Laboratorium CPO
Temat: Wyrównanie histogramu i Filtracja obrazów

Spis treści

1 Histogram. Wyrównanie histogramu.

1.1 Histogram.

- Histogram przedstawia ogólny rozkład intensywności obrazu, tzn. wartości pikseli (zwykle w zakresie od 0 do 255) znajdują się na osi X, a odpowiadająca im liczba pikseli obrazu na osi Y.
- **Histogram nieznormalizowany**:

$$\boldsymbol{n_k = h(r_k)}$$

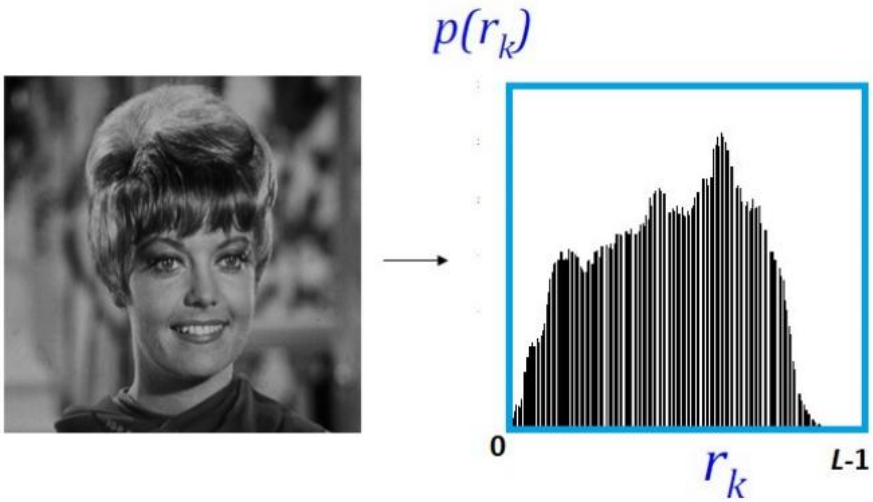
  - gdzie:
    - $r_k\ to\ k-ty\ poziom\ szarości,$
    - $n_k\ to\ liczba\ pikseli\ posiadających\ poziom\ szarości\ r_k$
    - $h(r_k) to\ histogram\ obrazu\ o\ poziomie\ szarości\ r_k$

- **Histogram znormalizowany:**

$$\boldsymbol{p(r_k) = \frac{n_k}{MN}}$$

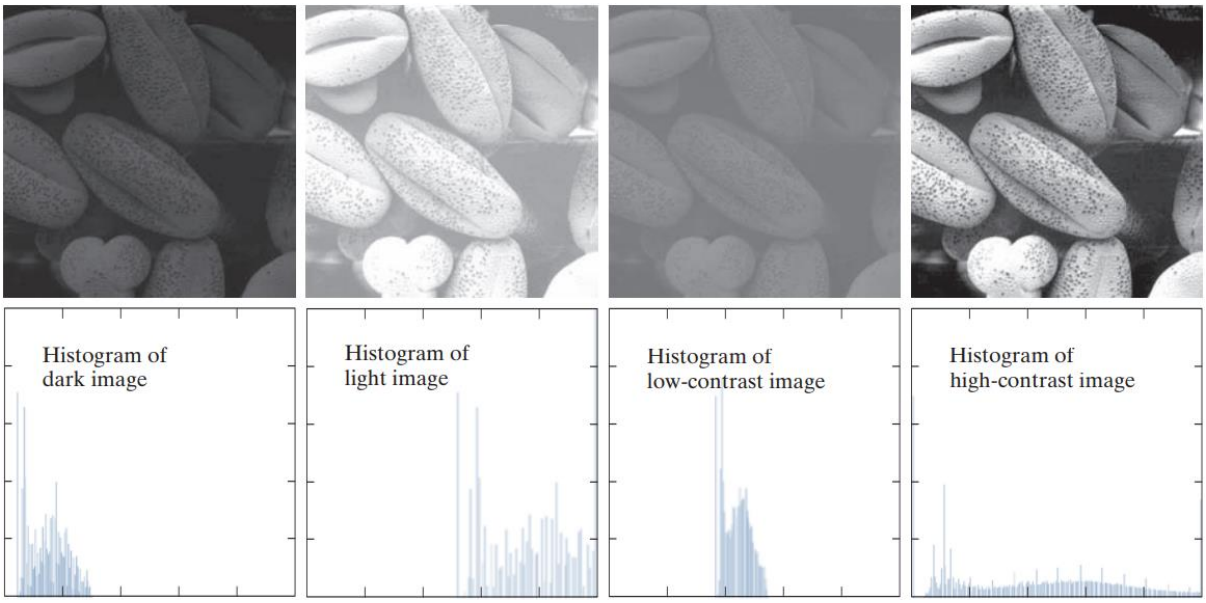
  - gdzie:
    - $p(r_k)\ to\ estymata\ prawdopodobieństwa\ wystąpienia\ poziomu\ szarości\ r_k$
    - $M, N\ to\ wysokość\ i\ szerokość\ obrazu\ a\ MxN\ to\ liczba\ pikseli\ obrazu$
  - suma wszystkich komponentów znormalizowanego histogramu jest równa 1.

- Analizując histogram obrazu, można uzyskać wiedzę dotyczącą kontrastu, jasności, rozkładu intensywności, itp. tego obrazu.

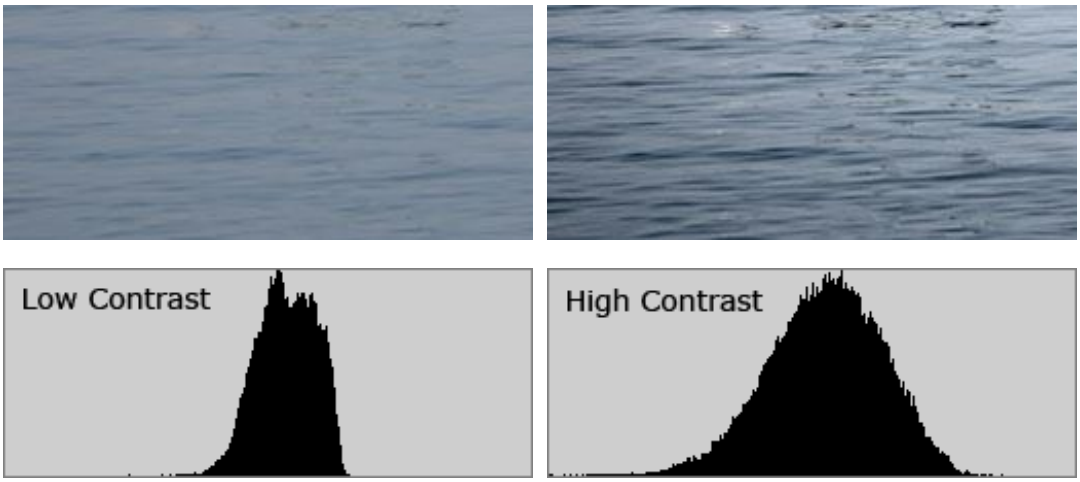


- Kontrast jest miarą różnicy jasności pomiędzy jasnymi i ciemnymi obszarami sceny.
- Szerokie histogramy odzwierciedlają scenę o znacznym kontraście, a wąskie histogramy odzwierciedlają mniejszy kontrast i mogą wydawać się płaskie lub matowe.
- Słaby kontrast zwykle jest spowodowany pewną kombinacją tematu sceny oraz warunkami oświetleniowymi – np. zdjęcia zrobione we mgle mają niski kontrast, a zdjęcia zrobione przy mocnym świetle dziennym mają większy kontrast.
- Kształt histogramu jest związany z wyglądem obrazu, np. poniższy przykład pokazuje cztery typowe charakterystyki intensywności:

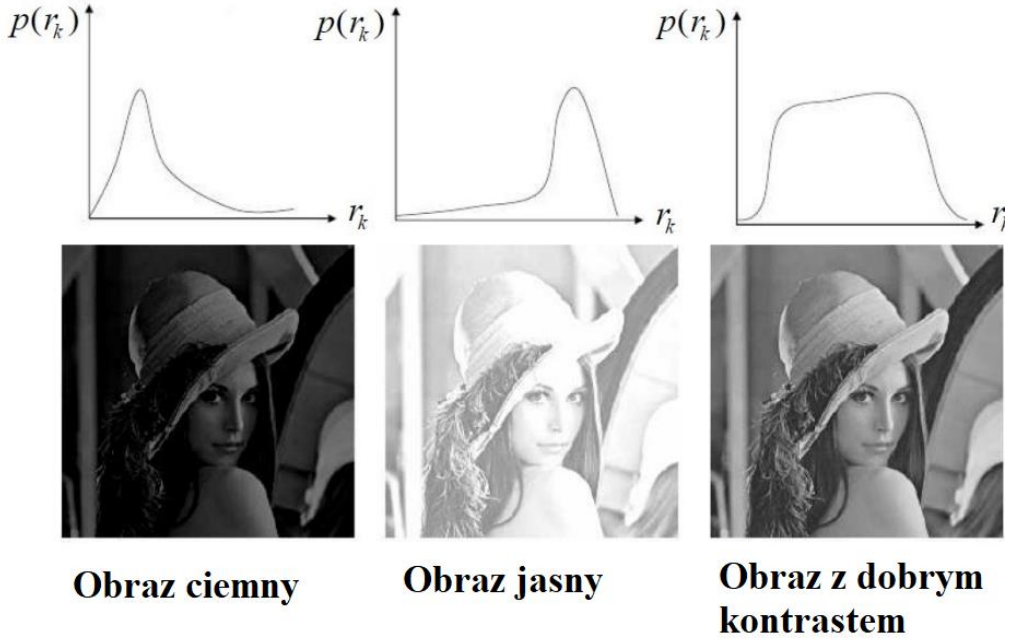
- ciemną – najbardziej zapełnione są przedziały histogramu na dolnym (ciemnym) końcu skali intensywności
- jasną – najbardziej zapełnione są przedziały histogramu odpowiadające górnej (jasnej) części skali intensywności
- niski kontrast – histogram jest wąski, zwykle umieszczony w środkowej części skali intensywności; w przypadku obrazu monochromatycznego obraz jest matowy i ma wyblakły szary wygląd
- wysoki kontrast – składniki histogramu są rozłożone równomiernie w całym zakresie skali intensywności, co sprawia, że obraz przedstawia dużą ilość szczegółów i charakteryzuje się wysokim zakresem dynamicznym.



- Kontrast ma znaczący wpływ wizualny na obraz i uwypukla teksturę, np. na poniższych zdjęciach można zaobserwować, że obszar wody o wysokim kontraście ma głębsze cienie i wyraźniejsze światło, tworząc znacznie wyraźniejszą teksturę (jakby "wyskakiwała" do obserwatora).



- Inny przykład obrazów o słabym kontraście i dobrym kontraście:



Obraz ciemny **Obraz jasny** **Obraz z dobrym kontrastem**

1.2 Obliczanie i rysowanie histogramów

- Obliczanie i rysowanie histogramów można wykonać za pomocą różnych bibliotek Pythona:
    - **NumPy** – funkcja **np.histogram()** oblicza histogram https://numpy.org/doc/stable/reference/generated/numpy.histogram.html
    - **OpenCV** – funkcja **cv2.calcHist**() oblicza histogram 40 razy szybciej niż **np.histogram()** https://docs.opencv.org/4.9.0/d6/dc7/group__imgproc__hist.html#ga4b2b5fd75503ff9e6844cc4dcdaed35d
    - **Matplotlib** – funkcja **plt.hist()** korzysta z **np.histogram()** i rysuje histogram https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html
    - Uwaga: biblioteka NumPy ma do obliczania histogramów jednowymiarowych 10 razy szybszą funkcję niż **np.histogram()** o nazwie **np.bincount()**
- Wszystkie powyższe funkcje mają szereg parametrów, które odpowiadają za określone działanie oraz uzyskanie konkretnych właściwości otrzymywanych wyników.
- Wykonując zadanie wyrównania kontrastu obrazu można zdecydowanie poprawić jego wygląd.

Obraz oryginalny

obraz z wyrównanym kontrastem

1.3 Jak działa wyrównanie histogramu?

- Metoda wykorzystuje tzw. histogram skumulowany (dystrybuantę) i na jego podstawie pozwala wyliczyć nowe wartości dla pikseli.
- Szczegóły działania algorytmu omówione są na stronach:
    - https://docs.opencv.org/4.9.0/d5/daf/tutorial_py_histogram_equalization.html
    - https://en.wikipedia.org/wiki/Histogram_equalization

1.4 Wykorzystanie funkcji `cv2.equalizeHist()` do wyrównania histogramu.

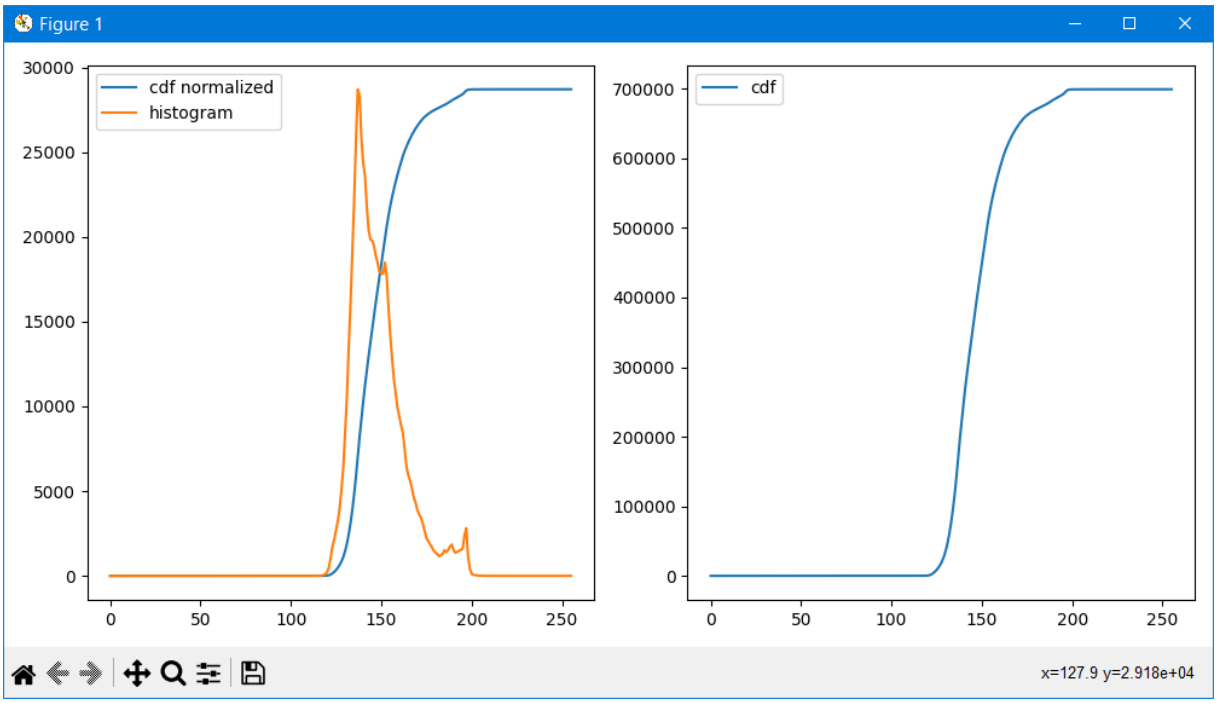
- Przykład zaczerpnięty ze strony `OpenCV`.
- Wprowadź do programu `Spyder` i wykonaj kod z komórki #1
    - przed uruchomieniem kodu w komórce #1 sprawdź ustawienie ścieżki do Twojego folderu roboczego oraz obecność w tym folderze `images` obrazu `wiki.jpg`
    - obraz `wiki.jpg` przedstawia scenę krajobrazu z fragmentem zbocza górskiego oraz dalszą perspektywą doliny, ale obraz technicznie prezentuje słaby kontrast (poniżej znajduje się jego czterokrotne pomniejszenie)

- następnie w komórce #1 obliczane są:
  - za pomocą funkcji `np.histogram()` histogram obrazu `wiki.jpg` oraz
  - histogram skumulowany (dystrybuanta) funkcja `np.cumsum()`
  - zapoznaj się z tymi funkcjami w dokumentacji `NumPy` ustalając przede wszystkim zastosowane parametry i ich wartości
- przy pomocy `Variable Explorer` programu `Spyder` dokonaj analizy uzyskanych wartości, zmienną `hist` oraz `cdf_img` i `cdf_img_normalized`

```
# %%
# 1

# import zależności
import cv2
import numpy as np
import matplotlib.pyplot as plt

# wczytanie obrazu w skali szarości
img = cv2.imread('images/wiki.jpg', cv2.IMREAD_GRAYSCALE)

# obliczenie histogramu za pomocą funkcji histogram() z biblioteki NumPy
hist_img, bins = np.histogram(a=img, bins=256, range=[0,256])

# obliczenie dystrybuanty i jej normalizacja
cdf_img = hist_img.cumsum()
cdf_img_normalized = cdf_img * float(hist_img.max()) / cdf_img.max()
```

