華東師範大學
East China Normal University

PixEditPro——基于 OpenCV 和深度学习模型的图片编辑网站项目报告

姓　　名：谢思羽、赵雅琪
学　　号：10213903419
10214900441
学　　院：软件工程学院
专　　业：软件工程
教　　师：曹桂涛
职　　称：教授

2024　年　7　月

目录

【摘要】

随着时间的推移，许多珍贵的老照片逐渐褪色，又或是在过去技术不够发达的年代，我们只能看到黑白的影像。对很多人来说，这会成为一大遗憾。现在，通过基于 OpenCV 和深度学习的图像着色技术，我们能够为黑白照片重新赋予色彩，使其更加生动。当我们希望让一幅作品的风格变得更加丰富时，风格迁移为我们提供了方向。利用深度学习技术，我们可以将一幅图像的艺术风格应用到另一幅图像上，实现如同名画作品般的效果。这不仅可以为图像增添艺术价值，还可以广泛应用于创意设计、广告制作等领域，为用户提供更多的创作可能性。

因此，本项目旨在开发一个基于 Web 的数字图像处理平台，采用 Vue+Flask+深度学习模型搭建了一个能够实现黑白图像修复和风格迁移以及多种基础数字图像处理功能的图片编辑网站。用户可以通过直观、简洁的界面进行图像处理操作，满足不同场景下的图像处理需求。

关键词：OpenCV；Python；Deep Learning；CNN；Fast Neural Style Transfer

1 背景

随着技术的不断发展，数字图像处理在各个领域中扮演着越来越重要的角色。无论是医学影像分析、卫星遥感图像处理，还是日常生活中的照片美化和修复，数字图像处理技术都发挥着至关重要的作用。在这些应用中，图像修复和风格迁移技术尤为引人注目。图像修复，特别是黑白老照片的上色，不仅能够让过去的历史照片焕发新生，还能为个人和家庭带来美好的回忆。而图像风格迁移技术则能够将不同艺术风格应用到照片上，创造出独特的艺术效果，极大地丰富了图像处理的可能性。

本次我们开发的 PixEditPro 项目旨在提供一个全面的图像处理平台，用户可以通过该平台实现多种图像处理功能，包括基础图像处理、黑白图像上色修复以及图像风格迁移。其中，图像修复功能可以对黑白老照片进行自动上色，恢复其原有的色彩和生动性；而风格迁移功能则可以将不同的艺术风格应用到用户的照片中，使普通的照片瞬间变成艺术作品。通过这些功能，我们希望能够为用户提供便利、高效的图像处理体验，同时也能探索图像处理技术在更多领域的应用可能性。

2 基本技术栈介绍

2.1 前端技术栈

在本项目前端开发中，我们使用了 Vue.js 作为前端开发框架，因其简单易用的特性和强大生态系统，特别适合构建单页面应用（SPA），完美切合我们课程项目这样轻量级的网站的实现。

在前后端交互上，我们采用了 Axios 这个流行的 HTTP 请求库，用于在前端发起异步请求，与后端 API 进行数据交互。在这里的核心代码中，前端向后端发送要处理的图片 url 以及对应的需要调用的方法 id，经过后端的处理后，返回处理后的图像 url，再展示到前端，以此实现了我们网页的主要核心功能。

在前端的页面开发中，我们利用 Vue Router 管理应用的路由，实现单页面设计。具体是通过定义不同的路由以及对应的组件，并使用路由导航实现页面的切换与渲染。同时也利用了 Vue.js 的组件化开发思想，将页面拆分为多个组件，每个组件专注于特定的功能或视图。组件之间通过 props 和事件进行通信，实现高内聚、低耦合的开发模式。

2.1.1 项目目录结构

```
|——src
|     |——assets
|           |——style.css
|     |——components
|     |     |——BasicTask.vue          基础功能页面
|     |     |——Colorizer.vue          图像修复页面
|     |     |——Header.vue             导航栏
|     |     |——HomePage.vue           首页
|     |     |——StyleTransfer.vue      风格迁移页面
|     |——router
```

```
|       |       |——index.js                页面路由配置文件
|       |——static
|       |       |——img                     存放一些展示图片
|       |       |——style-type              存放一些风格模板图片
|       |——theme    elment-ui              组件样式配置文件
|       |——utils
|       |       |——function.js            存放一些函数和数组
|       |——App.vue
|       |——Component.vue
|       |——main.js  配置文件
```

2.2 后端技术栈

Flask 是一个轻量级的 Python Web 框架，适合快速开发和部署 Web 应用程序。它提供了灵活的扩展性和简单的核心，方便我们快速有效地搭建一个后端平台。flask_cors 是一个用于 Flask 应用的跨源资源共享扩展，本项目也利用 flask_cors 实现了跨域请求，以便通过 HTTP 头部来控制请求行为。

2.2.1 项目目录结构

```
|——PixEditPro
|       |——core                    核心功能文件
|       |       |——__init__.py
|       |       |——main.py
|       |       |——process.py      主要功能处理文件
|       |       |——utils.py        相关函数
|       |——models                  一些训练模型
|       |——templates               网页文件
|       |——tmp                     用于存放处理前和处理后的图片
|       |       |——ct
|       |       |——draw
|       |——uploads                 用于存放上传给前端的图片
|       |       |——readmePic       存放 README 文件里的图片
|       |——app.py                  项目入口点，定义路由
|       |——config.py               项目配置文件
|       |——README.md
```

2.3 图像处理技术

图像处理采用 OpenCV 和深度学习技术。OpenCV 是一个强大的开源计算机视觉库，提供了丰富的图像处理功能。深度学习技术采用了预训练的神经网络模型，能够实现复杂的图像处理任务，如图像着色和风格迁移。

