

LAB 02: BADANIE ALGORYTMÓW DEMOZAIKINGU DLA DANYCH POBRANYCH Z CFA

Karol Działowski

nr albumu: 39259
przedmiot: Widzenie komputerowe

Szczecin, 11 marca 2021

Spis treści

1	Cel laboratorium	1
2	Wygenerowanie symulowanego obrazu z CFA	2
3	Prosta interpolacja	3
4	Algorytm Malvara	5
5	Porównanie wyników	7
5.1	PSNR	8
5.2	Zebrańe dane	9
	Bibliografia	9

1 Cel laboratorium

Celem laboratorium było zapoznanie z wybranymi algorytmami demozaikingu i badanie ich wpływu na jakość wynikowego obrazu RGB.

Za pomocą języka Python, przy użyciu podstawowych poleceń tego języka, należało wykonać

następujące algorytmy demosaicingu:

- algorytm interpolacji na podstawie najbliższych sąsiadów,
- algorytm Malvara.

Dodatkowo należało obiektywnie ocenić jakości obrazów po demozaikingu. W tym celu wybrano metrykę PSNR.

2 Wygenerowanie symulowanego obrazu z CFA

Wygenerowano obrazy zgodnie z schematem budowy filtra CFA (Bayera), tak aby w obrazie pozostawały pojedyncze składowe R, G i B w każdym punkcie obrazu. Przykładowe obrazy wynikowe przedstawiono na rysunku [1](#).

Kod źródłowy 1: Symulowanie obrazu z matrycy CFA

Źródło: Opracowanie własne

```
1 def img_to_cfa(img):
2     """
3     Creates mosaic from rgb image
4
5     Simulates output of a sensor with a Bayer filter
6
7     :param img: input image with 3 channels with shape (height, width, 3)
8     :return: mosaic image with shape (height, width)
9     """
10    mosaic = np.zeros((img.shape[0], img.shape[1]))
11    height, width, _ = img.shape
12
13    for i in range(height):
14        for j in range(width):
15            channel = bayer_channel_for_index(i, j)
16            mosaic[i, j] = img[i, j, channel]
17
18    return mosaic
19
20
21 def bayer_channel_for_index(i, j):
22     """
23     Calculates channel based on Bayer arrangement of color pixels
24
25     :param i: position in vertical axis (height)
26     :param j: position in horizontal axis (width)
27     :return: channel number, RED = 0, GREEN = 1, BLUE = 2
28     """
29    if i % 2 == 0:
30        if j % 2 == 0:
31            return GREEN
32    else:
```

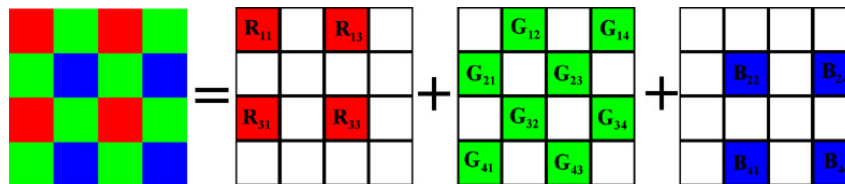
```

33         return RED
34     else:
35         if j % 2 != 0:
36             return GREEN
37         else:
38             return BLUE

```

3 Prosta interpolacja

Najprostszym sposobem na odtworzenie obrazu z matrycy CFA jest interpolacja koloru na podstawie 4 najbliższych sąsiadów.



Rysunek 2: Typowa matryca CFA. Na matrycy występuje dwa razy więcej pikseli zielonych od czerwonych i niebieskich. Interpolacje przeprowadza się na podstawie 4 najbliższych sąsiadów.

