Cwiczenie laboratoryjne

Temat: Restauracja starych fotografii

Cel ćwiczenia: Celem tego ćwiczenia jest zaprojektowanie narzędzia do przywracania i poprawiania starych oraz uszkodzonych fotografii. Zadaniem jest eliminacja zarysowań oraz wypełnianie brakujących fragmentów obrazu, wykorzystując narzędzia dostępne w bibliotekach do przetwarzania obrazów, takich jak Python z biblioteką OpenCV.

1. Podstawy teoretyczne

Eliminacja zarysowań (ang. scratch removal) to proces usuwania zarysowań z obrazów cyfrowych lub fotografii. Zarysowania to linie, smugi lub inne niechciane elementy, które pojawiły się na obrazie w wyniku uszkodzeń oryginalnego nośnika (np. filmu) lub w trakcie procesu przechowywania i użytkowania fotografii. W kontekście przetwarzania obrazów, eliminacja zarysowań może być realizowana za pomocą różnych technik. W przypadku restauracji starych fotografii, można stosować filtry i algorytmy, które pomagają w redukcji widoczności zarysowań na zdjęciu. Przykłady technik mogą obejmować:

1. **Filtracja medianowa:** Wykorzystuje się filtry medianowe do wygładzania obszarów zawierających zarysowania, co pomaga w ich redukcji.
2. **Algorytmy detekcji krawędzi:** Po wykryciu krawędzi na zdjęciu, można zastosować techniki, które pomagają w redukcji wpływu zarysowań na obszary krawędzi.
3. **Inpainting:** Wykorzystuje się algorytmy inpainting do zrekonstruowania obszarów, na których występują zarysowania, na podstawie otoczenia tych obszarów.
4. **Filtracja adaptacyjna:** Filtry adaptacyjne, takie jak filtr bilateralny, mogą być stosowane do wygładzania obrazu, jednocześnie zachowując ostrość krawędzi.

Eliminacja zarysowań jest istotnym krokiem w procesie przywracania starych fotografii, ponieważ pozwala na poprawę estetyki obrazu, przywracając mu pierwotny wygląd.

W języku Python istnieje kilka bibliotek dedykowanych przetwarzaniu obrazów, z których można skorzystać w kontekście zadania "Restauracja starych fotografii":

- OpenCV (Open Source Computer Vision Library): OpenCV to jedna z najbardziej rozpoznawalnych bibliotek do przetwarzania obrazów. Zapewnia bogaty zestaw narzędzi do manipulacji obrazami, operacji matematycznych oraz obsługi kamer.

```
pip install opencv-python
```

- NumPy: NumPy to biblioteka do obliczeń numerycznych w Pythonie. Jest często używana w połączeniu z OpenCV do efektywnego przetwarzania i manipulacji danymi obrazowymi.

```
pip install numpy
```

- Pillow (PIL Fork): Pillow to biblioteka do obsługi obrazów, która oferuje łatwy interfejs i obsługę wielu formatów plików obrazowych.

```
pip install Pillow
```

- scikit-image: scikit-image to biblioteka z pakietu scikit-learn, która dostarcza zestaw narzędzi do przetwarzania obrazów. Zawiera wiele algorytmów i funkcji przydatnych w analizie obrazów.

```
pip install scikit-image
```

- imageio: imageio to biblioteka umożliwiająca łatwe wczytywanie i zapisywanie różnych formatów plików obrazowych.

```
pip install imageio
```

Podczas prac nad zadaniem "Restauracja starych fotografii", korzystanie z kombinacji tych bibliotek pozwoli na skuteczne wczytywanie, przetwarzanie i analizę obrazów, co jest istotne dla eliminacji zarysowań i wypełniania brakujących fragmentów.