3 功能展示

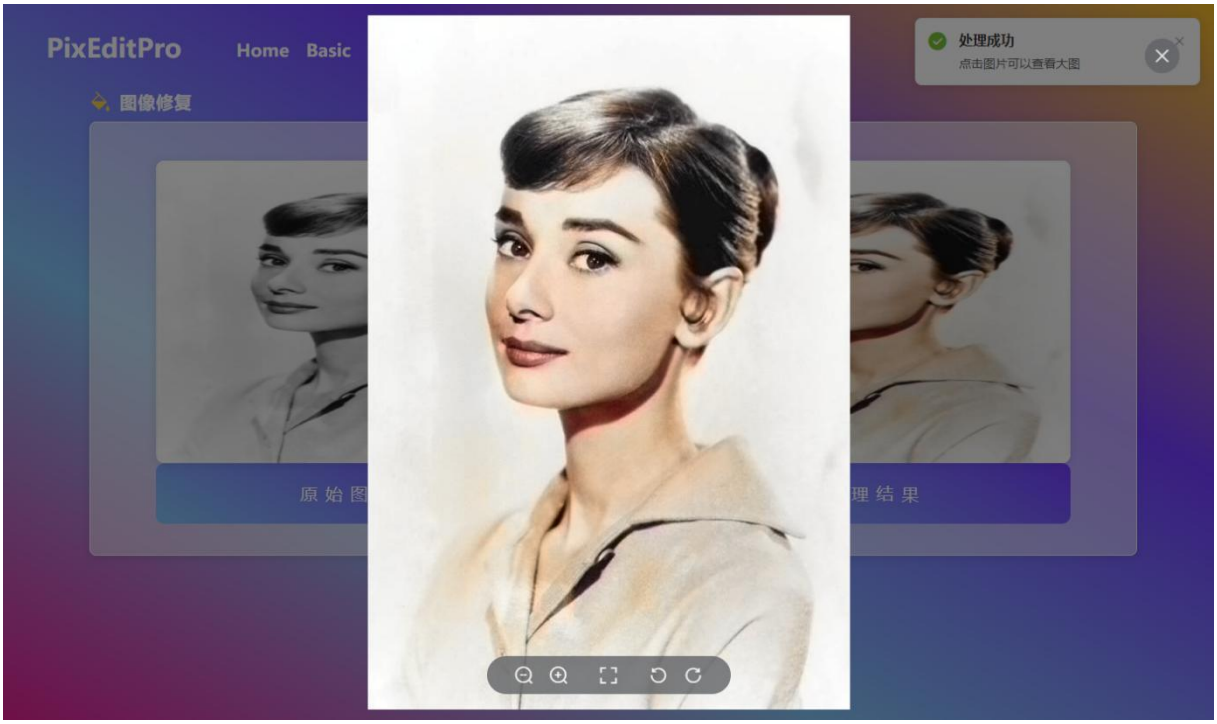


图 3-1 黑白老照片着色修复

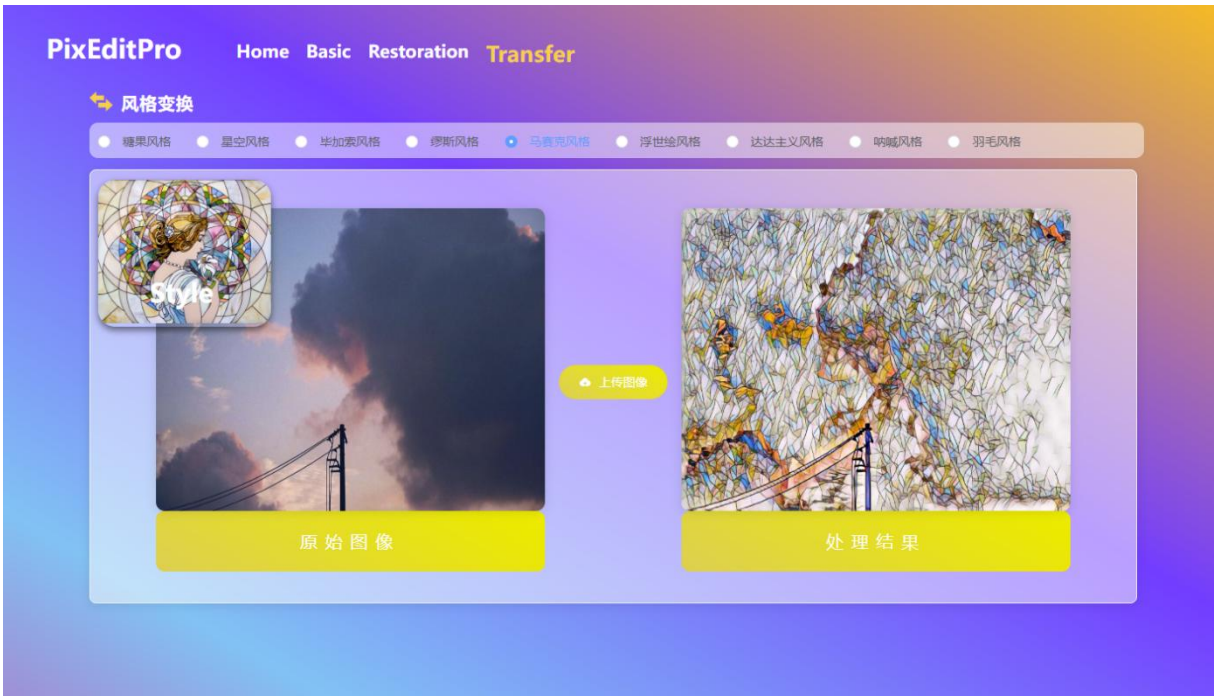


图 3-2 风格迁移-马赛克风格

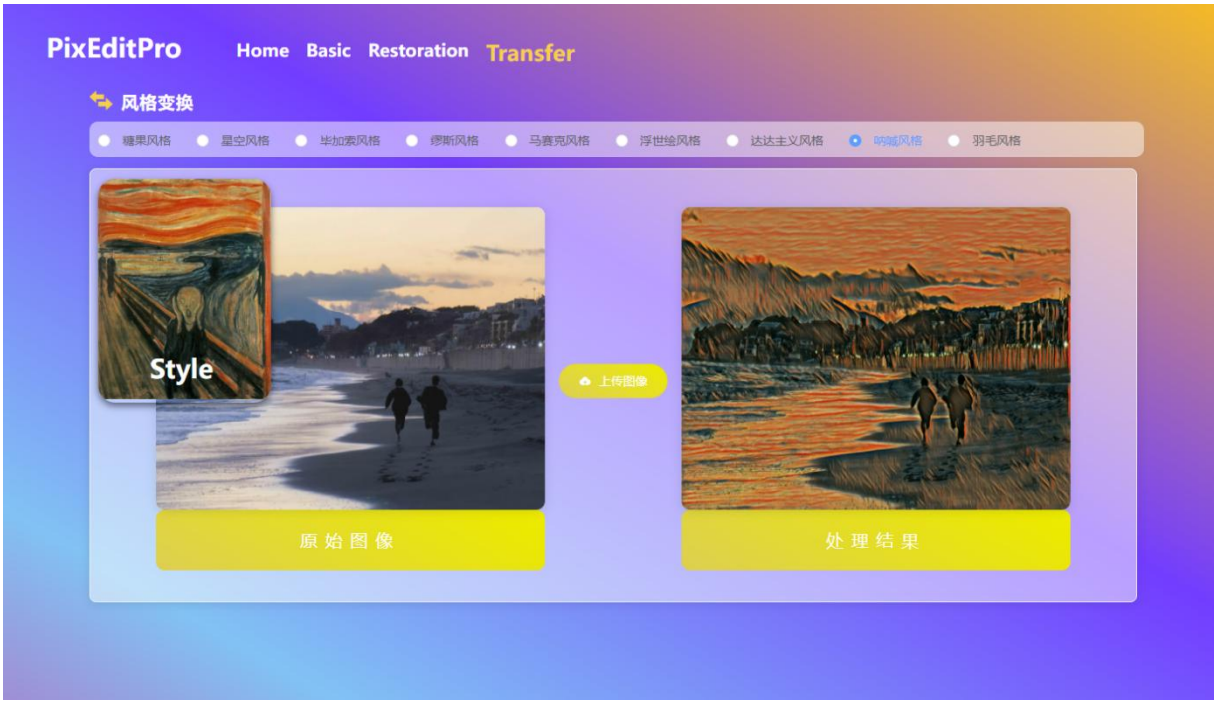


图 3-3 风格迁移-呐喊风格

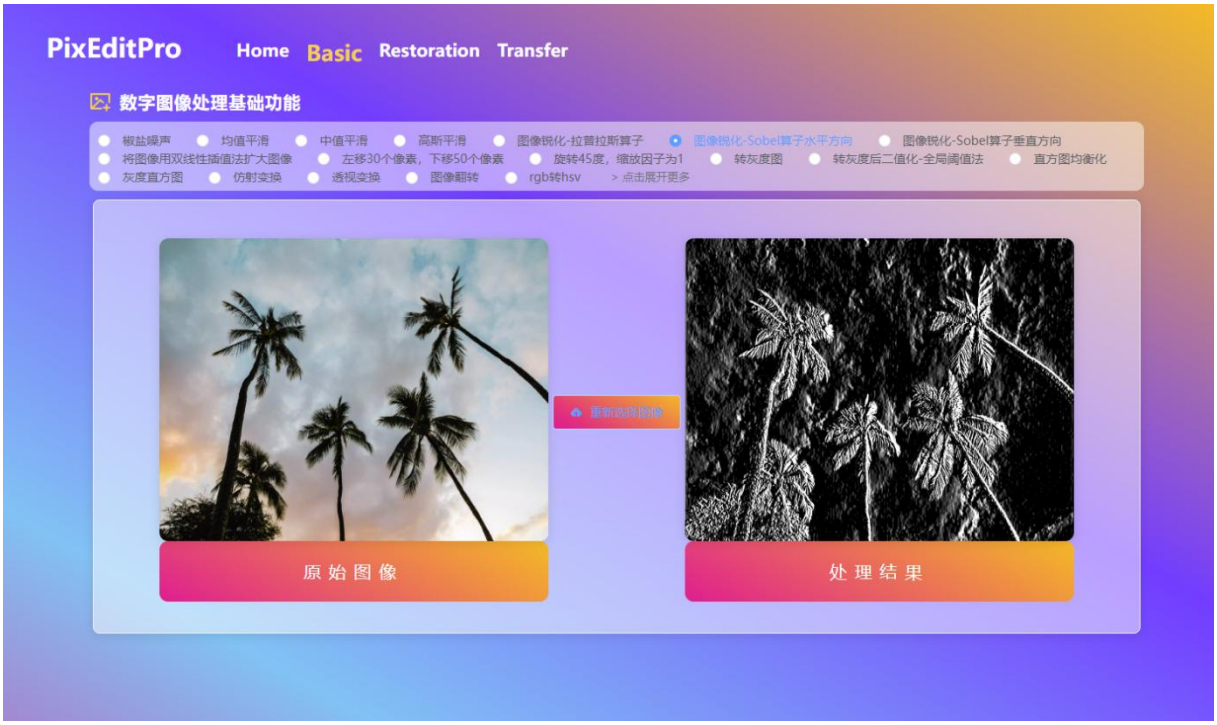


图 3-4 基础功能-图像锐化_Sobel 算子水平方向

4 功能实现

4.1 黑白图像着色修复

图像着色（Image Colorization）是一项将灰度图像转化为彩色图像的技术。它在恢复老照片、电影修复、增强视觉效果等领域有着广泛的应用。现代图像着色算法通常依赖于深度学习技术，通过训练神经网络来预测灰度图像中各个像素的颜色。下面将详细介绍我们使用的图像着色算法及其相关知识。

