Wstęp do multimediów

Podstawowe przetwarzanie obrazów

Jakub Robaczewski

Wstęp:

Na początku wyliczyłem indeks obrazu, na którym będę przeprowadzać operacje.

$$304119\ mod\ 36 = 27$$

Wyliczony indeks odpowiada obrazowi przedstawiającemu dwie papugi.

Pierwszym krokiem było stworzenie kilku funkcji pomocniczych do wyświetlania i zapisywania oraz zmiennych (zawierających obrazy). Wykorzystałem też funkcje zawarte w skrypcie.

```
IMAGE_COLOR_URL = "parrots_col.png"
IMAGE_MONO_URL = "parrots_mono.png"
IMAGE_COLOR = cv2.imread(IMAGE_COLOR_URL, cv2.IMREAD_UNCHANGED)
IMAGE_MONO = cv2.imread(IMAGE_MONO_URL, cv2.IMREAD_UNCHANGED)

def cv_imshow(img, img_title="image"):
    """
    Funkcja do wyświetlania obrazu w wykorzystaniem okna OpenCV.
    Wykonywane jest przeskalowanie obrazu z rzeczywistymi lub 16-bitowymi całkowitoliczbowymi wartościami
pikseli,
    żeby jedną funkcją wywietlać obrazy różnych typów.
    """
    # cv2.namedWindow(img_title, cv2.WINDOW_AUTOSIZE) # cv2.WINDOW_NORMAL

    if (img.dtype == np.float32) or (img.dtype == np.float64):
        img_ = img / 255
    elif img.dtype == np.int16:
        img_ = img * 128
    else:
        img_ = img
    cv2.imshow(img_title, img_)
    cv2.waitKey(1)


def cv_imwrite(path, name, image, params=None):
    if not os.path.isdir(path):
        os.makedirs(path)
    cv2.imwrite(path + "/" + name, image, params)


def calc_entropy(hist):
    pdf = hist / hist.sum()
    entropy = -sum([x * np.log2(x) for x in pdf if x != 0])
    return entropy


def plt_hist(x=None, y=None, title=None, xlim=None):
    if y is not None:
        if x is not None:
            plt.plot(x, y)
        else:
            plt.plot(y)
    else:
```

```
        return None

    if title is not None:
        plt.title(title)
    if xlim is not None:
        plt.xlim(xlim)


def dwt(img):
    """
    Bardzo prosta i podstawowa implementacja, nie uwzględniająca efektywnych metod obliczania DWT
    i dopuszczająca pewne niedokładności.
    """
    maskL = np.array([0.02674875741080976, -0.01686411844287795, -0.07822326652898785, 0.2668641184428723,
                      0.6029490182363579, 0.2668641184428723, -0.07822326652898785, -0.01686411844287795,
                      0.02674875741080976])
    maskH = np.array([0.09127176311424948, -0.05754352622849957, -0.5912717631142470, 1.115087052456994,
                      -0.5912717631142470, -0.05754352622849957, 0.09127176311424948])

    bandLL = cv2.sepFilter2D(img, -1, maskL, maskL)[::2, ::2]
    bandLH = cv2.sepFilter2D(img, cv2.CV_16S, maskL, maskH)[::2, ::2]
    bandHL = cv2.sepFilter2D(img, cv2.CV_16S, maskH, maskL)[::2, ::2]
    bandHH = cv2.sepFilter2D(img, cv2.CV_16S, maskH, maskH)[::2, ::2]

    return bandLL, bandLH, bandHL, bandHH


def calc_mse_psnr(img1, img2):
    """ Funkcja obliczająca MSE i PSNR dla podanych obrazów, zakładana wartość pikseli z przedziału [0, 255].
"""

    imax = 255. ** 2  ### maksymalna wartość sygnału -> 255
    """
    W różnicy obrazów istotne są wartości ujemne, dlatego img1 konwertowany jest do typu np.float64 (liczby
rzeczywiste)
    aby nie ograniczać wyniku do przedziału [0, 255].
    """
    mse = ((img1.astype(np.float64) - img2) ** 2).sum() / img1.size  ###img1.size - liczba elementów w img1,
==img1.shape[0]*img1.shape[1] dla obrazów mono, ==img1.shape[0]*img1.shape[1]*img1.shape[2] dla obrazów
barwnych
    psnr = 10.0 * np.log10(imax / mse)
    return mse, psnr
```