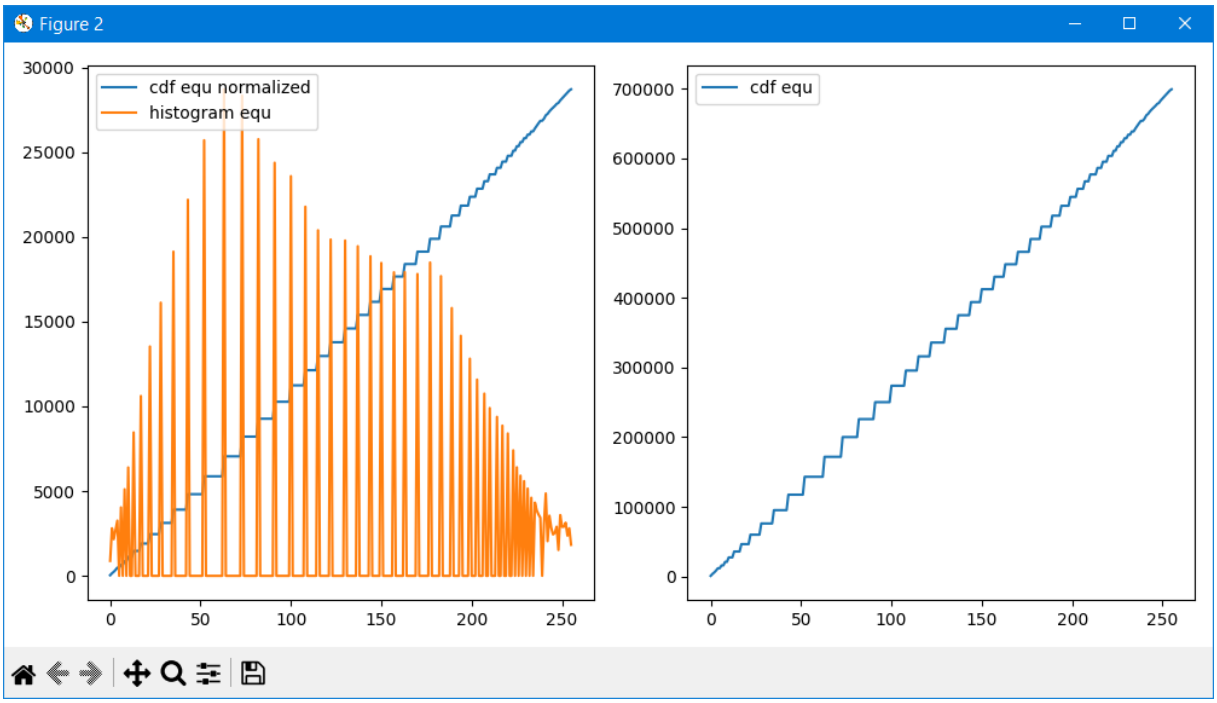
- Teraz wyświetlimy histogram obrazu i dystrybuanty.
- Będzie to wykres poglądowy – wykorzystamy obliczony histogram za pomocą `numpy`.

```
# %%
# 2

# Wykres podglądowy histogramu i dytrybuanty
plt.rcParams['figure.figsize'] = [10, 5]
fig, axs = plt.subplots(1, 2, tight_layout=True)

axs[0].plot(np.arange(256), cdf_img_normalized)
axs[0].plot(np.arange(256), hist_img)
axs[1].plot(np.arange(256), cdf_img)
axs[0].legend(('cdf normalized', 'histogram'), loc = 'upper left')
axs[1].legend(('cdf',), loc = 'upper left')
```



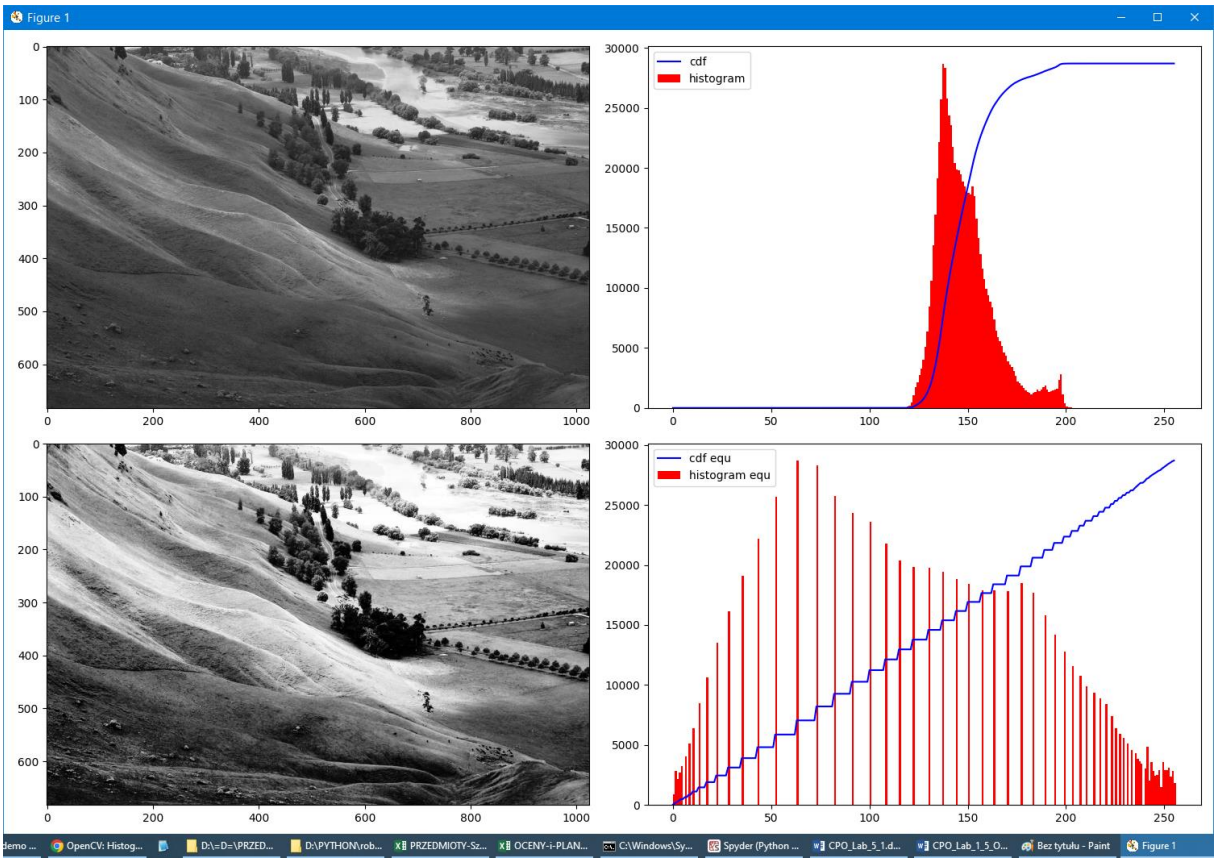
- Wyrównamy histogram obrazu za pomocą funkcji `cv2.equalizeHist()`.
- Wyświetlimy nowy histogram oraz dystrybuantę.

```
# %%
# 3

# Wyrównanie histogramu za pomocą funkcji cv2.equalizeHist
img_equ = cv2.equalizeHist(img)

# obliczenie histogramu za pomocą funkcji histogram() z biblioteki NumPy
hist_img_equ, bins = np.histogram(a=img_equ, bins=256, range=[0,256])

# obliczenie dystrybuanty obrazu img_equ i jej normalizacja
cdf_img_equ = hist_img_equ.cumsum()
cdf_img_equ_normalized = cdf_img_equ * float(hist_img_equ.max()) /
cdf_img_equ.max()


# Wykres podglądowy histogramu i dytrybuanty obrazu po wyrównaniu
histogramu
plt.rcParams['figure.figsize'] = [10, 5]
fig, axs = plt.subplots(1, 2, tight_layout=True)

axs[0].plot(np.arange(256), cdf_img_equ_normalized)
axs[0].plot(np.arange(256), hist_img_equ)
axs[1].plot(np.arange(256), cdf_img_equ)
axs[0].legend(('cdf equ normalized', 'histogram equ'), loc = 'upper left')
axs[1].legend(('cdf equ',), loc = 'upper left')
```



