Wstęp do cyfrowego przetwarzania obrazów

Opracowano na podstawie:

An Introduction to Digital Image Processing with MATLAB

Notes for SCM2511 Image Processing 1

Semester 1, 2004

Alasdair McAndrew

School of Computer Science and Mathematics

Victoria University of Technology

Rozdział 1. Wprowadzenie

1.1. Obrazy i przedstawienia wizualne

Ludzie są przede wszystkim istotami wzrokowymi: w bardzo dużym stopniu polegamy na wzroku, aby rozumieć otaczający nas świat. Nie tylko patrzymy na obiekty po to, aby je rozpoznawać i klasyfikować, lecz także potrafimy szybko wyszukiwać różnice oraz po krótkim spojrzeniu uzyskać ogólne, przybliżone wrażenie obserwowanego otoczenia (sceny wizualnej).

Człowiek wykształcił bardzo precyzyjne umiejętności wzrokowe: potrafimy rozpoznać twarz niemal natychmiast, rozróżniać kolory oraz bardzo szybko przetwarzać dużą ilość informacji wizualnej.

Świat jest jednak w ciągłym ruchu – obserwowane otoczenia zmieni się w jakiś sposób. Nawet duża, trwała struktura, taka jak budynek albo góra, zmienia swój wygląd w zależności od pory dnia, ilości światła słonecznego, zachmurzenia lub cieni padających na powierzchnię.

W tym rozdziale zajmujemy się pojedynczymi obrazami - można je traktować jako migawki sceny wizualnej. Dla naszych potrzeb obraz jest pojedynczym przedstawieniem, które reprezentuje coś konkretnego. Może to być zdjęcie osoby, grupy ludzi lub zwierząt, scena zewnętrzna, mikrofotografia elementu elektronicznego albo wynik obrazowania medycznego. Nawet jeśli obraz nie jest od razu rozpoznawalny, nie jest on przypadkową plamą, lecz przedstawia pewną strukturę.

1.2. Czym jest przetwarzanie obrazów?

Przetwarzanie obrazów polega na zmianie natury obrazu w jednym z dwóch głównych celów:

1. poprawa informacji obrazowej w celu ułatwienia interpretacji przez człowieka,
2. dostosowanie obrazu do autonomicznej percepcji maszynowej.

W tych materiałach interesuje nas cyfrowe przetwarzanie obrazów, czyli użycie komputera do modyfikowania obrazu cyfrowego. Należy zauważyć, że powyższe dwa cele są różne, choć oba są równie ważne. Procedura, która spełnia pierwszy cel - czyli sprawia, że obraz „wygląda lepiej” - może być bardzo zła dla celu drugiego. Ludzie lubią obrazy ostre, czytelne i szczegółowe. Maszyny często „wolą” obrazy proste, uproszczone i pozbawione zbędnych detali.

Przykłady działań poprawiających obraz dla człowieka obejmują:

- wyostrzanie krawędzi na obrazie, aby wyglądał na bardziej ostry i czytelny,
- usuwanie szumu, czyli losowych błędów w obrazie,
- usuwanie rozmycia ruchowego, które powstaje, gdy obiekt porusza się zbyt szybko względem czasu naświetlania.

Wyostrzanie krawędzi jest bardzo ważnym elementem przygotowania obrazu do druku. Aby obraz wyglądał na stronie możliwie dobrze, zwykle wykonuje się pewną formę wyostrzania.

(a) The original image (b) Result after “sharperning”

Figure 1.1: Image sharperning

Usuwanie szumu jest częstym problemem rozwiązywanym przy transmisji danych. Różne elementy elektroniczne mogą wpływać na przesyłane dane, a skutkiem mogą być niepożądane zakłócenia. Różne rodzaje szumu wymagają różnych metod usuwania.

(a) The original image (b) After removing noise

Figure 1.2: Removing noise from an image

Usuwanie rozmycia ruchowego jest przydatne na przykład wtedy, gdy chcemy odczytać numer rejestracyjny samochodu lub zobaczyć detale, które w obrazie pierwotnym są niewyraźne. Rozmycie ruchowe może powstać wtedy, gdy czas otwarcia migawki aparatu jest zbyt długi względem prędkości obiektu.

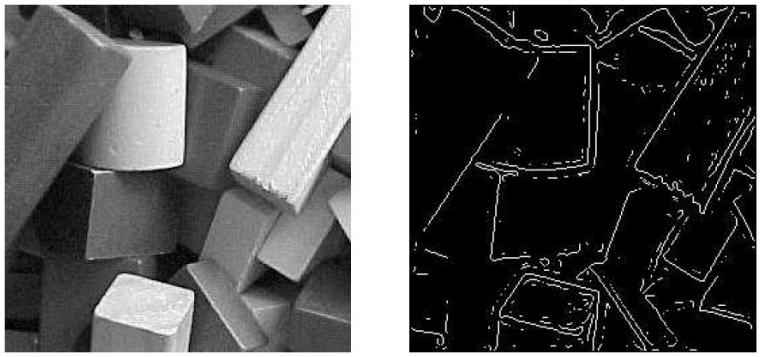
(a) The original image (b) After removing the blur

Figure 1.3: Image deblurring

Przykłady działań przygotowujących obraz do percepcji maszynowej obejmują:

- wyznaczanie/detekcja krawędzi obrazu,
- usuwanie szczegółów, które nie są istotne dla pomiaru lub rozpoznawania.

Wyznaczanie krawędzi może być konieczne do pomiaru obiektów na obrazie. Po znalezieniu krawędzi można mierzyć ich rozciągłość oraz pole powierzchni obszaru, który ograniczają. Detekcja krawędzi może być także pierwszym krokiem przed wyostrzaniem.



(a) The original image (b) Its edge image

Figure 1.4: Finding edges in an image

Usuwanie szczegółów - w zastosowaniach pomiarowych lub przy zliczaniu obiektów często nie interesują nas wszystkie drobne elementy sceny. Jeśli maszyna kontroluje produkty na linii produkcyjnej, najważniejsze mogą być kształt, rozmiar lub kolor. W takich przypadkach obraz można uprościć, aby pozostawić tylko zgrubną strukturę obrazu.

(a) The original image (b) Blurring to remove detail

Figure 1.5: Blurring an image

1.3. Akwizycja obrazu i próbkowanie

Próbkowanie oznacza proces cyfryzacji funkcji ciągłej. Jeśli pobieramy zbyt mało próbek funkcji, otrzymujemy przykład niedostatecznego próbkowania. W takim przypadku liczba punktów nie wystarcza do odtworzenia funkcji. Jeśli jednak pobierzemy wystarczająco dużo próbek, funkcję można zrekonstruować, a jej właściwości mogą być określone na podstawie próbek.

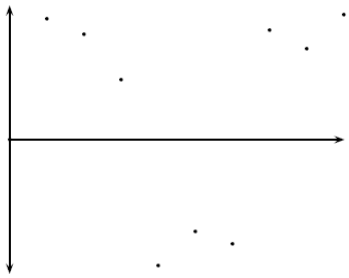


Figure 1.6: Sampling a function—undersampling

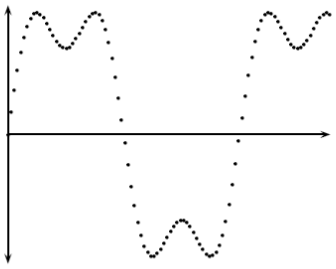


Figure 1.7: Sampling a function with more points

Aby mieć pewność, że liczba próbek jest wystarczająca, okres próbkowania nie powinien być większy niż połowa najmniejszego szczegółu występującego w funkcji. Zasada ta jest znana jako kryterium Nyquista. Można ją sformułować w postaci twierdzenia o próbkowaniu: funkcja ciągła może zostać zrekonstruowana ze swoich próbek, jeżeli częstotliwość próbkowania jest co najmniej dwa razy większa od maksymalnej częstotliwości występującej w funkcji.

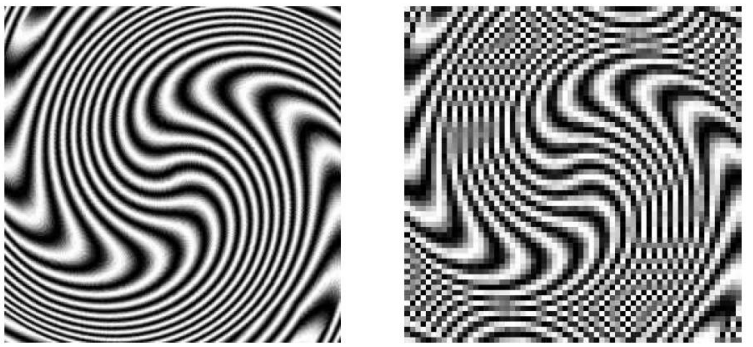
$$f_s \geq 2f_{max}$$

gdzie:

- f_s - częstotliwość próbkowania,
- f_{max} - maksymalna częstotliwość występująca w sygnale.

Próbkowanie obrazu wymaga zastosowania tej samej zasady. Obraz można traktować jako ciągłą funkcję dwóch zmiennych, którą próbkujemy, aby uzyskać obraz cyfrowy. Zbyt mała liczba próbek prowadzi do aliasingu. Na obrazie aliasing objawia się na przykład jako poszarpane krawędzie.

Aliasing - zniekształcenie wynikające ze zbyt rzadkiego próbkowania; pojawianie się fałszywych struktur i częstotliwości w obrazie wskutek niedostatecznego próbkowania.



Correct sampling; no aliasing

An undersampled version with aliasing

Figure 1.8: Effects of sampling

Aby otrzymać obraz cyfrowy, rozpoczynamy zwykle od ciągłej reprezentacji sceny wizyjnej. Scena odbija energię, którą rejestrujemy. Najczęściej używaną energią jest światło widzialne, lecz możliwe jest też użycie innych źródeł energii.

Obrazowanie z użyciem światła

Światło jest dominującym źródłem energii dla obrazów, ponieważ jest ono bezpośrednio obserwowalne przez człowieka. Znane nam fotografie są obrazowym zapisem sceny wizualnej.

Wiele obrazów cyfrowych jest rejestrowanych za pomocą światła widzialnego. Jest to rozwiązanie bezpieczne, tanie, łatwe do wykrycia i łatwe do przetwarzania przy użyciu odpowiedniego sprzętu. Dwa popularne sposoby pozyskiwania obrazów cyfrowych to kamera cyfrowa oraz skaner płaski.

Kamera CCD. W kamerze CCD zamiast klasycznego filmu znajduje się macierz fotositów (najmniejszy, fizyczny element światłoczuły znajdujący się na matrycy aparatu cyfrowego). Są to krzemowe elementy elektroniczne, których napięcie

wyjściowe jest proporcjonalne do natężenia padającego światła. Informacja z fotositów jest następnie przekazywana do odpowiedniego nośnika pamięci. Zwykle proces ten jest wykonywany sprzętowo, ponieważ jest to szybsze i bardziej wydajne niż rozwiązanie programowe.

Wynikiem działania kamery CCD jest tablica wartości. Każda wartość reprezentuje próbkę pierwotnej sceny. Elementy tej tablicy nazywamy elementami obrazu, czyli pikselami.

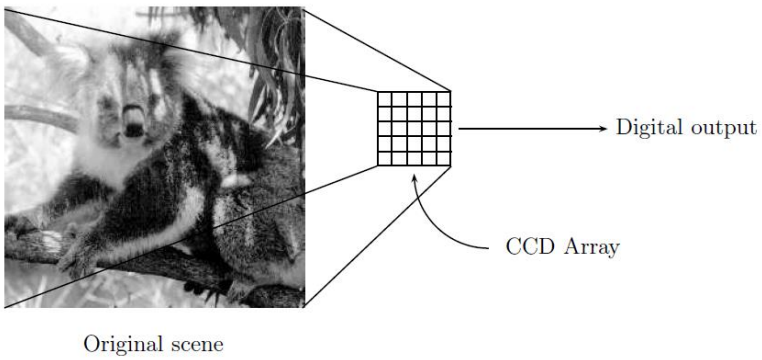


Figure 1.9: Capturing an image with a CCD array

Skaner płaski. Skaner działa podobnie do kamery CCD, ale zamiast rejestrować cały obraz jednocześnie, pojedynczy rząd fotositów przesuwa się nad obrazem i przechwytuje go wiersz po wierszu. Proces ten jest wolniejszy niż wykonanie zdjęcia kamerą, dlatego może być obsługiwany programowo.

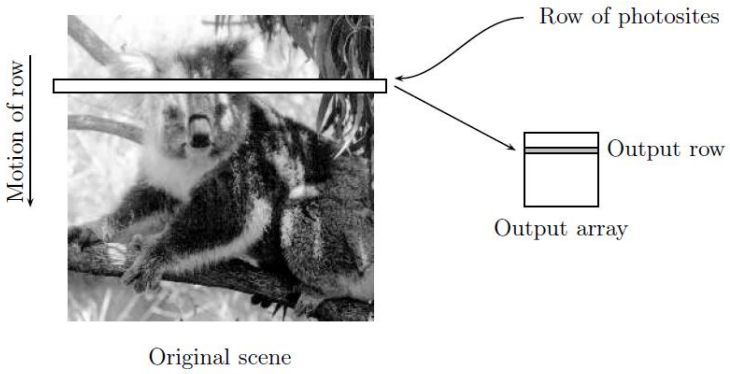


Figure 1.10: Capturing an image with a CCD scanner

Inne źródła energii

Chociaż światło jest popularne i łatwe w użyciu, do tworzenia obrazów można wykorzystać także inne źródła energii. Światło widzialne jest częścią widma elektromagnetycznego. Promieniowanie elektromagnetyczne obejmuje bardzo szeroki zakres długości fal: od promieniowania kosmicznego i gamma, przez promieniowanie rentgenowskie, ultrafiolet, światło widzialne, podczerwień, mikrofale, fale radiowe, aż po energię elektryczną o bardzo dużej długości fali.

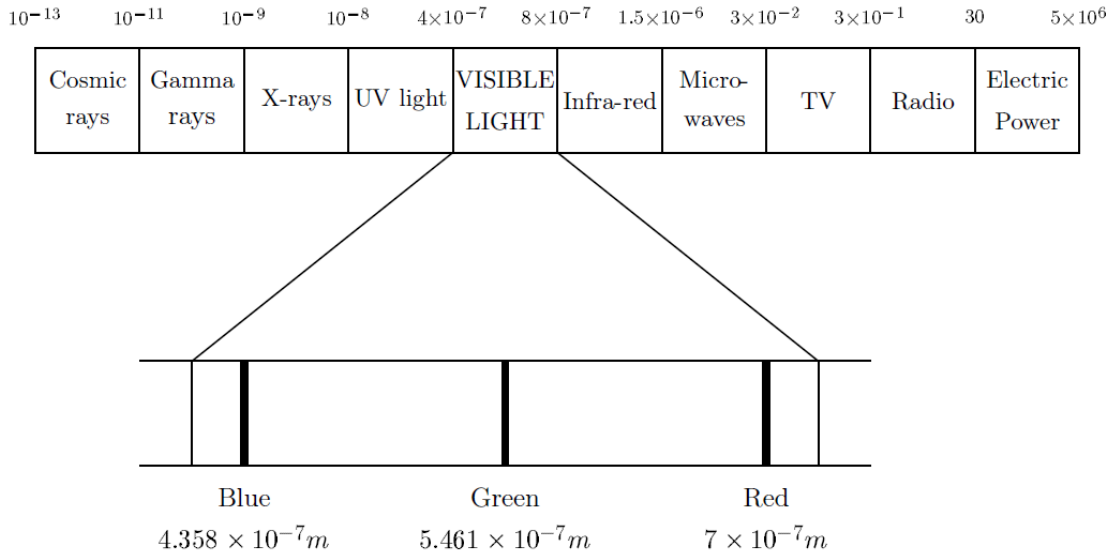


Figure 1.11: The electromagnetic spectrum

Promieniowanie rentgenowskie ma krótszą długość fali niż światło widzialne, dlatego może służyć do rozdzielania mniejszych obiektów. Jest także przydatne do badania struktur ukrytych przed wzrokiem, na przykład kości.

Inną metodą pozyskiwania obrazów jest tomografia rentgenowska. Obiekt jest otaczany wiązką promieniowania rentgenowskiego, a promienie przechodzące przez obiekt są rejestrowane przez detektory po przeciwnej stronie. Gdy wiązka obraca się wokół obiektu, można zrekonstruować obraz przekroju. Taki obraz nazywamy tomogramem. W tomografii komputerowej CAT pacjent leży w rurze, wokół której porusza się układ promieni rentgenowskich. Pozwala to utworzyć wiele przekrojów, które można połączyć w trójwymiarowy obraz.

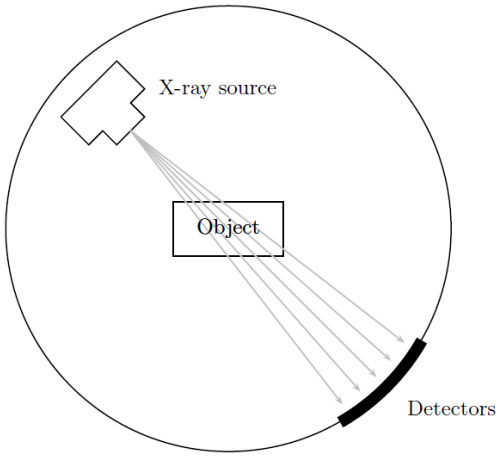


Figure 1.12: X-ray tomography

1.4. Obrazy i obrazy cyfrowe

Rozważmy fotografię monochromatyczną, czyli obraz w odcieniach szarości. Taki obraz można traktować jako funkcję dwóch zmiennych. Wartość funkcji oznacza jasność obrazu w danym punkcie.

$$f(x, y)$$

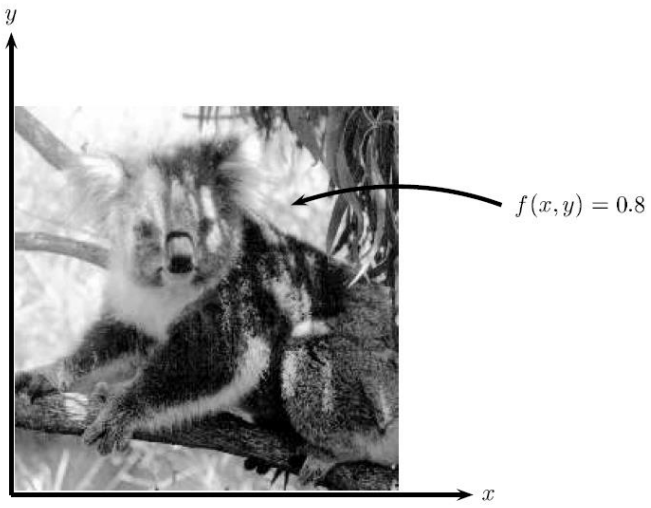


Figure 1.13: An image as a function

W obrazie ciągłym współrzędne x i y oraz wartości jasności mogą przyjmować wartości rzeczywiste. W obrazie cyfrowym współrzędne oraz wartości jasności są dyskretne. Zwykle przyjmują wartości całkowite. Przykładowo współrzędne mogą zmieniać się od 1 do 256, a jasność od 0 do 255, gdzie 0 oznacza czerń, a 255 biel.