Źródło: https://ivrlwww.epfl.ch/research/past_topics/demosaicing.html

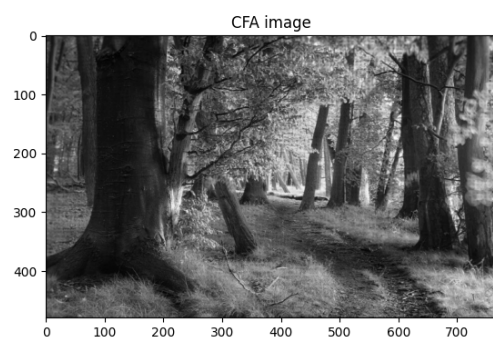
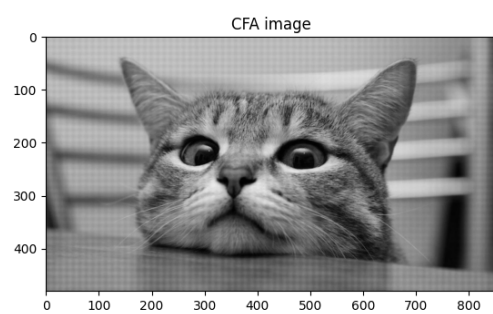
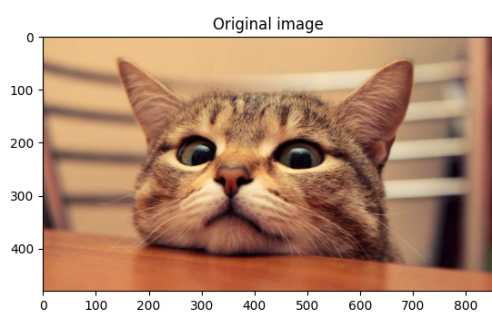
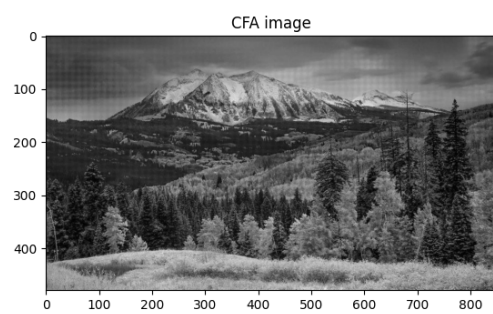
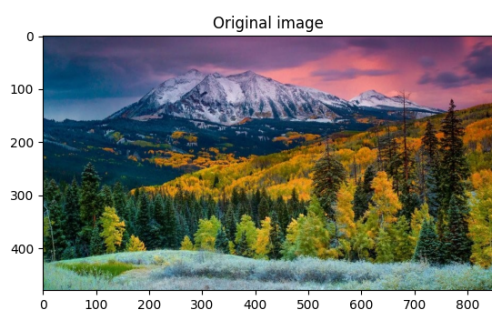
Kod źródłowy 2: Przykład kodu

Źródło: Opracowanie własne

```

1  def simple_interpolation(bayer_mosaics):
2      """
3      Simple interpolation using a mean of 9 nearest neighbors
4      :param bayer_mosaics: input image of shape (height, width, 3) with np.nan where
5      value is unknown
6      :return: rgb image with interpolated values in channels
7      """
8      height, width, _ = bayer_mosaics.shape
9      output = np.copy(bayer_mosaics)
10     for i in range(height):
11         for j in range(width):
12             # Interpolate red
13             if bayer_channel_for_index(i, j) != 0:
14                 output[i, j, 0] = mean_neighbors(bayer_mosaics[:, :, 0], i, j)
15             # Interpolate green
16             if bayer_channel_for_index(i, j) != 1:
17                 output[i, j, 1] = mean_neighbors(bayer_mosaics[:, :, 1], i, j)
18             # Interpolate blue
19             if bayer_channel_for_index(i, j) != 2:
20                 output[i, j, 2] = mean_neighbors(bayer_mosaics[:, :, 2], i, j)
21     return output.astype(int)
22
23 @jit
24 def mean_neighbors(channel, i, j):
25     """

```



Rysunek 1: Testowane obrazy i symulowane obrazy (RAW) z matrycy CFA

```

25     Calculate mean from 9 nearest neighbors for a channel
26
27     :param channel: matrix with (height, width) shape - single channel of mosaic.
    Filled with values and np.nan
28     :param i: horizontal index (height)
29     :param j: vertical index (width)
30     :return: mean of 9 nearest neighbors
31     """
32     height, width = channel.shape
33     lower_bound = i - 1 if i - 1 >= 0 else 0
34     upper_bound = i + 1 if i + 1 < height else height - 1
35     left_bound = j - 1 if j - 1 >= 0 else 0
36     right_bound = j + 1 if j + 1 < width else width - 1
37     return np.nanmean(
38         channel[lower_bound : upper_bound + 1, left_bound : right_bound + 1]
39     )

```

4 Algorytm Malvara

Algorytm zaprezentowany w pracy *High-Quality Linear Interpolation For Demosaicing Of Bayer-Patterned Color Images* umożliwia polepszenie wynikowego obrazu w metryce PSNR o ponad 5.5 dB w porównaniu z prostą interpolacją [1].

Algorytm polega na wykonaniu operacji splotowych korzystając z zdefiniowanych przez autorów współczynników zależnie od pozycji aktualnego piksela. Współczynniki przedstawiono na rysunku 3.