- Na kolejnym wykresie przedstawimy oba obrazy (oryginalny oraz po wyrównaniu histogramu), wraz z histogramami obliczonymi przez biblioteke `matplotlib` i ich dystrybuantami.

```
# %%
# 4

plt.rcParams['figure.figsize'] = [15, 10]
fig, axs = plt.subplots(2, 2, tight_layout=True)
axs[0,0].imshow(img, cmap='gray')

axs[0,1].hist(img.flatten(),256,[0,256], color = 'r')
axs[0,1].plot(cdf_img_normalized, color = 'b')
axs[0,1].legend(('cdf','histogram'), loc = 'upper left')


axs[1,0].imshow(img_equ, cmap='gray')

axs[1,1].hist(img_equ.flatten(),256,[0,256], color = 'r')
axs[1,1].plot(cdf_img_equ_normalized, color = 'b')
axs[1,1].legend(('cdf equ','histogram equ'), loc = 'upper left')
```

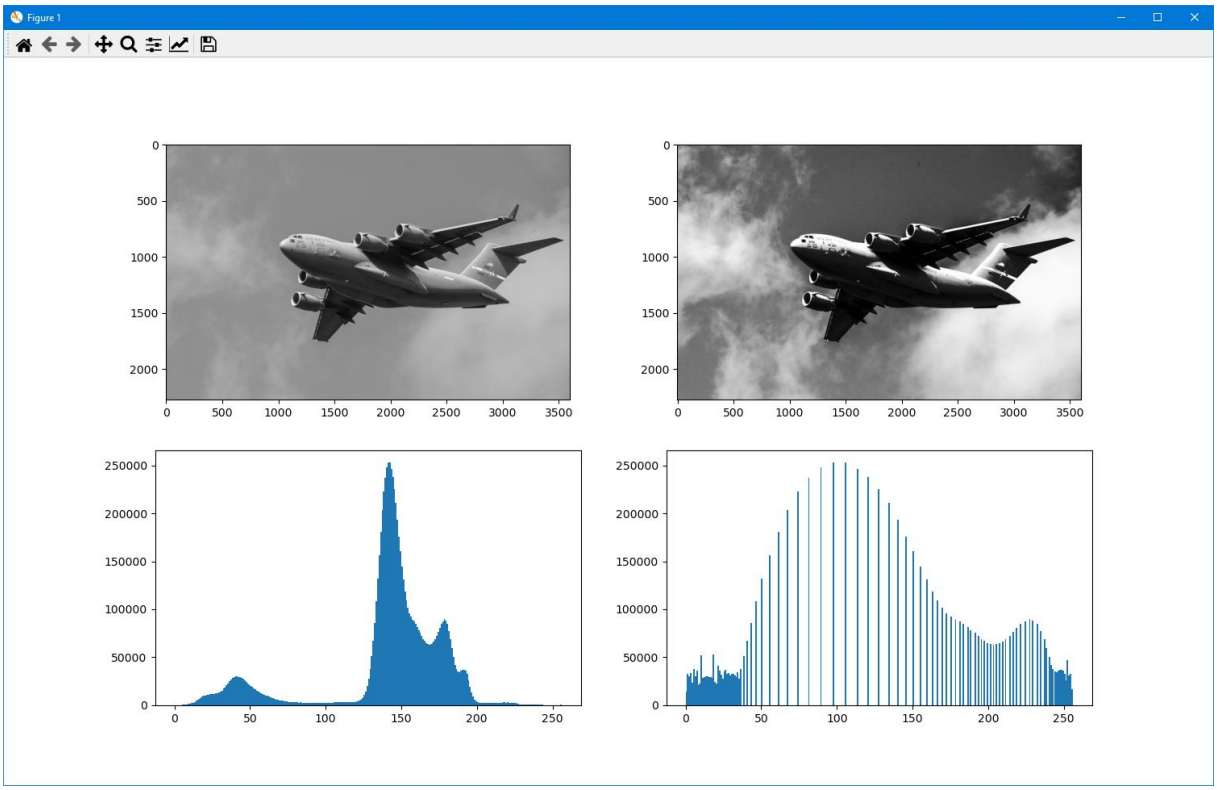


- Zanotuj uwagi z przerobionego materiału w sprawozdaniu.

1.5 Wykonaj poniższy przykład z zastosowaniem wyrónania histogramu.

- Wykonaj kod zawarty w komórkach, dokonując jego uzupełnienia np. w dodając tytuły wykresów, legendę, opisy osi, itp.

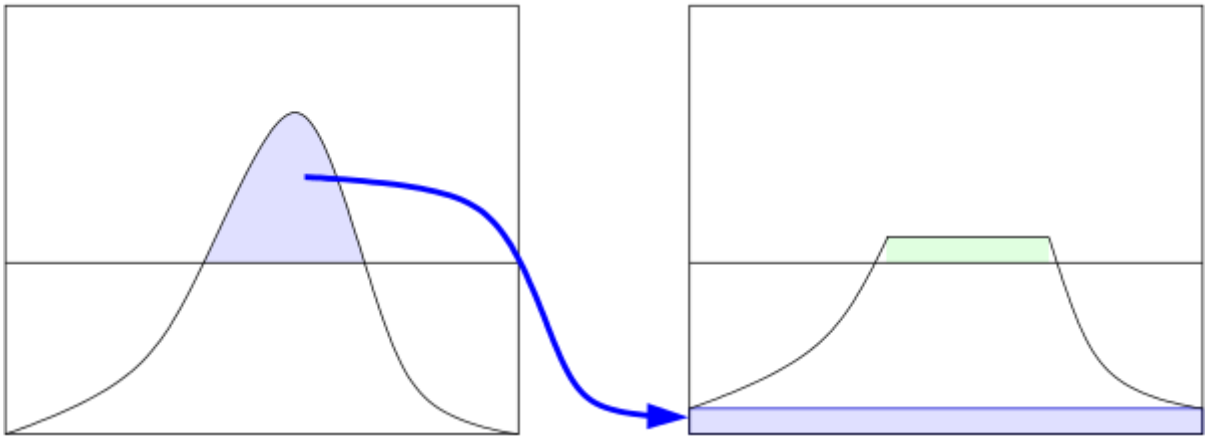
```
# %%

# import zależności
import cv2
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = [15, 10]

img = cv2.imread('images/plane.jpg', 0)
plt.subplot(221)
plt.imshow(img, cmap='gray')

plt.subplot(223)
plt.hist(img.ravel(), bins=256, range=[0, 256])

plt.subplot(222)
img_equ = cv2.equalizeHist(img)
plt.imshow(img_equ, cmap='gray')

plt.subplot(224)
plt.hist(img_equ.ravel(), bins=256, range=[0, 256])
plt.show()
```



- Zanotuj uwagi z przerobionego materiału w sprawozdaniu.