Obraz monochromatyczny:

Obliczenie przepływności

```
bitrate = 8 * os.stat(IMAGE_MONO_URL).st_size / (IMAGE_MONO.shape[0] * IMAGE_MONO.shape[1])
print(f"Z1: Przepływność: {bitrate:.4f}\n")
```

```
Z1: Przepływność: 4.2352
```

Obliczenie entropii obrazu, porównanie z przepływnością

```
hist_image = cv2.calcHist([IMAGE_MONO], [0], None, [256], [0, 256])
hist_image = hist_image.flatten()
# Suma wartości histogramu powinna być równa liczbie pikseli w obrazie
print(f"Z2: {hist_image.sum()} = {IMAGE_MONO.shape[0] * IMAGE_MONO.shape[1]}: {hist_image.sum() ==
IMAGE_MONO.shape[0] * IMAGE_MONO.shape[1]}")
H_image = calc_entropy(hist_image)
print(f"Z2: H(image) = {H_image:.4f}")
print(f"Z2: Przepływność = {bitrate:.4f}\n")
```

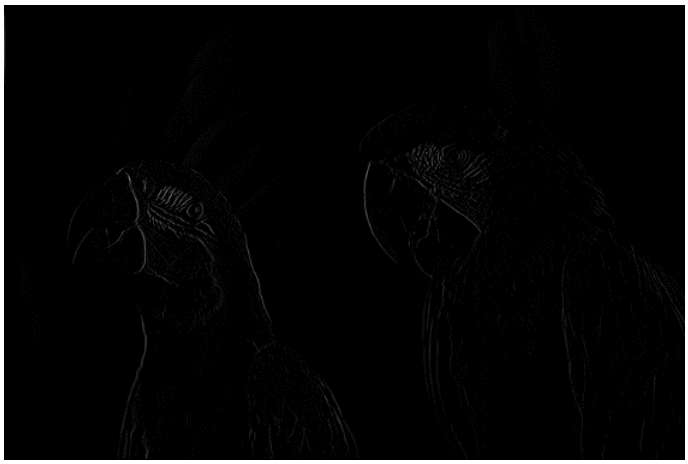
```
Z2: 393216.0 = 393216: True
Z2: H(image) = 7.2382
Z2: Przepływność = 4.2352
```

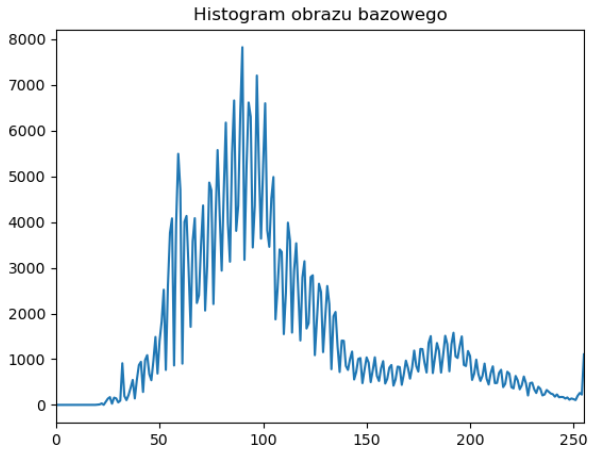
Pozornie wydaję się, że nierówność $l_{śr} \geq H$ nie zachodzi. Należy jednak pamiętać, że przepływność bitowa została policzona dla obrazu PNG (który jest skompresowany), a entropia dla wartości poszczególnych pikseli. Dlatego nie można porównywać tych dwóch rodzajów danych. By je porównać powinniśmy uwzględnić specyfikę informacji zawartej w formacie PNG.