Dostosowanie parametrów narzędzia do przywracania fotografii, takiego jak narzędzie ***inpaint*** *w OpenCV, może być kluczowe dla osiągnięcia optymalnych rezultatów. Poniżej znajdują się pewne parametry, które można dostosować, oraz wskazówki dotyczące ich optymalizacji:*

- ***Rozmiar filtra MedianBlur:***

Parametr domyślny: W przykładowym kodzie jest używany filtr medianowy o rozmiarze 5x5 (cv2.medianBlur(obraz, 5)).

Optymalizacja: Dostosuj rozmiar filtra medianowego w zależności od rozmiaru zarysowań na obrazie. Większy rozmiar filtra może być bardziej skuteczny w usuwaniu większych zarysowań, ale może również wprowadzać rozmycie.

```
# Przykład z większym rozmiarem filtra
obraz_wyjsciowy = cv2.medianBlur(obraz, 9)
```

- ***Próg koloru w inpaint:***

Parametr domyślny: W funkcji cv2.inpaint używany jest próg koloru do określenia obszarów do przywracania (cv2.inpaint(obraz, maska, 3, cv2.INPAINT_TELEA)).

Optymalizacja: Dostosuj próg koloru w zależności od konkretnych warunków na zdjęciu. Zbyt niski próg może spowodować zbyt szerokie wypełnienia, a zbyt wysoki może pomijać pewne obszary.

```
# Przykład z dostosowaniem progu koloru
obraz_wyjsciowy = cv2.inpaint(obraz, maska, 5, cv2.INPAINT_TELEA)
```

- ***Metoda inpaint:***

Parametr domyślny: W przykładowym kodzie używana jest metoda cv2.INPAINT_TELEA.

Optymalizacja: Możesz eksperymentować z innymi dostępnymi metodami, takimi jak cv2.INPAINT_NS lub dostosować je w zależności od efektów, jakie chcesz osiągnąć.

```
# Przykład z użyciem innej metody inpaint
obraz_wyjsciowy = cv2.inpaint(obraz, maska, 3, cv2.INPAINT_NS)
```

- ***Dostosowanie algorytmu eliminacji zarysowań:***

Parametr domyślny: Filtr MedianBlur jest używany do eliminacji zarysowań (cv2.medianBlur(obraz, 5)).

Optymalizacja: Jeśli obraz zawiera głównie drobne zarysowania, możesz zastosować mniejszy rozmiar filtra. Jednakże, jeśli zarysowania są większe, zwiększenie rozmiaru filtra może być bardziej skuteczne

 # Przykład z mniejszym rozmiarem filtra do eliminacji zarysowań

obraz_po_elim_zarysowan = eliminacja_zarysowan(obraz_wejsciowy, rozmiar_filtra=3)

Eksperymentuj z tymi parametrami, monitoruj wyniki i dostosuj je do konkretnego przypadku użycia. Często proces optymalizacji wymaga wielokrotnych prób i błędów, aby znaleźć najlepsze parametry dla danego zbioru danych.

2. Zadanie do wykonania

1. ***Importowanie bibliotek:***
   - Zainstaluj bibliotekę OpenCV, jeśli jeszcze nie jest zainstalowana.
   - Zaimportuj niezbędne biblioteki do pracy z obrazami w Pythonie.
2. ***Wczytanie obrazu:***
   - Wczytaj starą i uszkodzoną fotografię do programu.
3. ***Eliminacja zarysowań:***
   - Zaimplementuj algorytm do eliminacji zarysowań na fotografii. Możesz wykorzystać różne filtry i metody dostępne w OpenCV.
4. ***Wypełnianie brakujących fragmentów:***
   - Zastosuj techniki wypełniania brakujących fragmentów obrazu. Możesz skorzystać z algorytmów bazujących na kontekście otoczenia, takich jak Inpainting.
5. ***Testowanie:***
   - Przetestuj stworzone narzędzie na różnych starych fotografiach, aby zweryfikować skuteczność eliminacji zarysowań i wypełniania brakujących fragmentów.
6. ***Optymalizacja:***
   - Zastanów się, jakie dodatkowe kroki lub algorytmy mogą poprawić jakość przywracania zdjęć.