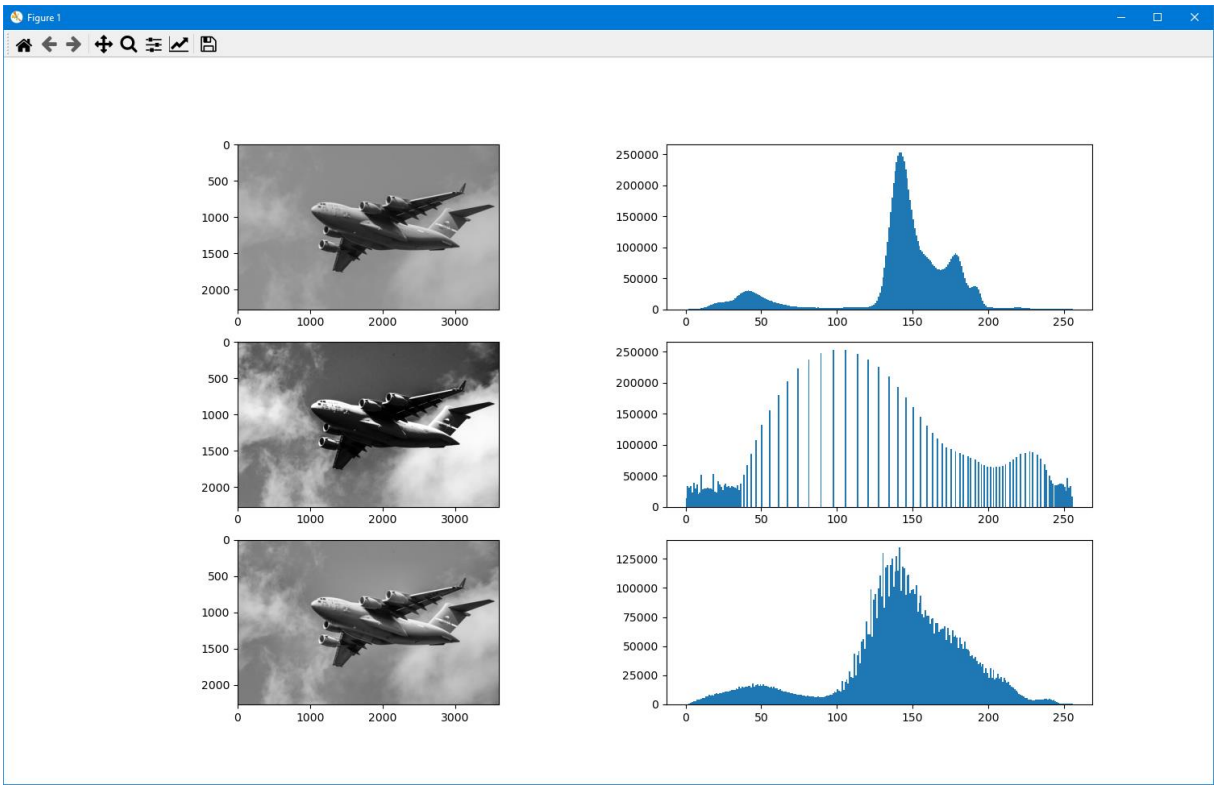
1.6 Technika CLAHE

- *Contrast Limited Adaptive Histogram Equalization*, czyli adaptacyjna metoda wyrównania histogramu.
- Opiera się na ograniczaniu wysokich wartości na histogramie i ich redystrybucję, zgodnie z rysunkiem:

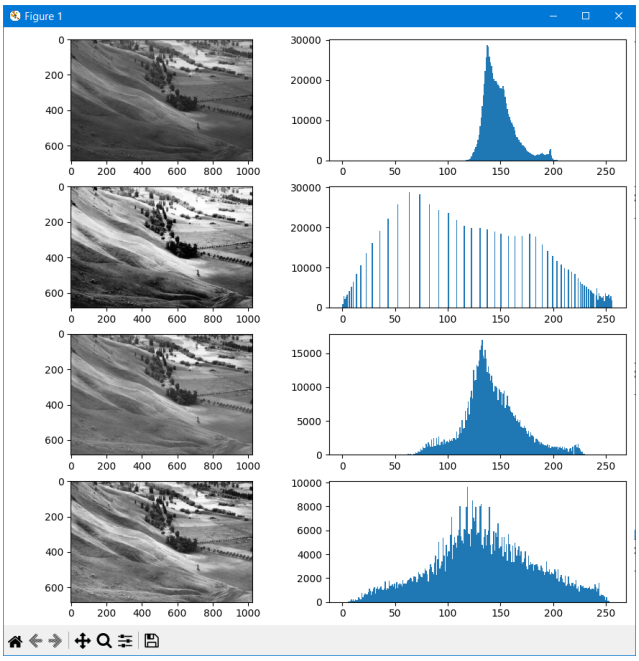


- W OpenCV wykorzystujemy w tym celu funkcję `cv2.createCLAHE()`, która przyjmuje parametry clipLimit, czyli wartość progu do limitowania kontrastu, a `tileGridSize` mówi o wielkości pojedynczych fragmentów, w których wyrównywany jest histogram.

```
# %%

# import zależności
import cv2
import numpy as np
import matplotlib.pyplot as plt


img = cv2.imread('images/plane.jpg', 0)

plt.subplot(321)
plt.imshow(img, cmap='gray')

plt.subplot(322)
plt.hist(img.ravel(), bins=256, range=[0, 256])

plt.subplot(323)
img_equ = cv2.equalizeHist(img)
plt.imshow(img_equ, cmap='gray')

plt.subplot(324)
plt.hist(img_equ.ravel(), bins=256, range=[0, 256])

plt.subplot(325)
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(4, 4))
img_equ_clahe = clahe.apply(img)
plt.imshow(img_equ_clahe, cmap='gray')

plt.subplot(326)
plt.hist(img_equ_clahe.ravel(), bins=256, range=[0, 256])
plt.show
```



- Zanotuj uwagi z przerobionego materiału w sprawozdaniu.
- Zmień wczytywany obraz na `wiki.jpg` i powtórz ostatnie doświadczenie.
    - Porównaj wizualnie obraz wyrównany metodą CLAHE z obrazem wyrównanym metodą tradycyjną.
- Na koniec zmodyfikuj kod tego doświadczenia w taki sposób, aby zastosować metodę CLAHE do obrazu `img_equ_clahe` (efekt widoczny na poniższym wykresie).



1.7 Obrazy kolorowe

- W przypadku obrazów kolorowych najczęściej stosuje się przejście do innej przestrzeni barwowej (np. LAB), a wyrównanie histogramu stosuje się tylko dla składowej L (jasność).
- Przygotuj nowy skrypt, zresetuj konsolę, wykonaj poniższy kod.

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

img_bgr = cv2.imread('images/fruits.jpg', )

img_lab = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2LAB)
clahe = cv2.createCLAHE(clipLimit=2.0,tileGridSize=(8, 8))
img_lab[..., 0] = clahe.apply(img_lab[..., 0])
img_equ = cv2.cvtColor(img_lab, cv2.COLOR_LAB2BGR)

plt.rcParams['figure.figsize'] = [8, 6]

plt.subplot(221)
plt.imshow(img_bgr[...,::-1])
plt.title("BGR")

plt.subplot(222)
plt.hist(img_bgr[..., 0].ravel(), bins=256, range=[0, 256], color='b')
plt.hist(img_bgr[..., 1].ravel(), bins=256, range=[0, 256], color='g')
plt.hist(img_bgr[..., 2].ravel(), bins=256, range=[0, 256], color='r')

plt.subplot(223)
plt.imshow(img_equ[...,::-1])
plt.title("LAB")

plt.subplot(224)
plt.hist(img_equ[..., 0].ravel(), bins=256, range=[0, 256], color='b')
plt.hist(img_equ[..., 1].ravel(), bins=256, range=[0, 256], color='g')
plt.hist(img_equ[..., 2].ravel(), bins=256, range=[0, 256], color='r')

