Transformaty Ortogonalne

Klaudia Lota
Numer indeksu: 272576
Grupa: Wtorek TNP 17:05

1 lutego 2024

1 Odszumianie

W dziedzinie przetwarzania obrazów cyfrowych, szum odnosi się do wizualnych zakłóceń o wysokiej częstotliwości na obrazie. Algorytmy przetwarzania obrazów często są wykorzystywane w celu eliminacji tego szumu. Jedną z efektywnych metod filtrowania obrazu jest zastosowanie dyskretnej transformaty Fouriera. Proces ten obejmuje wykonanie transformaty, usunięcie składowych o wysokich częstotliwościach z obrazu, a następnie zastosowanie odwrotnej transformacji. Poniżej przedstawiono fragment kodu implementujący tę operację:

```
def Fourier_denoise(image):
    denoised = np.zeros_like(image, dtype=np.complex128)
    denoised = filter(image)
    return denoised
```

Funkcja `Fourier_denoise` przyjmuje obraz `image` jako argument. Tworzy zmienną `denoised`, będącą macierzą zer o tym samym kształcie co oryginalny obraz, ale w formie liczb zespolonych (`np.complex128`). Następnie, wynik funkcji `filter(image)` przypisywany jest do tej zmiennej.

Aby rozwijać tę implementację na obrazy kolorowe, gdzie mamy trzy kanały (R, G, B), zastosowano poniższy kod:

```
def Fourier_color_denoise(image):
    denoised = np.zeros_like(image, dtype=np.complex128)
    for i in range(image.shape[2]):
        denoised[:, :, i] = filter(image[:, :, i])

    return denoised
```

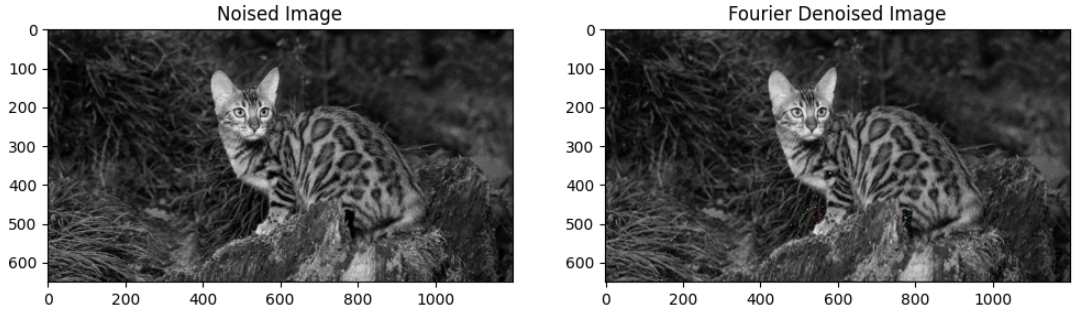
Funkcja `Fourier_color_denoise` inicjalizuje zmienną `denoised` jako macierz zer o tym samym kształcie co oryginalny obraz, ale w formie liczb zespolonych. Następnie, iteruje po kanałach kolorowych obrazu (czerwonym, zielonym, niebieskim), przypisując wyniki funkcji `filter` do odpowiednich kanałów w macierzy `denoised`. W rezultacie otrzymujemy odszumioną macierz zespoloną, gdzie każdy kanał obrazu kolorowego jest przetwarzany niezależnie.

Dodatkowo, można zaimplementować mechanizm odszumiania poprzez transformację falkową, co przedstawia poniższy kod:

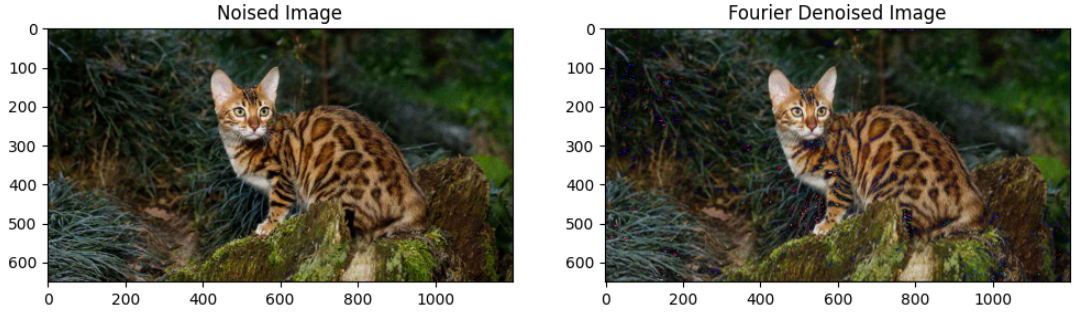
```
def Wavelet_denoise(image, threshold=0.1):
    coeffs_2d = pywt.dwt2(image, 'bior1.3')
    coeffs_2d = tuple(map(lambda x: pywt.threshold(x, threshold * np.max(np.abs(x)),
        mode='soft'), coeffs_2d))

    image = pywt.idwt2(coeffs_2d, 'bior1.3')
    image = np.round(image).astype(np.uint8)

    return image
```

Funkcja `Wavelet_denoise` przeprowadza dwuwymiarową transformację falkową (DWT) na wejściowym obrazie kolorowym, eliminując składowe o niskich częstotliwościach. Następnie, stosuje progowanie dla każdej składowej transformacji falkowej w celu redukcji szumów. Ostatecznie, odwraca transformację, uzyskując odszumiony obraz, który jest zaokrąglany do najbliższych całkowitych i przekształcany do formatu `uint8`. Ta implementacja umożliwia skuteczne odszumianie obrazów, zachowując istotne cechy oryginału.

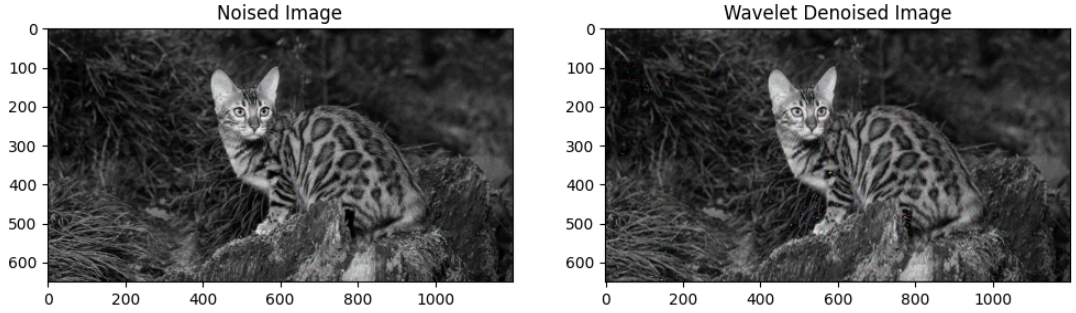
Poniżej znajduje się zobrazowanie działania każdego z algorytmów odszumiania.



Rysunek 1: Odszumianie transformacją Fouriera dla obrazu w odcieniach szarości.



Rysunek 2: Odszumianie transformacją Fouriera dla obrazu kolorowego.



Rysunek 3: Odszumianie transformacją Falkową dla obrazu w odcieniach szarości.

Przedstawione ilustracje obrazują efektywność każdego z algorytmów. Zastosowanie tych metod umożliwia skuteczne odszumianie obrazów, przy jednoczesnym zachowaniu kluczowych cech oryginalnych obrazów. Każdy z algorytmów może być optymalny w zależności od konkretnego przypadku użycia, pozostawiając pole do wyboru w zależności od specyfiki zadania i wymagań dotyczących jakości odszumiania.

2 Kompresja

Algorytm kompresji oparty na transformacji ortogonalnej obejmuje zastosowanie dyskretnej wersji jednej z takich transformacji, na przykład transformacji Fouriera, do reprezentacji obrazu 2D. W przypadku obrazów RGB konieczne jest przeprowadzenie transformacji dla każdego koloru osobno. Następnie, współczynniki o niskiej amplitudzie są redukowane do zera, a obraz poddawany jest odwrotnej transformacji (dekompresji).