Upewnij się, że podczas ćwiczenia korzystasz z różnych technik przetwarzania obrazu dostępnych w bibliotece OpenCV. Eksperymentuj z parametrami i zastosuj różne filtry, aby uzyskać jak najlepsze efekty przywracania starych fotografii.

Przykład 1:

```
import cv2
import numpy as np

def wczytaj_obraz(sciezka):
```

```
    # Wczytanie obrazu z podanej ścieżki
    obraz = cv2.imread(sciezka)
    return obraz

def eliminacja_zarysowan(obraz):
    # Zastosowanie filtra do eliminacji zarysowań (np. MedianBlur)
    obraz_wyjsciowy = cv2.medianBlur(obraz, 5)
    return obraz_wyjsciowy

def wypelnij_brakujace_fragmenty(obraz):
    # Zastosowanie algorytmu Inpainting do wypełnienia brakujących fragmentów
    maska = cv2.inRange(obraz, np.array([0, 0, 0]), np.array([10, 10, 10]))
    obraz_wyjsciowy = cv2.inpaint(obraz, maska, 3, cv2.INPAINT_TELEA)
    return obraz_wyjsciowy

def testowanie_i_ocena_skutecznosci(obraz_wejsciowy):
    # Przetestowanie narzędzia na różnych starych fotografiach
    obraz_po_elim_zarysowan = eliminacja_zarysowan(obraz_wejsciowy)
    obraz_po_wypelnieniu = wypelnij_brakujace_fragmenty(obraz_po_elim_zarysowan)

    # Wyświetlenie obrazów przed i po zastosowaniu narzędzia
    cv2.imshow('Obraz przed', obraz_wejsciowy)
    cv2.imshow('Obraz po eliminacji zarysowań', obraz_po_elim_zarysowan)
    cv2.imshow('Obraz po wypełnieniu brakujących fragmentów', obraz_po_wypelnieniu)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Wczytanie starnej i uszkodzonej fotografii
    sciezka_do_obrazu = 'sciezka/do/starej/fotografii.jpg'
    stara_fotografia = wczytaj_obraz(sciezka_do_obrazu)

    # Testowanie i ocena skuteczności narzędzia
    testowanie_i_ocena_skutecznosci(stara_fotografia)
```

W powyższym rozwiązaniu, wczytaj_obraz, eliminacja_zarysowan i wypelnij_brakujace_fragmenty są funkcjami pomocniczymi, a testowanie_i_ocena_skutecznosci przeprowadza testy narzędzia na zadanej fotografii. Możesz dostosować ścieżkę do własnej fotografii i eksperymentować z różnymi algorytmami dostępnymi w OpenCV, aby uzyskać optymalne wyniki przywracania starych fotografii.

Przykład 2:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def restore_old_photo(image_path, mask_path):
    # Wczytaj obraz starych fotografii
```

```
    old_photo = cv2.imread(image_path)

    # Wczytaj maskę obszarów do przywrócenia (maska czarna - uszkodzone obszary)
    damaged_mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)

    # Wykonaj inpainting, aby przywrócić uszkodzone obszary
    restored_photo = cv2.inpaint(old_photo, damaged_mask, inpaintRadius=3,
flags=cv2.INPAINT_TELEA)

    # Wyświetl obraz przed i po przywracaniu
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(old_photo, cv2.COLOR_BGR2RGB))
    plt.title("Old Photo")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(restored_photo, cv2.COLOR_BGR2RGB))
    plt.title("Restored Photo")
    plt.axis("off")

    plt.show()

# Ścieżki do obrazu starych fotografii i odpowiadającej mu maski
image_path = "sciezka/do/twojej/starej/fotografii.jpg"
mask_path = "sciezka/do/twojej/maski/uszkodzen.jpg"