plt.show()
```

- Zanotuj uwagi z przerobionego materiału w sprawozdaniu.
- Wykonaj podobny przykład dla przestrzeni kolorów HSV (wykres po prawej).

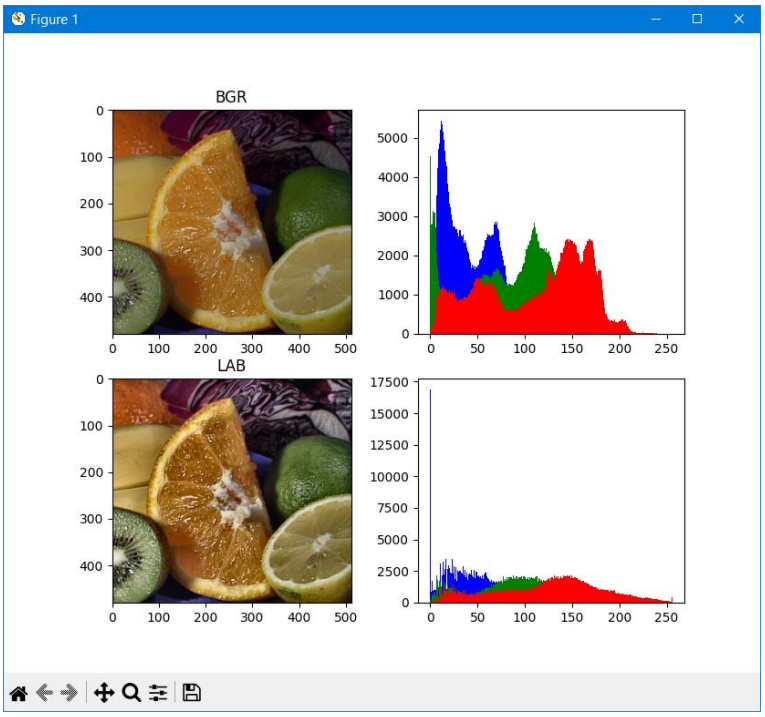


Figure 1
BGR
LAB

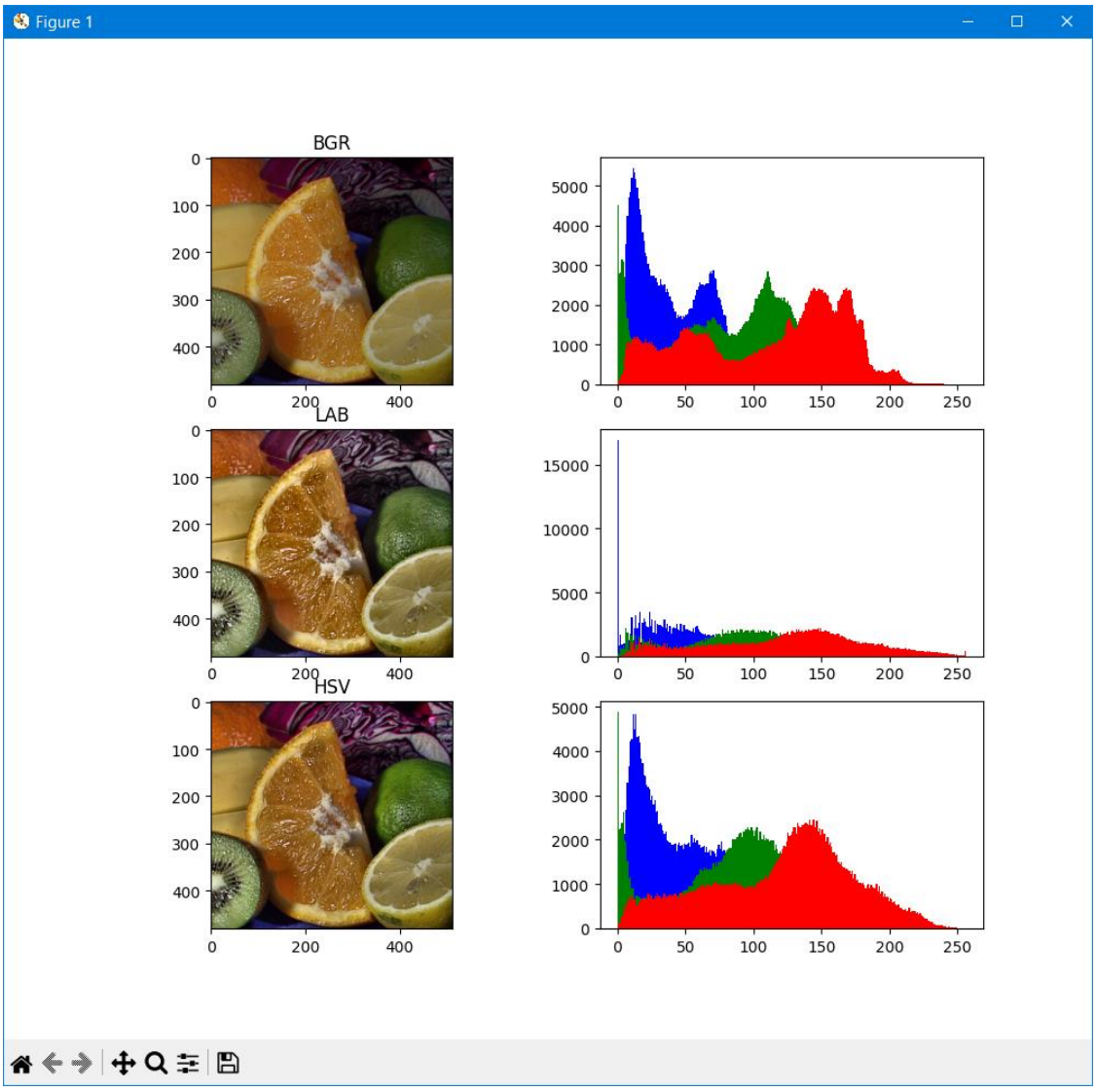


Figure 1
BGR
LAB
HSV

2 Techniki filtracji obrazu

- Podczas wykonywania ćwiczeń notuj swoje uwagi w sprawozdaniu.
- W pierwszej komórce kodu przygotowana jest funkcja `imshow`, która obsłuży zarówno obrazy w skali szarości oraz obrazy kolorowe, a ponadto odwróci kolejność kolorów: RGB->BGR.

```
# %%
# 1

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

def imshow(image):
    if len(image.shape) == 2 or (len(image.shape) == 3 and image.shape[-1]
== 1):
        plt.imshow(image, cmap='gray')
    else:
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

2.1 Badanie operacji korelacji i splotu 2D.

Zapoznanie z funkcją `filter2D` z biblioteki OpenCV
Zapoznanie z funkcją `convolve` z biblioteki SciPy

- Biblioteka OpenCV posiada funkcję `cv2.filter2D()`, która stosuje do obrazu dowolny filtr liniowy.
- Otwórz dokumentację OpenCV, znajdź opis tej funkcji i zapoznaj się z nią.
- Używane jest tu pojęcie konwolucji, ale funkcja `filter2D` wykonuje korelację!
- Na początku w ćwiczeniu sprawdzimy działanie korelacji i konwolucji za pomocą funkcji filter2D, a później użyjemy tej funkcji jako filtr uśredniający.

Sprawdzenie różnicy działania operacji konwolucji i korelacji:

- przygotujemy filtr o niesymetrycznych wartościach współczynników

```
# %%
# 2

# import biblioteki scipy
from scipy import ndimage

# wczytujemy obraz w skali szarości
src_gray = cv2.imread('images/okret.jpg', 0)
print(src_gray.shape)

# przygotowanie filtra
kernel = np.array([[1,1,1],[1,1,0],[1,0,0]])

# zastosujemy filtr konwolucyjny z biblioteki scipy
dst_conv  = ndimage.convolve(src_gray, kernel, mode='constant', cval=1.0)

# zastosowanie oryginalnego filtra do celów korelacji
dst_corr = cv2.filter2D(src_gray, -1, kernel)

# złączenie obrazów
result = np.concatenate((src_gray, dst_corr, dst_conv), axis=1)
cv2.imshow('wynik', result)
```

2.2 Rozmycie obrazu

- Rozmycie obrazu uzyskuje się przez operację konwolucji przy użyciu odpowiedniego filtra.
- Jest to przydatne przy usuwaniu szumów.
- W rzeczywistości operacja ta usuwa z obrazu treść o wysokiej częstotliwości (szumy i krawędzie), co powoduje rozmycie brzegów po zastosowaniu filtra.
- OpenCV oferuje kilka technik rozmycia.

2.2.1 Technika uśredniania

- Odbywa się to poprzez operację konwolucji za pomocą znormalizowanego filtra.
- Pobiera on średnią wszystkich pikseli znajdujących się pod obszarem filtra i zastępuje centralny element tą średnią.
- Technika może być wykonywana za pomocą funkcji `cv2.blur()`.
- Przykład: zastosowanie filtra rozmywającego z maską 3x3:

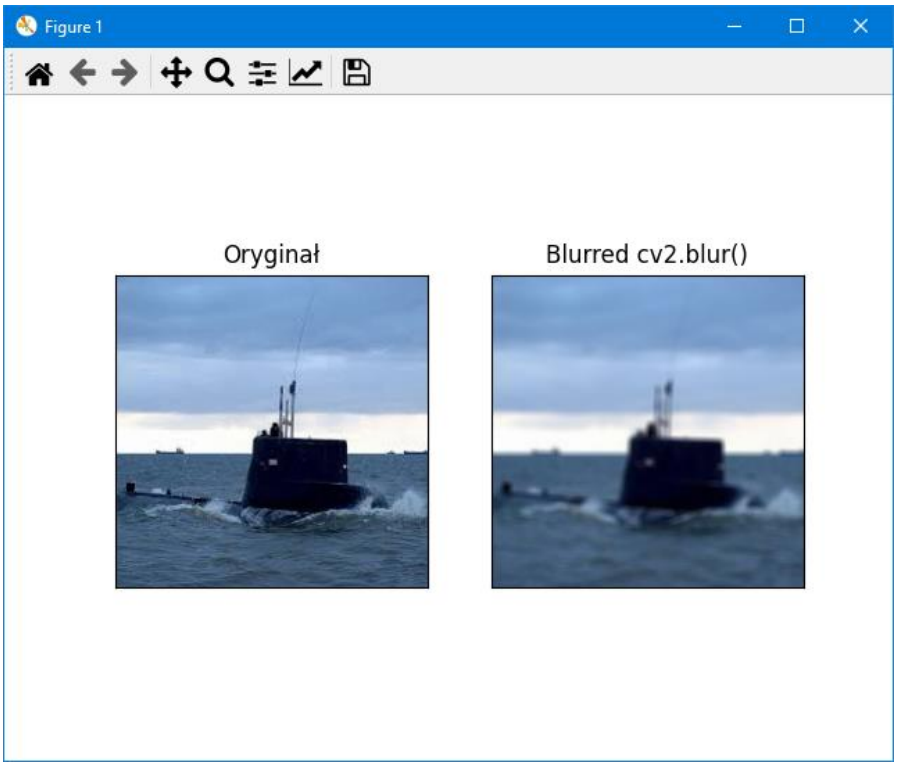
$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Filtracja uśredeniająca obrazu RGB

Funkcja `blur` z biblioteki OpenCV

```
# %%
# 3
# Filtr uśredniający - obraz RGB
# Funkcja blur()
```

```
# wczytujemy obraz RGB
src = cv2.imread('images/okret.jpg')
print(src.shape)


# zastosowanie funkcji blur jako filtra uśredniającego
blur1 = cv2.blur(src, (5,5))


# wyświetlenie obrazów oryginalnego i po uśrednieniu za pomocą plt
plt.figure()
plt.subplot(121), imshow(src), plt.title('Oryginał')
plt.xticks([]), plt.yticks([])
plt.subplot(122), imshow(blur1), plt.title('Blurred cv2.blur()')
plt.xticks([]), plt.yticks([])
plt.show()
```

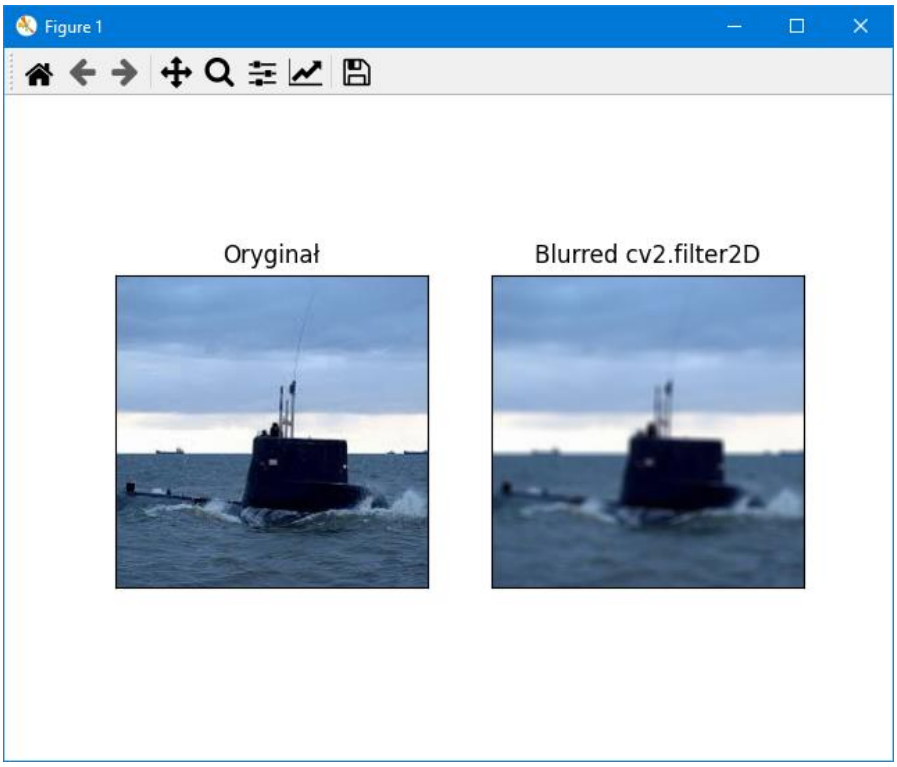


Funkcja `filter2D` z biblioteki OpenCV

```
# %%
# 4
# Filtr uśredniający - obraz RGB
# Funkcja filter2D()


# przygotowanie i wydruk maski
size = 5
kernel = 1 / size**2 * np.ones((size, size))
print(kernel)
```

```
# zastosowanie funkcji filter2D jako filtra uśredniającego
blur2 = cv2.filter2D(src, cv2.CV_8U, kernel)
print(blur2.shape)


# wyświetlenie obrazów oryginalnego i po uśrednieniu za pomocą plt
plt.figure()
plt.subplot(121), imshow(src), plt.title('Oryginał')
plt.xticks([]), plt.yticks([])
plt.subplot(122), imshow(blur2), plt.title('Blurred cv2.filter2D')
plt.xticks([]), plt.yticks([])
plt.show()
```

```
[[0.04 0.04 0.04 0.04 0.04]
 [0.04 0.04 0.04 0.04 0.04]
 [0.04 0.04 0.04 0.04 0.04]
 [0.04 0.04 0.04 0.04 0.04]
 [0.04 0.04 0.04 0.04 0.04]]
(225, 225, 3)
(225, 225, 3)
```



Filtracja uśredniająca obrazu w skali szarości

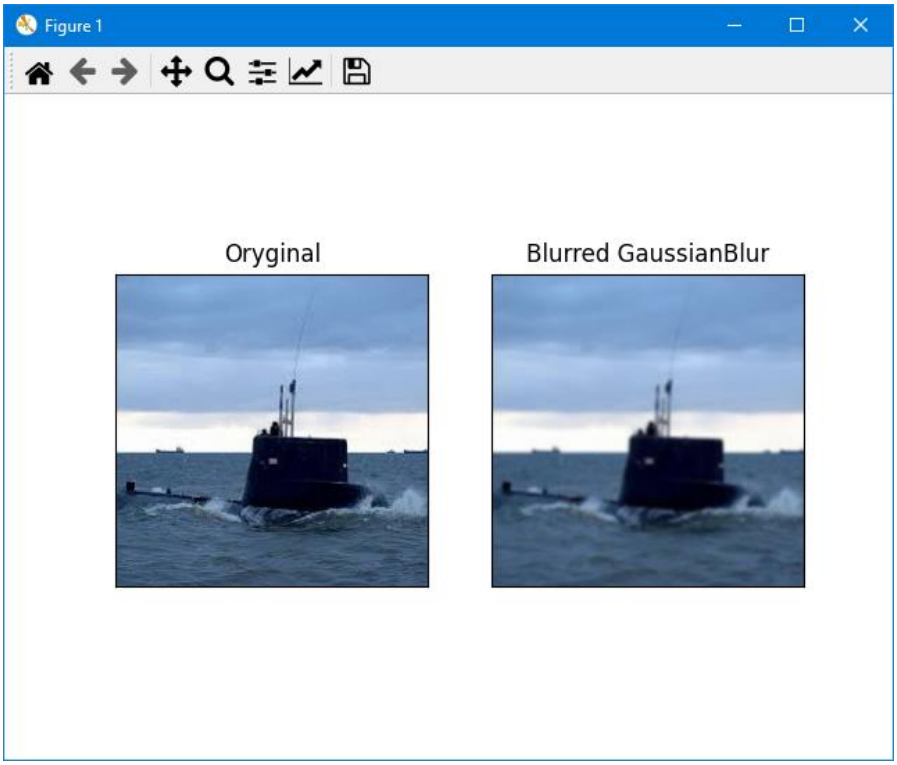
```
# %%
# 5
# Filtr uśredniający - obraz w skali szarości
# Funkcja filter2D()
```

```
# przygotowanie i wydruk maski
size = 5
kernel = 1 / size**2 * np.ones((size, size))
print(kernel)

# zastosowanie funkcji filter2D jako filtra uśredniającego
blur3 = cv2.filter2D(src_gray, -1, kernel)
print(blur3.shape)


# połączenie obrazów w poziomie i wyświetlenie za pomocą cv2
result = np.concatenate((src_gray, blur3), axis=1)
cv2.imshow('blured gray', result)
```