我们使用的图像着色算法基于论文《Colorful Image Colorization》实现。

在图像处理和计算机视觉中，颜色空间（Color Space）的选择对于特定任务的成功与否有着重要影响。对于图像着色任务，Lab 颜色空间相比传统的 RGB 颜色空间更为有利。其中，L 表示亮度（Luminance），范围通常是 0 到 100；a 表示从绿色到红色的颜色分量，负值表示绿色，正值表示红色；b：表示从蓝色到黄色的颜色分量，负值表示蓝色，正值表示黄色。

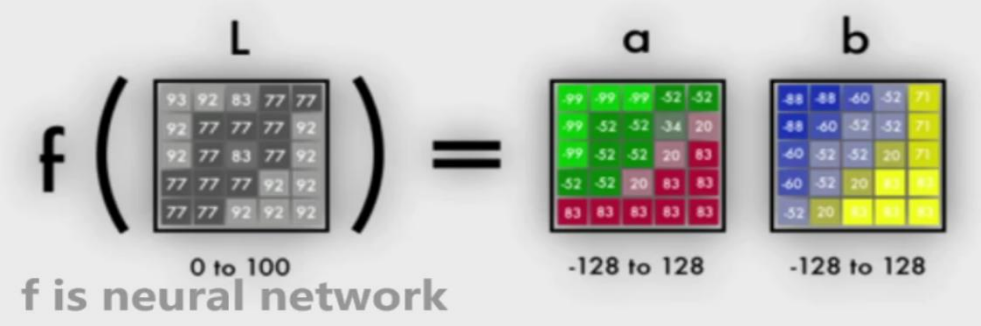


图 4-1-1 利用 Lab 颜色空间的色彩预测

在图像着色中，典型的流程包括以下步骤：
图像转换：将输入的灰度图像从 RGB 颜色空间转换到 Lab 颜色空间。
亮度分离：分离出 L 通道（亮度信息）。
模型预测：使用深度学习模型预测 a 和 b 通道的值。
颜色合并：将预测得到的 a 和 b 通道与原始 L 通道合并。
颜色转换：将合并后的 Lab 图像转换回 RGB 颜色空间。

