Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

Wydział Informatyki

# LAB 02: BADANIE ALGORYTMÓW DEMOZAIKINGU DLA DANYCH POBRANYCH Z CFA

**Karol Działowski**

nr albumu: 39259
przedmiot: Widzenie komputerowe

Szczecin, 20 czerwca 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 Zebrane dane . . . . . 9

**Bibliografia** **9**

## 1 Cel laboratorium

Celem laboratorium było zapoznanie z wybranymi algorytmami demozaikingu i badanie ich wpływu na jakośc 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.

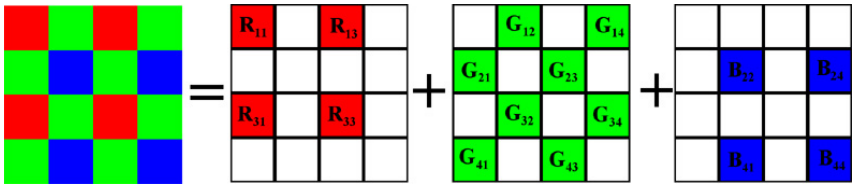
**Kod źródłowy 1:** Symulowanie obrazu z matrycy CFA
Źródło: Opracowanie własne

```
def img_to_cfa(img):
    """
    Creates mosaic from rgb image

    Simulates output of a sensor with a Bayer filter

    :param img: input image with 3 channels with shape (height, width, 3)
    :return: mosaic image with shape (height, width)
    """
    mosaic = np.zeros((img.shape[0], img.shape[1]))
    height, width, _ = img.shape

    for i in range(height):
        for j in range(width):
            channel = bayer_channel_for_index(i, j)
            mosaic[i, j] = img[i, j, channel]

    return mosaic


def bayer_channel_for_index(i, j):
    """
    Calculates channel based on Bayer arrangement of color pixels

    :param i: position in vertical axis (height)
    :param j: position in horizontal axis (width)
    :return: channel number, RED = 0, GREEN = 1, BLUE = 2
    """
    if i % 2 == 0:
        if j % 2 == 0:
            return GREEN
        else:
```

```
            return RED
    else:
        if j % 2 != 0:
            return GREEN
        else:
            return BLUE
```

# 3 Prosta interpolacja

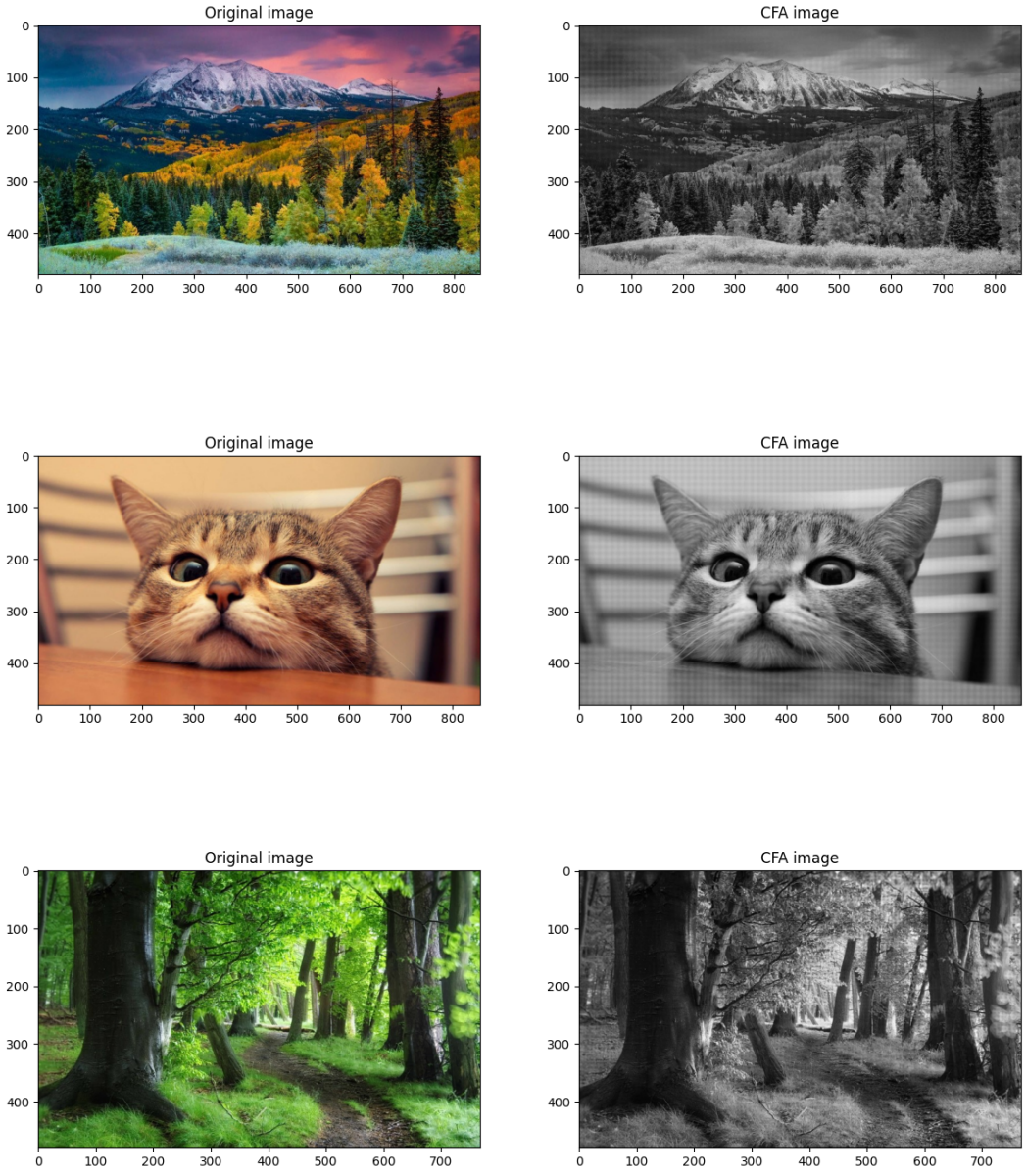
Najprostszym sposobem na odtworzenie obrazu z matrycy CFA jest interpolacja koloru na podstawie 4 najbliższych sąsiadów.