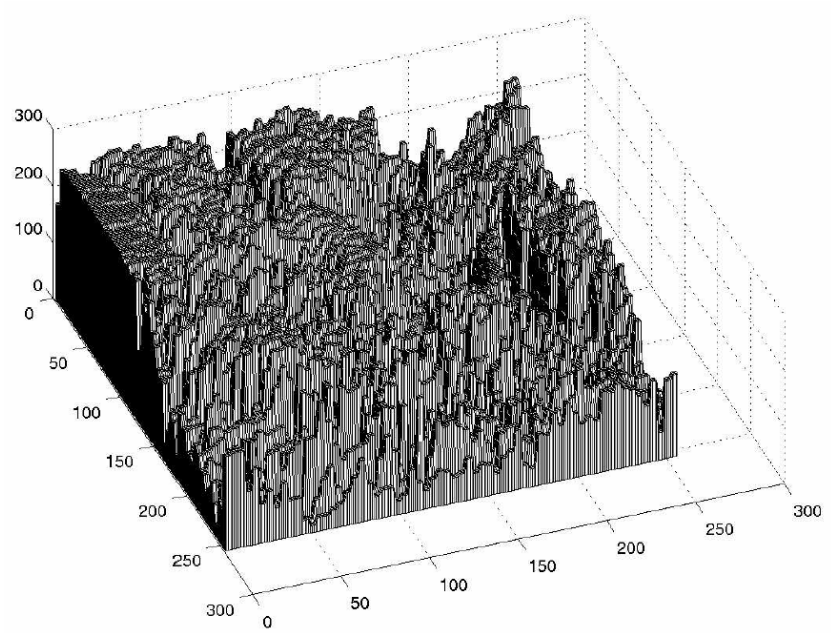


Figure 1.14: The image of figure 1.13 plotted as a function of two variables

Obraz cyfrowy można więc traktować jako dużą tablicę próbek pobranych z obrazu ciągłego. Każda próbka ma określoną, skwantowaną jasność. Próbki te są pikselami tworzącymi obraz cyfrowy.

Piksele otaczające dany piksel tworzą jego sąsiedztwo. Sąsiedztwo może być opisane tak jak macierz, np. jako sąsiedztwo 3×3 albo 5×5. Z wyjątkiem szczególnych przypadków sąsiedztwa mają nieparzystą liczbę wierszy i kolumn, ponieważ wtedy bieżący piksel znajduje się dokładnie w środku sąsiedztwa.

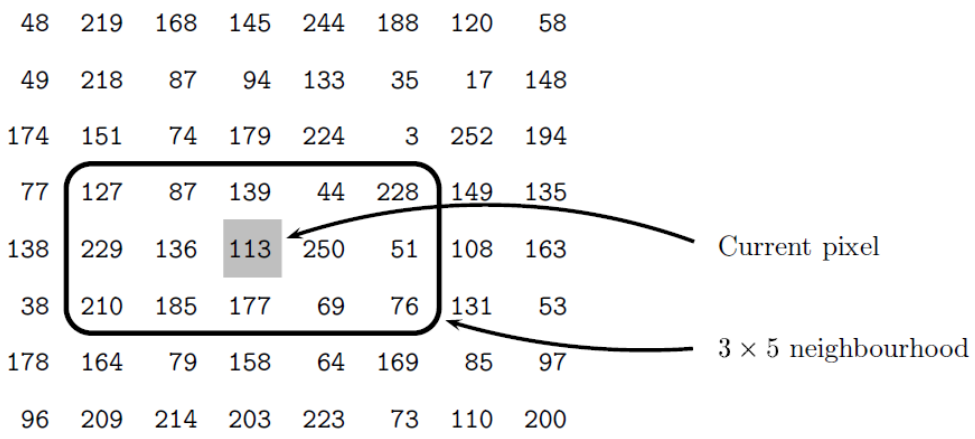


Figure 1.15: Pixels, with a neighbourhood

1.5. Wybrane zastosowania

Przetwarzanie obrazów ma ogromną liczbę zastosowań. Prawie każda dziedzina nauki i techniki może korzystać z metod przetwarzania obrazów. Przykładowe zastosowania obejmują:

1. **Medycyna** - inspekcja i interpretacja obrazów rentgenowskich, MRI lub CAT, analiza obrazów komórek oraz kariotypów chromosomów.
2. **Rolnictwo** - analiza zdjęć satelitarnych i lotniczych w celu określenia sposobu wykorzystania terenów, badanie przydatności regionów do różnych upraw, kontrola jakości owoców i warzyw.
3. **Przemysł** - automatyczna kontrola elementów na linii produkcyjnej, analiza próbek papieru.
4. **Organy ścigania** - analiza odcisków palców, wyostrzanie lub odszumianie obrazów z fotoradarów.

1.6. Aspekty przetwarzania obrazów

Wygodnie jest dzielić algorytmy przetwarzania obrazów na klasy. Różne algorytmy służą różnym zadaniom i problemom.

Poprawa jakości obrazu (image enhancement). Oznacza przetwarzanie obrazu tak, aby wynik był bardziej odpowiedni dla konkretnego zastosowania. Przykłady:

- wyostrzanie lub usuwanie rozmycia,
- podkreślanie krawędzi,
- poprawa kontrastu,
- rozjaśnianie obrazu,
- usuwanie szumu.

Restauracja obrazu (image restoration). Można ją traktować jako odwracanie uszkodzeń obrazu spowodowanych znaną przyczyną. Przykłady:

- usuwanie rozmycia spowodowanego ruchem liniowym,
- usuwanie zniekształceń optycznych,
- usuwanie zakłóceń okresowych.

Segmentacja obrazu (image segmentation). Polega na podziale obrazu na części składowe albo izolowaniu wybranych aspektów obrazu. Przykłady:

- znajdowanie linii, okręgów lub określonych kształtów,
- identyfikacja samochodów, drzew, budynków albo dróg na zdjęciu lotniczym.

Klasy te nie są rozłączne. Ten sam algorytm może być użyty zarówno do poprawy jakości obrazu, jak i do jego restauracji. Zawsze należy jednak określić cel działania: czy chcemy jedynie poprawić wygląd obrazu, czy też usunąć konkretne uszkodzenie.

1.7. Przykład zadania przetwarzania obrazów

Rozważmy praktyczne zadanie: automatyczne odczytywanie kodów pocztowych z kopert. Można je wykonać w kilku etapach.

Akwizycja obrazu. Najpierw należy utworzyć obraz cyfrowy papierowej koperty. Można użyć kamery CCD albo skanera.

Wstępne przetwarzanie. Ten etap poprzedza główne zadanie. Celem jest wykonanie podstawowych operacji, które ułatwią dalszą analizę. Można poprawić kontrast, usunąć szum albo znaleźć obszary, w których prawdopodobnie znajduje się kod pocztowy.

Segmentacja. Na tym etapie właściwie „wydobywa się" kod pocztowy z obrazu, czyli wycina się część obrazu zawierającą wyłącznie kod.

Reprezentacja i opis. Oznacza to wyznaczanie cech, które pozwalają odróżniać obiekty. Dla cyfr będą to na przykład krzywe, otwory i naroża.

Rozpoznawanie i interpretacja. Oznacza przypisywanie etykiet obiektom na podstawie ich opisów oraz nadawanie tym etykietom znaczenia. W ten sposób rozpoznajemy cyfry i interpretujemy ciąg czterech cyfr na końcu adresu jako kod pocztowy.

1.8. Typy obrazów cyfrowych

Wyróżnimy cztery podstawowe typy obrazów cyfrowych.

Obrazy binarne

Każdy piksel jest tylko czarny albo biały. Ponieważ istnieją tylko dwie możliwe wartości piksela, do reprezentacji jednego piksela wystarcza jeden bit. Obrazy binarne są więc bardzo oszczędne pamięciowo. Nadają się do przedstawiania tekstu, pisma odręcznego, odcisków palców lub planów architektonicznych.

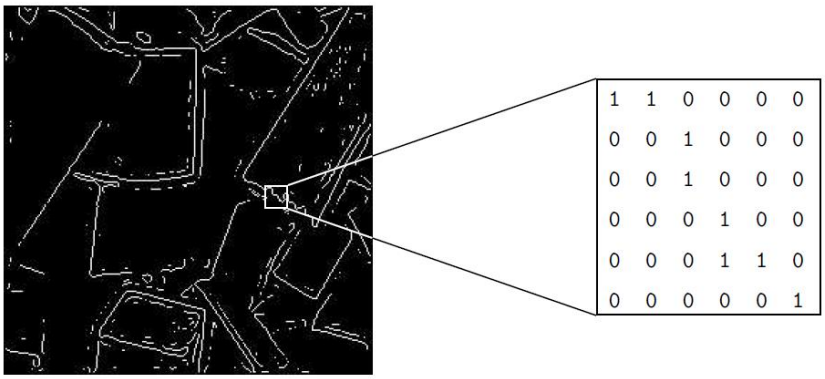


Figure 1.16: A binary image

Obrazy w odcieniach szarości

Każdy piksel ma pewien odcień szarości, zwykle z zakresu od 0 do 255. Taki zakres oznacza, że każdy piksel można zapisać za pomocą 8 bitów, czyli jednego bajtu. Jest to bardzo naturalny zakres przy zapisie plików obrazowych. Inne zakresy też są stosowane; zwykle są potęgami liczby 2.

Obrazy w odcieniach szarości występują w medycynie, w obrazach dokumentów drukowanych oraz w wielu innych zastosowaniach. W praktyce 256 poziomów szarości wystarcza do rozpoznania większości naturalnych obiektów.

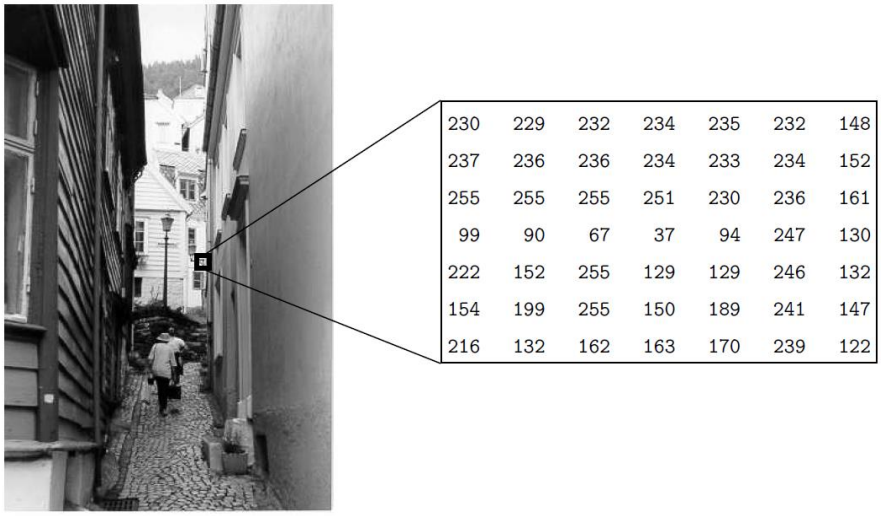


Figure 1.17: A greyscale image

Obrazy RGB, czyli true colour

W obrazie RGB każdy piksel ma kolor opisany przez ilość czerwieni, zieleni i błękitu. Jeśli każda składowa ma zakres od 0 do 255, łączna liczba możliwych kolorów wynosi:

$$256^3 = 16\,777\,216$$

Jest to wystarczająca liczba kolorów dla praktycznie dowolnego obrazu. Ponieważ do zapisu trzech składowych potrzeba 24 bitów, takie obrazy nazywa się obrazami 24-bitowymi.

Obraz RGB można traktować jako „stos” trzech macierzy: jednej dla składowej czerwonej, jednej dla zielonej i jednej dla niebieskiej. Dla każdego piksela mamy więc trzy wartości.

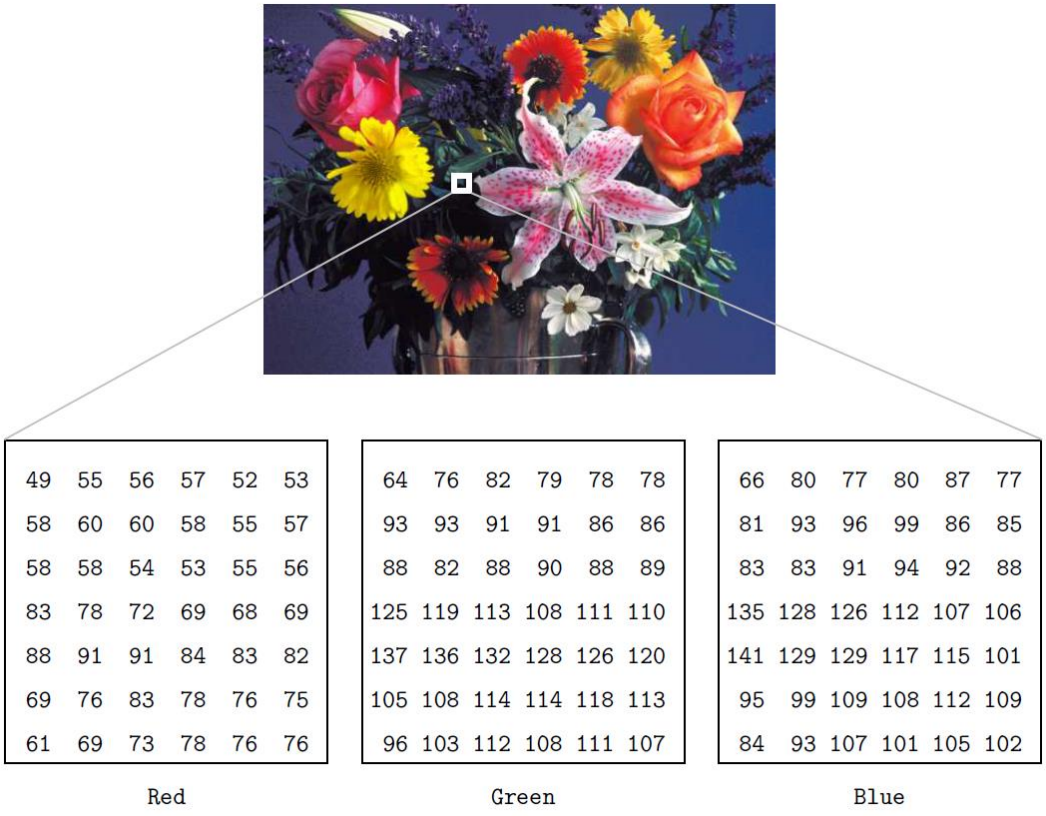


Figure 1.18: A true colour image

Obrazy indeksowane

Większość obrazów kolorowych używa tylko niewielkiego podzbioru ponad szesnastu milionów możliwych kolorów. Dla wygody zapisu i obsługi plików obraz może mieć przypisaną mapę kolorów, czyli paletę. Paleta jest listą kolorów użytych w obrazie. W takim przypadku wartość piksela nie oznacza bezpośrednio koloru, lecz indeks do mapy kolorów.

Jest wygodne, gdy obraz ma 256 kolorów lub mniej, ponieważ wtedy indeks każdego piksela można zapisać jednym bajtem. Niektóre formaty plików, takie jak GIF, ograniczają liczbę kolorów właśnie z tego powodu.

Figure 1.19: An indexed colour image

1.9. Rozmiary plików obrazowych

Pliki obrazowe mają zwykle duże rozmiary. Ilość informacji zależy od rodzaju obrazu, jego wymiarów oraz liczby bitów przypadających na piksel.

Dla obrazu 512×512 wartości są następujące.

Obraz binarny:

$$512 \times 512 \times 1 \text{ bit} = 262\,144 \text{ bity}$$

$$262\,144 \text{ bity} = 32\,768 \text{ bajtów} \approx 32{,}77 \text{ KB}$$

Obraz w odcieniach szarości:

$$512 \times 512 \times 1 \text{ bajt} = 262\,144 \text{ bajty} \approx 262{,}14 \text{ KB}$$

Obraz RGB:

$$512 \times 512 \times 3 \text{ bajty} = 786\,432 \text{ bajty} \approx 786{,}43 \text{ KB}$$

Wiele obrazów jest znacznie większych. Obrazy satelitarne mogą mieć po kilka tysięcy pikseli w każdym kierunku.

1.10. Percepcja obrazu

Duża część przetwarzania obrazów dotyczy sprawiania, aby obraz wyglądał „lepiej” dla człowieka. Trzeba więc znać ograniczenia ludzkiego systemu wzrokowego. Percepcja obrazu składa się z dwóch podstawowych etapów:

1. przechwycenie obrazu przez oko,
2. rozpoznanie i interpretacja obrazu przez korę wzrokową mózgu.

Połączenie tych etapów oraz ich ogromna zmienność wpływa na to, jak postrzegamy świat.

Należy pamiętać o kilku faktach.

Po pierwsze, obserwowana intensywność zależy od tła. Ten sam szary kwadrat będzie wydawał się ciemniejszy na jasnym tle niż na ciemnym tle. Nie postrzegamy więc poziomów szarości absolutnie, lecz w relacji do otoczenia.