# Przywróć i popraw starą fotografię
restore_old_photo(image_path, mask_path)
```

W powyższym kodzie używamy funkcji **cv2.inpaint** do przywracania uszkodzonych obszarów na starym zdjęciu na podstawie wcześniej przygotowanej maski. Obraz przed i po przywracaniu jest następnie wyświetlany za pomocą biblioteki matplotlib.

Efekty inpaintingu mogą zależeć od rodzaju uszkodzeń na zdjęciu, a wyniki mogą być różne w zależności od konkretnego przypadku. Możesz dostosować parametry inpaintingu w funkcji **cv2.inpaint** w zależności od potrzeb.
Aby stworzyć ***mask_path*** dla tego kodu, możemy utworzyć maskę, która zawiera obszary, gdzie chcemy przeprowadzić operacje eliminacji zarysowań i wypełnienia brakujących fragmentów. W poniższym przykładzie, maska zostanie stworzona jako binarna maska, gdzie białe obszary oznaczają obszary do przetworzenia, a czarne obszary są pomijane. Zakładamy, że czarne obszary w oryginalnym obrazie są obszarami, które nie wymagają operacji ***inpaint.***

```
import cv2
import numpy as np

def create_mask(image):
    # Utworzenie maski - tutaj zakładamy, że obszary do przetworzenia to obszary o niskiej
intensywności pikseli
    mask = cv2.inRange(image, np.array([0, 0, 0]), np.array([50, 50, 50]))
    return mask

if __name__ == "__main__":
```

```
# Wczytanie starej i uszkodzonej fotografii
sciezka_do_obrazu = 'sciezka/do/starej/fotografii.jpg'
stara_fotografia = wczytaj_obraz(sciezka_do_obrazu)