**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

```
def simple_interpolation(bayer_mosaics):
    """
    Simple interpolation using a mean of 9 nearest neighbors
    :param bayer_mosaics: input image of shape (height, width, 3) with np.nan where value is unknown
    :return: rgb image with interpolated values in channels
    """
    height, width, _ = bayer_mosaics.shape
    output = np.copy(bayer_mosaics)
    for i in range(height):
        for j in range(width):
            # Interpolate red
            if bayer_channel_for_index(i, j) != 0:
                output[i, j, 0] = mean_neighbors(bayer_mosaics[:, :, 0], i, j)
            # Interpolate green
            if bayer_channel_for_index(i, j) != 1:
                output[i, j, 1] = mean_neighbors(bayer_mosaics[:, :, 1], i, j)
            # Interpolate blue
            if bayer_channel_for_index(i, j) != 2:
                output[i, j, 2] = mean_neighbors(bayer_mosaics[:, :, 2], i, j)
    return output.astype(int)

@jit
def mean_neighbors(channel, i, j):
    """
```




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

```
    Calculate mean from 9 nearest neighbors for a channel

    :param channel: matrix with (height, width) shape - single channel of mosaic. Filled with values and np.nan
    :param i: horizontal index (height)
    :param j: vertical index (width)
    :return: mean of 9 nearest neighbors
    """
    height, width = channel.shape
    lower_bound = i - 1 if i - 1 >= 0 else 0
    upper_bound = i + 1 if i + 1 < height else height - 1
    left_bound = j - 1 if j - 1 >= 0 else 0
    right_bound = j + 1 if j + 1 < width else width - 1
    return np.nanmean(
        channel[lower_bound : upper_bound + 1, left_bound : right_bound + 1]
    )
```

# 4 Algorytm Malvara

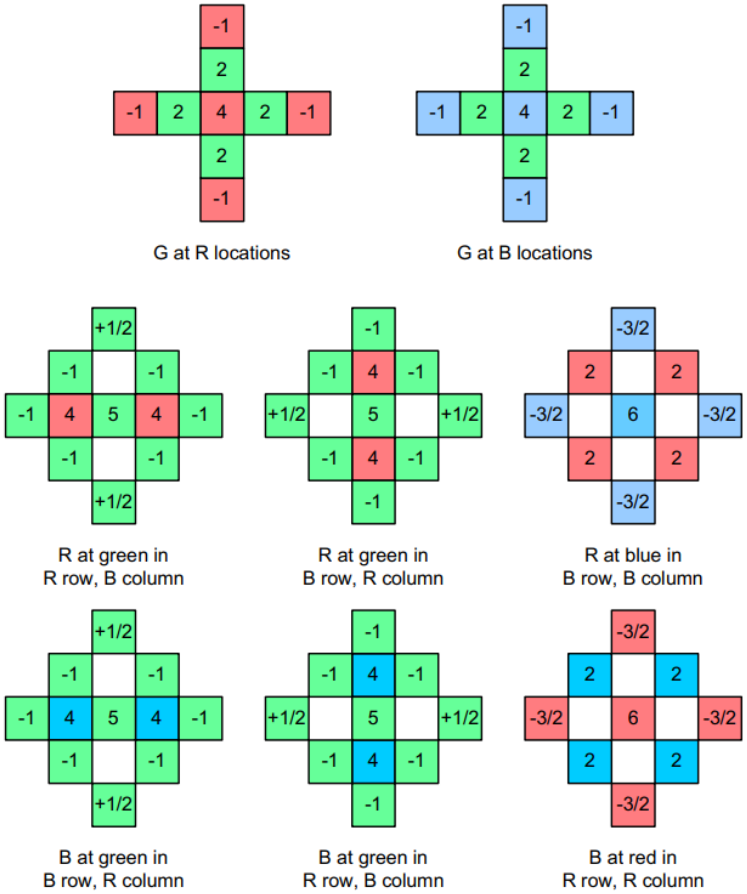
Algorytm zaprezentowany w pracy *High-Quality Linear Interpolation For Demosaicing Of Bayer-Patterned Color Images* umożliwia polepszenie wynikowego obrazu w metryce PSNR o ponad 5.5 dB w porównaniu z prostą interpolacją [1].