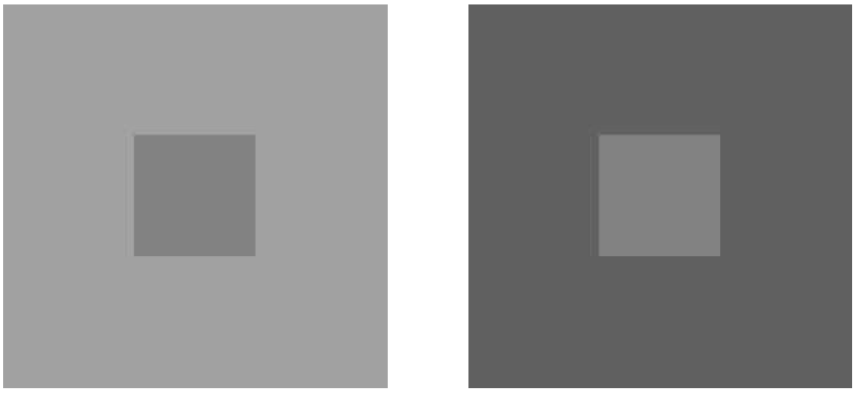


Figure 1.20: A grey square on different backgrounds

Po drugie, możemy widzieć nieistniejące intensywności w obrazie, którego poziomy szarości zmieniają się płynnie. W ciągłym gradiencie oko może dostrzegać sztuczne pasma albo krawędzie.

Figure 1.21: Continuously varying intensities

Po trzecie, system wzrokowy ma tendencję do przeszacowania lub niedoszacowania jasności w pobliżu granicy obszarów o różnych intensywnościach. Jasna krawędź może wydawać się jaśniejsza niż reszta jasnego obszaru, a ciemna krawędź może wydawać się ciemniejsza.

1.11. Obrazy w skali szarości w Matlabie

Obrazy mogą być traktowane jako macierze, których elementami są wartości pikseli. Matlab bardzo sprawnie obsługuje macierze, dlatego nadaje się do pracy z obrazami.

Załóżmy, że pracujemy w oknie poleceń Matlab. Polecenie:

```
w = imread('wombats.tif');
```

odczytuje wartości szarości wszystkich pikseli obrazu `wombats.tif` i umieszcza je w macierzy w. Macierz w staje się zmienną Matlaba, na której można wykonywać operacje macierzowe.

Należy zauważyć dwie rzeczy:

1. Polecenie kończy się średnikiem. Dzięki temu wynik nie jest wyświetlany w oknie poleceń. Ponieważ wynik może być dużą macierzą, nie chcemy wypisywać wszystkich wartości.
2. Nazwa pliku jest podana w apostrofach. Bez apostrofów Matlab potraktowałby `wombats.tif` jako nazwę zmiennej.

Obraz można wyświetlić poleceniem:

```
figure, imshow(w), pixval on
```

Jest to w rzeczywistości ciąg trzech poleceń:

- `figure` tworzy nowe okno graficzne,
- `imshow(w)` wyświetla macierz jako obraz,
- `pixval on` włącza wyświetlanie wartości piksela pod kursorem.

Wartości piksela są wyświetlane w postaci współrzędnych i wartości szarości. Ponieważ obraz `wombats.tif` jest 8-bitowym obrazem w skali szarości, wartości pikseli są liczbami całkowitymi od 0 do 255.

Obraz można też wyświetlić bez wcześniejszego zapisu do macierzy:

```
imshow('wombats.tif')
```

Lepiej jednak używać macierzy, ponieważ Matlab bardzo efektywnie je przetwarza.

Figure 1.22: The wombats image with `pixval on`

1.12. Obrazy RGB

Kolory trzeba opisywać w pewien standardowy sposób, zwykle jako podzbiór trójwymiarowego układu współrzędnych. Taki opis nazywa się modelem barw. Dla wyświetlania i zapisu obrazów standardowym modelem jest RGB.

W modelu RGB wszystkie kolory można wyobrazić sobie jako punkty wewnątrz sześcianu barw. Oś czerwona, zielona i niebieska odpowiadają składowym R, G i B. Kolory na przekątnej od czerni do bieli mają równe wartości wszystkich trzech składowych, dlatego są odcieniami szarości.

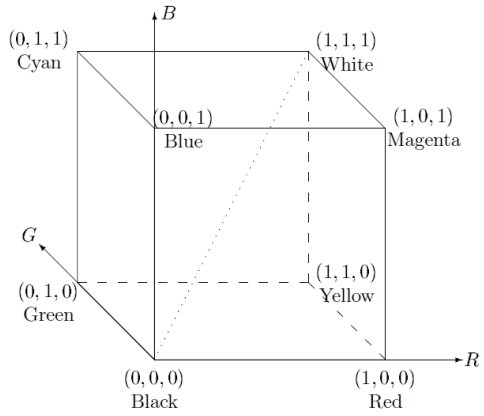


Figure 1.23: The colour cube for the RGB colour model

W Matlabie obrazy 24-bitowe RGB obsługuje się podobnie jak obrazy w skali szarości:

```
a = imread('autumn.tif');

figure, imshow(a), pixval on
```

Wartość piksela składa się wtedy z trzech liczb: czerwonej, zielonej i niebieskiej składowej koloru.

Istotna różnica polega na tym, że obraz RGB jest tablicą trójwymiarową. Polecenie:

```
size(a)
```

zwraca liczbę wierszy, liczbę kolumn i liczbę „warstw”. Dla obrazu RGB są to trzy warstwy odpowiadające kanałom R, G i B.

Aby odczytać wartość zielonej składowej piksela w wierszu 100 i kolumnie 200, można użyć:

```
a(100,200,2)
```

Aby odczytać wszystkie trzy składowe koloru:

```
a(100,200,:)
```

Funkcja `impixel` jest wygodna do pobierania wartości RGB:

```
impixel(a,200,100)
```

1.13. Obrazy kolorowe indeksowane

Polecenie:

```
figure, imshow('emu.tif'), pixval on
```

wyświetla kolorowy obraz emu. Wartości pikseli nie są jednak trzema liczbami całkowitymi, jak w obrazie RGB, lecz trzema ułamkami z zakresu od 0 do 1.

Jeżeli zapiszemy obraz tylko do jednej macierzy:

```
em = imread('emu.tif');

figure, imshow(em), pixval on
```

otrzymamy ciemny, trudny do rozpoznania obraz. Oznacza to, że macierz `em` zawiera tylko indeksy do mapy kolorów. Aby poprawnie wczytać obraz indeksowany, trzeba wczytać także mapę kolorów:

```
[em, emap] = imread('emu.tif');

figure, imshow(em, emap), pixval on
```

Matlab przechowuje wartości RGB mapy kolorów obrazu indeksowanego jako wartości typu `double` z zakresu od 0 do 1.

Informacje o obrazie

Funkcja `imfinfo` zwraca wiele informacji o obrazie. Na przykład:

```
imfinfo('emu.tif')
```

może zwrócić nazwę pliku, datę modyfikacji, rozmiar pliku, format, szerokość, wysokość, głębię bitową, typ koloru oraz mapę kolorów. Poniżej przykład dla imfinfo("ngc6543a.jpg"):

```
            Filename: 'C:\Program Files\MATLAB\R2025a\toolbox\matlab\demos\ngc6543a.jpg'
         FileModDate: '01-Oct-1996 22:19:44'
            FileSize: 27387
              Format: 'jpg'
       FormatVersion: ''
               Width: 600
              Height: 650
            BitDepth: 24
           ColorType: 'truecolor'
     FormatSignature: ''
     NumberOfSamples: 3
        CodingMethod: 'Huffman'
      CodingProcess: 'Sequential'
             Comment: {'CREATOR: XV Version 3.00b  Rev: 6/15/94  Quality = 75, Smoothing = 0↵'}
  AutoOrientedWidth: 600
 AutoOrientedHeight: 650
```

Dla obrazu true colour `imfinfo` pokazuje na przykład `BitDepth: 24` i `ColorType: truecolor`. Dla obrazu binarnego Matlab może podać `ColorType: grayscale`, ponieważ obraz binarny traktuje jako szczególny przypadek obrazu w skali szarości z tylko dwoma intensywnościami. O tym, że obraz jest binarny, świadczy głębia bitowa równa 1.

1.14. Typy danych i konwersje

Elementy macierzy w Matlabie mogą mieć różne typy numeryczne. Najczęściej używane typy to:

Typ danych	Opis	Zakres
`int8`	8-bitowa liczba całkowita	od -128 do 127
`uint8`	8-bitowa liczba całkowita bez znaku	od 0 do 255
`int16`	16-bitowa liczba całkowita	od -32768 do 32767
`uint16`	16-bitowa liczba całkowita bez znaku	od 0 do 65535
`double`	liczba rzeczywista podwójnej precyzji	zależny od maszyny

Dane możemy konwertować na inny typ, np.:

```
a = 23;

b = uint8(a);

whos a b
```

Zmienne `a` i `b` mają tę samą wartość liczbową, lecz różne typy danych.

Obraz w skali szarości może mieć piksele typu `uint8`. Jest to oszczędne pamięciowo, ponieważ każdy piksel zajmuje jeden bajt. Operacje arytmetyczne nie zawsze są jednak dozwolone bez konwersji. Przed obliczeniami obraz `uint8` często trzeba przekształcić do typu `double`.

Matlab udostępnia funkcje konwersji między typami obrazów:

Funkcja	Zastosowanie	Format
`ind2gray`	indeksowany na skala szarości	`y = ind2gray(x,map);`
`gray2ind`	skala szarości na indeksowany	`[y,map] = gray2ind(x);`
`rgb2gray`	RGB na skala szarości	`y = rgb2gray(x);`
`gray2rgb`	skala szarości na RGB	`y = gray2rgb(x);`
`rgb2ind`	RGB na indeksowany	`[y,map] = rgb2ind(...);`
`ind2rgb`	indeksowany na RGB	`y = ind2rgb(x,map);`

Funkcja `gray2rgb` nie tworzy nowej informacji kolorystycznej. Tworzy obraz RGB, w którym składowe czerwone, zielone i niebieskie są równe wartościom szarości.

1.15. Podstawy wyświetlania obrazów

Obraz można reprezentować jako macierz wartości szarości. Problem polega na tym, jak taką macierz wyświetlić na ekranie. Na wygląd obrazu wpływają między innymi:

1. oświetlenie otoczenia,
2. typ i ustawienia monitora,
3. karta graficzna,
4. rozdzielczość monitora.

Ten sam obraz może wyglądać bardzo różnie na różnych monitorach. Wysoka rozdzielczość może sprawić, że obraz zajmie mniejszy obszar fizyczny na ekranie. Jasne światło otoczenia może utrudnić oglądanie. Również indywidualny system wzrokowy człowieka wpływa na odbiór obrazu.

Podstawową funkcją Matlaba do wyświetlania obrazu jest `image`. Funkcja ta wyświetla macierz jako obraz, ale nie zawsze daje właściwy efekt. Na przykład:

```
c = imread('cameraman.tif');

image(c)
```

może wyświetlić obraz w dziwnych kolorach. Wynika to z faktu, że `image` korzysta z aktualnej mapy kolorów. Domyślna mapa `jet` zawiera jaskrawe kolory i nie nadaje się do wyświetlania obrazu w skali szarości.

Aby poprawnie wyświetlić obraz, można użyć:

```
image(c), truesize, axis off, colormap(gray(247))
```

Polecenie `truesize` wyświetla jeden piksel obrazu jako jeden piksel ekranu. `axis off` wyłącza osie, a `colormap(gray(247))` ustawia skalę szarości z odpowiednią liczbą poziomów.

Dla obrazów indeksowanych trzeba wczytać mapę kolorów:

```
[x,map] = imread('cat.tif');

image(x), truesize, axis off, colormap(map)
```

Dla obrazów true colour funkcja `image` ignoruje aktualną mapę kolorów i używa wartości RGB z tablicy.

1.16. Funkcja `imshow`

Obrazy w skali szarości

Jeżeli `x` jest macierzą typu `uint8`, polecenie:

```
imshow(x)
```

wyświetla `x` jako obraz. Jest to naturalne, ponieważ wartości `uint8` należą do zakresu od 0 do 255.

Wiele funkcji przetwarzania obrazów zwraca jednak macierze typu `double`. Dla takiej macierzy są dwie możliwości:

1. przekształcić ją do typu `uint8` i wyświetlić,
2. wyświetlić bezpośrednio jako macierz typu `double`.

`imshow` wyświetla macierz typu `double` jako obraz w skali szarości tylko wtedy, gdy wartości znajdują się w zakresie od 0 do 1. Wartość 0 oznacza czerń, a 1 biel. Wartości większe od 1 są wyświetlane jako biel, a mniejsze od 0 jako czerń.

Przykład:

```
c = imread('caribou.tif');

cd = double(c);

imshow(c), figure, imshow(cd)
```

Drugi obraz będzie prawie biały, ponieważ wartości typu `double` nie zostały przeskalowane do zakresu od 0 do 1.

(a) The original image (b) After conversion to type `double`

Figure 1.24: An attempt at data type conversion

Aby wyświetlić go poprawnie, trzeba podzielić przez 255:

```
imshow(cd/255)
```

Można też użyć funkcji `im2double`, która wykonuje poprawne skalowanie:

```
cd = im2double(c)
```

```
imshow(cd)
```

Różnica między `double` i `im2double` jest ważna. `double` zmienia typ danych, ale nie zmienia wartości liczbowych. `im2double` zmienia zarówno typ, jak i skalę wartości.

Analogicznie istnieją funkcje `uint8` i `im2uint8`. Zaleca się używanie `im2uint8`, ponieważ zwraca poprawny wynik dla różnych typów wejściowych.

Obrazy binarne

Obraz binarny ma tylko dwie wartości: 0 i 1. Matlab nie ma osobnego typu „binary image”, ale może interpretować macierz jako logiczną. Jeśli utworzymy macierz:

```
cl = c > 120;
```

to `cl` będzie macierzą logiczną i polecenie:

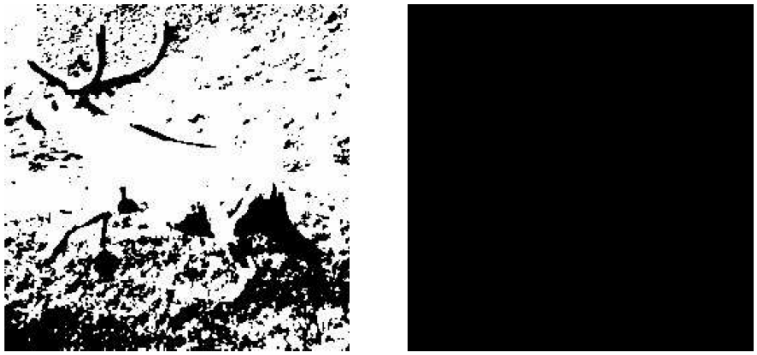
```
imshow(cl)
```

wyświetli ją jako obraz binarny.

Jeśli jednak usuniemy flagę logiczną:

```
cl = +cl;
```

obraz może wyglądać prawie całkowicie na czarny, ponieważ w typie `uint8` wartość 1 jest bardzo ciemną szarością, a nie bielą.



(a) The caribou image turned binary (b) After conversion to type `uint8`

Figure 1.26: Making the image binary

Aby ponownie poprawnie wyświetlić obraz, można użyć:

```
imshow(logical(cl))
```

albo:

```
imshow(double(cl))
```

1.17. Płaszczyzny bitowe

Obrazy w skali szarości można przekształcić w sekwencję obrazów binarnych przez rozbicie ich na płaszczyzny bitowe. W 8-bitowym obrazie wartość piksela można traktować jako 8-bitowe słowo binarne.

Najmniej znaczący bit tworzy płaszczyznę najmniej znaczącego bitu. Ma on najmniejszy wpływ na wartość piksela. Najbardziej znaczący bit tworzy płaszczyznę najbardziej znaczącego bitu i ma największy wpływ na wartość piksela.

Aby wyodrębnić płaszczyzny bitowe w Matlabie, można użyć:

```
c = imread('cameraman.tif');
cd = double(c);
c0 = mod(cd,2);
c1 = mod(floor(cd/2),2);
c2 = mod(floor(cd/4),2);
c3 = mod(floor(cd/8),2);
c4 = mod(floor(cd/16),2);
c5 = mod(floor(cd/32),2);
c6 = mod(floor(cd/64),2);
c7 = mod(floor(cd/128),2);
```

Najmniej znacząca płaszczyzna bitowa wygląda zazwyczaj jak losowy szum. Wraz ze wzrostem numeru płaszczyzny coraz wyraźniej pojawia się struktura obrazu. Najbardziej znacząca płaszczyzna bitowa jest równoważna progowaniu obrazu na poziomie 127:

```
ct = c > 127;
```

```
all(c7(:) == ct(:))
```

Wynik 1 oznacza, że warunek jest prawdziwy dla wszystkich pikseli.