Ocena skuteczności kompresji uwzględnia dwa kryteria: zawartość informacji (MAE) oraz stopień kompresji (C), gdzie X_o oznacza oryginalny obraz, X_c to obraz po kompresji, a X_d to obraz po dekompresji. N to liczba pikseli obrazu. Zwiększenie liczby współczynników niezerowych poprawi jakość kompresji, ale może jednocześnie obniżyć zawartość informacji.

Poniżej przedstawiono kod implementujący metodę kompresji przy użyciu transformacji Fouriera.

```
def Fourier_compress(img, ratio):
    transformed = np.fft.fft2(img)
    sorted_transformed = np.sort(np.abs(transformed.flatten()))
    threshold = sorted_transformed[int((1 - ratio) * len(sorted_transformed))]
    transformed[np.abs(transformed) < threshold] = 0
    return np.fft.ifft2(transformed).real
```

W algorytmie kompresji stratnej za pomocą transformacji Fouriera dla obrazów szarych, obraz w odcieniach szarości (`img`) jest przekształcany dwuwymiarową transformacją Fouriera (`np.fft.fft2`). To przenosi obraz do dziedziny częstotliwości, umożliwiając analizę składowych częstotliwości.

Określa się próg kompresji na podstawie współczynnika kompresji (`ratio`). Wyższy współczynnik oznacza większą kompresję przez odrzucenie części współczynników. Sortuje się wartości współczynników, a próg jest ustawiany na poziomie odpowiadającym pewnemu percentylowi posortowanych wartości.

Następnie, współczynniki poniżej progu są eliminowane, co prowadzi do utraty nieistotnych informacji i redukcji danych obrazu. Proces ten ma na celu uzyskanie kompresji bez znaczącej utraty jakości wizualnej.

Po odrzuceniu części współczynników, zmodyfikowany obraz przechodzi przez odwrotną transformację Fouriera (`np.fft.ifft2`). To przywraca obraz do pierwotnej dziedziny przestrzennej, zachowując osiągniętą kompresję.

Można łatwo przekształcić powyższy kod, aby obsługiwał również obrazy kolorowe. Przekształcona funkcja, wykorzystująca wcześniejszą implementację, została przedstawiona poniżej.

```
def Fourier_color_compress(image, ratio):
    RED, GREEN, BLUE = image[:,:,0], image[:,:,1], image[:,:,2]
    red, green, blue = Fourier_compress(RED, ratio), Fourier_compress(GREEN, ratio),
        Fourier_compress(BLUE, ratio)

    return np.dstack((red, green, blue))
```

Obraz kolorowy (`image`) najpierw jest rozdzielany na trzy kanały: czerwony, zielony i niebieski (R, G, B), co umożliwia niezależne przetwarzanie każdego kanału. Następnie, dla każdego z kanałów, stosuje się algorytm kompresji stratnej, używając transformacji Fouriera (`Fourier_compress`). Proces ten obejmuje przekształcenie, odrzucenie części współczynników na podstawie zadanego współczynnika kompresji, a potem przywrócenie kanału do dziedziny przestrzennej poprzez odwrotną transformację Fouriera. Finalnie, odszumione kanały (R, G, B) są łączone, tworząc ostateczny, odszumiony obraz kolorowy, zachowujący korelację między kanałami.

Możemy również zainplementować kompresję stratną korzystającą z transformacji falkowej. Poniżej przedstawiono kod.

```
def Wavelet_compress(image, ratio):
    img_array = np.array(image)
    coeffs = pywt.dwt2(img_array, "bior1.3")
    cA, (cH, cV, cD) = coeffs
    threshold = np.max(np.abs(cD)) * ratio / 10.0
    cD_threshold = pywt.threshold(cD, threshold, mode="soft")
    coeefs_threshold = (cA, (cH, cV, cD_threshold))
    result = pywt.idwt2(coeefs_threshold, "bior1.3")
    return result
```

Algorytm rozpoczyna się od konwersji obrazu wejściowego (`image`) do macierzy NumPy (`img_array`). Następnie, dwuwymiarowa transformacja falkowa (`pywt.dwt2`) generuje współczynniki przybliżone (`cA`) i detaliczne (`cH`, `cV`, `cD`). Wartości detaliczne w pionie (`cV`) i poziomie (`cH`) są kluczowe.