# Utworzenie maski
maska = create_mask(stara_fotografia)

# Ścieżka do zapisu maski
mask_path = 'sciezka/do/zapisu/maski.jpg'

# Zapis maski do pliku
cv2.imwrite(mask_path, maska)

# Testowanie i ocena skuteczności narzędzia
testowanie_i_ocena_skutecznosci(stara_fotografia)
```

W tym przykładzie maska jest utworzona przy użyciu funkcji **create_mask**, która tworzy maskę binarną, gdzie obszary o niskiej intensywności pikseli są traktowane jako obszary do przetworzenia. Następnie maska jest zapisywana do pliku o ścieżce **mask_path.** Możesz dostosować kryteria do utworzenia maski w zależności od specyfiki twoich obrazów.

Przyklad 3

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def load_image(file_path):
    """
    Wczytuje obraz z podanej ścieżki.

    Parameters:
    - file_path (str): Ścieżka do pliku obrazu.

    Returns:
    - image (numpy.ndarray): Wczytany obraz.
    """
    image = cv2.imread(file_path)
    return image

def create_damage_mask(image_shape):
    """
    Tworzy pustą maskę o wymiarach zgodnych z obrazem.

    Parameters:
    - image_shape (tuple): Kształt obrazu (wysokość, szerokość, liczba kanałów).

    Returns:
    - damage_mask (numpy.ndarray): Pusta maska uszkodzeń.
    """
    damage_mask = np.zeros(image_shape[:2], dtype=np.uint8)
    return damage_mask

def mark_damaged_areas(image, damage_mask):
```

```
    """
    Oznacza obszary uszkodzeń na obrazie na podstawie podanej maski.

    Parameters:
    - image (numpy.ndarray): Obraz do oznaczenia.
    - damage_mask (numpy.ndarray): Maska uszkodzeń.

    Returns:
    - marked_image (numpy.ndarray): Obraz z oznaczonymi uszkodzonymi obszarami.
    """
    marked_image = image.copy()
    marked_image[damage_mask > 0] = [0, 0, 255]  # Oznaczenie na czerwono
    return marked_image

def restore_damaged_areas(image, damage_mask):
    """
    Przywraca uszkodzone obszary na obrazie przy użyciu inpaintingu.

    Parameters:
    - image (numpy.ndarray): Obraz z uszkodzeniami.
    - damage_mask (numpy.ndarray): Maska uszkodzeń.

    Returns:
    - restored_image (numpy.ndarray): Obraz z przywróconymi obszarami.
    """
    restored_image = cv2.inpaint(image, damage_mask, inpaintRadius=3,
flags=cv2.INPAINT_TELEA)
    return restored_image

def display_images(original_image, marked_image, restored_image):
    """
    Wyświetla obraz przed i po procesie restauracji.

    Parameters:
    - original_image (numpy.ndarray): Oryginalny obraz.
    - marked_image (numpy.ndarray): Obraz z oznaczonymi uszkodzonymi obszarami.
    - restored_image (numpy.ndarray): Obraz z przywróconymi obszarami.
    """
    plt.figure(figsize=(15, 5))

    plt.subplot(1, 3, 1)
    plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 3, 2)
    plt.imshow(cv2.cvtColor(marked_image, cv2.COLOR_BGR2RGB))
    plt.title("Damaged Areas Marked")
    plt.axis("off")

    plt.subplot(1, 3, 3)
    plt.imshow(cv2.cvtColor(restored_image, cv2.COLOR_BGR2RGB))
    plt.title("Restored Image")
    plt.axis("off")
```

```
    plt.show()

# Ścieżka do pliku z obrazem starych fotografii
image_path = "sciezka/do/twojej/starej/fotografii.jpg"

# Wczytaj obraz
old_photo = load_image(image_path)

# Stwórz maskę uszkodzeń (na razie pustą)
damage_mask = create_damage_mask(old_photo.shape)

# Oznacz obszary uszkodzeń (interaktywne narzędzie graficzne lub ręczne zaznaczanie)
# Tutaj możesz dodać kod do interaktywnego zaznaczania obszarów na obrazie.

# Przywróć uszkodzone obszary
restored_photo = restore_damaged_areas(old_photo, damage_mask)

# Wyświetl wyniki
display_images(old_photo, mark_damaged_areas(old_photo, damage_mask), restored_photo)
```

3. Treść sprawozdania i jego forma

1. Numer i nazwa pracy laboratoryjnej.
2. Cele pracy laboratoryjnej.
3. Odpowiedzi na pytania kontrolne.
4. Skriny przedstawiające kolejność prac laboratoryjnych oraz wyniki uzyskane w trakcie ich realizacji.

4. Pytania kontrolne

1. Jak można zdefiniować pojęcie "Eliminacja zarysowań" w kontekście przetwarzania obrazów?
2. Jakie biblioteki w języku Python mogą być używane do przetwarzania obrazów w kontekście tego zadania?
3. Jakie są kroki procesu restauracji starych fotografii obejmujące wczytywanie obrazu?
4. Jakie są metody i filtry dostępne w bibliotece OpenCV do eliminacji zarysowań na fotografii?
5. W jaki sposób algorytmy bazujące na kontekście otoczenia, takie jak Inpainting, mogą być wykorzystane do wypełniania brakujących fragmentów obrazu?
6. Jakie są potencjalne wyzwania związane z przywracaniem starych fotografii, które mogą wymagać dodatkowych kroków lub algorytmów?
7. Jakie są korzyści stosowania filtra MedianBlur w kontekście eliminacji zarysowań na zdjęciach?
8. Jakie są inne popularne metody eliminacji zarysowań, które mogą być skuteczne w procesie restauracji fotografii?
9. W jaki sposób można dostosować parametry narzędzia do przywracania fotografii w celu osiągnięcia optymalnych rezultatów?
10. Jakie kryteria mogą być używane do oceny skuteczności narzędzia do przywracania i poprawiania starych fotografii?