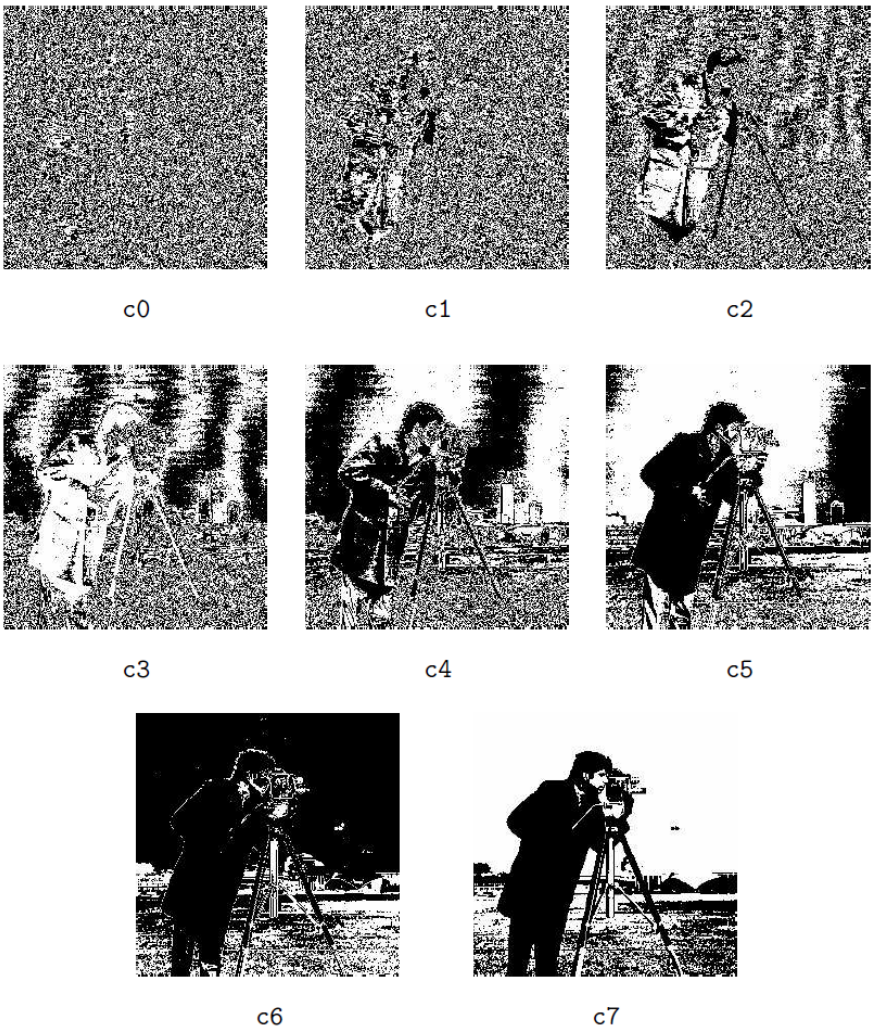


Figure 1.27: The bit planes of an 8-bit greyscale image

Oryginalny obraz można odzyskać z płaszczyzn bitowych:

```
cc = 2*(2*(2*(2*(2*(2*(2*c7+c6)+c5)+c4)+c3)+c2)+c1)+c0;

imshow(uint8(cc))
```

1.18. Rozdzielczość przestrzenna

Rozdzielczość przestrzenna oznacza gęstość pikseli w obrazie. Im większa rozdzielczość przestrzenna, tym więcej pikseli używa się do wyświetlenia obrazu.

W Matlabie można eksperymentować z rozdzielczością za pomocą funkcji `imresize`. Jeśli obraz 256×256 zapisany jest w macierzy x, polecenie:

```
imresize(x,1/2)
```

zmniejsza obraz dwukrotnie w każdym kierunku. Można potem ponownie powiększyć obraz:

```
x2 = imresize(imresize(x,1/2),2);
```

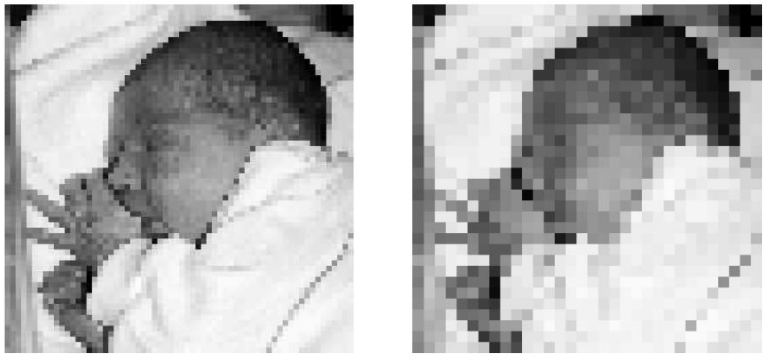
Obraz będzie miał pierwotny rozmiar, lecz efektywna rozdzielczość będzie mniejsza. Można zmieniać parametry, aby otrzymać rozdzielczości efektywne:

Polecenie	Efektywna rozdzielczość
`imresize(imresize(x,1/4),4)`	64×64
`imresize(imresize(x,1/8),8)`	32×32
`imresize(imresize(x,1/16),16)`	16×16
`imresize(imresize(x,1/32),32)`	8×8

Gdy rozdzielczość maleje, pojawia się blokowość i pikselizacja. Przy umiarkowanym zmniejszeniu szczegóły stają się mniej wyraźne. Przy dużym zmniejszeniu obraz przestaje być rozpoznawalny.

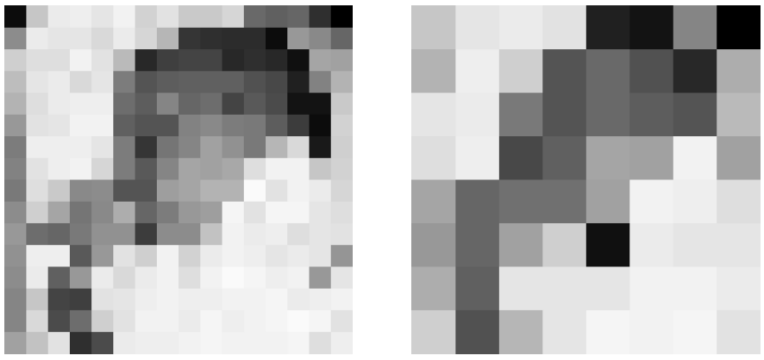
(a) The original image (b) at 128×128 resolution

Figure 1.28: Reducing resolution of an image



(a) At 64×64 resolution (b) At 32×32 resolution

Figure 1.29: Further reducing the resolution of an image



(a) At 16×16 resolution (b) at 8×8 resolution

Figure 1.30: Even more reducing the resolution of an image

Rozdział 2. Przetwarzanie punktowe

2.1 Wprowadzenie

Każda operacja przetwarzania obrazu przekształca wartości szarości pikseli. Jednak operacje przetwarzania obrazu można podzielić na trzy klasy, zależnie od informacji wymaganej do wykonania przekształcenia. Od najbardziej złożonych do najprostszych są to:

3. **Transformaty.** "Transformata" przedstawia wartości pikseli w pewnej innej, lecz równoważnej postaci. Transformaty pozwalają na stosowanie bardzo wydajnych i silnych algorytmów, jak zobaczymy później. Możemy uważać, że przy użyciu transformaty cały obraz jest przetwarzany jako jeden duży blok. Można to zilustrować schematem pokazanym na rysunku 2.1.

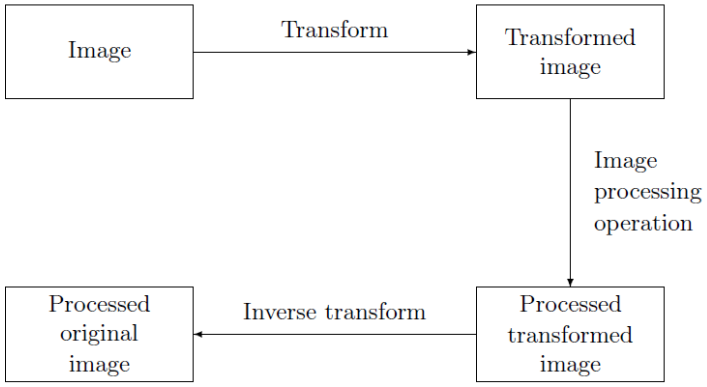


Figure 2.1: Schema for transform processing

4. **Przetwarzanie sąsiedztwa.** Aby zmienić poziom szarości danego piksela, musimy znać jedynie wartości poziomów szarości w małym sąsiedztwie pikseli otaczających dany piksel.

5. **Operacje punktowe.** Wartość szarości piksela jest zmieniana bez żadnej wiedzy o jego otoczeniu.

Chociaż operacje punktowe są najprostsze, zawierają jedne z najpotężniejszych i najpowszechniej używanych operacji przetwarzania obrazów. Są one szczególnie użyteczne we wstępnym przetwarzaniu obrazu, gdy obraz musi zostać zmodyfikowany przed wykonaniem głównego zadania.

2.2 Operacje arytmetyczne

Operacje te działają przez zastosowanie prostej funkcji

$$y = T(x)$$

opisywanej dla każdej wartości szarości w obrazie. Zatem $T(x)$ jest funkcją, która odwzorowuje zakres $0, \dots, 255$ na siebie.

Do prostych funkcji należy dodawanie stałej wartości do każdego piksela lub odejmowanie jej od każdego piksela:

$$y = x + c$$

albo mnożenie każdego piksela przez stałą:

$$y = cx.$$

W każdym przypadku możemy musieć nieznacznie poprawić wynik, aby upewnić się, że rezultaty są liczbami całkowitymi z zakresu $0, \dots, 255$. Możemy to zrobić przez najpierw zaokrąglenie wyniku, jeśli jest to konieczne, aby otrzymać liczbę całkowitą, a następnie przez "obcięcie" wartości:

$$y' = \begin{cases} 255, & \text{jeśli } y > 255, \\ 0, & \text{jeśli } y < 0, \\ round(y), & \text{w przeciwnym razie.} \end{cases}$$

Możemy zrozumieć, jak operacje te wpływają na obraz, wykreślając $y = T(x)$. Rysunek 2.2 pokazuje wynik dodania 128 do każdego piksela obrazu albo odjęcia 128 od każdego piksela obrazu. Zauważmy, że gdy dodajemy 128, wszystkie wartości szarości równe 127 lub większe zostaną odwzorowane na 255. A gdy odejmujemy 128, wszystkie wartości szarości równe 128 lub mniejsze zostaną odwzorowane na 0. Patrząc na te wykresy, obserwujemy, że ogólnie dodanie stałej rozjaśnia obraz, a odjęcie stałej go przyciemnia.

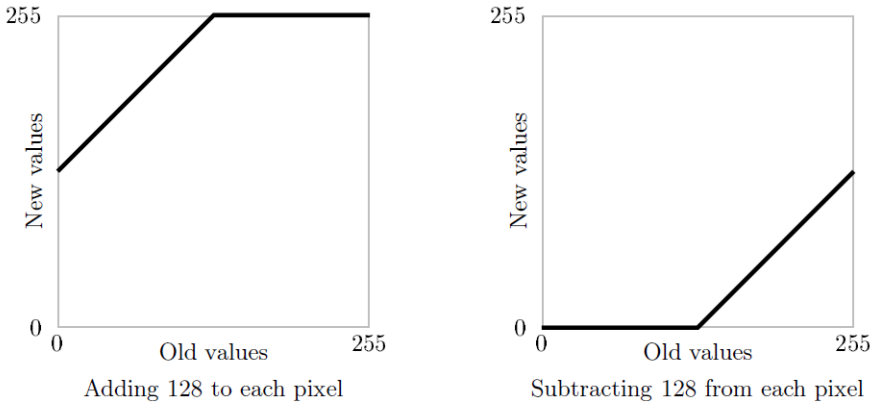


Figure 2.2: Adding and subtracting a constant

Możemy przetestować to na obrazie `blocks.tif`, który widzieliśmy na rysunku 1.4. Zaczynamy od wczytania obrazu:

```
>> b=imread('blocks.tif');
>> whos b
Name      Size          Bytes  Class
b         256x256       65536  uint8 array
```

Celem drugiego polecenia było znalezienie typu danych numerycznych macierzy b; jest to `uint8`. Typ danych `uint8` jest używany tylko do przechowywania danych; nie możemy wykonywać na nim operacji arytmetycznych. Jeśli spróbujemy, otrzymamy jedynie komunikat o błędzie:

```
>> b1=b+128
??? Error using ==> +
Function '+' not defined for variables of class 'uint8'.
```

Możemy obejść to na dwa sposoby. Możemy najpierw zamienić b na macierz typu `double`, dodać 128, a następnie zamienić wynik z powrotem na `uint8`, aby go wyświetlić:

```
>> b1=uint8(double(b)+128);
```

Drugim, bardziej eleganckim sposobem, jest użycie funkcji Matlaba `imadd`, która jest zaprojektowana dokładnie do tego celu:

```
>> b1=imadd(b,128);
```

Odejmowanie jest podobne; możemy przekształcić naszą macierz do typu `double` i z powrotem albo użyć funkcji `imsubtract`:

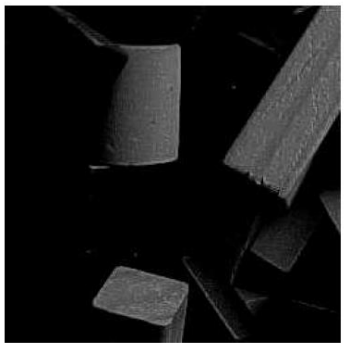
```
>> b2=imsubtract(b,128);
```

Teraz możemy je obejrzeć:

```
>> imshow(b1),figure,imshow(b2)
```

a wyniki widać na rysunku 2.3.

b1: Adding 128



b2: Subtracting 128

Figure 2.3: Arithmetic operations on an image: adding or subtracting a constant

Możemy także rozjaśnić lub przyciemnić obraz przez mnożenie; rysunek 2.4 pokazuje kilka przykładów funkcji, które będą miały takie efekty.

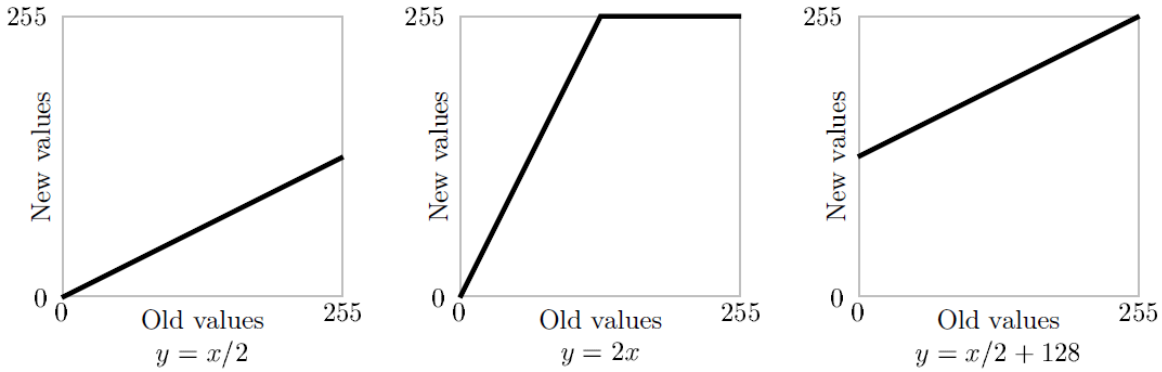


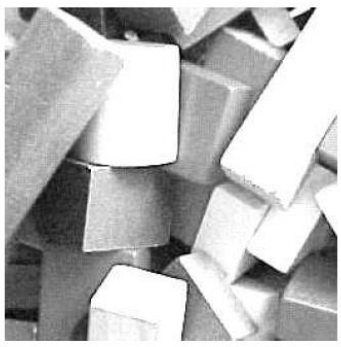
Figure 2.4: Using multiplication and division

Aby zaimplementować te funkcje, używamy funkcji `immultiply`. Tabela 2.1 pokazuje konkretne polecenia wymagane do zaimplementowania funkcji z rysunku 2.4. Wszystkie te obrazy można obejrzeć za pomocą `imshow`; pokazano je na rysunku 2.5.

$y = x/2$	`b3=immultiply(b,0.5);` or `b3=imdivide(b,2)`
$y = 2x$	`b4=immultiply(b,2);`
$y = x/2 + 128$	`b5=imadd(immultiply(b,0.5),128);` or `b5=imadd(imdivide(b,2),128);`

Table 2.1: Implementing pixel multiplication by MATLAB commands

b3: $y = x/2$



b4: $y = 2x$

b5: $y = x/2 + 128$

Figure 2.5: Arithmetic operations on an image: multiplication and division

Porównaj wyniki przyciemniania b2 i b3. Zauważ, że b3, chociaż ciemniejszy niż oryginał, jest nadal całkiem wyraźny, podczas gdy wiele informacji zostało utraconych w procesie odejmowania, co można zobaczyć w obrazie b2. Dzieje się tak, ponieważ w obrazie b2 wszystkie piksele o wartościach szarości 128 lub mniejszych stały się zerami.

Podobna utrata informacji wystąpiła w obrazach b1 i b4. Zauważ szczególnie krawędzie jasnego bloku w dolnej części pośrodku; w obrazach b1 i b4 prawa krawędź zniknęła. Jednak krawędź ta jest wyraźnie widoczna w obrazie b5.

Dopełnienia

Dopełnienie obrazu w skali szarości jest jego negatywem fotograficznym. Jeśli macierz obrazu m jest typu double, a więc jej wartości szarości leżą w zakresie od 0 do 1, możemy otrzymać jej negatyw poleceniem

```
>> 1-m
```

Jeśli obraz jest binarny, możemy użyć

```
>> ~m
```

Jeśli obraz jest typu uint8, najlepszym podejściem jest funkcja imcomplement. Rysunek 2.6 pokazuje funkcję dopełnienia

$$s = 255 - r,$$

oraz wynik poleceń

```
>> bc=imcomplement(b);
>> imshow(bc)
```

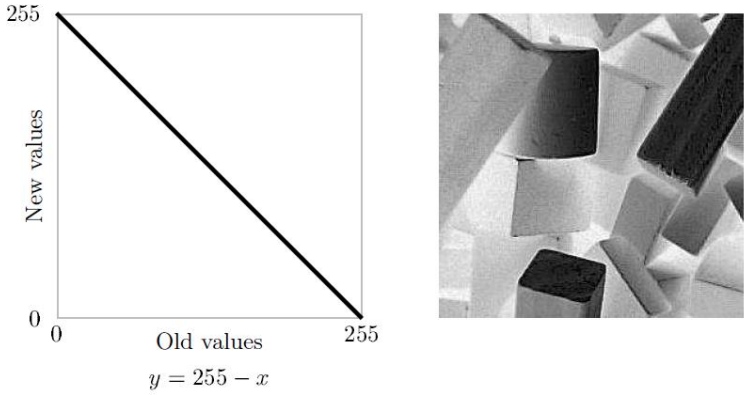


Figure 2.6: Image complementation

Interesujące efekty specjalne można uzyskać przez dopełnienie tylko części obrazu; na przykład przez wzięcie dopełnienia pikseli o wartości szarości 128 lub mniejszej i pozostawienie pozostałych pikseli bez zmian. Albo moglibyśmy wziąć dopełnienie pikseli, które są równe 128 lub większe, i pozostawić inne piksele bez zmian. Rysunek 2.7 pokazuje te funkcje. Efekt tych funkcji nazywa się **solaryzacją**.