4.1.1 算法概述

我们使用的是基于 OpenCV 和深度学习的图像着色算法。该算法利用预训练的 Caffe 模型和聚类中心点来预测并赋予灰度图像颜色。主要步骤包括模型加载、图像预处理、颜色预测和后处理等。

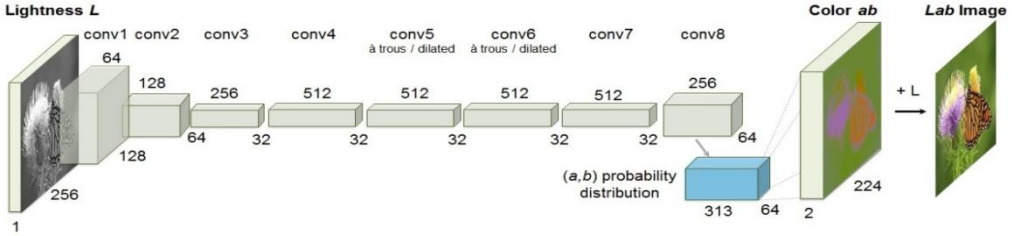


图 4-1-2 模型架构

我们的架构使用卷积神经网络（CNN）来预测黑白图像中的颜色信息。

输入的图像是一个单通道的灰度图像，表示亮度信息（L 通道）。

图像通过一系列的卷积层进行处理，每一层的卷积核（filters）数量和尺寸如图所示：

- conv1: 使用 64 个卷积核，输出尺寸为 128x128。
- conv2: 使用 128 个卷积核，输出尺寸为 64x64。
- conv3: 使用 256 个卷积核，输出尺寸为 32x32。
- conv4: 使用 512 个卷积核，输出尺寸为 32x32。
- conv5, conv6, conv7: 使用 512 个卷积核，进行à trous 卷积（空洞卷积），输出尺寸为 32x32。
- conv8: 使用 256 个卷积核，输出尺寸为 64x64。

在经过多层卷积处理后，网络生成一个概率分布图，表示每个像素可能的颜色（a 和 b 通道）。这一概率分布图通过一个 313 通道的卷积层得到，其中 313 表示颜色的聚类中心数。将预测的 ab 通道（64x64）上采样到与输入图像相同的尺寸（224x224）。将输入图像的 L 通道与预测得到的 ab 通道合并，形成一个完整的 Lab 颜色空间图像。最后，将合并后的 Lab 图像转换回 RGB 颜色空间，得到着色后的彩色图像。

4.1.2 模型加载

```
# 设置模型路径
prototxt = r'models/colorization_deploy_v2.prototxt'
model = r'models/colorization_release_v2.caffemodel'
points = r'models/pts_in_hull.npy'
net = cv.dnn.readNetFromCaffe(prototxt, model)  # 加载模型
pts = np.load(points)
```

使用预训练的 Caffe 模型进行图像着色。模型文件包括结构文件（prototxt）、权重文件（caffemodel）和聚类中心点文件（npy）。加载模型和聚类中心点，并将其添加到模型中，以便在预测过程中使用。

4.1.3 图像预处理

```
# 读取图像并转换颜色空间
scaled = image.astype("float32") / 255.0  # 将图像像素值缩放到[0,1]范围
lab = cv.cvtColor(scaled, cv.COLOR_BGR2LAB)  # 将图像从 BGR 转换到 LAB 颜色空间
# 调整图像大小并分离 L 通道
resized = cv.resize(lab, (224, 224))  # 将图像调整到 224x224 大小
L = cv.split(resized)[0]  # 分离 L 通道
L -= 50  # 中心化 L 通道
```

将输入的灰度图像读取并转换为浮点数格式，同时将像素值缩放到[0, 1]范围。将图像从 BGR 颜色空间转换到 LAB 颜色空间，其中 L 通道表示亮度，A 和 B 通道表示颜色信息。

4.1.4 颜色预测

```
# 预测 a 和 b 通道
net.setInput(cv.dnn.blobFromImage(L))  # 将 L 通道作为输入
ab = net.forward()[0, :, :, :].transpose((1, 2, 0))  # 预测 ab 通道并调整形状
```

将图像调整到指定的输入尺寸，并分离出 L 通道。使用预训练的神经网络模型预测 A 和 B 通道的颜色值。将预测得到的 A 和 B 通道大小调整到原图尺寸，以便进行后续合并。

4.1.5 图像后处理

```
# 合并 L 通道和预测的 ab 通道
L = cv.split(lab)[0]  # 分离原图的 L 通道
colorized = np.concatenate((L[:, :, np.newaxis], ab), axis=2)  # 合并 L 和 ab 通道
# 转换回 BGR 颜色空间并裁剪无效值
colorized = cv.cvtColor(colorized, cv.COLOR_LAB2BGR)  # 将图像从 LAB 转换回 BGR
```

将预测得到的 A 和 B 通道与原图的 L 通道合并，形成完整的 LAB 图像。将 LAB 图像转换回 BGR 颜色空间，并裁剪无效的像素值，使其在有效范围内。将图像像素值转换为 8 位无符号整数格式，以便显示和保存。