Próg kompresji jest określany na podstawie maksymalnej wartości bezwzględnej współczynnika detalicznego (`cD`), pomnożonej przez współczynnik kompresji (`ratio`), a następnie podzielonej przez 10.0. Współczynnik detaliczny `cD` jest progowany funkcją `pywt.threshold` z "soft thresholding".

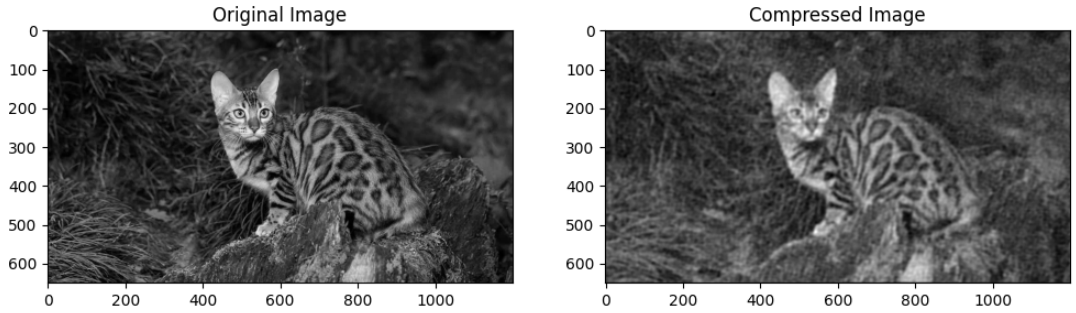
Wartości progowane są wstawiane z powrotem do struktury współczynników, przygotowując dane do odwrotnej transformacji. Ostateczny krok to odwrotna transformacja falkowa (`pywt.idwt2`), prowadząca do odszumionego obrazu, eliminującego nieistotne składowe detaliczne, zachowujące istotne dla rekonstrukcji obrazu.

Wynikiem działania funkcji jest odszumiony obraz, który jest zwracany jako rezultat.

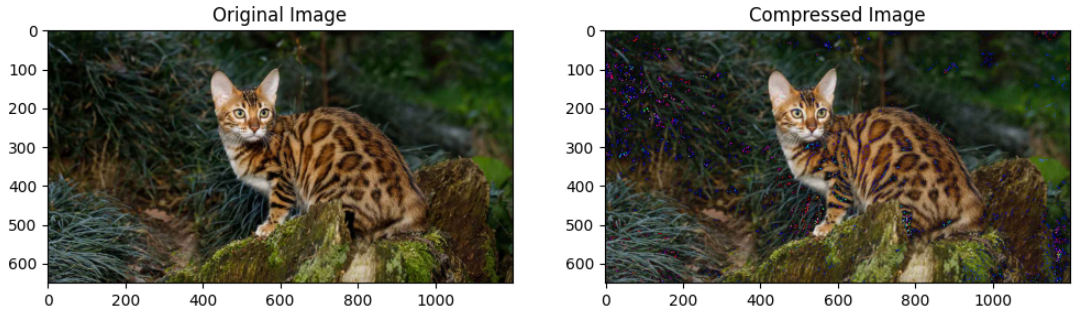
```
def Wavelet_compress_color(img, ratio):
    img_c = np.zeros_like(img, dtype=np.uint8)
    for ch in range(3):
        img_c[:,:,ch] = Wavelet_compress(img[:,:,ch], ratio)
    return img_c
```

`Wavelet_compress_color` przyjmuje obraz kolorowy `img` i wykorzystuje algorytm kompresji stratnej oparty na transformacji falkowej. Funkcja inicjalizuje macierz `img_c` o zerowych wartościach typu danych `uint8` do gromadzenia odszumionych kanałów kolorowych. Następnie, w pętli `for` dla każdego kanału kolorowego stosuje funkcję `Wavelet_compress` z zadanym współczynnikiem kompresji `ratio`. Otrzymane odszumione kanały są przypisywane do odpowiednich miejsc w macierzy wynikowej `img_c`. Finalnie, funkcja zwraca odszumiony obraz kolorowy. Implementacja pozwala na niezależne odszumianie kanałów, zachowując równowagę między redukcją szumów a zachowaniem informacji kolorystycznych.