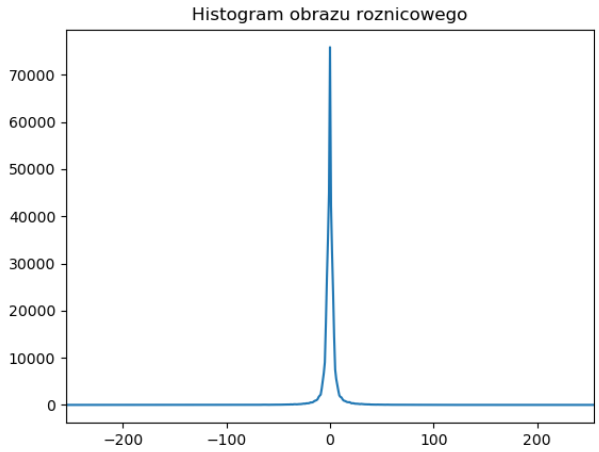
Wyznaczenie obrazu różnicowego

```
img_tmp1 = IMAGE_MONO[:, 1:]
img_tmp2 = IMAGE_MONO[:, :-1]

image_hdiff = cv2.addWeighted(img_tmp1, 1, img_tmp2, -1, 0, dtype=cv2.CV_16S)
image_hdiff_0 = cv2.addWeighted(IMAGE_MONO[:, 0], 1, 0, 0, -127, dtype=cv2.CV_16S)
```

```
image_hdiff = np.hstack((image_hdiff_0, image_hdiff))

cv_imshow(image_hdiff, "image_hdiff")

image_tmp = (image_hdiff + 255).astype(np.uint16)
hist_hdiff = cv2.calcHist([image_tmp], [0], None, [511], [0, 511]).flatten()
hist_image = cv2.calcHist([IMAGE_MONO], [0], None, [256], [0, 256])
plt_hist(y=hist_image, title="Histogram obrazu bazowego", xlim=[0, 255])
plt.savefig("out/Z3/basehist.png")
plt_hist(x=np.arange(-255, 256, 1), y=hist_hdiff, title="Histogram obrazu różnicowego", xlim=[-255, 255])
plt.savefig("out/Z3/diffhist.png")
H_image = calc_entropy(hist_hdiff)
H_base_image = calc_entropy(hist_image)
print(f"Z3: H(diffimage) = {H_image:.4f}")
print(f"Z3: H(baseimage) = {H_base_image:.4f}\n")
```





