape[:2]

sumL = suma = sumb = 0.0

sumL2 = suma2 = sumb2 = 0.0

count = 0

for y in range(height):

for x in range(width):

b, g, r = img[y, x] / 255.0

L, a, b_ = self.rgb_to_lab(r, g, b)

sumL += L

suma += a

sumb += b_
```

```
sumL2 += L * L

suma2 += a * a

sumb2 += b_ * b_

count += 1

if count == 0:

return 0.0, 0.0, 0.0, 0.0, 0.0, 0.0

# 计算均值

meanL = sumL / count

meana = suma / count

meanb = sumb / count

# 计算标准差

stdL = np.sqrt((sumL2 - sumL * sumL / count) / (count - 1))

stda = np.sqrt((suma2 - suma * suma / count) / (count - 1))

stdb = np.sqrt((sumb2 - sumb * sumb / count) / (count - 1))

return meanL, meana, meanb, stdL, stda, stdb

def color_transfer(self, source_img, target_img, luminance_ratio=50, chrominance_ratio=80):

"""执行颜色迁移"""

# 计算源图像和目标图像的 Lab 统计量

source_meanL, source_meana, source_meanb, source_stdL, source_stda, source_stdb = self.compute_lab_stats(source_img)

target_meanL, target_meana, target_meanb, target_stdL, target_stda, target_stdb = self.compute_lab_stats(target_img)

# 应用迁移强度比例

luminance_scale = luminance_ratio / 100.0

chrominance_scale = chrominance_ratio / 100.0

# 创建结果图像

result_img = np.copy(target_img)

height, width = target_img.shape[:2]

# 对每个像素进行颜色迁移

for y in range(height):

for x in range(width):

# 获取原始像素值

b, g, r = target_img[y, x] / 255.0

# 转换到 Lab 空间

L, a, b_ = self.rgb_to_lab(r, g, b)

# 应用亮度迁移

newL = target_meanL + (source_meanL - target_meanL) + \

(source_stdL / (target_stdL if target_stdL > 0 else 1.0)) * (L - target_meanL)

# 应用色度迁移

newa = target_meana + (source_meana - target_meana) + \

(source_stda / (target_stda if target_stda > 0 else 1.0)) * (a - target_meana)

newb = target_meanb + (source_meanb - target_meanb) + \

(source_stdb / (target_stdb if target_stdb > 0 else 1.0)) * (b_ - target_meanb)

# 应用迁移强度

newL = newL * luminance_scale + (1.0 - luminance_scale) * L

newa = newa * chrominance_scale + (1.0 - chrominance_scale) * a

newb = newb * chrominance_scale + (1.0 - chrominance_scale) * b_
```

```
# 转换回 RGB 空间
r_trans, g_trans, b_trans = self.lab_to_rgb(newL, newa, newb)
# 更新像素值
result_img[y, x] = [int(b_trans * 255), int(g_trans * 255), int(r_trans * 255)]
return result_img
```

```
# 应用程序界面
class ColorTransferApp:
def __init__(self, root):
self.root = root
self.root.title("颜色追色工具")
self.root.geometry("1200x700")
self.root.resizable(True, True)
# 确保中文显示正常
self.style = ttk.Style()
if sys.platform.startswith('win'):
self.style.configure('.', font=('Microsoft YaHei UI', 10))
elif sys.platform.startswith('darwin'):
self.style.configure('.', font=('Arial Unicode MS', 10))
else:
self.style.configure('.', font=('SimHei', 10))
# 初始化颜色迁移器
self.color_transfer = ColorTransfer()
# 图像路径和数据
self.source_path = None
self.target_path = None
self.source_img = None
self.target_img = None
self.result_img = None
# 创建 UI 组件
self.create_widgets()
def create_widgets(self):
# 顶部控制区域
control_frame = ttk.Frame(self.root, padding=10)
control_frame.pack(fill=tk.X)
# 左侧源图像区域
left_frame = ttk.LabelFrame(self.root, text="源图像", padding=10)
left_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5, pady=5)
# 中间目标图像区域
middle_frame = ttk.LabelFrame(self.root, text="目标图像", padding=10)
middle_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5, pady=5)
# 右侧结果图像区域
right_frame = ttk.LabelFrame(self.root, text="追色结果", padding=10)
right_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=5, pady=5)
# 控制区域 - 源图像选择
```

