Е. В. Купчина, Д. В. Коршикова

АЛГОРИТМ ВОССТАНОВЛЕНИЯ ИЗОБРАЖЕНИЙ: ПРИНЦИПЫ И ПРИМЕРЫ

В данной статье будут рассмотрены основные методы восстановления изображений и приведем пример модификации кода на Python, использующего библиотеку OpenCV для выполнения этой задачи.

Введение

Восстановление изображений – это процесс улучшения качества изображений, которые были искажены или повреждены. Такие искажения могут быть вызваны шумом, размытием, недостаточной резкостью или даже потерей информации.

I. Принципы восстановления изображений

Существует несколько методов восстановления изображений, включая:

1. Фильтрация: фильтры (такие как гауссов фильтр или медианный фильтр) используются для удаления шума. Они работают на основе анализа соседних пикселей и исправляют значения пикселей на основании их окружения.

2. Деконволюция: этот метод позволяет восстановить оригинальное изображение, основываясь на модели процесса искажения (например, размытия). Часто используется для корректировки размытых изображений.

3. Интерполяция: применяется для восстановления потерянной информации в изображении, когда оно было уменьшено или обрезано. Методы интерполяции (линейная, кубическая) помогают достичь визуально приемлемых результатов.

4. Преобразование Фурье: позволяет анализировать частоты в изображении и восстанавливать утраченные детали, работая с частотной областью.

II. Пример кода

Рассмотрим пример восстановления изображения с использованием медианного фильтра для удаления шума и функции cv2.inpaint для восстановления поврежденных областей. Для этого потребуется библиотека OpenCV.

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

Происходит загрузка изображения с помощью функции cv2.imread.

```
image = cv2.imread('input_image.jpg')
```

Добавление шума. Случайный шум добавляется к изображению для имитации искажения. Создание маски. Маска создается для областей изображения, которые необходимо восстановить. В данном примере условие создания маски основано на значении одного из цветовых каналов. Происходит восстановление изображения с помощью фильтрации – применяется медианный фильтр для удаления шума и инпейнтинга – функция cv2.inpaint восстанавливает определенные области на изображении, основываясь на окружающих пикселях.

```
noise = np.random.randint
(0, 50, image.shape, dtype='uint8')
noisy_image = cv2.add(image, noise)
mask = np.zeros
(noisy_image.shape[:2], dtype='uint8')
mask[noisy_image[:,:,0] > 200] = 255
denoised_image = cv2.medianBlur
(noisy_image, 5)
inpainted_image = cv2.inpaint
(denoised_image, mask, inpaintRadius=3,
flags=cv2.INPAINT_TELEA)
```

Отображение оригинального, зашумленного и восстановленного изображений

```
cv2.imshow('Original Image', image)
cv2.imshow('Noisy Image', noisy_image)
cv2.imshow('Inpainted Image',
inpainted_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

III. Пример работы кода

IV. Изменения в коде

Чтобы улучшить представленный код необходимо внести несколько изменений, чтобы повысить качество результатов и сделать его более универсальным. Вот некоторые изменения и улучшения, которые можно внедрить:

1. Изменение типа шума. Вместо добавления случайного шума, можно использовать более распространенные типы шумов, такие как соль и перец. Этот тип шума более реалистичен и часто встречается в реальных изображениях.

```
def add_salt_and_pepper_noise(image,
salt_prob, pepper_prob):
noisy_image = np.copy(image)
num_salt = np.ceil
(salt_prob * total_pixels)
coords = [np.random.randint(0, i - 1,
int(num_salt)) for i in image.shape]
noisy_image[coords[0], coords[1], :]
= 255
num_pepper = np.ceil
(pepper_prob * total_pixels)
coords = [np.random.randint(0, i - 1,
int(num_pepper)) for i in image.shape]
noisy_image[coords[0], coords[1], :]
= 0
return noisy_image
noisy_image = add_salt_and_pepper_noise
(image, salt_prob=0.02,
pepper_prob=0.02)
```

2. Улучшение создания маски. Необходимо добавить размытие к маске, чтобы сделать края менее резкими и сгладить переходы. Для этого создается маска на основе цветов пикселей, которые представляют шум, что позволяет более точно отчитывать области, которые нужно восстановить.

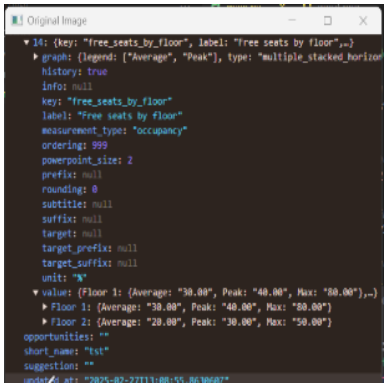
```
mask = np.zeros
(noisy_image.shape[:2],
dtype='uint8')
mask[noisy_image[:, :, 0] == 255]
= 255
mask[noisy_image[:, :, 0] == 0]
= 255
```

3. Применение Гауссово размытия на маске. Это улучшит плавность распределения маски вокруг шумных пикселей, что сделает восстановление более естественным.

```
mask = cv2.GaussianBlur(mask,
(5, 5), 0)
```

Эти улучшения помогут получить более качественное восстановление поврежденных областей в изображении.

V. Пример работы кода после внесений изменений



VI. Выводы

Восстановление изображений – это важная область компьютерного зрения, используемая в различных приложениях, включая медицинскую визуализацию, фотографию и обработку видео. Методы, такие как фильтрация, деконволюция и инпейнтинг, помогают восстанавливать поврежденные изображения и улучшать их качество. Приведенный выше код демонстрирует способ удаления шума и восстановления части изображения с использованием Python и OpenCV. Эти принципы могут быть адаптированы и расширены для более сложных задач восстановления.

1. Васильев, А.. Программирование на Python в примерах и задачах / А. Васильев // 2020. – С. 16-69.
2. Брэдски, Г., Кэлер, А.. Изучаем OpenCV 3 / Г. Брэдски, А. Кэлер // 2017. – С. 81-101.

Купчина Екатерина Валерьевна, магистрант кафедры информационных технологий автоматизированных систем БГУИР, ekupchina@bsuir.by.

Коршикова Дарья Валерьевна, магистрант кафедры информационных технологий автоматизированных систем БГУИР, korshikova@bsuir.by.

Научный руководитель: Трофимович Алексей Фёдорович, старший преподаватель, заместитель декана ФИТУ, trofimaf@bsuir.by.