3.1.6 模型训练

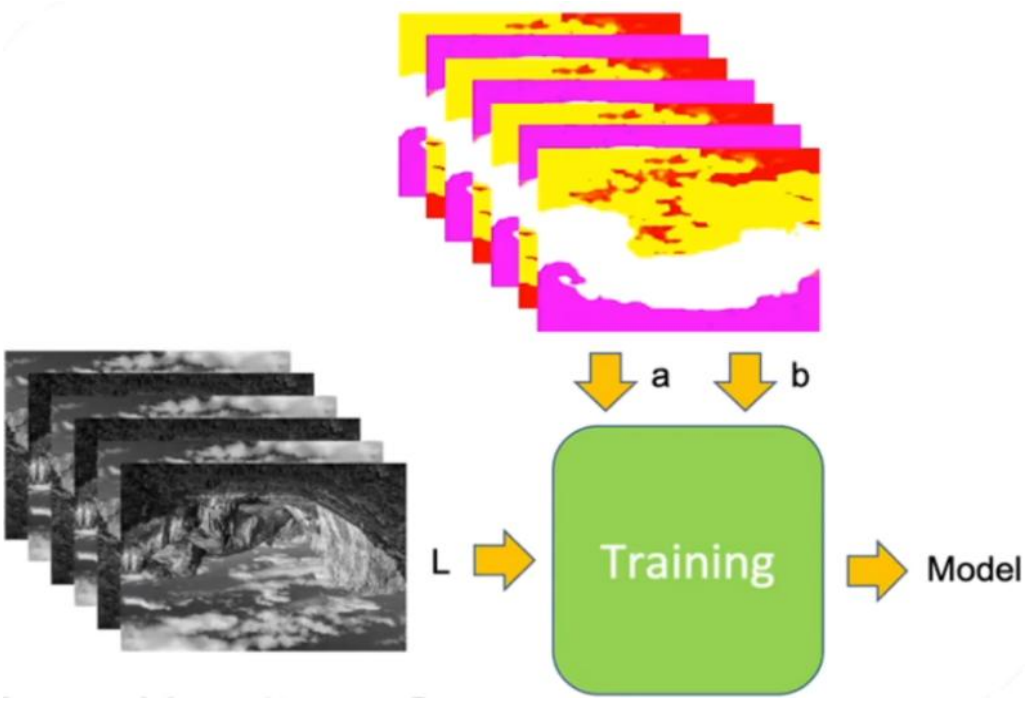


图 4-1-3 模型训练示意图

我们在这里所采用的是已经是在 Imagenet 数据集中的一百万张图像上进行大规模数据训练过后的模型。

由于 L 通道仅编码强度，我们可以使用 L 通道作为网络的灰度输入。网络必须学会预测 a 和 b 通道。给定输入的 L 通道和预测的 ab 通道，我们就可以形成最终的输出图像。

整个训练过程概括为如下步骤：

1. 将所有训练图像从 RGB 颜色空间转换为 Lab 颜色空间。
2. 使用 L 通道作为网络的输入，并训练网络预测 ab 通道。
3. 将输入的 L 通道与预测的 ab 通道结合起来。
4. 将 Lab 图像转换回 RGB。

模型文件包含如下三个：

• colorization_deploy_v2.prototxt：

定义了神经网络的结构，包括每一层的类型、参数、连接方式等。

• colorization_release_v2.caffemodel：

包含了神经网络各层的权重和偏置参数。

• pts_in_hull.npy：

存储了 Lab 颜色空间中的 ab 通道的 313 个聚类中心点。

在实际操作中，首先通过 colorization_deploy_v2.prototxt 文件加载网络架构，然后通过 colorization_release_v2.caffemodel 文件加载预训练的权重，最后将 pts_in_hull.npy 中的聚类中心点添加到网络的特定层中，确保模型能够更好地处理和生成颜色信息。

图 4-1-4 黑白老照片着色修复效果图

可以看到，对于过去的黑白老照片的修复效果非常自然。

3.1.7 不同模型尝试

在使用这个新的模型和算法前，我们尝试了使用其他模型来完成图像着色，但效果不尽如人意。

图 4-1-5 其他模型的着色效果

图 4-1-6 新模型的着色效果

我们认为可能的原因如下：

• 模型训练数据：不同模型使用的训练数据集可能不同。用于训练模型的数据集如果不包含足够的多样性，模型在面对未见过的图像时可能表现不佳。

• 网络架构：不同模型的网络架构可能存在差异。深度神经网络的层数、层类型和每层的参数设置都会影响模型的性能和输出效果。

• 模型权重初始化：权重的初始化方式对模型的训练结果有很大影响。如果权重初始化不当，模型可能无法充分学习到有效的特征。

3.1.7 相关优化

我们的优化主要是从三个方面入手，以提高图像的清晰度。

首先是应用了双边滤波，它不同于传统的高斯滤波，它能够在保留边缘细节的同时减少噪声，使图像更平滑。

```
colorized = cv.bilateralFilter(colorized, d=3, sigmaColor=30, sigmaSpace=30)
```

接着使用了自适应直方图均衡化，它也不同于传统的直方图均衡化，CLAHE 在图像的不同区域分别应用直方图均衡化，从而在全局对比度增强的同时避免了过度增强带来的噪声。

```
lab = cv.cvtColor(colorized, cv.COLOR_BGR2LAB)
l, a, b = cv.split(lab)
clahe = cv.createCLAHE(clipLimit=1.5, tileGridSize=(8, 8))
cl = clahe.apply(l)
lab = cv.merge((cl, a, b))
colorized = cv.cvtColor(lab, cv.COLOR_LAB2BGR)
```

最后我们使用了一个锐化滤波器，用于增强图像的边缘细节。

```
kernel = np.array([[0, -0.5, 0],
                   [-0.5, 3, -0.5],
                   [0, -0.5, 0]])
```

我们采用的卷积核

1.中心值 3 是卷积核的主要权重。它通过增加中心像素的权重来增强该像素的亮度。提高中心像素的亮度，使其在锐化后的图像中更加突出。

2.周围值 -0.5 用于减去中心像素周围的像素值。这个负值通过减小相邻像素的亮度来实现边缘增强。减少周围像素的亮度，使中心像素与其邻近像素之间的对比度增加，从而突出边缘和细节。

3.其他值为 0，这些位置不影响卷积操作。这些位置对滤波结果没有贡献，只关注增强中心像素和其周围像素之间的对比。

优化后的效果展示如下：

优化前　　优化后

图 4-1-7 优化前后对比图

4.2 风格迁移

4.2.1 背景简介

近年来，由深度学习引领的人工智能技术浪潮越来越广泛地应用到社会各个领域。这其中，手机应用 Prisma，尝试为用户的照片生成名画效果，一经推出就吸引了海量用户，登顶 App Store 下载排行榜。这神奇背后的核心技术就是基于深度学习的图像风格迁移。

风格迁移又称风格转换，直观点的类比就是给输入的图像加个滤镜，但是又不同于传统滤镜。风格迁移基于人工智能，每个风格都是由真正的艺术家作品训练、创作而成。只需要给定原始图片，并选择艺术家的风格图片，就能把原始图片转化成具有相应艺术家风格的图片。如下图所示，给定一张风格图片和一张内容图片，神经网络能够生成"梵高风格"的建筑图。

风格迁移这一想法与纹理生成的想法密切相关，在 2015 年开发出神经风格迁移之前，这一想法就已经在图像处理领域有着悠久的历史。但事实证明，与之前经典的计算机视觉技术实现相比，基于深度学习的风格迁移实现得到的结果是无与伦比的，

并且还在计算机视觉的创造性应用中引发了惊人的复兴。

风格迁移其主要应用场景如在艺术创作场景，将不同艺术风格应用于图像，可以创造出独特的艺术效果，使作品具有新的视觉呈现。或者在社交平台上风格化滤镜，图像增强等。

图 4-2-1 风格迁移示意图

4.2.2 研究现状

1. Neural Style Transfer（神经风格迁移）

这种风格迁移的方法是基于优化的算法。该算法通过最小化内容图像和风格图像的差异，同时最大化内容图像与风格图像的相似性来实现风格迁移的。因此需要大量的迭代优化，计算成本较高，也容易生成的图片不够清晰。