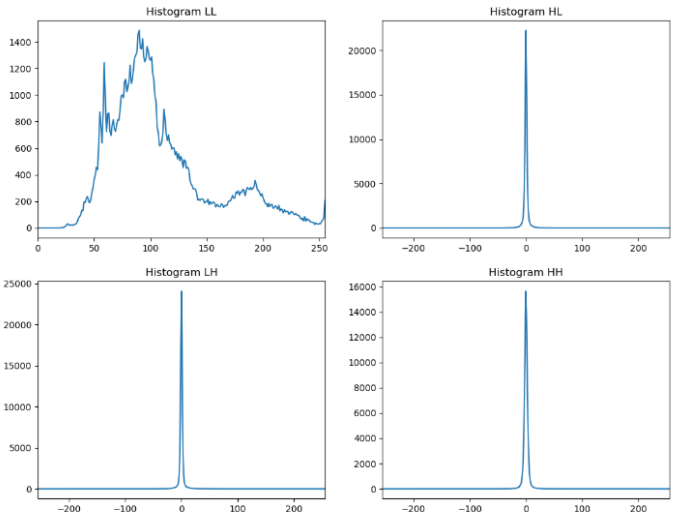


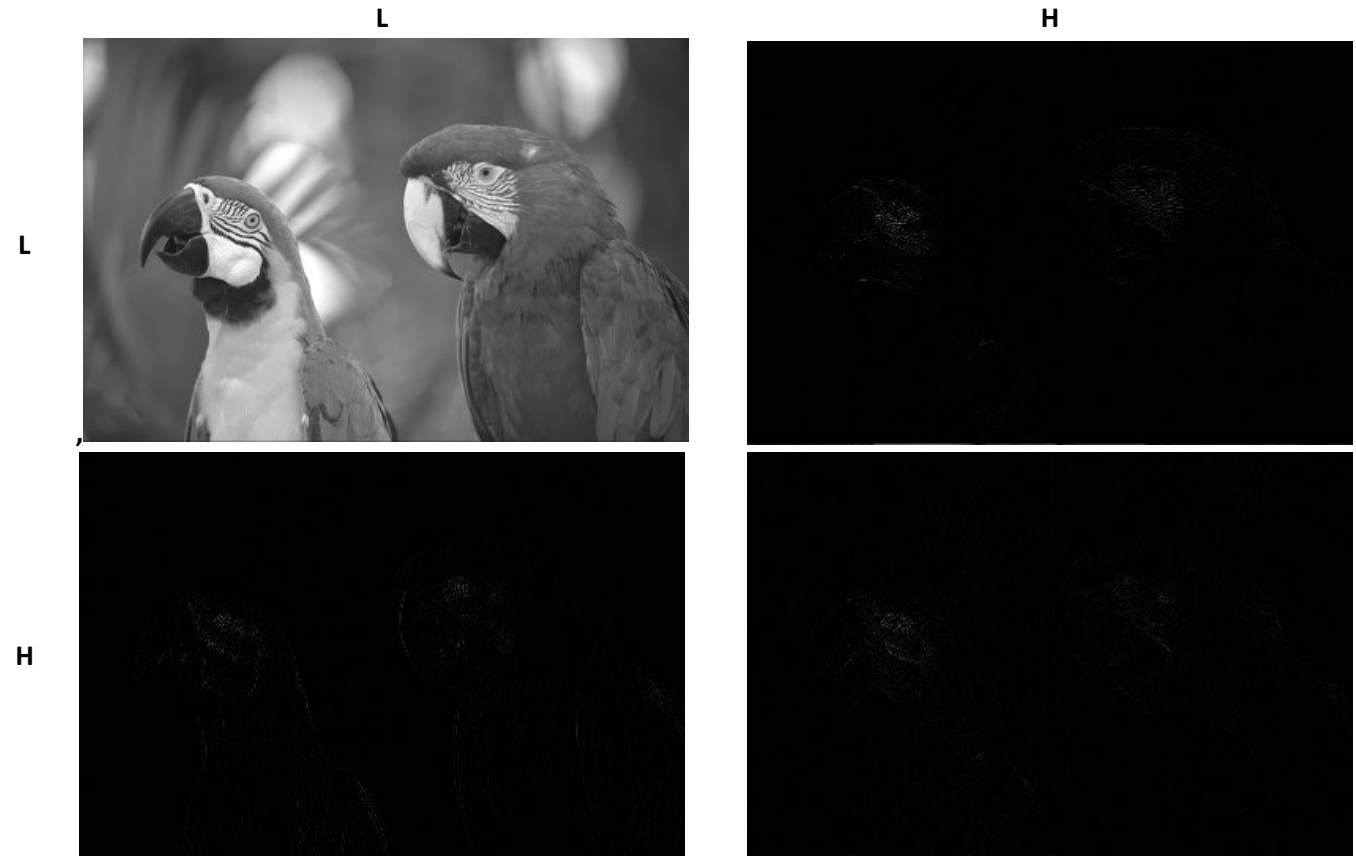

```
Z3: H(diffimage) = 4.1908
Z3: H(baseimage) = 7.2382
```