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

```
def demosaicing_malvar(mosaic):
    """
    Malvar algorithm

    Henrique S. Malvar, Li-wei He, and Ross Cutler , High-Quality Linear Interpolation For Demosaicing Of Bayer-Patterned Color Images
    :param bayer_mosaics: input image of shape (height, width, 3) with np.nan where value is unknown
    :return: rgb image with interpolated values in channels
    """
    mosaic = add_padding(mosaic, 2)
    height, width = mosaic.shape

    bayer_mosaics = np.full((height, width, 3), np.nan, dtype=int)

    # ... ommited definitions of kernels

    for i in range(2, height - 2):
        for j in range(2, width - 2):
```

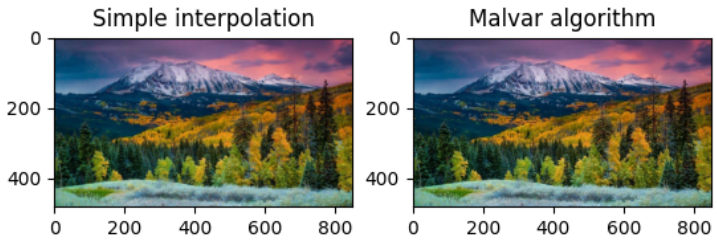
```
            # Green channel
            if (
                bayer_channel_for_index(i, j) == 0 or bayer_channel_for_index(i, j)
== 2
            ):  # if in Red or Blue
                c = mosaic[i - 2 : i + 2 + 1, j - 2 : j + 2 + 1]
                value = (GR_GB * c).sum()
                bayer_mosaics[i, j, 1] = int(value)
            else:
                bayer_mosaics[i, j, 1] = mosaic[i, j]
            # Red channel
            if bayer_channel_for_index(i, j) == 1:  # if R at green
                if i % 2 == 0 and j % 2 == 0:  # if R row and B column
                    c = mosaic[i - 2 : i + 2 + 1, j - 2 : j + 2 + 1]
                    value = (Rg_RB_Bg_BR * c).sum()
                    bayer_mosaics[i, j, 0] = int(value)
                else:
                    c = mosaic[i - 2 : i + 2 + 1, j - 2 : j + 2 + 1]
                    value = (Rg_BR_Bg_RB * c).sum()
                    bayer_mosaics[i, j, 0] = int(value)
            elif bayer_channel_for_index(i, j) == 2:  # if R at blue in B row B
column
                c = mosaic[i - 2 : i + 2 + 1, j - 2 : j + 2 + 1]
                value = (Rb_BB_Br_RR * c).sum()
                bayer_mosaics[i, j, 0] = int(value)
            else:
                bayer_mosaics[i, j, 0] = mosaic[i, j]
            # Blue channel
            if bayer_channel_for_index(i, j) == 1:  # if B at green
                if i % 2 == 0 and j % 2 == 0:  # if R row and B column
                    c = mosaic[i - 2 : i + 2 + 1, j - 2 : j + 2 + 1]
                    value = (Rg_BR_Bg_RB * c).sum()
                    bayer_mosaics[i, j, 2] = int(value)
                else:
                    c = mosaic[i - 2 : i + 2 + 1, j - 2 : j + 2 + 1]
                    value = (Rg_RB_Bg_BR * c).sum()
                    bayer_mosaics[i, j, 2] = int(value)
            elif bayer_channel_for_index(i, j) == 0:  # if B at red in B row B column
                c = mosaic[i - 2 : i + 2 + 1, j - 2 : j + 2 + 1]
                value = (Rb_BB_Br_RR * c).sum()
                bayer_mosaics[i, j, 2] = int(value)
            else:
                bayer_mosaics[i, j, 2] = mosaic[i, j]

    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 demozaikingu 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:

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

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:

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

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

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