```
ttk.Label(control_frame, text="目标图像:").grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)
self.source_path_var = tk.StringVar()
ttk.Entry(control_frame, textvariable=self.source_path_var, width=50).grid(row=0, column=1, padx=5, pady=5)
ttk.Button(control_frame, text="浏览...", command=self.browse_source).grid(row=0, column=2, padx=5, pady=5)
# 控制区域 - 目标图像选择
ttk.Label(control_frame, text="原始图像:").grid(row=1, column=0, padx=5, pady=5, sticky=tk.W)
self.target_path_var = tk.StringVar()
ttk.Entry(control_frame, textvariable=self.target_path_var, width=50).grid(row=1, column=1, padx=5, pady=5)
ttk.Button(control_frame, text="浏览...", command=self.browse_target).grid(row=1, column=2, padx=5, pady=5)
# 控制区域 - 迁移强度调整
ttk.Label(control_frame, text="亮度迁移强度:").grid(row=0, column=3, padx=5, pady=5, sticky=tk.W)
self.luminance_var = tk.IntVar(value=50)
ttk.Scale(control_frame, variable=self.luminance_var, from_=0, to=100, orient=tk.HORIZONTAL, length=200).grid(row=0, column=4,
padx=5, pady=5)
self.luminance_label = ttk.Label(control_frame, text="50%")
self.luminance_label.grid(row=0, column=5, padx=5, pady=5)
self.luminance_var.trace_add("write", self.update_luminance_label)
ttk.Label(control_frame, text="色度迁移强度:").grid(row=1, column=3, padx=5, pady=5, sticky=tk.W)
self.chrominance_var = tk.IntVar(value=80)
ttk.Scale(control_frame, variable=self.chrominance_var, from_=0, to=100, orient=tk.HORIZONTAL, length=200).grid(row=1, column=4,
padx=5, pady=5)
self.chrominance_label = ttk.Label(control_frame, text="80%")
self.chrominance_label.grid(row=1, column=5, padx=5, pady=5)
self.chrominance_var.trace_add("write", self.update_chrominance_label)
# 控制区域 - 执行按钮
ttk.Button(control_frame, text="开始追色", command=self.process_images, style='Accent.TButton').grid(row=0, column=6, rowspan=2,
padx=20, pady=5, ipadx=10, ipady=5)
# 图像显示区域
self.source_canvas = tk.Canvas(left_frame, bg="white")
self.source_canvas.pack(fill=tk.BOTH, expand=True)
self.target_canvas = tk.Canvas(middle_frame, bg="white")
self.target_canvas.pack(fill=tk.BOTH, expand=True)
self.result_canvas = tk.Canvas(right_frame, bg="white")
self.result_canvas.pack(fill=tk.BOTH, expand=True)
# 状态栏
self.status_var = tk.StringVar(value="就绪")
ttk.Label(self.root, textvariable=self.status_var, relief=tk.SUNKEN, anchor=tk.W).pack(side=tk.BOTTOM, fill=tk.X)
# 绑定窗口调整事件
self.root.bind("<Configure>", self.on_resize)
# 设置按钮样式
self.style.configure('Accent.TButton', font=('Arial', 10, 'bold'))
def update_luminance_label(self, *args):
self.luminance_label.config(text=f"{self.luminance_var.get()}%")
def update_chrominance_label(self, *args):
self.chrominance_label.config(text=f"{self.chrominance_var.get()}%")
```