Wyznaczyć współczynniki DWT

```
ll, lh, hl, hh = dwt(IMAGE_MONO)

cv_imshow(ll, "LL2")
cv_imshow(cv2.multiply(lh, 2), "LH2")
cv_imshow(cv2.multiply(hl, 2), "HL2")
cv_imshow(cv2.multiply(hh, 2), "HH2")
cv_imwrite("out/Z4", "LL.png", ll)
cv_imwrite("out/Z4", "LH.png", lh)
cv_imwrite("out/Z4", "HL.png", hl)
cv_imwrite("out/Z4", "HH.png", hh)

hist_ll = cv2.calcHist([ll], [0], None, [256], [0, 256]).flatten()
hist_lh = cv2.calcHist([(lh + 255).astype(np.uint16)], [0], None, [511], [0, 511]).flatten()
hist_hl = cv2.calcHist([(hl + 255).astype(np.uint16)], [0], None, [511], [0, 511]).flatten()
hist_hh = cv2.calcHist([(hh + 255).astype(np.uint16)], [0], None, [511], [0, 511]).flatten()
H_ll = calc_entropy(hist_ll)
H_lh = calc_entropy(hist_lh)
H_hl = calc_entropy(hist_hl)
H_hh = calc_entropy(hist_hh)
```

```
print(f"Z4: H(LL) = {H_ll:.4f})")
print(f"Z4: H(LH) = {H_lh:.4f}")
print(f"Z4: H(HL) = {H_hl:.4f}")
print(f"Z4: H(HH) = {H_hh:.4f}")
print(f"Z4: H_śr = {(H_ll + H_lh + H_hl + H_hh) / 4:.4f}")

fig = plt.figure()
fig.set_figheight(fig.get_figheight() * 2)
fig.set_figwidth(fig.get_figwidth() * 2)
plt.subplot(2, 2, 1)
plt.plot(hist_ll)
plt.title("Histogram LL")
plt.xlim([0, 255])
plt.subplot(2, 2, 3)
plt.plot(np.arange(-255, 256, 1), hist_lh)
plt.title("Histogram LH")
plt.xlim([-255, 255])
plt.subplot(2, 2, 2)
plt.plot(np.arange(-255, 256, 1), hist_hl)
plt.title("Histogram HL")
plt.xlim([-255, 255])
plt.subplot(2, 2, 4)
plt.plot(np.arange(-255, 256, 1), hist_hh)
plt.title("Histogram HH")
plt.xlim([-255, 255])
plt.savefig("out/Z4/DWT.png")
plt.show()
cv2.waitKey()
```






```
Z4: H(LL) = 7.2881)
Z4: H(LH) = 3.3975
Z4: H(HL) = 3.5237
Z4: H(HH) = 3.8170
Z4: H_śr = 4.5066
```

Najbardziej wyróżnia się pasmo LL, które jest bardzo zbliżone do obrazu bazowego. Pasma LH, HL i HH mają histogramy symetryczne względem punktu 0, podobnie jak obraz różnicowy. Histogram LL wygląda podobnie jak obrazu bazowego, ale „wygładził się” na skutek odfiltrowania pasm LH, HL, HH.

Obraz barwny:

Obliczenie entropii dla składowych RGB

```
image_R = IMAGE_COLOR[:, :, 2]
image_G = IMAGE_COLOR[:, :, 1]
image_B = IMAGE_COLOR[:, :, 0]

hist_R = cv2.calcHist([image_R], [0], None, [256], [0, 256]).flatten()
hist_G = cv2.calcHist([image_G], [0], None, [256], [0, 256]).flatten()
hist_B = cv2.calcHist([image_B], [0], None, [256], [0, 256]).flatten()

H_R = calc_entropy(hist_R)
H_G = calc_entropy(hist_G)
H_B = calc_entropy(hist_B)
print(f"Z5: H(R) = {H_R:.4f}")
print(f"Z5: H(G) = {H_G:.4f}")
print(f"Z5: H(B) = {H_B:.4f}")
print(f"Z5: H_śr = {(H_R + H_G + H_B) / 3:.4f}\n")
```

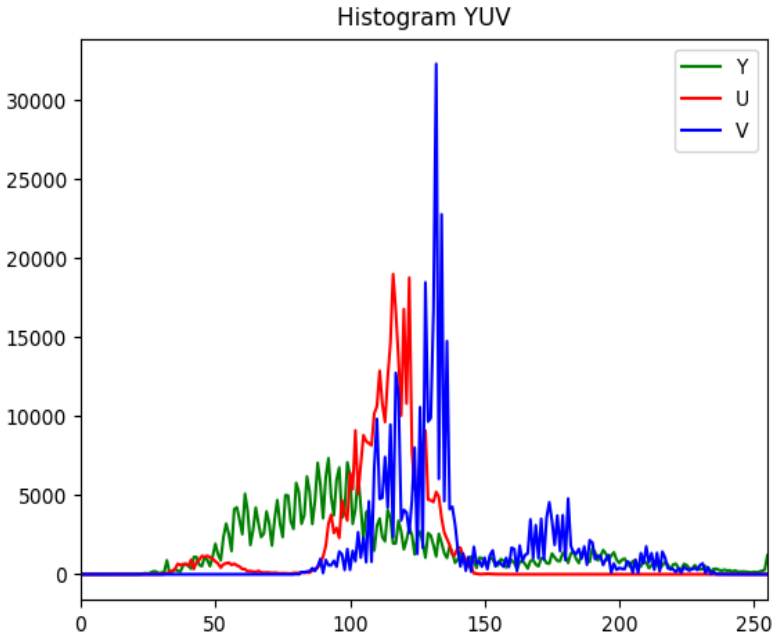
```
Z5: H(R) = 7.4692
Z5: H(G) = 7.4796
Z5: H(B) = 7.1344
Z5: H_śr = 7.3610
```