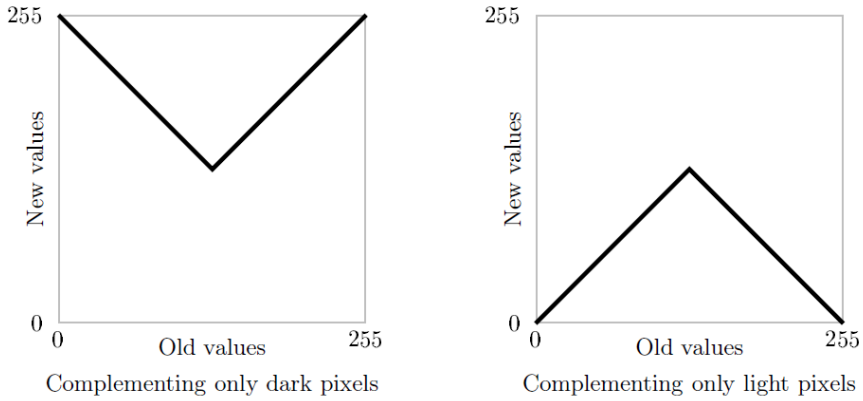


Figure 2.7: Part complementation

2.3 Histogramy

Dla danego obrazu w skali szarości jego histogram składa się z histogramu jego poziomów szarości; to znaczy z wykresu wskazującego, ile razy każdy poziom szarości występuje w obrazie. Z histogramu możemy wywnioskować bardzo wiele o wyglądzie obrazu, jak wskazują następujące przykłady:

- W ciemnym obrazie poziomy szarości, a więc i histogram, byłyby skupione przy dolnym końcu zakresu.
- W jednolicie jasnym obrazie poziomy szarości byłyby skupione przy górnym końcu zakresu.
- W obrazie o dobrym kontraście poziomy szarości byłyby dobrze rozłożone na dużej części zakresu.

Histogram obrazu możemy obejrzeć w Matlabie za pomocą funkcji `imhist`:

```
>> p=imread('pout.tif');
>> imshow(p),figure,imhist(p),axis tight
```

Polecenie `axis tight` zapewnia, że osie histogramu zostaną automatycznie przeskalowane tak, aby obejmować wszystkie wartości. Wynik pokazano na rysunku 2.8. Ponieważ wartości szarości są wszystkie skupione razem w środku histogramu, oczekiwalibyśmy, że obraz będzie miał słaby kontrast - i rzeczywiście tak jest.

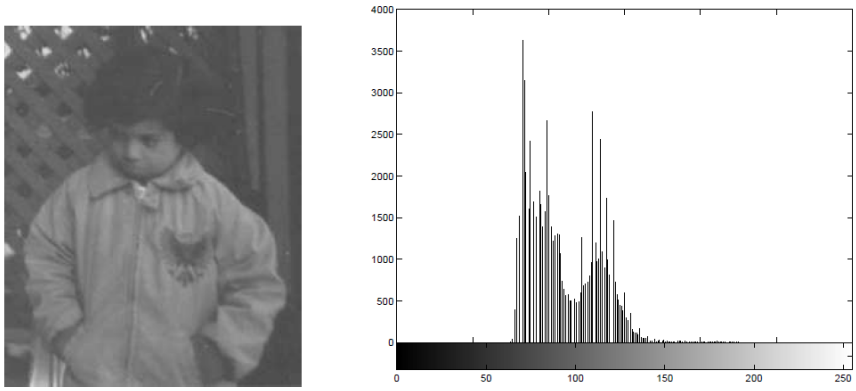


Figure 2.8: The image `pout.tif` and its histogram

Mając obraz o słabym kontraście, chcielibyśmy poprawić jego kontrast przez rozciągnięcie histogramu. Są dwa sposoby, aby to zrobić.

2.3.1 Rozciąganie histogramu (rozciąganie kontrastu)

Przypuśćmy, że mamy obraz z histogramem pokazanym na rysunku 2.9, związanym z tabelą liczby n_i wartości szarości:

Grey level i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n_i	15	0	0	0	0	70	110	45	70	35	0	0	0	0	0	15

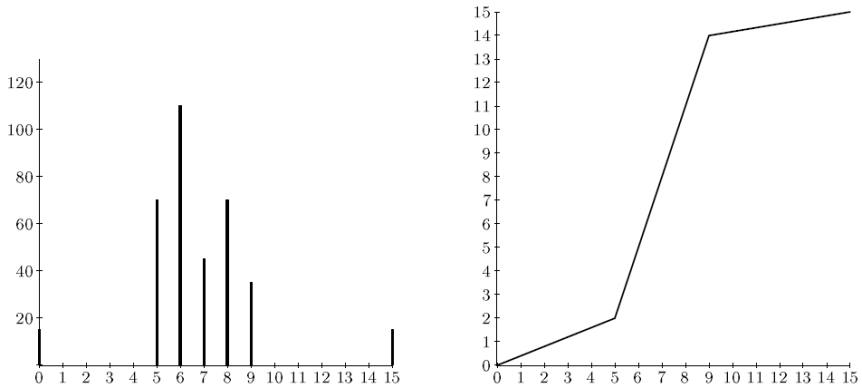


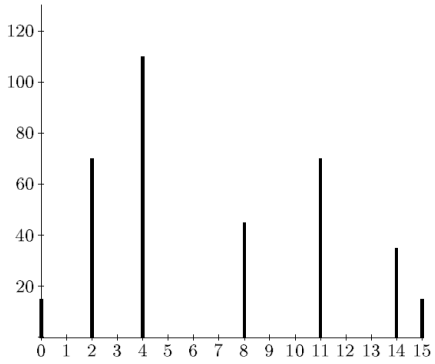
Figure 2.9: A histogram of a poorly contrasted image, and a stretching function

Możemy rozciągnąć poziomy szarości w środku zakresu, stosując funkcję liniową odcinkami pokazaną po prawej stronie rysunku 2.9. Funkcja ta rozciąga poziomy szarości od 5 do 9 na poziomy od 2 do 14 zgodnie z równaniem

$$j = \frac{14-2}{9-5}(i-5)+2.$$

Tutaj i jest oryginalnym poziomem szarości, a j jest wynikiem po transformacji. Poziomy szarości poza tym zakresem są albo pozostawiane bez zmian, jak w tym przypadku, albo przekształcane zgodnie z funkcjami liniowymi na końcach wykresu. Daje to odpowiadający histogram, który wskazuje obraz o większym kontraście niż oryginalny:

i	5	6	7	8	9
j	2	5	8	11	14



Użycie imadjust

Aby wykonać rozciąganie histogramu w Matlabie, można użyć funkcji `imadjust`. W najprostszej postaci polecenie

```
imadjust(im,[a,b],[c,d])
```

rozciąga obraz zgodnie z funkcją pokazaną na rysunku 2.10.

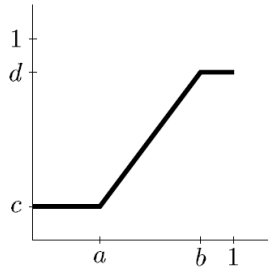


Figure 2.10: The stretching function given by `imadjust`

Ponieważ `imadjust` jest zaprojektowana tak, aby działać równie dobrze na obrazach typu `double`, `uint8` albo `uint16`, wartości a, b, c i d muszą leżeć między 0 a 1; funkcja automatycznie konwertuje obraz, jeśli trzeba, do typu `double`.

Zauważ, że `imadjust` nie działa dokładnie w taki sam sposób jak pokazano na rysunku 2.9. Wartości pikseli mniejsze niż a są wszystkie zamieniane na c, a wartości pikseli większe niż b są wszystkie zamieniane na d. Jeśli którykolwiek z przedziałów `[a,b]` albo `[c,d]` zostanie wybrany jako `[0,1]`, można użyć skrótu `[]`. Tak więc na przykład polecenie

```
>> imadjust(im,[],[])
```

nic nie robi, a polecenie

```
>> imadjust(im,[],[1,0])
```

odwraca wartości szarości obrazu, dając wynik podobny do negatywu fotograficznego.

Funkcja `imadjust` ma jeszcze jeden opcjonalny parametr: wartość gamma, która opisuje kształt funkcji między współrzędnymi (a, c) i (b, d). Jeśli gamma jest równa 1, co jest wartością domyślną, używane jest odwzorowanie liniowe, jak pokazano wcześniej na rysunku 2.10. Jednak wartości mniejsze niż jeden dają funkcję wklęsłą ku dołowi, jak pokazano po lewej stronie rysunku 2.11, a wartości większe niż jeden dają funkcję wklęsłą ku górze, jak pokazano po prawej stronie rysunku 2.11.

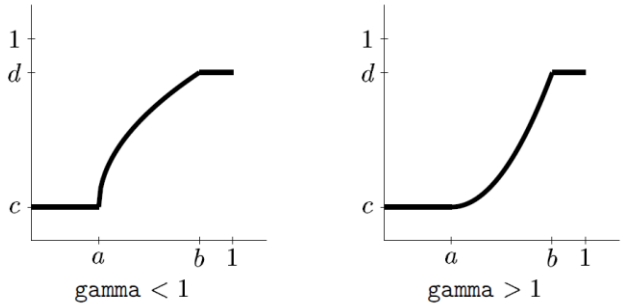


Figure 2.11: The `imadjust` function with `gamma` not equal to 1

Używana funkcja przekształcająca:

$$y = c + (d - c)\left(\frac{x - a}{b - a}\right)^{\gamma}.$$

Samo użycie wartości gamma może wystarczyć do znacznej zmiany wyglądu obrazu. Na przykład:

```
>> t=imread('tire.tif');
>> th=imadjust(t,[],[],0.5);
>> imshow(t),figure,imshow(th)
```

daje wynik pokazany na rysunku 2.12.

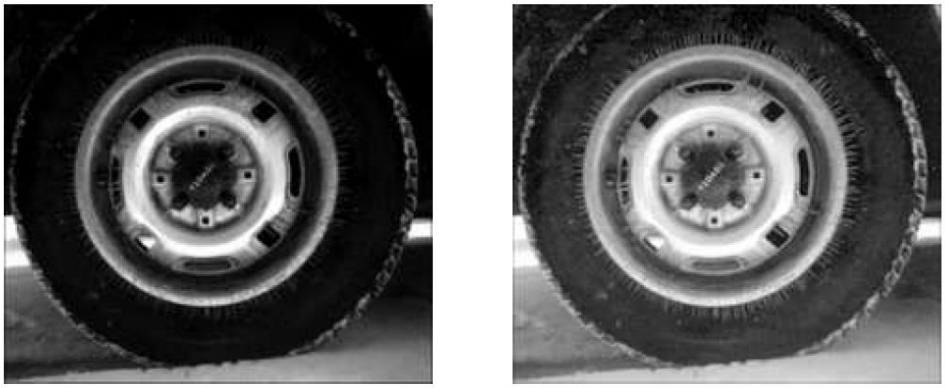


Figure 2.12: The tire image and after adjustment with the `gamma` value

Funkcję rozciągającą `imadjust` możemy obejrzeć za pomocą funkcji `plot`. Na przykład

```
>> plot(t,th,'.'),axis tight
```

tworzy wykres pokazany na rysunku 2.13.

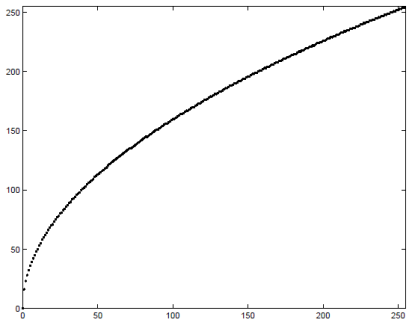


Figure 2.13: The function used in figure 2.12

Ponieważ `t` i `th` są macierzami, które zawierają wartości oryginalne i wartości po zastosowaniu funkcji `imadjust`, funkcja `plot` po prostu wykreśla je, używając do tego kropek.

Funkcja rozciągania liniowego odcinkami

Możemy łatwo napisać własną funkcję wykonującą rozciąganie liniowe odcinkami, jak pokazano na rysunku 2.14.

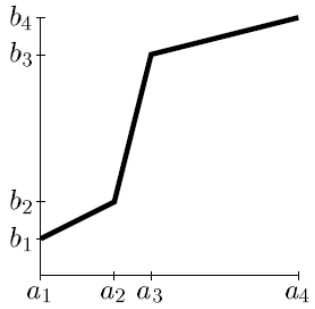


Figure 2.14: A piecewise linear stretching function

W tym celu skorzystamy z funkcji `find`, aby znaleźć wartości pikseli w obrazie między a_i i a_{i+1}. Ponieważ prosta między współrzędnymi (a_i, b_i) i (a_{i+1}, b_{i+1}) ma równanie

$$y = \frac{b_{i+1} - b_i}{a_{i+1} - a_i}(x - a_i) + b_i,$$

sercem naszej funkcji będą linie

```
pix=find(im >= a(i) & im < a(i+1));
out(pix)=(im(pix)-a(i))*(b(i+1)-b(i))/(a(i+1)-a(i))+b(i);
```

gdzie `im` jest obrazem wejściowym, a `out` obrazem wyjściowym. Prosta procedura, która przyjmuje jako wejście obrazy typu `uint8` albo `double`, jest pokazana na rysunku 2.15.

```
function out = histpwl(im,a,b)
%
% HISTPWL(IM,A,B) applies a piecewise linear transformation to the pixel values
% of image IM, where A and B are vectors containing the x and y coordinates
% of the ends of the line segments.  IM can be of type UINT8 or DOUBLE,
% and the values in A and B must be between 0 and 1.
%
% For example:
%
%   histpwl(x,[0,1],[1,0])
%
% simply inverts the pixel values.
%
classChanged = 0;
if ~isa(im, 'double'),
    classChanged = 1;
    im = im2double(im);
end

if length(a) ~= length (b)
  error('Vectors A and B must be of equal size');
end

N=length(a);
out=zeros(size(im));

for i=1:N-1
  pix=find(im>=a(i) & im<a(i+1));
  out(pix)=(im(pix)-a(i))*(b(i+1)-b(i))/(a(i+1)-a(i))+b(i);
end

pix=find(im==a(N));
out(pix)=b(N);

if classChanged==1
  out = uint8(255*out);
end
```

Figure 2.15: A MATLAB function for applying a piecewise linear stretching function

Jako przykład użycia tej funkcji:

```
>> th=histpwl(t,[0 .25 .5 .75 1],[0 .75 .25 .5 1]);
>> imshow(th)
>> figure,plot(t,th,'.'),axis tight
```

daje rysunki pokazane na rysunku 2.16.

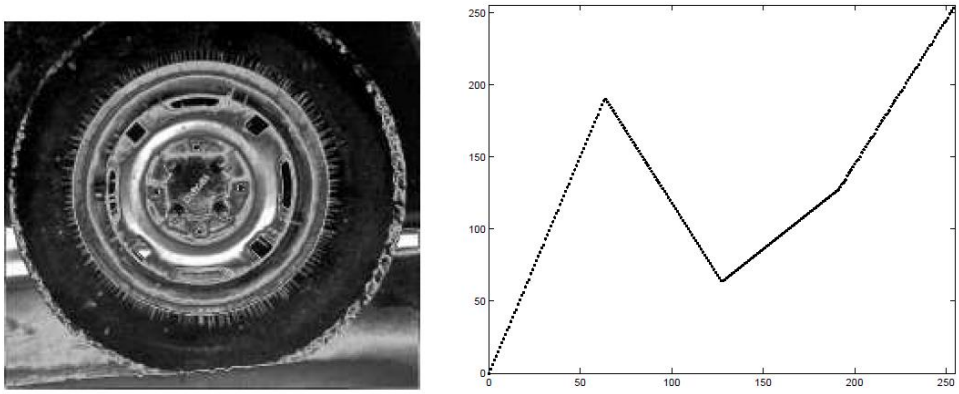


Figure 2.16: The tire image and after adjustment with the gamma value

2.3.2 Wyrównywanie histogramu

Problem ze wszystkimi powyższymi metodami rozciągania histogramu polega na tym, że wymagają one danych wejściowych od użytkownika. Czasami lepsze podejście zapewnia wyrównywanie histogramu, które jest procedurą całkowicie automatyczną. Pomysł polega na zmianie histogramu na taki, który jest jednostajny; to znaczy taki, że każdy słupek histogramu ma tę samą wysokość, albo innymi słowy, że każdy poziom szarości w obrazie występuje z taką samą częstością. W praktyce nie jest to na ogół możliwe, chociaż, jak zobaczymy, wynik wyrównywania histogramu daje bardzo dobre rezultaty.

Przypuśćmy, że nasz obraz ma L różnych poziomów szarości $0,1,\ldots,L-1$ i że poziom szarości i występuje w obrazie n_i razy. Przypuśćmy także, że całkowita liczba pikseli w obrazie wynosi N, tak że

$$n_0 + n_1 + \cdots + n_{L-1} = N.$$

Aby przekształcić poziomy szarości i uzyskać obraz o lepszym kontraście, zmieniamy poziom szarości i na

$$\text{round}\left((L-1)\frac{n_0 + n_1 + \cdots + n_i}{N}\right).$$

Przykład

Przypuśćmy, że 4-bitowy obraz w skali szarości ma histogram pokazany na rysunku 2.17, związany z tabelą liczby n_i wartości szarości:

Grey level i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n_i	15	0	0	0	0	0	0	0	0	70	110	45	80	40	0	0

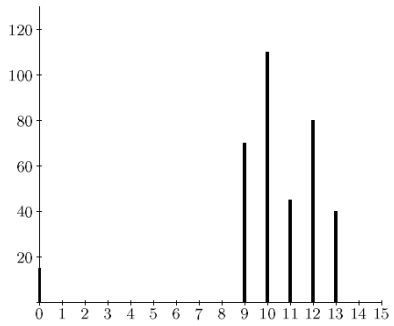


Figure 2.17: Another histogram indicating poor contrast

Oczekiwalibyśmy, że obraz ten będzie jednolicie jasny, z kilkoma ciemnymi punktami. Aby wyrównać ten histogram, tworzymy sumy narastające wartości n_i i mnożymy każdą przez

$$\frac{L-1}{N} = \frac{15}{360} = \frac{1}{24}$$

Grey level i	n_i	Σn_i	$(1/24)\Sigma n_i$	Rounded value
0	15	15	0.63	1
1	0	15	0.63	1
2	0	15	0.63	1
3	0	15	0.63	1
4	0	15	0.63	1
5	0	15	0.63	1
6	0	15	0.63	1
7	0	15	0.63	1
8	0	15	0.63	1
9	70	85	3.65	4
10	110	195	8.13	8
11	45	240	10	10
12	80	320	13.33	13
13	40	360	15	15
14	0	360	15	15
15	0	360	15	15

Otrzymujemy transformację wartości szarości przez odczytanie pierwszej i ostatniej kolumny odpowiedniej tabeli. Histogram wartości po transformacji pokazano na rysunku 2.18. Jest on znacznie bardziej rozciągnięty niż oryginalny histogram, więc obraz wynikowy powinien wykazywać większy kontrast.