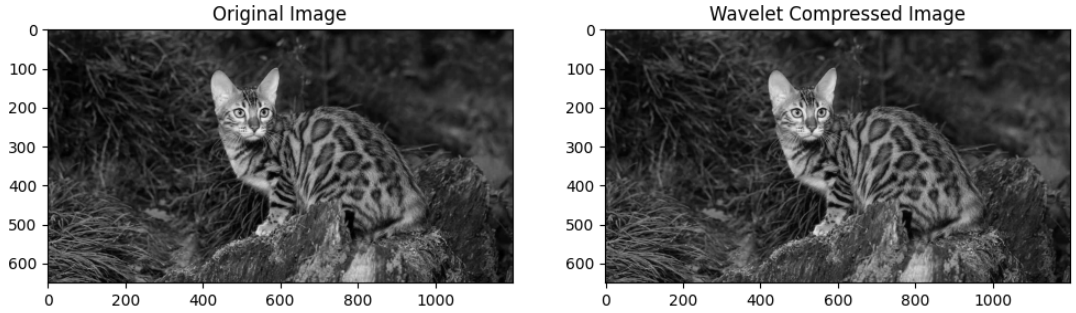
Poniżej znajduje się zobrazowanie działania każdego z algorytmów kompresji.



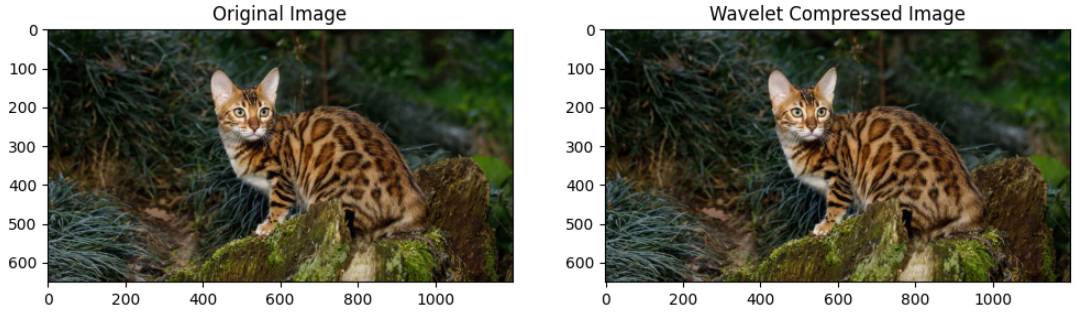
Rysunek 4: Kompresja transformacją Fouriera dla obrazu w odcieniach szarości.



Rysunek 5: Kompresja transformacją Fouriera dla obrazu kolorowego.



Rysunek 6: Kompresja transformacją Falkową dla obrazu w odcieniach szarości.



Rysunek 7: Kompresja transformacją Falkową dla obrazu kolorowego.

Ilustracje prezentują kontrast między kompresją obrazów w odcieniach szarości a kolorowych, stosując oba analizowane algorytmy.

3 Wnioski

Przedstawione eksperymenty z odszumianiem i kompresją obrazów wykorzystujące transformaty ortogonalne, w tym transformację Fouriera i falkową, dostarczają wglądu w skuteczność tych metod. W przypadku odszumiania, zastosowane algorytmy, takie jak odszumianie Fourierem czy falkowym, wykazują skuteczność w redukcji szumów, przy jednoczesnym zachowaniu kluczowych cech obrazów.

W kontekście kompresji, algorytmy oparte na transformacji Fouriera umożliwiają efektywną redukcję ilości danych poprzez eliminację współczynników o niskiej amplitudzie. Dodatkowo, zaimplementowany mechanizm kompresji falkowej dostarcza alternatywną strategię kompresji stratnej, gdzie eliminowane są niskie częstotliwości, przyczyniając się do skompresowania obrazu.

Oba podejścia do kompresji prezentują pewne zalety i ograniczenia. W przypadku transformacji Fouriera, skuteczność kompresji zależy od odpowiedniego doboru współczynnika kompresji, który wpływa na zachowanie informacji obrazowej. Z kolei, kompresja falkowa oferuje dodatkową elastyczność w redukcji szumów i zachowaniu szczegółów.