2. Fast Neural Style Transfer（快速神经风格迁移）

由于第一种方法的做法在实现上过于复杂，每次进行风格迁移都需要几十分钟甚至几个小时的训练。斯坦福博士生 Justin Johnson 在 2016 年提出了一种快速实现风格迁移的方法，当使用 Fast Neural Style 训练好一个模型之后，CPU 只需要运行几秒，就能生成对应风格的迁移结果，并且可以反复利用。因此本次我们的项目就选择采取此种方法训练得到结果。

3. CycleGAN（循环一致性对抗网络）

除了上述两种方法之外，也有研究者提出了一个基于生成对抗网络（GAN）的方法，这是一种无需成对训练数据的风格迁移的方法，它可以在不同风格的图像之间进行转换，并且生成的图片质量很高；但是对于较复杂的图片，这种方法可能会需要采用更多的训练数据。

因此，综合考虑，我们的项目选用了 Fast Neural Style Transfer 的方法进行风格迁移的实现，并且得到了不错的效果。

4.2.2 实现原理

要生成逼真的风格迁移图片有两个要求。一是要生成的图片在内容、细节上与输入的图片尽可能相似；二是要生成的图片在风格上尽可能与风格图片类似。因此，需要定义两个损失函数 content loss 和 style loss，来分别衡量上述两个指标。

在 Neural Style Transfer 的方法中，通过卷积神经网络在数学上定义 content loss 和 style loss。在 CNN 卷积网络进行对象识别任务中，随着层次的加深对象的信

息输出更加的明确，较前的层数特征图输出到一些更加通用的结构，比如猫狗分类中的基础边缘线条，而更深的层可以捕捉到更加全局和抽象的结构，如猫耳，猫眼睛。根据这个模式，我们可以通过不同深度的层特征图重建输入图像以可视化层所包含输入图像的信息，如下图可以看到深层的特征图包含了图像中对象的全局排列信息（高级、抽象），但是像素值信息会丢失。浅层的层特征图重建图像几乎是完整的精确像素值。

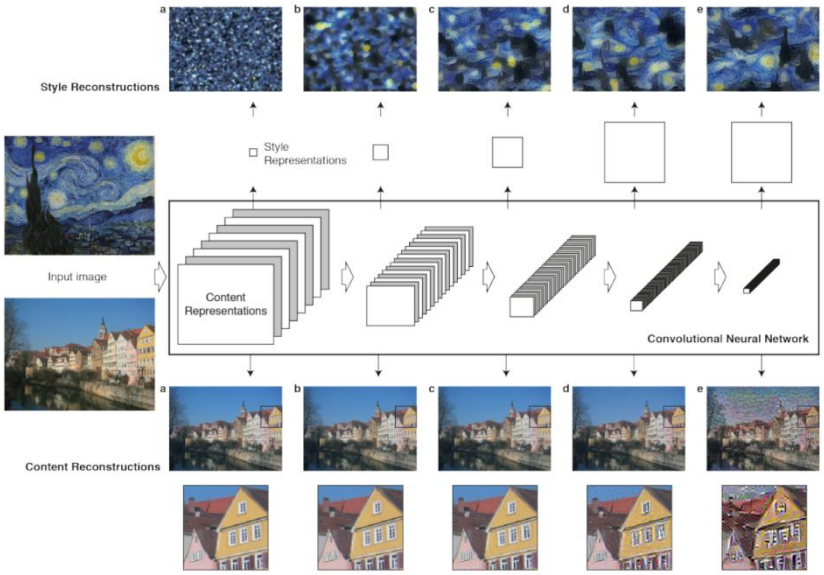


图 4-2-2 不同层次的 content 和 style 输出

图像的内容和风格含义广泛，并且没有严格的数学统一，具有很大程度上的主观性，因此很难表示。常用的 content loss 是采用逐像素计算差值，但是这样太过于死板，因此我们采用的方法是 perceptual loss，也就是感知损失，它计算的是在更高层语义上的差异，需要将预训练好的神经网络的高层输入作为图片的知觉特征；这样的得到的结果会更加自然。

4.2.2.1 Gram 矩阵

在 Fast Neural Style 中，采用 Gram 矩阵来表示图像的风格特征，对于每一张图片，卷积层的输出形状为：

$$C \times H \times W$$

C 是卷积核的通道数，每个卷积核学习图像的不同特征。每一个卷积核输出的 $H \times W$ 代表这张图像的一个 feature map,而过计算每个 feature map 之间的相似性，我们可以得到图像的风格特征。对于一个 $C \times H \times W$ 的 feature map F 来说，Gram Matrix 的形状为 $C \times C$，其第 i，j 个元素 $G_{i,j}$ 定义如下：

$$G_{i,j} = \sum_k F_{i,k} F_{j,k}$$

其中$F_{i,k}$代表第 i 个 feature map 的第 k 个像素点。

经过实践证明利用 Gram Matrix 表征图像的风格特征在风格迁移、纹理合成等任务中的表现十分出众。

4.2.2.2 Fast Neural Style 的具体训练步骤

Fast Neural Style 的具体训练步骤如下：

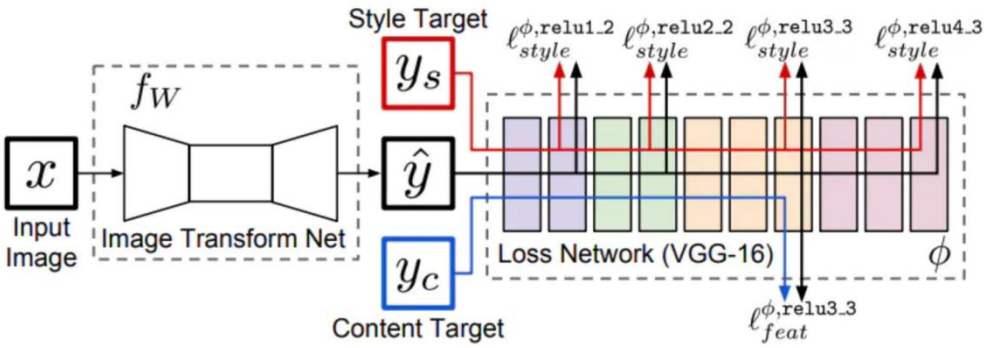


图 4-2-3 Fast Neural Style 训练步骤示意图

（1）输入一张图片x到fw中得到结果$\hat{y}$

（2）将y和y_c输入到 Loss Network 中，计算它在 relu1_2、relu2_2、relu3_3 和 relu4_3 的输出，再计算它们的 Gram Matrix 的均方误差作为 style loss。