Kod źródłowy 3: Przykład kodu

Źródło: Opracowanie własne

```

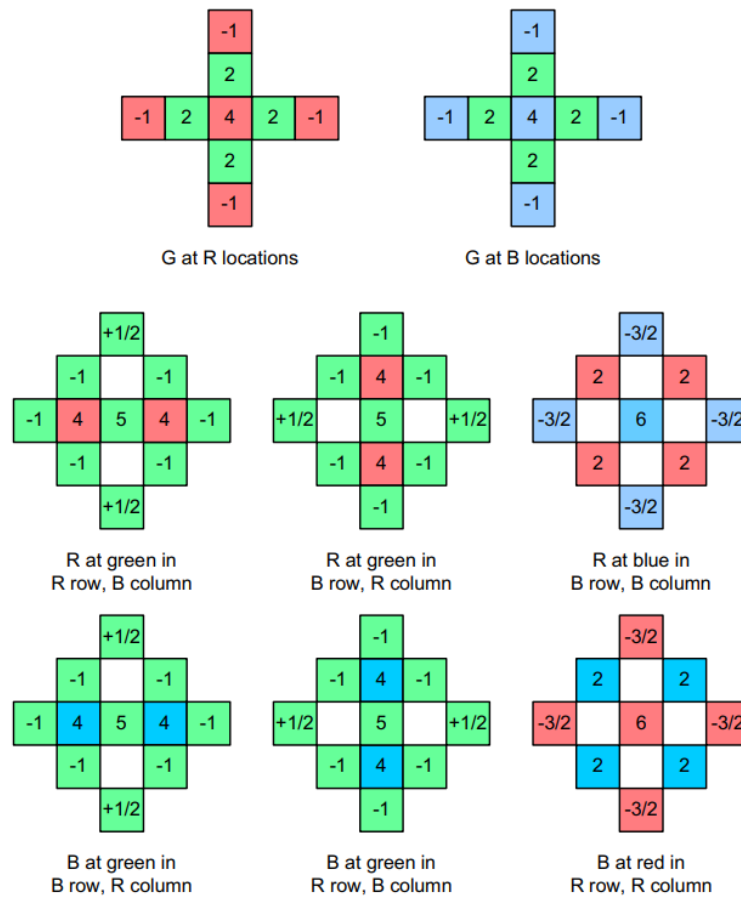
1  def demosaicing_malvar(mosaic):
2      """
3      Malvar algorithm
4
5      Henrique S. Malvar, Li-wei He, and Ross Cutler , High-Quality Linear
    Interpolation For Demosaicing Of Bayer-Patterned Color Images
6      :param bayer_mosaics: input image of shape (height, width, 3) with np.nan where
    value is unknown
7      :return: rgb image with interpolated values in channels
8      """
9      mosaic = add_padding(mosaic, 2)
10     height, width = mosaic.shape
11
12     bayer_mosaics = np.full((height, width, 3), np.nan, dtype=int)
13
14     # ... ommited definitions of kernels
15
16     for i in range(2, height - 2):
17         for j in range(2, width - 2):

```

```

18         # Green channel
19         if (
20             bayer_channel_for_index(i, j) == 0 or bayer_channel_for_index(i, j)
21             == 2
22         ): # if in Red or Blue
23             c = mosaic[i - 2 : i + 2 + 1, j - 2 : j + 2 + 1]
24             value = (GR_GB * c).sum()
25             bayer_mosaics[i, j, 1] = int(value)
26         else:
27             bayer_mosaics[i, j, 1] = mosaic[i, j]
28         # Red channel
29         if bayer_channel_for_index(i, j) == 1: # if R at green
30             if i % 2 == 0 and j % 2 == 0: # if R row and B column
31                 c = mosaic[i - 2 : i + 2 + 1, j - 2 : j + 2 + 1]
32                 value = (Rg_RB_Bg_BR * c).sum()
33                 bayer_mosaics[i, j, 0] = int(value)
34             else:
35                 c = mosaic[i - 2 : i + 2 + 1, j - 2 : j + 2 + 1]
36                 value = (Rg_BR_Bg_RB * c).sum()
37                 bayer_mosaics[i, j, 0] = int(value)
38         elif bayer_channel_for_index(i, j) == 2: # if R at blue in B row B
39             column
40             c = mosaic[i - 2 : i + 2 + 1, j - 2 : j + 2 + 1]
41             value = (Rb_BB_Br_RR * c).sum()
42             bayer_mosaics[i, j, 0] = int(value)
43         else:
44             bayer_mosaics[i, j, 0] = mosaic[i, j]
45         # Blue channel
46         if bayer_channel_for_index(i, j) == 1: # if B at green
47             if i % 2 == 0 and j % 2 == 0: # if R row and B column
48                 c = mosaic[i - 2 : i + 2 + 1, j - 2 : j + 2 + 1]
49                 value = (Rg_BR_Bg_RB * c).sum()
50                 bayer_mosaics[i, j, 2] = int(value)
51             else:
52                 c = mosaic[i - 2 : i + 2 + 1, j - 2 : j + 2 + 1]
53                 value = (Rg_RB_Bg_BR * c).sum()
54                 bayer_mosaics[i, j, 2] = int(value)
55         elif bayer_channel_for_index(i, j) == 0: # if B at red in B row B column
56             c = mosaic[i - 2 : i + 2 + 1, j - 2 : j + 2 + 1]
57             value = (Rb_BB_Br_RR * c).sum()
58             bayer_mosaics[i, j, 2] = int(value)
59         else:
60             bayer_mosaics[i, j, 2] = mosaic[i, j]
61
62     return bayer_mosaics[2:-2, 2:-2, :]