2.2.2 Technika filtrowania gaussowskiego

- W tym podejściu zamiast filtra typu box składającego się z jednakowych współczynników filtracji stosuje się filtr gaussowski.
- Odbywa się to za pomocą funkcji `cv2.GaussianBlur()`.
- Trzeba w niej podać wymiary jądra - wymiar ten musi być dodatni i nieparzysty.
- Powinniśmy również określić odchylenie standardowe w kierunkach X i Y, odpowiednio `sigmaX` i `sigmaY`. Jeśli podano tylko `sigmaX`, `sigmaY` jest rozumiane jako równe `sigmaX`.
- Jeśli oba podane są jako zera, to wówczas są one obliczane na podstawie wielkości filtra.
- Filtrowanie gaussowskie jest bardzo skuteczne w usuwaniu szumów gaussowskich z obrazu.
- Możliwe jest również ręczne utworzenie filtra gaussowskiego z użyciem funkcji `cv2.getGaussianKernel()`.

```
# %%
# 6
# Filtr gaussowski - obraz RGB
# Funkcja GaussianBlur()

blur_4 = cv2.GaussianBlur(src,(5,5), 0)

plt.subplot(121), imshow(src), plt.title('Oryginal')
plt.xticks([]), plt.yticks([])
plt.subplot(122), imshow(blur_4), plt.title('Blurred GaussianBlur')
plt.xticks([]), plt.yticks([])
plt.show()
```



2.2.3 Filtrowanie medianowe

- W tym przypadku funkcja `cv2.medianBlur()` pobiera medianę wszystkich pikseli znajdujących się pod obszarem jądra i centralny element zostaje zastąpiony tą wartością mediany.
- Funkcja ta jest bardzo skuteczna w przypadku zakółceń typu "sól-i-pieprz" na obrazie.
- W poprzednich filtrach centralnym elementem jest nowo obliczona wartość, która może być wartością piksela na obrazie lub nową wartością, jednak w przypadku filtra medianowego centralny element jest zawsze zastępowany pewną wartością piksela na obrazie.

- Skutecznie redukuje szum.
- Rozmiar jądra powinien być dodatnią nieparzystą liczbą całkowitą.
- W tym przykładzie jest 50%-owy szum dodany do oryginalnego obrazu.

```
# %%
# 7
# Filtr medianowy - obraz RGB
# Funkcja medianBlur()


# Generowanie szumu gaussowskiego
gauss = np.random.normal(0,1,src.size)

gauss =
gauss.reshape(src.shape[0],src.shape[1],src.shape[2]).astype('uint8')

# Dodanie szumu gaussowskiego do obrazu
src_gauss = cv2.add(src,gauss)


# zastosowanie filtra do obrazu pierwotnego, filtr staje się tu tablicą np
blur5 = cv2.medianBlur(src_gauss, 5)

# połączenie obrazów do wyświetlenia
result = np.concatenate((src, src_gauss, blur5), axis=1)
cv2.imshow('wynik', result)
```

2.2.4 Filtrowanie bilateralne

- Jest to kolejna metoda wygładzania obrazu.
- Jej ważną cechą jest zdolność do usuwania szumu z obrazu przy jednoczesnym zachowaniu ostrych, wyraźnych krawędzi.
- Jego działanie polega na zastępowaniu wartości w każdym pikselu przez ważoną wartość przylegających pikseli.
- Funkcja `cv2.bilateralFilter()` oprócz obrazu przyjmuje parametry:
    - `d` mówiący o średnicy analizowanego otoczenia (wielkość filtra - im większy, tym wolniejszy),

- `sigmaColor` i `sigmaSpace` mówią o stopniu, w jakim sąsiednie piksele wpływają na siebie przy wygładzaniu (dla pierwszego: większa wartość sprawi, że mieszane będą bardziej odległe od siebie kolory; dla drugiego - bardziej odległe piksele).

```
# %%
# 8
# Filtr bilateralny - obraz RGB
# Funkcja bilateralFilter()


blur6 = cv2.bilateralFilter(src,
                            d=11,
                            sigmaColor=60,
                            sigmaSpace=60)

blur7 = cv2.bilateralFilter(src_gauss,
                            d=11,
                            sigmaColor=60,
                            sigmaSpace=60)

plt.figure()
plt.subplot(221), imshow(src), plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(222), imshow(blur6), plt.title('Blur6 bilateralFilter')
plt.xticks([]), plt.yticks([])
plt.subplot(223), imshow(src_gauss), plt.title('Gaussian noise')
plt.xticks([]), plt.yticks([])
plt.subplot(224), imshow(blur7), plt.title('Blur7 bilateralFilter')
plt.xticks([]), plt.yticks([])
plt.show()

# result = np.concatenate((src_gauss, blur7), axis=1)
# cv2.imshow('wynik', result)
```

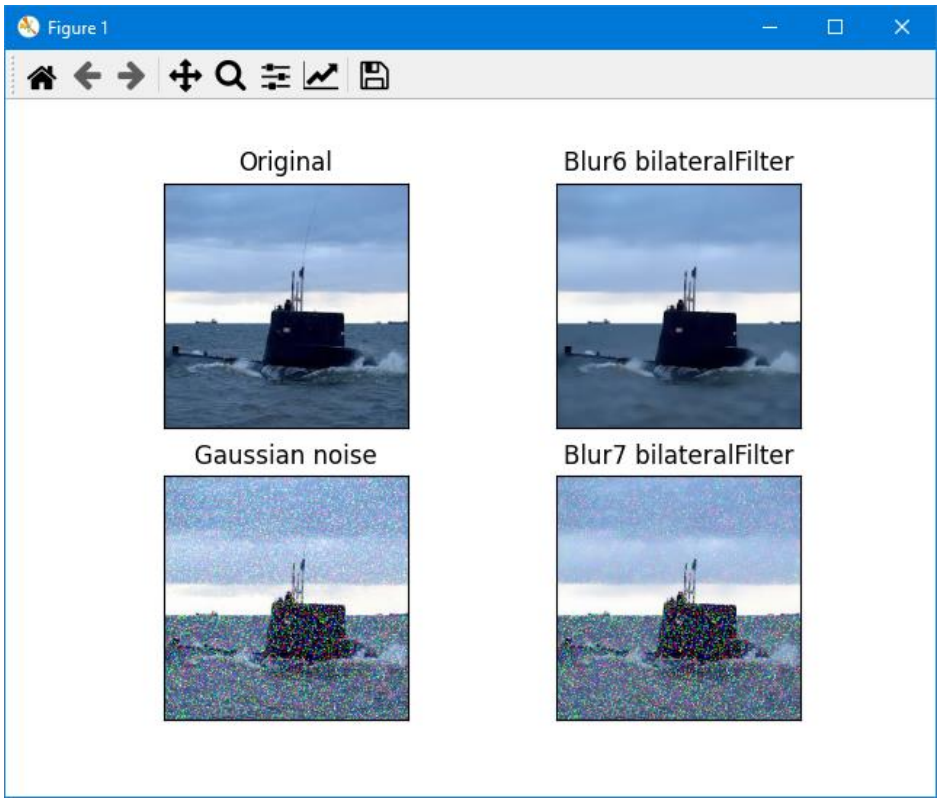


Figure 1
Original
Blur6 bilateralFilter
Gaussian noise
Blur7 bilateralFilter

2.3 Wyostrzanie obrazu

- Wysotrzanie obrazu uzyskuje się, podobnie jak rozmycie przez operację konwolucji przy użyciu odpowiedniego filtra.
- Wynikiem działania tego typu filtrów jest podkreślenie, uwypuklenie elementów obrazu o dużej częstotliwości poprzez zwiększenie ich jasności, koloru itp.
- Dla obrazu jako całości efektem jest zazwyczaj zwiększenie kontrastu poprzez podkreślenie ostrych krawędzi obiektów.
- Tak wyglądałby przykładowy filtr wyostrzający 3x3:

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

```
# %%
# 9
# Wyostrzenie obrazu

kernel = np.array([[-1,-1,-1], [-1,9,-1], [-1,-1,-1]])
print(kernel)

sharp = cv2.filter2D(src, cv2.CV_8U, kernel)

plt.subplot(121), imshow(src), plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122), imshow(sharp), plt.title('Sharpened')
plt.xticks([]), plt.yticks([])
plt.show()
```

```
[[-1 -1 -1]
 [-1  9 -1]
 [-1 -1 -1]]
```