（3）两个损失相加，并反向传播。更新fw参数，固定 loss network 不动。

（4）跳回第一步，继续训练fw

在进行风格迁移时，我们并不要求生成图片的像素和原始图片中的每一个像素都一样，我们追求的是生成图片和原始图片具有相同的特征：例如原图中有大象，我们希望风格迁移之后的图片依旧有大象。图片中“有大象”这个概念是我们分类问题最后一层的输出。但最后一层的特征对我们来说抽象程度太高，因为我们不仅希望图片中有大象，还希望保存图片的部分细节信息，例如它的形状、动作等信息，这些信息相对来说没有那么高的层次。因此我们使用中间某些层的特征作为目标，希望原图像和风格迁移的结果在这些层输出的结果尽可能相似，即将图片在深度模型的中间某些层的输出作为图像的知觉特征。下图展示了通过优化来找到合适的层级的过程，可以

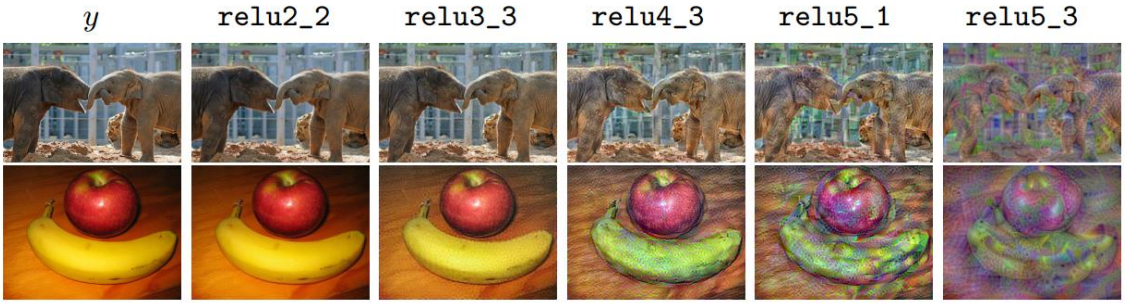


图 4-2-4 深度学习不同层次的输出结果

可以看到在较高层次风格会更加接近所需，但是损失了图片的细节特征，因此需

要恰当地选取图像的知觉特征。

4.2.2 实现过程

因为本地的电脑难以训练风格迁移这么大量的数据集，因此我们利用了开源项目 fast-neural-style 里预训练的 Torch 模型进行风格迁移的实现。

下面以 candy 风格为例，介绍具体的实现思路和过程。

4.2.2.1 加载模型

```
net = cv.dnn.readNetFromTorch('.\models\candy.t7') # 选择一个模型的地址
net.setPreferableBackend(cv.dnn.DNN_BACKEND_OPENCV) # 创建后端
```

通过 cv.dnn.readNetFromTorch 方法加载一个预训练的 Torch 模型，模型文件路径为 .\models\candy.t7。

使用 cv.dnn.DNN_BACKEND_OPENCV 设置 DNN 后端为 OpenCV 实现，以此执行深度学习模型的计算引擎或框架。

4.2.2.2 预处理图像

```
(h, w) = image.shape[:2]
blob = cv.dnn.blobFromImage(image, 1, (w, h), (103.939, 116.779, 123.680), swapRB=False,
crop=False)
```

获取输入图像的高度和宽度。

使用 cv.dnn.blobFromImage 方法将图像转换为网络输入所需的格式。具体操作包括：

归一化我们采用将图像像素值减去均值 (103.939, 116.779, 123.680)的方法。(103.939, 116.779, 123.680) 是 ImageNet 数据集的 RGB 均值。ImageNet 数据集是一个大型图像数据库，常用于训练和评估计算机视觉算法。在训练深度学习模型时，通常会对输入图像进行标准化处理，即减去每个通道的均值，以便使数据更加集中于零附近，从而加快收敛速度。

swapRB=False 表示不交换 R 和 B 通道，因为图像已经是 BGR 格式。

crop=False 表示不裁剪图像。

这主要是为了将图像进行归一化处理，转换为适合网络输入的格式，以便为了后续在神经网络中传输以进行训练。`

4.2.2.3 向前传播

```
net.setInput(blob)
out = net.forward()
```

将预处理后的图像（blob）作为输入传递给网络。
使用 net.forward 方法进行前向传播，进行计算，生成输出图像。

4.2.2.4 图像后处理

```
out = out.reshape(3, out.shape[2], out.shape[3])
out[0] += 103.939
out[1] += 116.779
out[2] += 123.68
out /= 255
out = out.transpose(1, 2, 0)
out = out * 255
```

将输出 out 进行形状调整，使其变为 (3, height, width)。
将每个通道的像素值加上原始均值 (103.939, 116.779, 123.68)，进行反归一化。
将像素值除以 255，使其归一化到 [0, 1] 范围。
将输出从 (channels, height, width) 转置为 (height, width, channels)，即回到图像的常规格式。再将像素值乘以 255，恢复到原始像素值范围。
这样就得到了经过快速神经网络训练之后的风格迁移图像。

4.2.3 实现结果

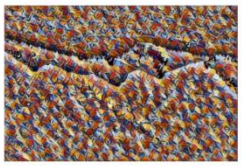
在本项目中，我们基于 Fast Neural Style 的训练模型实现了 9 种风格的风格变换，具体如下（以下是我们的效果展示图）：