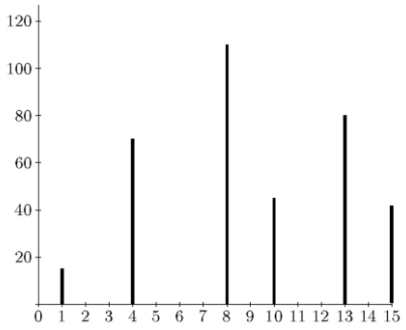


Figure 2.18: The histogram of figure 2.17 after equalization

Aby zastosować wyrównywanie histogramu w Matlabie, użyj funkcji `histeq`; na przykład:

```
>> p=imread('pout.tif');
>> ph=histeq(p);
>> imshow(ph),figure,imhist(ph),axis tight
```

stosuje wyrównywanie histogramu do obrazu `pout` i tworzy wynikowy histogram. Wyniki te pokazano na rysunku 2.19. Zauważ znacznie większe rozciągnięcie histogramu. Odpowiada to większemu wzrostowi kontrastu w obrazie.

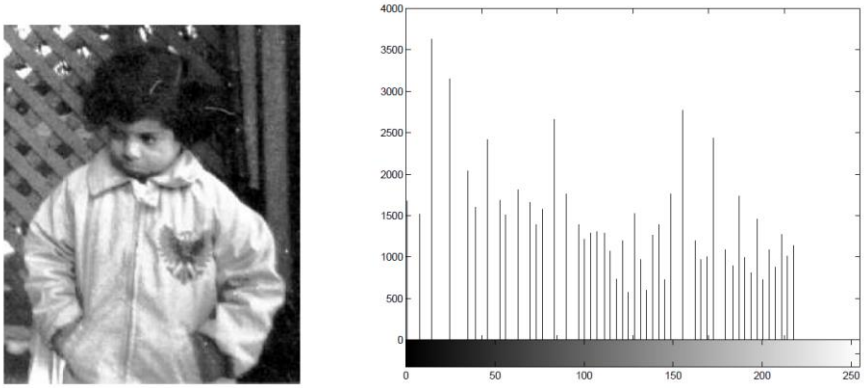


Figure 2.19: The histogram of figure 2.8 after equalization

Podamy jeszcze jeden przykład, bardzo ciemnego obrazu. Możemy otrzymać ciemny obraz przez użycie `imdivide`.

```
>> en=imread('engineer.tif');
>> e=imdivide(en,4);
```

Ponieważ macierz `e` zawiera tylko niskie wartości, będzie wyglądać bardzo ciemno po wyświetleniu. Możemy wyświetlić tę macierz i jej histogram poleceniami:

```
>> imshow(e),figure,imhist(e),axis tight
```

a wyniki pokazano na rysunku 2.20.

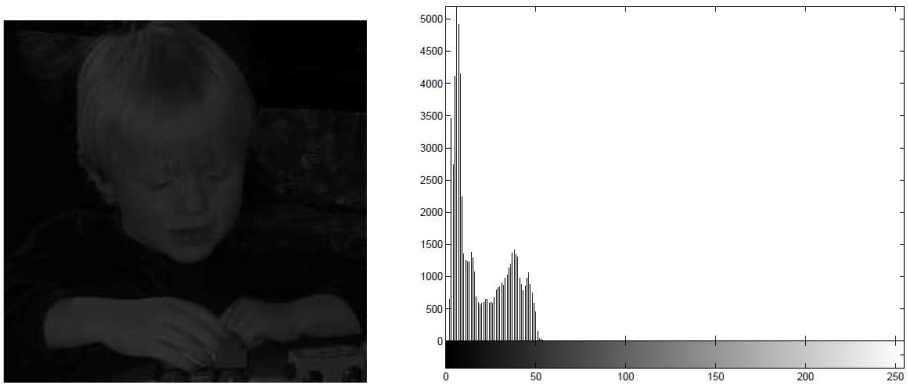


Figure 2.20: The darkened version of `engineer.tif` and its histogram

Jak widać, bardzo ciemny obraz ma odpowiadający mu histogram silnie skupiony przy dolnym końcu skali.

Możemy jednak zastosować wyrównywanie histogramu do tego obrazu i wyświetlić wyniki:

```
>> eh=histeq(e);
>> imshow(eh),figure,imhist(eh),axis tight
```

a wyniki pokazano na rysunku 2.21.

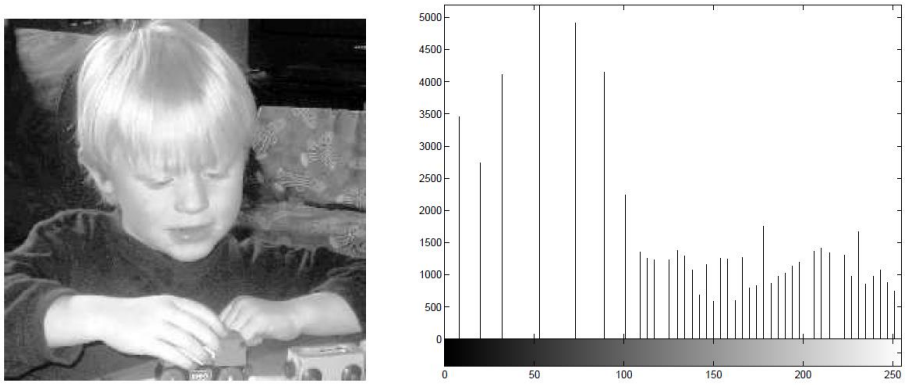


Figure 2.21: The image from 2.20 equalized and its histogram

2.4 Tablice przeglądowe

Operacje punktowe można wykonywać bardzo efektywnie przez użycie tablicy przeglądowej, znanej prościej jako LUT. Dla operacji na obrazach typu `uint8` taka tablica składa się z jednej tablicy 256 wartości, z których każda jest liczbą całkowitą z zakresu $0, \ldots, 255$. Wtedy naszą operację można zaimplementować przez zastąpienie każdej wartości piksela r odpowiadającą jej wartością $T(r)$ w tablicy.

Na przykład LUT odpowiadająca dzieleniu przez 2 wygląda następująco:

Indeks	0	1	2	3	4	…	250	251	252	253	254	255
LUT	0	0	1	1	2	…	125	125	126	126	127	127

Oznacza to na przykład, że piksel o wartości szarości 4 zostanie zastąpiony wartością szarości 2; piksel o wartości szarości 253 zostanie zastąpiony wartością szarości 126.

Jeśli T jest tablicą przeglądową w Matlabie, a `im` jest naszym obrazem, to tablicę przeglądową można zastosować prostym poleceniem

```
T(im)
```

Na przykład przypuśćmy, że chcemy zastosować powyższą tablicę przeglądową do obrazu `blocks`. Możemy utworzyć tabelę poleceniem

```
>> T=uint8(floor(0:255)/2);
```

zastosować ją do obrazu `blocks` `b` poleceniem

```
>> b2=T(b);
```

Obraz `b2` jest typu `uint8`, więc można go oglądać bezpośrednio za pomocą `imshow`.

Jako inny przykład przypuśćmy, że chcemy zastosować LUT, aby zaimplementować funkcję rozciągania kontrastu pokazaną na rysunku 2.23.

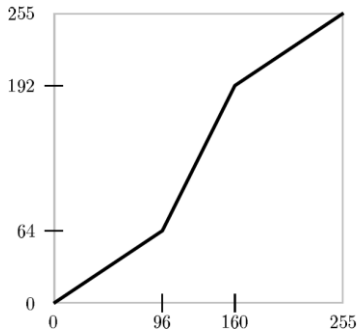


Figure 2.23: A piecewise linear contrast stretching function

Równania trzech użytych odcinków są następujące (patrz sekcja 2.3.1):

$$s = 0.6667r, \qquad dla\ 0 \leq r \leq 64,$$

$$s = 2r - 128, \qquad dla\ 65 \leq r \leq 160,$$

$$s = 0.6632r + 85.8947, \qquad dla\ 161 \leq r \leq 255.$$

Możemy następnie skonstruować LUT poleceniami:

```
>> t1=0.6667*[0:64];
>> t2=2*[65:160]-128;
>> t3=0.6632*[161:255]+85.8947;
>> T=uint8(floor([t1 t2 t3]));
```

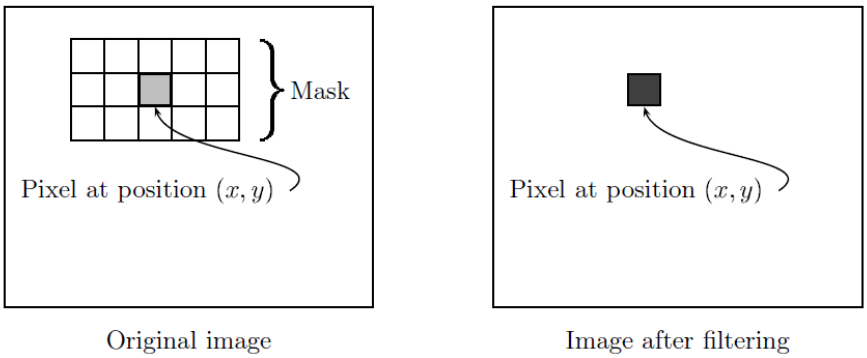
Zauważ, że polecenia dla `t1`, `t2` i `t3` są bezpośrednimi tłumaczeniami równań prostych na Matlaba, z tym wyjątkiem, że w każdym przypadku stosujemy równanie tylko na jego dziedzinie.

Rozdział 3. Przetwarzanie sąsiedztwa/Filtrowanie przestrzenne

3.1 Wprowadzenie

Widzieliśmy w rozdziale 2, że obraz można zmodyfikować przez zastosowanie określonej funkcji do każdej wartości piksela. Przetwarzanie sąsiedztwa można uważać za rozszerzenie tego podejścia, w którym funkcję stosuje się do sąsiedztwa każdego piksela.

Idea polega na przesuwaniu „maski": prostokąta, zwykle o bokach nieparzystej długości, albo innego kształtu, nad danym obrazem. Kiedy to robimy, tworzymy nowy obraz, którego piksele mają wartości szarości obliczone z wartości szarości znajdujących się pod maską, jak pokazano na rysunku 3.1. Połączenie maski i funkcji nazywa się **filtrem**. Jeżeli funkcja, za pomocą której obliczana jest nowa wartość szarości, jest funkcją liniową wszystkich wartości szarości w masce, to filtr nazywa się **filtrem liniowym**.



Rysunek 3.1: Użycie maski przestrzennej na obrazie.

Filtr liniowy można zaimplementować przez pomnożenie wszystkich elementów maski przez odpowiadające im elementy obrazu i zsumowanie wszystkich tych iloczynów. Załóżmy, że mamy maskę 3×5, taką jak zilustrowano na rysunku 3.1. Załóżmy, że wartości maski są dane przez:

$$\begin{bmatrix} a_{-1,-2} & a_{-1,-1} & a_{-1,0} & a_{-1,1} & a_{-1,2} \\ a_{0,-2} & a_{0,-1} & a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,-2} & a_{1,-1} & a_{1,0} & a_{1,1} & a_{1,2} \end{bmatrix}$$

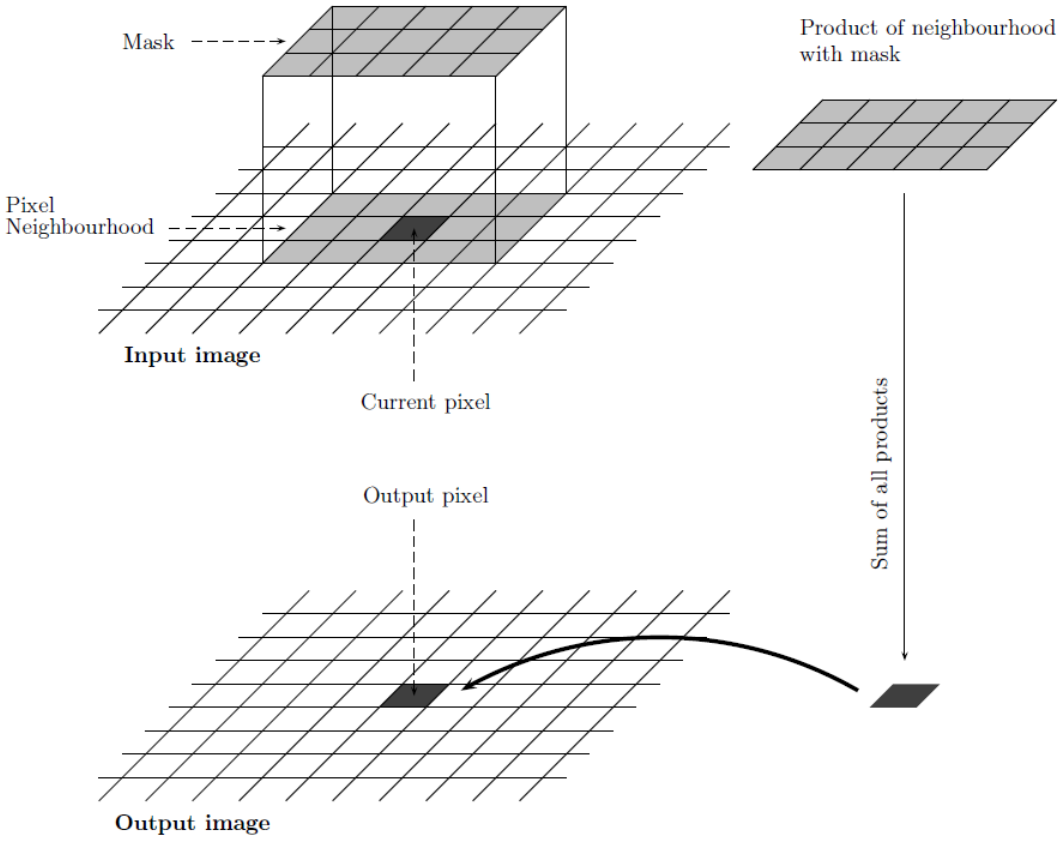
a odpowiadające im wartości pikseli są następujące:

$$\begin{bmatrix} f(x-1,y-2) & f(x-1,y-1) & f(x-1,y) & f(x-1,y+1) & f(x-1,y+2) \\ f(x,y-2) & f(x,y-1) & f(x,y) & f(x,y+1) & f(x,y+2) \\ f(x+1,y-2) & f(x+1,y-1) & f(x+1,y) & f(x+1,y+1) & f(x+1,y+2) \end{bmatrix}.$$

Teraz mnożymy i dodajemy:

$$\sum_{i=-1}^{1} \sum_{j=-2}^{2} a_{i,j}\, f(x+i, y+j).$$

Schemat ilustrujący proces wykonywania filtracji przestrzennej podano na rysunku 3.2.



Rysunek 3.2: Wykonywanie filtracji przestrzennej.

Filtracja przestrzenna wymaga trzech kroków:

1. umieścić maskę nad bieżącym pikselem,
2. utworzyć wszystkie iloczyny elementów filtru z odpowiadającymi im elementami obrazu,
3. zsumować wszystkie iloczyny.

Należy to powtórzyć dla każdego piksela obrazu.

Z filtracją przestrzenną związany jest **splot przestrzenny**. Metoda wykonywania splotu jest taka sama jak metoda filtracji, z wyjątkiem tego, że przed mnożeniem i dodawaniem filtr musi zostać obrócony o $180°$. Używając jak poprzednio notacji $a(i,j)$ oraz $f(x,y)$, wynik splotu z maską 3×5 dla pojedynczego piksela ma postać:

$$\sum_{i=-1}^{1} \sum_{j=-2}^{2} a_{-i,-j}\ f(x+i, y+j).$$

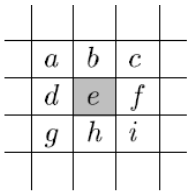
Zwróćmy uwagę na znaki minus przy indeksach a. Ten sam wynik można uzyskać przez:

$$\sum_{i=-1}^{1} \sum_{j=-2}^{2} a_{i,j}\ f(x-i, y-j).$$

Przykład. Jednym ważnym filtrem liniowym jest użycie maski 3×3 i wzięcie średniej ze wszystkich dziewięciu wartości znajdujących się w masce. Ta wartość staje się wartością szarości odpowiadającego piksela w nowym obrazie. Operację tę można opisać następująco:

$$g(x,y) = \frac{1}{9} \sum_{i=-1}^{1} \sum_{j=-1}^{1} f\ (x+i, y+j),$$

gdzie $f(x,y)$ jest wartością szarości bieżącego piksela w obrazie oryginalnym, a średnia jest wartością szarości odpowiadającego piksela w nowym obrazie.



$$\longrightarrow \frac{1}{9}(a+b+c+d+e+f+g+h+i)$$

Aby zastosować to do obrazu, rozważmy „obraz” 5×5 otrzymany przez:

```
>> x=uint8(10*magic(5))
x =
   170   240    10    80   150
   230    50    70   140   160
    40    60   130   200   220
   100   120   190   210    30
   110   180   250    20    90
```

Możemy traktować tę tablicę jako złożoną z dziewięciu nakładających się sąsiedztw 3×3. Wynik naszych obliczeń będzie więc składał się tylko z dziewięciu wartości. Później zobaczymy, jak otrzymać 25 wartości na wyjściu.