```
def browse_source(self):
"""浏览并选择源图像"""
file_path = filedialog.askopenfilename(
title="选择源图像",
filetypes=[("图像文件", "*.png *.jpg *.jpeg *.bmp *.tiff")]
)
if file_path:
self.source_path = file_path
self.source_path_var.set(file_path)
self.load_source_image()
def browse_target(self):
"""浏览并选择目标图像"""
file_path = filedialog.askopenfilename(
title="选择目标图像",
filetypes=[("图像文件", "*.png *.jpg *.jpeg *.bmp *.tiff")]
)
if file_path:
self.target_path = file_path
self.target_path_var.set(file_path)
self.load_target_image()
def load_source_image(self):
"""加载源图像并显示"""
if self.source_path:
try:
self.source_img = cv2.imread(self.source_path)
self.display_image(self.source_canvas, self.source_img, "源图像")
self.status_var.set(f"已加载源图像: {os.path.basename(self.source_path)}")
except Exception as e:
messagebox.showerror("错误", f"无法加载图像: {str(e)}")
self.status_var.set("加载图像失败")
def load_target_image(self):
"""加载目标图像并显示"""
if self.target_path:
try:
self.target_img = cv2.imread(self.target_path)
self.display_image(self.target_canvas, self.target_img, "目标图像")
self.status_var.set(f"已加载目标图像: {os.path.basename(self.target_path)}")
except Exception as e:
messagebox.showerror("错误", f"无法加载图像: {str(e)}")
self.status_var.set("加载图像失败")
def process_images(self):
"""执行颜色迁移处理"""
if self.source_img is None or self.target_img is None:
messagebox.showwarning("警告", "请先选择源图像和目标图像")
return
```

```
try:
self.status_var.set("正在处理图像...")
self.root.update() # 更新 UI 以显示状态
# 获取强度设置
luminance_ratio = self.luminance_var.get()
chrominance_ratio = self.chrominance_var.get()
# 执行颜色迁移
self.result_img = self.color_transfer.color_transfer(
self.source_img,
self.target_img,
luminance_ratio,
chrominance_ratio
)
# 显示结果
self.display_image(self.result_canvas, self.result_img, "追色结果")
self.status_var.set("处理完成")
except Exception as e:
messagebox.showerror("错误", f"处理图像时出错: {str(e)}")
self.status_var.set("处理失败")
def display_image(self, canvas, img_cv, title):
"""在 Canvas 上显示 OpenCV 图像"""
if img_cv is None:
return
# 转换 BGR 到 RGB
img_rgb = cv2.cvtColor(img_cv, cv2.COLOR_BGR2RGB)
# 调整图像大小以适应 Canvas
canvas_width = canvas.winfo_width() - 10
canvas_height = canvas.winfo_height() - 30 # 留出标题空间
if canvas_width <= 0 or canvas_height <= 0:
# Canvas 尺寸尚未确定，使用默认大小
canvas_width, canvas_height = 400, 300
# 计算缩放比例
h, w = img_rgb.shape[:2]
scale = min(canvas_width / w, canvas_height / h)
new_size = (int(w * scale), int(h * scale))
# 调整图像大小
img_resized = cv2.resize(img_rgb, new_size)
# 转换为 PIL 图像
img_pil = Image.fromarray(img_resized)
# 转换为 Tkinter 可用的 PhotoImage
img_tk = ImageTk.PhotoImage(image=img_pil)
# 在 Canvas 上显示图像
canvas.delete("all")
canvas.create_image(canvas_width/2, canvas_height/2, anchor=tk.CENTER, image=img_tk)
canvas.create_text(canvas_width/2, 15, text=title, font=("Arial", 12, "bold"))
```

```
# 保存对图像的引用，防止被垃圾回收
canvas.image = img_tk
def on_resize(self, event):
"""窗口大小改变时重绘图像"""
if event.widget == self.root:
self.display_image(self.source_canvas, self.source_img, "源图像")
self.display_image(self.target_canvas, self.target_img, "目标图像")
self.display_image(self.result_canvas, self.result_img, "追色结果")
```

```
if __name__ == "__main__":
root = tk.Tk()
app = ColorTransferApp(root)
root.mainloop()
```