Konwersja z RGB do YUV

```
IMAGE_YUV = cv2.cvtColor(IMAGE_COLOR, cv2.COLOR_BGR2YUV)

hist_Y = cv2.calcHist([IMAGE_YUV[:, :, 0]], [0], None, [256], [0, 256]).flatten()
hist_U = cv2.calcHist([IMAGE_YUV[:, :, 1]], [0], None, [256], [0, 256]).flatten()
hist_V = cv2.calcHist([IMAGE_YUV[:, :, 2]], [0], None, [256], [0, 256]).flatten()

H_Y = calc_entropy(hist_Y)
H_U = calc_entropy(hist_U)
H_V = calc_entropy(hist_V)
print(f"Z6: H(Y) = {H_Y:.4f}")
print(f"Z6: H(U) = {H_U:.4f}")
print(f"Z6: H(V) = {H_V:.4f}")
print(f"Z6: H_śr = {(H_Y + H_U + H_V) / 3:.4f}\n")

cv_imshow(IMAGE_YUV[:, :, 0], "Obraz Y")
cv_imwrite("out/Z6", "imageY.png", IMAGE_YUV[:, :, 0])
cv_imshow(IMAGE_YUV[:, :, 1], "Obraz U")
cv_imwrite("out/Z6", "imageU.png", IMAGE_YUV[:, :, 1])
cv_imshow(IMAGE_YUV[:, :, 2], "Obraz V")
cv_imwrite("out/Z6", "imageV.png", IMAGE_YUV[:, :, 2])

plt.figure()
plt.plot(hist_Y, color="green")
plt.plot(hist_U, color="red")
plt.plot(hist_V, color="blue")
plt.title("Histogram YUV")
plt.xlim([0, 255])
plt.legend(["Y", "U", "V"])
plt.savefig("out/Z6/hist.png")
plt.show()
cv2.waitKey()
```

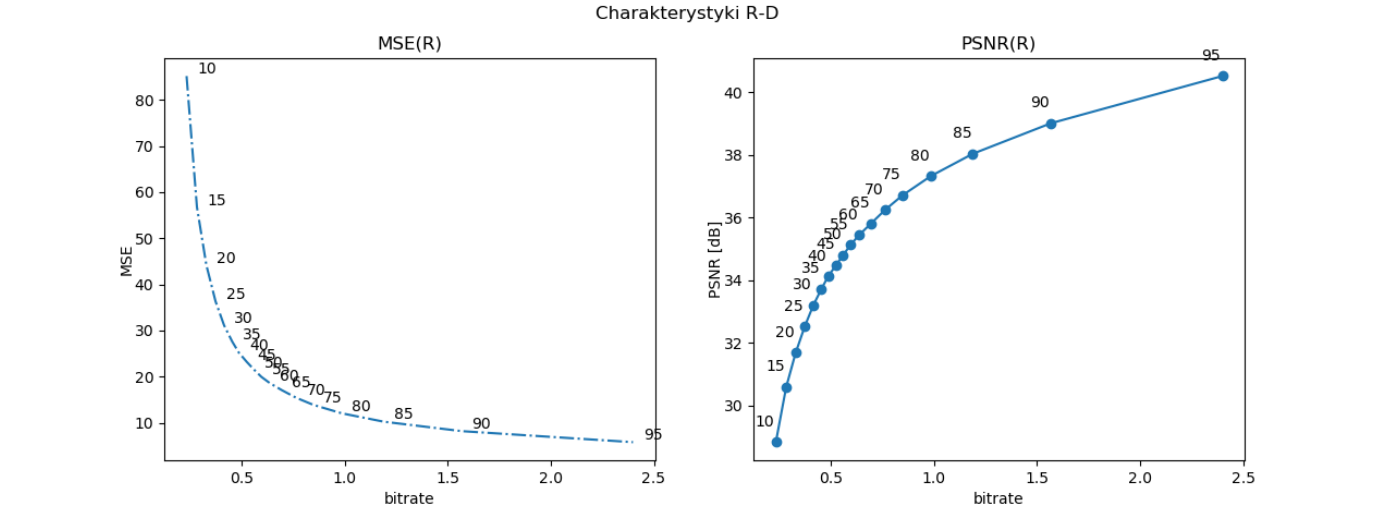
```
Z6: H(Y) = 7.2498
Z6: H(U) = 5.7606
Z6: H(V) = 6.1165
Z6: H_śr = 6.3756
```