Rozważmy górne lewe sąsiedztwo 3×3 naszego obrazu x:

```
170 240  10
230  50  70
 40  60 130
```

Teraz bierzemy średnią ze wszystkich tych wartości:

```
>> mean2(x(1:3,1:3))
ans =
  111.1111
```

co można zaokrąglić do 111. Teraz możemy przejść do drugiego sąsiedztwa:

```
240  10  80
 50  70 140
 60 130 200
```

i obliczyć jego średnią:

```
>> mean2(x(1:3,2:4))
ans =
  108.8889
```

Wartość tę można zaokrąglić w dół do 108 albo do najbliższej liczby całkowitej 109. Jeżeli będziemy kontynuować w ten sposób, otrzymamy następujący wynik:

```
111.1111  108.8889  128.8889
110.0000  130.0000  150.0000
131.1111  151.1111  148.8889
```

Ta tablica jest wynikiem filtrowania x filtrem uśredniającym 3×3.

3.2 Notacja

Wygodnie jest opisywać filtr liniowy po prostu za pomocą współczynników dla wszystkich wartości szarości pikseli znajdujących się w masce. Można to zapisać jako macierz.

Na przykład powyższy filtr uśredniający mógłby mieć swoje wyjście zapisane jako:

$$\frac{1}{9}f(x-1,y-1)+\frac{1}{9}f(x-1,y)+\frac{1}{9}f(x-1,y+1)+\frac{1}{9}f(x,y-1)+\frac{1}{9}f(x,y)+\frac{1}{9}f(x,y+1)+\frac{1}{9}f(x+1,y-1)$$
$$+\frac{1}{9}f(x+1,y)+\frac{1}{9}f(x+1,y+1).$$

Dlatego filtr ten można opisać macierzą:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}.$$

Przykład. Filtr

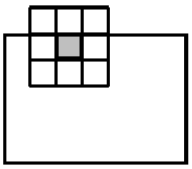
$$\begin{bmatrix}0 & -1 & 0\\ -1 & 4 & -1\\ 0 & -1 & 0\end{bmatrix}$$

działałby na wartościach szarości jako:

$$g(x,y)=4f(x,y)-f(x-1,y)-f(x+1,y)-f(x,y-1)-f(x,y+1).$$

Brzegi obrazu

Istnieje oczywisty problem przy stosowaniu filtru — co dzieje się na brzegu obrazu, gdzie maska częściowo wypada poza obraz? W takim przypadku, jak zilustrowano na rysunku 3.3, będzie brakowało wartości szarości potrzebnych do użycia w funkcji filtru.



Rysunek 3.3: Maska na brzegu obrazu.

Istnieje kilka różnych podejść do rozwiązania tego problemu.

Ignorowanie brzegów. Oznacza to, że maska jest stosowana tylko do tych pikseli obrazu, dla których maska mieści się całkowicie wewnątrz obrazu. Obejmuje to wszystkie piksele poza brzegami i daje obraz wyjściowy mniejszy od oryginału. Jeżeli maska jest bardzo duża, tą metodą można utracić znaczną ilość informacji.

Tę metodę zastosowaliśmy w naszym powyższym przykładzie.

Dopełnianie zerami. Zakładamy, że wszystkie potrzebne wartości poza obrazem są równe zeru. Daje nam to wszystkie wartości potrzebne do pracy i zwraca obraz wyjściowy tego samego rozmiaru co oryginał, ale może wprowadzić niepożądane artefakty, na przykład krawędzie, wokół obrazu.

3.3 Filtrowanie w Matlabie

Funkcja `filter2` wykonuje za nas zadanie filtracji liniowej; jej użycie ma postać:

```
filter2(filter,image,shape)
```

a wynikiem jest macierz typu danych `double`. Parametr `shape` jest opcjonalny; opisuje on metodę radzenia sobie z brzegami.

`filter2(filter,image,'same')` jest wartością domyślną; tworzy macierz o takim samym rozmiarze jak oryginalna macierz obrazu. Używa dopełniania zerami:

```
>> a=ones(3,3)/9
a =
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111

>> filter2(a,x,'same')
ans =
   76.6667   85.5556   65.5556   67.7778   58.8889
   87.7778  111.1111  108.8889  128.8889  105.5556
   66.6667  110.0000  130.0000  150.0000  106.6667
   67.7778  131.1111  151.1111  148.8889   85.5556
   56.6667  105.5556  107.7778   87.7778   38.8889
```

`filter2(filter,image,'valid')` stosuje maskę tylko do pikseli „wewnętrznych”. Wynik zawsze będzie mniejszy niż oryginał:

```
>> filter2(a,x,'valid')
ans =
  111.1111  108.8889  128.8889
  110.0000  130.0000  150.0000
  131.1111  151.1111  148.8889
```

Wynik trybu `'same'` podany powyżej można także otrzymać przez dopełnienie zerami i użycie trybu `'valid'`:

```
>> x2=zeros(7,7);
>> x2(2:6,2:6)=x
x2 =
     0     0     0     0     0     0     0
     0   170   240    10    80   150     0
     0   230    50    70   140   160     0
     0    40    60   130   200   220     0
     0   100   120   190   210    30     0
     0   110   180   250    20    90     0
     0     0     0     0     0     0     0
>> filter2(a,x2,'valid')
```

`filter2(filter,image,'full')` zwraca wynik większy niż oryginał; robi to przez dopełnienie zerami i stosowanie filtru we wszystkich miejscach na obrazie oraz wokół obrazu, w których maska przecina macierz obrazu.

```
>> filter2(a,x,'full')
ans =
   18.8889   45.5556   46.6667   36.6667   26.6667   25.5556   16.6667
   44.4444   76.6667   85.5556   65.5556   67.7778   58.8889   34.4444
   48.8889   87.7778  111.1111  108.8889  128.8889  105.5556   58.8889
   41.1111   66.6667  110.0000  130.0000  150.0000  106.6667   45.5556
   27.7778   67.7778  131.1111  151.1111  148.8889   85.5556   37.7778
   23.3333   56.6667  105.5556  107.7778   87.7778   38.8889   13.3333
   12.2222   32.2222   60.0000   50.0000   40.0000   12.2222   10.0000
```

Parametr `shape`, ponieważ jest opcjonalny, można pominąć; w takim przypadku wartością domyślną jest `'same'`. Nie ma jednego „najlepszego” podejścia; metoda musi wynikać z rozpatrywanego problemu, z używanego filtru oraz z wymaganego wyniku.

Możemy tworzyć filtry ręcznie albo używać funkcji `fspecial`; ma ona wiele opcji, które ułatwiają tworzenie wielu różnych filtrów. Użyjemy opcji `average`, która tworzy filtry uśredniające o zadanym rozmiarze; zatem:

```
>> fspecial('average',[5,7])
```

zwróci filtr uśredniający rozmiaru 5×7;

```
>> fspecial('average',11)
```

zwróci filtr uśredniający rozmiaru 11×11. Jeżeli pominiemy końcową liczbę albo wektor, zwracany jest filtr uśredniający 3×3.

Na przykład załóżmy, że stosujemy filtr uśredniający 3×3 do obrazu w następujący sposób:

```
>> c=imread('cameraman.tif');
>> f1=fspecial('average');
>> cf1=filter2(f1,c);
```

Mamy teraz macierz typu danych `double`. Aby ją wyświetlić, możemy zrobić dowolną z następujących rzeczy:

- przekształcić ją do macierzy typu `uint8`, aby użyć `imshow`,
- podzielić jej wartości przez 255, aby otrzymać macierz z wartościami w zakresie $0–1$ do użycia z `imshow`,
- użyć `mat2gray`, aby przeskalować wynik do wyświetlenia. Użycie tej funkcji omówimy później.

Używając drugiej metody:

```
>> figure,imshow(c),figure,imshow(cf1/255)
```

otrzymamy obrazy pokazane na rysunkach 3.4(a) oraz 3.4(b).

Filtr uśredniający rozmywa obraz; szczególnie krawędzie są mniej wyraźne niż w oryginale. Obraz można rozmyć jeszcze bardziej, używając filtru uśredniającego o większym rozmiarze. Pokazano to na rysunku 3.4(c), gdzie użyto filtru uśredniającego 7×7, oraz na rysunku 3.4(d), gdzie użyto filtru uśredniającego 15×15.

(a) Original image (b) Average filtering

(c) Using a 9×9 filter (d) Using a 25×25 filter

Rysunek 3.4: Filtrowanie uśredniające.

Zauważmy, że dopełnianie zerami użyte na brzegach spowodowało pojawienie się ciemnej ramki wokół obrazu. Jest to szczególnie widoczne, gdy używany jest duży filtr. Jeżeli jest to niepożądany artefakt filtracji, na przykład jeżeli zmienia on średnią jasność obrazu, wtedy bardziej odpowiednie może być użycie opcji kształtu `'valid'`.

Obraz wynikowy po zastosowaniu tych filtrów może wydawać się znacznie „gorszy" niż oryginał. Jednak zastosowanie filtru rozmywającego w celu zmniejszenia szczegółów w obrazie może być idealną operacją dla autonomicznego rozpoznawania maszynowego albo wtedy, gdy koncentrujemy się tylko na „ogólnych" aspektach obrazu: liczbie obiektów, ilości obszarów ciemnych i jasnych. W takich przypadkach zbyt wiele szczegółów może zaciemniać wynik.

Filtry separowalne

Niektóre filtry można zaimplementować przez kolejne zastosowanie dwóch prostszych filtrów. Na przykład, ponieważ:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1\\ 1\\ 1\end{bmatrix}\frac{1}{3}[1 \quad 1 \quad 1],$$

filtr uśredniający 3×3 można zaimplementować przez najpierw zastosowanie filtru uśredniającego 3×1, a następnie zastosowanie filtru uśredniającego 1×3 do wyniku. Filtr uśredniający 3×3 jest zatem separowalny na dwa mniejsze filtry.

Separowalność może dawać duże oszczędności czasu. Załóżmy, że filtr $n \times n$ jest separowalny na dwa filtry o rozmiarach $n \times 1$ oraz $1 \times n$. Zastosowanie filtru $n \times n$ wymaga n^2 mnożeń oraz $n^2 - 1$ dodawań dla każdego piksela obrazu. Natomiast zastosowanie filtru $n \times 1$ wymaga tylko n mnożeń i $n - 1$ dodawań. Ponieważ trzeba to zrobić dwa razy, łączna liczba mnożeń i dodawań wynosi odpowiednio $2n$ oraz $2n - 2$. Jeżeli n jest duże, oszczędność efektywności może być dramatyczna.

Wszystkie filtry uśredniające są separowalne; innym filtrem separowalnym jest laplasjan:

$$\begin{bmatrix}1 & -2 & 1\\ -2 & 4 & -2\\ 1 & -2 & 1\end{bmatrix} = \begin{bmatrix}1\\ -2\\ 1\end{bmatrix}[1 \quad -2 \quad 1]$$

3.4 Częstotliwości; filtry dolnoprzepustowe i górnoprzepustowe

Wygodnie będzie mieć pewną standardową terminologię, za pomocą której będziemy mogli omawiać efekty, jakie filtr będzie miał na obrazie, oraz być w stanie wybrać najbardziej odpowiedni filtr dla danego zadania przetwarzania obrazu. Jednym ważnym aspektem obrazu, który nam to umożliwia, jest pojęcie **częstotliwości**. Z grubsza rzecz biorąc, częstotliwości obrazu są miarą tego, o ile wartości szarości zmieniają się wraz z odległością. **Składowe wysokoczęstotliwościowe** charakteryzują się dużymi zmianami wartości szarości na małych odległościach; przykładami składowych wysokoczęstotliwościowych są krawędzie i szum. **Składowe niskoczęstotliwościowe**, z drugiej strony, są częściami obrazu charakteryzującymi się niewielką zmianą wartości szarości. Mogą one obejmować tła, tekstury skóry. Mówimy wtedy, że filtr jest

- **filtrem górnoprzepustowym**, jeżeli "przepuszcza" składowe wysokoczęstotliwościowe oraz redukuje lub eliminuje składowe niskoczęstotliwościowe,
- **filtrem dolnoprzepustowym**, jeżeli "przepuszcza" składowe niskoczęstotliwościowe oraz redukuje lub eliminuje składowe wysokoczęstotliwościowe,

Na przykład filtr uśredniający 3×3 jest filtrem dolnoprzepustowym, ponieważ ma tendencję do rozmywania krawędzi. Filtr

$$\begin{bmatrix}1 & -2 & 1\\ -2 & 4 & -2\\ 1 & -2 & 1\end{bmatrix}$$

jest filtrem górnoprzepustowym.

Zauważamy, że suma współczynników (to znaczy suma wszystkich elementów w macierzy) w filtrze górnoprzepustowym wynosi zero. Oznacza to, że w niskoczęstotliwościowej części obrazu, gdzie wartości szarości są podobne, wynikiem użycia tego filtru będzie to, że odpowiadające wartości szarości w nowym obrazie będą bliskie zeru. Aby to zobaczyć, rozważmy blok 4×4 pikseli o podobnych wartościach i zastosujmy powyższy filtr górnoprzepustowy do centralnych czterech:

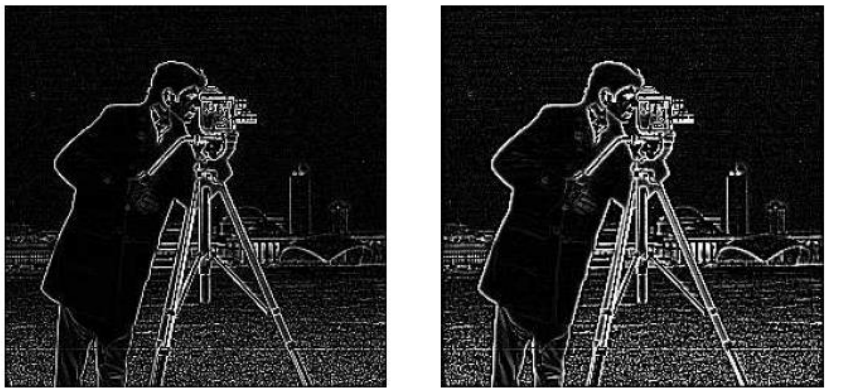
$$\begin{bmatrix} 150 & 152 & 148 & 149 \\ 147 & 152 & 151 & 150 \\ 152 & 148 & 149 & 151 \\ 151 & 149 & 150 & 148 \end{bmatrix} \rightarrow \begin{bmatrix} 11 & 6 \\ -13 & -5 \end{bmatrix}$$

Otrzymane wartości są bliskie zeru, co jest oczekiwanym wynikiem zastosowania filtru górnoprzepustowego do składowej niskoczęstotliwościowej. Zobaczymy poniżej, jak radzić sobie z wartościami ujemnymi.

Filtry górnoprzepustowe mają szczególną wartość w wykrywaniu krawędzi i wzmacnianiu krawędzi. Np.

```
>> f=fspecial('laplacian')

f =

    0.1667    0.6667    0.1667
    0.6667   -3.3333    0.6667
    0.1667    0.6667    0.1667

>> cf=filter2(f,c);
>> imshow(cf/100)
>> f1=fspecial('log')

f1 =

    0.0448    0.0468    0.0564    0.0468    0.0448
    0.0468    0.3167    0.7146    0.3167    0.0468
    0.0564    0.7146   -4.9048    0.7146    0.0564
    0.0468    0.3167    0.7146    0.3167    0.0468
    0.0448    0.0468    0.0564    0.0468    0.0448

>> cf1=filter2(f1,c);
>> figure,imshow(cf1/100)
```

Obrazy są pokazane na rysunku 3.5. Obraz (a) jest wynikiem filtru Laplace'a; obraz (b) pokazuje wynik filtru Laplasjana Gaussowskiego ("log"). W każdym przypadku suma wszystkich elementów filtru wynosi zero.



(a) Laplacian filter (b) Laplacian of Gaussian ("log") filtering

Rysunek 3.5: Filtrowanie górnoprzepustowe. (a) Filtr Laplace'a. (b) Filtrowanie Laplasjanem Gaussowskim ("log").

Wartości spoza zakresu 0-255

Widzieliśmy, że do wyświetlania obrazu chcielibyśmy, aby wartości szarości pikseli leżały pomiędzy 0 a 255. Jednak wynik zastosowania filtru liniowego może być wartościami, które leżą poza tym zakresem. Możemy rozważyć sposoby radzenia sobie z wartościami poza tym "wyświetlalnym" zakresem.

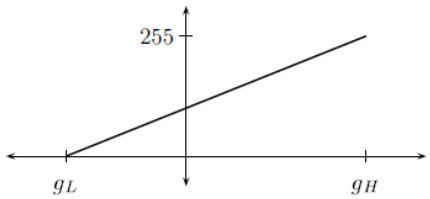
Zamień wartości ujemne dodatnimi. To z pewnością poradzi sobie z wartościami ujemnymi, ale nie z wartościami większymi niż 255. Dlatego może to być użyte tylko w szczególnych okolicznościach; na przykład wtedy, gdy jest tylko kilka wartości ujemnych i gdy same te wartości są bliskie zeru.

Obetnij wartości. Stosujemy następującą operację typu progowania do wartości szarości x wytworzonych przez filtr, aby otrzymać wyświetlalną wartość y:

$$y = \begin{cases} 0, & \text{jeśli } x < 0, \\ x, & \text{jeśli } 0 \le x \le 255, \\ 255, & \text{jeśli } x > 255. \end{cases}$$

To wytworzy obraz ze wszystkimi wartościami pikseli w wymaganym zakresie, ale nie jest odpowiednie, jeżeli istnieje wiele wartości szarości poza zakresem 0-255; w szczególności, jeżeli wartości szarości są równomiernie rozłożone na większym zakresie. W takim przypadku ta operacja będzie miała tendencję do niszczenia wyników filtru.