style

candy

style

Composition VII,
Vasily Kandinsky(Russia),
1913



style

feathers

style

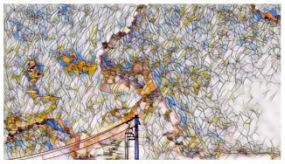
A muse,
Pablo Picasso(Spain),
1935

style

The Scream,
Pablo Picasso(Spain),
1895

style

Mosaic

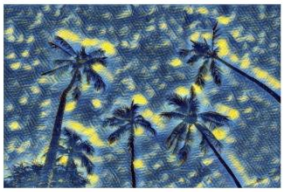


style

Church,
Francis Picabia(Russia),
1913

style

Starry Night,
Pablo Picasso(Spain),
1889



style

かながわおきなみうら,
Katsushika Hokusai(Japan),
19C

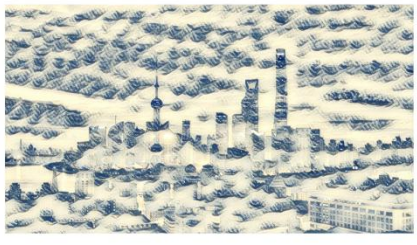


图 4-2-5 9 种风格的效果展示图

4.3 基础功能

基础功能中包含了课程中所学过的部分基础图像处理方法，这些方法为诸多复杂的图像处理奠定了基础。因此，作为数字图像处理的入门学生，学好这些方法对我们而言尤为重要。

下面我们将介绍每种功能的大致原理以及它们在高级图像处理中的应用价值。

4.3.1 椒盐噪声

随机将图像的某些像素设置为黑色或白色，模拟噪声。

4.3.2 均值平滑/中值平滑/高斯平滑

通过计算邻域内像素的平均值来平滑图像，减小噪声；用邻域内像素的中值替代中心像素值，保留边缘的同时去除噪声；使用高斯核对图像进行平滑处理，减小噪声。

在图像去噪和平滑处理中常用，选择不同的平滑方式来消除不同的噪声。

4.3.3 图像锐化-拉普拉斯算子/Sobel 算子水平方向/图像锐化-Sobel 算子垂直方向

使用拉普拉斯算子增强图像中的细节和边缘、使用 Sobel 算子在水平方向上计算图像梯度，检测边缘、使用 Sobel 算子在垂直方向上计算图像梯度，检测边缘。

一般可用于边缘检测，如需特定的垂直或水平方向的边缘可以选择不同方向的 Sobel 算子。

4.3.4 将图像用双线性插值法扩大

通过双线性插值方法将图像放大，保留更多细节。

可用于图像的缩放操作，保证放大后的图像不失真。

4.3.5 转灰度图

将彩色图像转换为灰度图像，保留亮度信息。

用于预处理步骤，减少计算复杂度。

在我们的黑白图像着色修复中，就应用到了该技术。在训练图像着色模型时，将训练数据中的彩色图像转换为灰度图像（即只保留 L 通道），然后用这个灰度图像作为输入，来训练模型预测其对应的 a 和 b 通道。模型的目标是根据输入的灰度图像（L 通道）预测彩色信息（a 和 b 通道）。训练过程中，模型不断调整其权重，以使其输出的 a 和 b 通道尽可能接近实际的彩色信息。

4.3.6 转灰度后二值化-全局阈值法

将灰度图像转换为二值图像，使用全局阈值分割。

用于图像分割和边缘检测，简化后续处理。

4.3.7 直方图均衡化

调整图像的亮度分布，增强对比度。

用于增强图像的对比度和细节，改善视觉效果。

4.3.8 灰度直方图

统计图像中各灰度值出现的频率，形成直方图。
用于图像分析和增强，对比度调整的基础。

4.3.9 仿射变换

通过线性变换将图像进行旋转、缩放、平移等操作。
用于图像几何校正和对齐。

4.3.10 透视变换

将图像按照透视关系进行变换，改变图像的视角。
用于图像校正和视角调整。

4.3.11 rgb 转 hsv

将 RGB 颜色空间转换为 HSV 颜色空间。
用于颜色分割和增强处理。

4.3.12 hsv 获取 h、s、v

从 HSV 颜色空间中提取色调（H）、饱和度（S）、明度（V）分量。
用于颜色分析和分割。

4.3.13 rgb 获取 b、g、r

从 RGB 颜色空间中提取蓝色（B）、绿色（G）、红色（R）分量。
用于颜色分析和分割。

3.3.14 腐蚀/膨胀

使用结构元素腐蚀图像，减小物体边界、使用结构元素膨胀图像，扩展物体边界。
用于去除小物体和噪声和填补物体内的孔洞。

4.3.28 图像开/闭运算

先腐蚀后膨胀，去除小的噪声点、先膨胀后腐蚀，填补小的孔洞。
用于去除图像中的噪声和填补图像中的小孔。

4.3.15 顶帽/底帽运算

原图减去开运算结果，突出局部高亮区域、闭运算结果减去原图，突出局部暗区域。
用于提取亮斑和暗斑。

4.3.16 HoughLinesP 实现线条检测

使用概率霍夫变换检测直线。
用于检测图像中的直线。

4.3.17 Canny 边缘检测

使用多级边缘检测算法，提取图像边缘。
用于图像轮廓检测。

4.3.18 图像增强

通过对比度拉伸和直方图均衡化增强图像细节。
用于改善图像的视觉效果。

4.3.19 Roberts 算子/Prewitt 算子/Laplacian 算子提取图像边缘

使用 Roberts 算子、Prewitt 算子计算图像梯度，使用 Laplacian 算子计算图像二阶导数，提取边缘。
用于边缘检测。

4.3.20 LoG 边缘提取

使用高斯拉普拉斯算子进行边缘检测，减少噪声影响。
用于边缘检测和噪声抑制。

5 总结

通过数字图像处理课程的学习，我们开发了一款图像处理 web 平台。该平台的开发旨在为用户提供一个便捷、高效的图像处理工具，通过整合基础图像处理功能、黑白图像上色修复以及图像风格迁移三大主要功能，满足用户在不同应用场景下的需求。

在基础功能部分，我们实现了包括添加噪声、平滑处理、图像锐化、图像翻转等多种图像处理操作。这些操作是数字图像处理的基础，能够对图像进行预处理，为后续的高级处理奠定了基础。

黑白图像上色修复和图像风格迁移是本平台的高级功能。这两部分功能的实现基于卷积神经网络（CNN）和深度学习技术。通过 OpenCV 库和训练好的深度学习模型，我们能够将黑白图像转化为彩色图像，或将图像转换为另一种风格。模型的训练过程使用了大量的图像数据，利用深度学习的方法提取图像的特征，学习颜色和风格的转换规则。

在开发平台的过程中，我们深入学习并实践了 OpenCV 和深度学习技术。使用预训练的卷积神经网络模型，我们能够快速、高效地进行复杂的图像处理任务。整个项目增强了我们对数字图像处理和深度学习的理解，尤其是在实际应用中的效果和价值。这次的项目经历让我们对图像处理技术有了更全面和深入的认识，为未来的学习和工作打下了坚实的基础。

参考文献

[1]Johnson, J., Alahi, A., & Fei-Fei, L. (2016). Perceptual losses for real-time style transfer and super-resolution. In European conference on computer vision (pp. 694-711). Springer, Cham.

[2]Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576.

[3]Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE international conference on computer vision (pp. 2223-2232).

[4]arXiv:1603.08511[cs.CV]