```

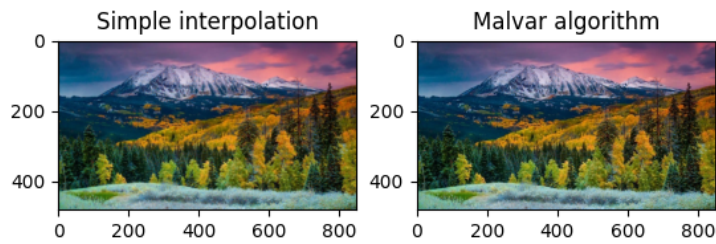


Rysunek 3: Współczynniki przedstawione w pracy Malvara

Źródło: High-Quality Linear Interpolation For Demosaicing Of Bayer-Patterned Color Images [1]

5 Porównanie wyników

Przykładowy obraz po demosaikingu przedstawiono na rysunku 4. W celu porównania obrazów z oryginałem wykorzystano metrykę PSNR.



Rysunek 4: Wynik demozaikingu dla obrazu 1

Źródło: Opracowanie własne

5.1 PSNR

Szczytowy stosunek sygnału do szumu, (ang. *peak signal-to-noise ratio (PSNR)*) – stosunek maksymalnej mocy sygnału do mocy szumu zakłócającego ten sygnał.

Najczęściej PSNR stosowany jest do oceny jakości kodeków wykorzystujących stratną kompresję obrazków. W takim przypadku sygnałem są nieskompresowane dane źródłowe, a szumem – artefakty (zniekształcenia) spowodowane zastosowaniem kompresji stratnej.

W celu wyznaczenie PSNR, należy najpierw obliczyć współczynnik MSE (błąd średniokwadratowy) bazując na obu porównywanych obrazkach za pomocą wzoru:

$$MSE = \frac{1}{n \cdot m} \sum_{i=1}^N \sum_{j=1}^M ([f(i, j) - f'(i, j)])^2 \quad (1)$$

gdzie:

n, m - wymiary obrazu w pikselach,

$f(i, j)$ - wartość piksela o współrzędnych (i, j) obrazu oryginalnego

$f'(i, j)$ - wartość piksela o współrzędnych (i, j) obrazu poddanego kompresji i dekompresji

Następnie wyliczoną wartość MSE należy podstawić do końcowego wzoru:

$$PSNR = 10 \cdot \log_{10} \frac{[\max(f(i, j))]^2}{MSE} \quad (2)$$

gdzie:

$\max(f(i, j))$ - wartość maksymalna danego sygnału; w przypadku obrazów zwykle jest to wartość stała, np. dla obrazów monochromatycznych o reprezentacji 8-bitowej wynosi 255.

Kod źródłowy 4: Implementacja PSNR

Źródło: Opracowanie własne

```
1 def psnr(original_image, referenced_image):
2     mse = np.mean((original_image - referenced_image) ** 2)
3     if mse == 0: # MSE is zero means no noise is present in the signal .
4         # Therefore PSNR have no importance.
5         return 100
6     max_pixel = 255.0
7     psnr = 20 * math.log10(max_pixel / math.sqrt(mse))
8     return psnr
```

5.2 Zebrane dane

Tabela 1: Porównanie PSNR dla zaimplementowanych metod

	PSNR interpolacja [dB]	PSNR Malvar [dB]
image1.jpg	26.949965	31.641772
image2.jpg	35.317820	36.663621
image3.jpg	26.968871	31.604858
średnia	29.745552	33.303417

Dla badanych obrazów wynik średni wynik PSNR jest lepszy o 3.5 dB dla algorytmu Malvara. Czas demozaikingu algorytmem Malvara dla pojedynczego obrazu wynosi średnio 31 sekund w porównaniu do 26 sekund dla prostej interpolacji dwuliniowej.

Bibliografia

- [1] Malvar H. S., He L.-w., Cutler R.: High-quality linear interpolation for demosaicing of bayer-patterned color images, *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, pp. iii–485, 2004.