Transformacja skalująca. Przypuśćmy, że najniższa wartość szarości wytworzona przez filtr to g_L, a najwyższa wartość to g_H. Możemy przekształcić wszystkie wartości z zakresu $g_L - g_H$ na zakres 0-255 za pomocą transformacji liniowej zilustrowanej poniżej:



Ponieważ nachylenie prostej wynosi $255/(g_H - g_L)$, możemy zapisać równanie prostej jako

$$y = 255\frac{x - g_L}{g_H - g_L}$$

a zastosowanie tej transformacji do wszystkich poziomów szarości x wytworzonych przez filtr da w wyniku (po ewentualnym koniecznym zaokrągleniu) obraz, który można wyświetlić.

Jako przykład zastosujmy filtr górnoprzepustowy podany w sekcji 3.4 do obrazu cameraman:

```
>> f2=[1 -2 1;-2 4 -2;1 -2 1];
>> cf2=filter2(f2,c);
```

Teraz wartości maksymalna i minimalna macierzy `cf2` wynoszą odpowiednio 593 oraz -541. Funkcja `mat2gray` automatycznie skaluje elementy macierzy do wartości wyświetlalnych; dla dowolnej macierzy M stosuje transformację liniową do jej elementów, z najniższą wartością odwzorowaną na 0.0, a najwyższą wartością odwzorowaną na 1.0. Oznacza to, że wyjście funkcji `mat2gray` jest zawsze typu `double`. Funkcja wymaga również, aby typ wejściowy był `double`.

```
>> figure,imshow(mat2gray(cf2));
```

Aby zrobić to ręcznie, że tak powiem, stosując powyższą transformację liniową, możemy użyć:

```
>> maxcf2=max(cf2(:));
>> mincf2=min(cf2(:));
>> cf2g=(cf2-mincf2)/(maxcf2-mncf2);
```

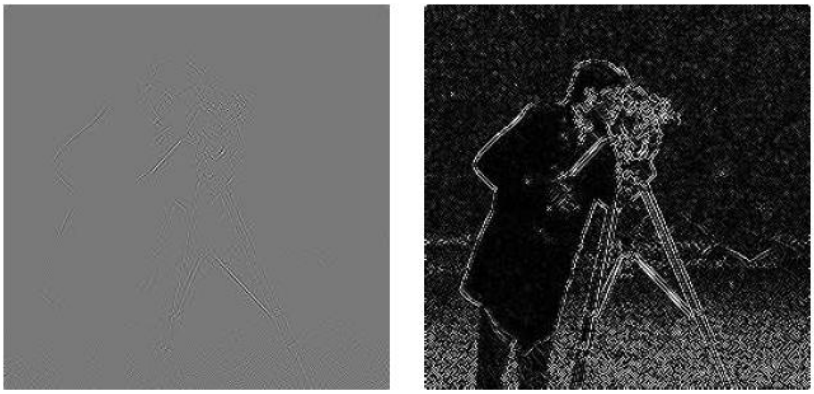
Wynik będzie macierzą typu `double`, z elementami w zakresie 0.0-1.0. Można ją obejrzeć za pomocą `imshow`. Możemy uczynić ją obrazem `uint8`, mnożąc najpierw przez 255. Wynik można zobaczyć na rysunku 3.6.

Zazwyczaj możemy uzyskać lepszy wynik, dzieląc wynik filtrowania przez stałą przed wyświetleniem go:

```
>> figure,imshow(cf2/60)
```

i to także pokazano na rysunku 3.6.

Filtry górnoprzepustowe są często używane do wykrywania krawędzi. Można je zobaczyć dość wyraźnie na prawym obrazie rysunku 3.6.



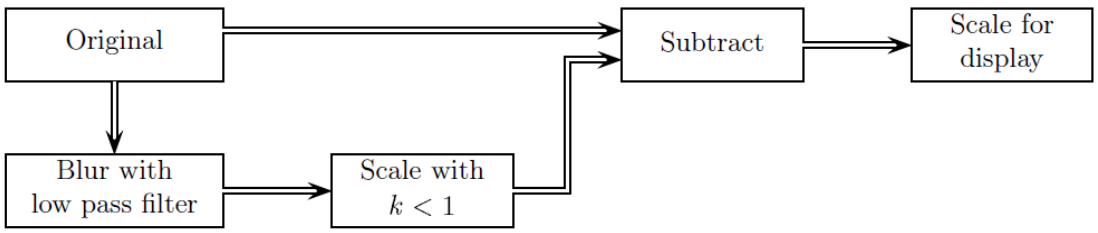
Rysunek 3.6: Użycie filtru górnoprzepustowego i wyświetlenie wyniku. (a) Użycie `mat2gray`. (b) Dzielenie przez stałą.

*3.5 Wyostrzanie krawędzi

Filtrowanie przestrzenne może być użyte do uczynienia krawędzi w obrazie nieco ostrzejszymi i bardziej wyrazistymi, co ogólnie skutkuje obrazem przyjemniejszym dla ludzkiego oka. Operacja ta jest różnie nazywana “wzmacnianiem krawędzi”, “wyostrzaniem krawędzi” albo “maskowaniem wyostrzającym”. Ten ostatni termin pochodzi z przemysłu drukarskiego.

Maskowanie wyostrzające

Idea maskowania wyostrzającego polega na odjęciu przeskalowanej “wyostrzającej” wersji obrazu od oryginału. W praktyce możemy osiągnąć ten efekt, odejmując przeskalowany rozmyty obraz od oryginału. Schemat maskowania wyostrzającego pokazano na rysunku 3.7.



Rysunek 3.7: Schemat maskowania wyostrzającego.

Przypuśćmy, że obraz x jest typu `uint8`. Maskowanie wyostrzające może być zastosowane za pomocą następującej sekwencji poleceń:

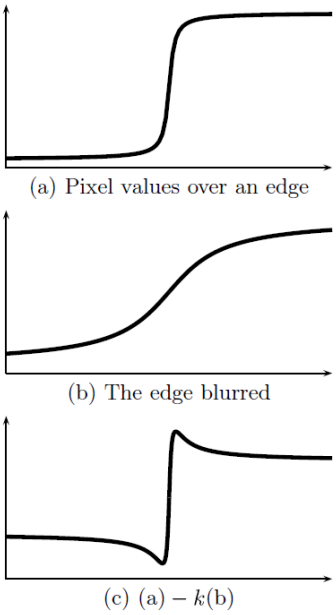
```
>> f=fspecial('average');
>> xf=filter2(f,x);
>> xu=double(x)-xf/1.5
>> imshow(xu/70)
```

Ostatnie polecenie skaluje wynik tak, aby `imshow` wyświetliło odpowiedni obraz; wartość może wymagać dostosowania zgodnie z obrazem wejściowym. Przypuśćmy, że x jest obrazem pokazanym na rysunku 3.8(a), wtedy wynik maskowania wyostrzającego jest podany na rysunku 3.8(b). Wynik wydaje się być lepszym obrazem niż oryginał; krawędzie są bardziej wyraziste i jaśniej zdefiniowane.

(a) Original image (b) The image after unsharp masking

Rysunek 3.8: Przykład maskowania wyostrzającego. (a) Obraz oryginalny. (b) Obraz po maskowaniu wyostrzającym.

Aby zobaczyć, dlaczego to działa, możemy rozważyć funkcję wartości szarości, gdy przechodzimy przez krawędź, jak pokazano na rysunku 3.9.



(a) Pixel values over an edge

(b) The edge blurred

(c) (a) − k(b)

Rysunek 3.9: Maskowanie wyostrzające. (a) Wartości pikseli przez krawędź. (b) Krawędź rozmyta. (c) (a) - k(b).

Ponieważ przeskalowane rozmycie jest odejmowane od oryginału, wynikiem jest to, że krawędź zostaje wzmocniona, jak pokazano na wykresie (c) rysunku 3.9.

Możemy faktycznie wykonać operację filtrowania i odejmowania w jednym poleceniu, używając liniowości filtru oraz tego, że filtr 3×3

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

jest “filtrem identycznościowym”.

Stąd maskowanie wyostrzające może być zaimplementowane za pomocą filtru postaci

$$f = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{k}\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

gdzie k jest stałą wybraną tak, aby dać najlepszy wynik. Alternatywnie filtr maskowania wyostrzającego może być zdefiniowany jako

$$f = k\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

tak że w efekcie odejmujemy rozmycie od przeskalowanej wersji oryginału; współczynnik skalowania może być także podzielony pomiędzy filtry identycznościowy i rozmywający.

Opcja `unsharp` funkcji `fspecial` tworzy takie filtry; utworzony filtr ma postać

$$\frac{1}{\alpha+1}\begin{bmatrix} -\alpha & \alpha-1 & -\alpha \\ \alpha-1 & \alpha+5 & \alpha-1 \\ -\alpha & \alpha-1 & -\alpha \end{bmatrix}$$

gdzie α jest opcjonalnym parametrem, którego wartość domyślna wynosi 0.2. Jeżeli $\alpha = 0.5$, filtr jest

$$\frac{1}{3}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 11 & -1 \\ -1 & -1 & -1 \end{bmatrix} = 4\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - 3\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Rysunek 3.10 został utworzony za pomocą poleceń MATLAB-a

```
>> p=imread('pelicans.tif');
>> u=fspecial('unsharp',0.5);
>> pu=filter2(u,p);
>> imshow(p),figure,imshow(pu/255)
```

Rysunek 3.10(b) wydaje się znacznie ostrzejszy i “czystszy” niż oryginał. Zauważ w szczególności skały i drzewa w tle oraz zmarszczki na wodzie.

(a) The original (b) After unsharp masking

Rysunek 3.10: Wzmocnienie krawędzi z maskowaniem wyostrzającym. (a) Oryginał. (b) Po maskowaniu wyostrzającym.

Chociaż powyżej używaliśmy filtrów uśredniających, możemy w rzeczywistości użyć dowolnego filtru dolnoprzepustowego do maskowania wyostrzającego.

Filtrowanie high boost

Powiązane z filtrami maskowania wyostrzającego są filtry **high boost**, które otrzymuje się przez

$$\mathrm{HB} = AO - L.$$

gdzie O oznacza obraz oryginalny, L oznacza wynik działania filtru dolnoprzepustowego, a A jest "współczynnikiem wzmocnienia". Jeżeli $A = 1$, wtedy filtr high boost staje się zwykłym filtrem górnoprzepustowym. Jeżeli jako filtr dolnoprzepustowy weźmiemy filtr uśredniający 3×3, wtedy filtr high boost będzie miał postać

$$\frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ -1 & z & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

gdzie $z > 8$. Jeżeli podstawimy $z = 11$, otrzymamy filtrowanie bardzo podobne do powyższego filtru unsharp, z wyjątkiem współczynnika skalowania. Zatem polecenia:

```
>> f=[-1 -1 -1;-1 11 -1;-1 -1 -1]/9;
>> xf=filter2(x,f);
>> imshow(xf/80)
```

wytworzą obraz podobny do tego na rysunku 3.8. Wartość 80 została uzyskana metodą prób i błędów, aby wytworzyć obraz o intensywności podobnej do oryginału.

Możemy również zapisać powyższy wzór high boost jako (przyjmując, że O oznacza oryginał, L filtr dolnoprzepustowy, a H filtr górnoprzepustowy):

$$\mathrm{HB} = AO - L$$

$$\mathrm{HB} = AO - (O - H)$$

$$\mathrm{HB} = (A - 1)O + H.$$

Najlepsze wyniki filtrowania high boost uzyskuje się, jeżeli mnożymy równanie przez współczynnik w tak, aby wartości filtru sumowały się do 1; wymaga to

$$wA - w = 1$$

lub

$$w = \frac{1}{A - 1}.$$

Zatem ogólny wzór maskowania wyostrzającego jest

$$\frac{A}{A - 1}O - \frac{1}{A - 1}L.$$

Inna wersja tego wzoru to

$$\frac{A}{2A - 1}O - \frac{1 - A}{2A - 1}L$$

gdzie dla najlepszych wyników A przyjmuje się tak, aby

$$\frac{3}{5} \leq A \leq \frac{5}{6}.$$

Jeżeli weźmiemy $A = 3/5$, wzór staje się

$$\frac{3/5}{2(3/5) - 1}O - \frac{1 - (3/5)}{2(3/5) - 1}L = 3O - 2L.$$

Jeżeli weźmiemy $A = 5/6$, otrzymujemy

$$\frac{5}{4}O - \frac{1}{4}L.$$

Używając filtrów identycznościowego i uśredniającego, możemy otrzymać filtry high boost przez:

```
>> id=[0 0 0;0 1 0;0 0 0];
>> f=fspecial('average');
>> hb1=3*id-2*f

hb1 =

   -0.2222   -0.2222   -0.2222
   -0.2222    2.7778   -0.2222
   -0.2222   -0.2222   -0.2222

>> hb2=1.25*id-0.25*f

hb2 =

   -0.0278   -0.0278   -0.0278
   -0.0278    1.2222   -0.0278
   -0.0278   -0.0278   -0.0278
```

Jeżeli każdy z filtrów `hb1` i `hb2` zostanie zastosowany do obrazu za pomocą `filter2`, wynik będzie miał wzmocnione krawędzie. Obrazy na rysunku 3.11 pokazują te wyniki; rysunek 3.11(a) został otrzymany przez

```
>> x1=filter2(hb1,x);
>> imshow(x1/255)
```

a rysunek 3.11(b) podobnie.

(a) High boost filtering with `hb1`

(b) High boost filtering with `hb2`

Rysunek 3.11: Filtrowanie high boost. (a) Filtrowanie high boost z `hb1`. (b) Filtrowanie high boost z `hb2`.

Z dwóch filtrów `hb1` wydaje się dawać lepszy wynik; `hb2` daje obraz niewiele bardziej wyrazisty niż oryginał.

3.6 Filtry nieliniowe

Filtry liniowe, jak widzieliśmy w poprzednich sekcjach, są łatwe do opisania i mogą być stosowane bardzo szybko i wydajnie przez MATLAB-a.

Filtr nieliniowy uzyskuje się przez nieliniową funkcję zaimplementowana w masce przekształcającą wartości w skali szarości. Prostymi przykładami są **filtr maksimum**, którego wynikiem jest maksymalna wartość pod maską, oraz odpowiadający mu **filtr minimum**, którego wynikiem jest minimalna wartość pod maską.

Zarówno filtr maksimum, jak i filtr minimum są przykładami **filtrów porządkowych**. W takim filtrze elementy pod maską są porządkowane, a konkretna wartość jest zwracana jako wynik. Zatem jeżeli wartości są podane w porządku rosnącym, filtr

minimum jest filtrem porządkowym, dla którego zwracany jest pierwszy element, a filtr maksimum jest filtrem porządkowym, dla którego zwracany jest ostatni element.

Do implementowania ogólnego filtru nieliniowego w MATLAB-ie funkcja, której należy użyć, to `nlfilter`, która stosuje filtr do obrazu zgodnie ze wstępnie zdefiniowaną funkcją. Jeżeli funkcja nie jest już zdefiniowana, musimy utworzyć plik m, który ją definiuje.

Oto kilka przykładów; najpierw, aby zaimplementować filtr maksimum na sąsiedztwie 3×3:

```
>> cmax=nlfilter(c,[3,3],'max(x(:))');
```

Funkcja `nlfilter` wymaga trzech argumentów: macierzy obrazu, rozmiaru filtru oraz funkcji, która ma zostać zastosowana. Funkcja musi być funkcją macierzową, która zwraca wartość skalarną. Wynik tej operacji pokazano na rysunku 3.12(a).

Odpowiadająca implementacja filtru minimum to:

```
>> cmin=nlfilter(c,[3,3],'min(x(:))');
```

a wynik pokazano na rysunku 3.12(b).

(a) Using a maximum filter (b) Using a minimum filter

Rysunek 3.12: Użycie filtrów nieliniowych. (a) Użycie filtru maksimum. (b) Użycie filtru minimum.

Zauważ, że w każdym przypadku obraz stracił nieco ostrości i został rozjaśniony przez filtr maksimum oraz przyciemniony przez filtr minimum.

Filtrowanie nieliniowe za pomocą `nlfilter` może być bardzo wolne. Szybszą alternatywą jest użycie funkcji `colfilt`, która najpierw przestawia obraz w kolumny. Na przykład, aby zastosować filtr maksimum do obrazu cameraman, możemy użyć

```
>> cmax=colfilt(c,[3,3],'sliding',@max);
```

Parametr `sliding` wskazuje, że używane są nakładające się sąsiedztwa (co oczywiście ma miejsce przy filtrowaniu). Ta konkretna operacja jest niemal natychmiastowa w porównaniu z użyciem `nlfilter`.

Aby zaimplementować filtry maksimum i minimum jako filtry porządkowe, możemy użyć funkcji MATLAB-a `ordfilt2`. Wymaga ona trzech danych wejściowych: obrazu, wartości indeksu uporządkowanych wyników do wybrania jako wyjście oraz definicji maski. Aby więc zastosować filtr maksimum na masce 3×3, używamy

```
>> cmax=ordfilt2(c,9,ones(3,3));
```

a filtr minimum może być zastosowany przez

```
>> cmin=ordfilt2(c,1,ones(3,3));
```

Bardzo ważnym filtrem porządkowym jest **filtr medianowy**, który bierze środkową wartość uporządkowanej listy. Moglibyśmy zastosować filtr medianowy przez

```
>> cmed=ordfilt2(c,5,ones(3,3));
```

Jednak filtr medianowy ma własne polecenie, `medfilt2`, które omawiamy bardziej szczegółowo w rozdziale 5.

Innymi filtrami nieliniowymi są **filtr średniej geometrycznej**, który jest zdefiniowany jako

$$\left(\prod_{(i,j)\in M} x\,(i,j) \right)^{1/|M|}$$

gdzie M jest maską filtru, a $|M|$ jej rozmiarem; oraz **filtr średniej alfa-obciętej**, który najpierw porządkuje wartości pod maską, odcina elementy z każdego końca uporządkowanej listy i bierze średnią z pozostałych. Na przykład, jeżeli mamy maskę 3×3 i uporządkujemy elementy jako

$$x_1 \leq x_2 \leq x_3 \leq \cdots \leq x_9$$

i odetniemy po dwa elementy z każdego końca, wynikiem filtru będzie

$$\frac{x_3 + x_4 + x_5 + x_6 + x_7}{5}.$$

Oba te filtry mają zastosowania w odtwarzaniu obrazów.