Dla składowej Y wynik jest bardzo zbliżony do wyniku obrazu pierwotnego. Pozostałe składowe mają wyniki niższe. Jest to spowodowane tym, że składowa Y występuje na najszerszym przedziale i nie jest skupiona wokół jednej wartości.



Wyznaczenie zależności zniekształcenia D od przepływności R

```
xx = []  ### tablica na wartości osi X -> bitrate
ym = []  ### tablica na wartości osi Y dla MSE
yp = []  ### tablica na wartości osi Y dla PSNR
images = {} ### słownik na obrazy
quality_values = np.arange(10, 100, 5)

for quality in quality_values:
    out_file_name = f"image_q{quality:03d}.jpg"
    path = "out/Z7"
    pathname = path + "/" + out_file_name

    cv_imwrite(path, out_file_name, IMAGE_COLOR, (cv2.IMWRITE_JPEG_QUALITY, int(quality)))
    image_compressed = cv2.imread(pathname, cv2.IMREAD_UNCHANGED)
    bitrate = 8 * os.stat(pathname).st_size / (IMAGE_COLOR.shape[0] * IMAGE_COLOR.shape[1])
    mse, psnr = calc_mse_psnr(IMAGE_COLOR, image_compressed)
    images[quality] = image_compressed
    xx.append(bitrate)
    ym.append(mse)
    yp.append(psnr)

fig = plt.figure()
fig.set_figwidth(fig.get_figwidth() * 2)
plt.suptitle("Charakterystyki R-D")
plt.subplot(1, 2, 1)
plt.plot(xx, ym, "-.")
plt.title("MSE(R)")
plt.xlabel("bitrate")
plt.ylabel("MSE", labelpad=0)
for a, b, c in zip(xx, ym, quality_values):
    plt.text(a+0.05, b+0.5, str(c))
plt.subplot(1, 2, 2)
plt.plot(xx, yp, "-o")
plt.title("PSNR(R)")
plt.xlabel("bitrate")
plt.ylabel("PSNR [dB]", labelpad=0)
for a, b, c in zip(xx, yp, quality_values):
    plt.text(a-0.1, b+0.5, str(c))
plt.savefig("out/Z7/hist.png")
plt.show()

bit_png = 8 * os.stat(IMAGE_COLOR_URL).st_size / (IMAGE_COLOR.shape[0] * IMAGE_COLOR.shape[1])
bit_jpg = 8 * os.stat("out/Z7/image_q050.jpg").st_size / (images[50].shape[0] * images[50].shape[1])
print(f"Z7: Przepływność PNG: {bit_png}")
print(f"Z7: Przepływność JPG (jakość 50): {bit_jpg}")
cv2.waitKey()
```



Jakość 10

Jakość 45

Obraz PNG

Do sprawdzania wybrałem obrazu z jakością z przedziału od 10 do 95 z krokiem 5. Początkowo (10-20) obrazy były kiepskiej jakości, widoczne były na nich zniekształcenia. Następnie (20-50) jakość obrazu stopniowo się poprawiała i była już całkiem zadowalająca: zniekształcenia był widoczne w tle, na jednolitych powierzchniach. Powyżej 50 obrazy były już nieodróżnialne gołym okiem i nie odbiegały jakością od obrazu PNG.

```
Z7: Przepływność PNG: 11.404561360677084
Z7: Przepływność JPG (jakość 50): 0.5604654947916666
```

Mimo tego, że JPG jest nieodróżnialny gołym okiem, przepływność jest ponad 20 razy mniejsza.