ŁUKASZ STANISZEWSKI, NR INDEKSU: 304098

WPROWADZENIE DO MULTIMEDIÓW

SPRAWOZDANIE Z LABORATORIUM 5 – STATYSTYCZNE WŁAŚCIWOŚCI OBRAZÓW

I. Przygotowanie

Koniecznym było pozyskanie odpowiednich plików do wykonania zadania. W wyniku działania $304098 \% 36 = 6$, pobrano pliki: **boat2_mono.png** oraz **boat2_col.png**.

II. Przepływność i entropia obrazu mono

1. KOD ŹRÓDŁOWY

```python
def calc_bitrate_and_entropy(image, path):
    bitrate = 8 * os.stat(path).st_size / (image.shape[0] * image.shape[1])
    print(f"bitrate: {bitrate:.4f} bpp") # obliczanie przeplywnosci / bitrate
    hist_image = cv2.calcHist([image], [0], None, [256], [0, 256])
    hist_image = hist_image.flatten()  # otrzymanie histogramu
    H_image = calc_entropy(hist_image) # obliczanie entropii
    print(f"H(image) = {H_image:.4f}")

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

def task12():
    imageBW = cv2.imread("boat2_mono.png", cv2.IMREAD_UNCHANGED)
    calc_bitrate_and_entropy(imageBW, "boat2_mono.png")
```

KOD 2.1 – PEŁNE ROZWIĄZANIE PUNKTU II

2. ROZWIĄZANIE

W wyniku wykonania przedstawionych wyżej funkcji, otrzymano **przepływność/bitrate** (liczbę bitów przypadającą na 1 piksel) dla obrazu skompresowanego **PNG**, a także obliczono **entropię obrazu** (H) i porównano z wyliczoną **przepływnością**. Wynikiem jest:

```
bitrate: 5.0293 bpp
H(image) = 6.8484
```

WYNIK 2.1 – OBLICZONY BITRATE I ENTROPIA DLA OBRAZU MONO PNG

Można tu zauważyć, że **entropia obrazu jest większa od jego bitrate'u/przepływności**. Fakt ten jednak **nie oznacza**, że zależność mówiąca, że **średnia długość kodu przedrostkowego nie może być mniejsza niż entropia jest fałszywa**. Jest to spowodowane faktem, że **zależność ta dotyczy kodów przedrostkowych**, a zastosowana w zadaniu **kompresja PNG** jest **kompresją bezstratną** i opiera się na **idei kodowania słownikowego** (stąd zależność może być tu łamana). Poprzez zastosowanie kompresji PNG bit-rate obrazu oryginalnego zmalał przy jednoczesnym zachowaniu się prawdopodobieństw wartości pikseli (kompresja bezstratna – zachowanie entropii), przez co zależność przestała być spełniona, ale nie dotyczy tu już kodu przedrostkowego.

III. Obraz różnicowy

1. KOD ŹRÓDŁOWY

```python
def get_differential_image(image):
    img_tmp1 = image[:, 1:]   # kolumny od 1 do ostatniej
    img_tmp2 = image[:, :-1]  # kolumny od 0 do przedostatniej
    image_hdiff = cv2.addWeighted(img_tmp1, 1, img_tmp2, -1, 0, dtype=cv2.CV_16S)  # predykcja w poziomie
    image_hdiff_0 = cv2.addWeighted(image[:, 0], 1, 0, 0, -127, dtype=cv2.CV_16S)  # od 0 kolumny odejmowane 127
    image_hdiff = np.hstack((image_hdiff_0, image_hdiff))  # połączenie tablic w kierunku poziomym
    cv_imshow(image_hdiff, "image_hdiff")  # funkcja pomocnicza do wyświetlania obrazów
    return image_hdiff

def task3():
    imageBW = cv2.imread("boat2_mono.png", cv2.IMREAD_UNCHANGED)
    image_differential = get_differential_image(imageBW)    # otrzymanie obrazu różnicowego i jego wyświetlenie
    compare_hist_entropy(imageBW, image_differential)   # funkcja do wyswietlania histogramow i entropii
```

KOD 3.1 – OBLICZENIE OBRAZU RÓŻNICOWEGO I JEGO WYŚWIETLENIE

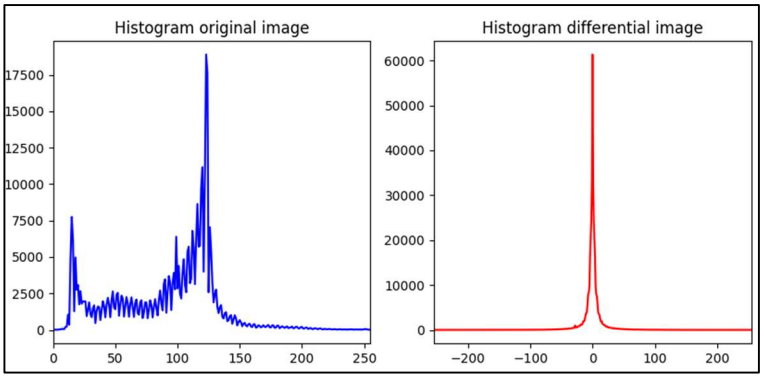
```python
def compare_hist_entropy(image1, image2):
    image_tmp = (image2 + 255).astype(np.uint16) # calcHist() wymaga liczb całkowitych bez znaku, odp. skalowanie
    hist_hdiff = cv2.calcHist([image_tmp], [0], None, [511], [0, 511]).flatten() # histogram obrazu różnicowego
    hist_image = cv2.calcHist([image1], [0], None, [256], [0, 256]).flatten() # histogram obrazu oryginalnego
    print(f"Entropy original image: {calc_entropy(hist_image):.4f}") # obliczanie entropii
    print(f"Entropy differential image: {calc_entropy(hist_hdiff):.4f}")
    plt.figure()  # tworzenie zestawu histogramów
    plt.subplot(121)
    plt.plot(hist_image, color="blue")
    plt.title("Histogram original image")
    plt.xlim([0, 255])
    plt.subplot(122)
    plt.plot(np.arange(-255, 256, 1), hist_hdiff, color="red") # dziedzina w różnicowym to <-255, 255>
    plt.title("Histogram differential image")
    plt.xlim([-255, 255])
    plt.show()
```

KOD 3.2 – FUNKCJA WYŚWIETLAJĄCA HISTOGRAMY I ENTROPIE OBRAZU ORYGINALNEGO I RÓŻNICOWEGO

2. ROZWIĄZANIE

Na początku koniecznym było wyznaczenie **obrazu różnicowego** (z kodowaniem różnicowym / predykcyjnym **poziomym**), gdzie dla pierwszego piksela w wierszu konieczne było przyjęcie wartości sąsiada jako **127**. Następnie **wyświetlono obraz różnicowy**. Na końcu **wyznaczono histogram dla obrazu różnicowego i oryginalnego**, oraz **entropię** obu **obrazów**.

RYS. 3.1 - OBRAZ RÓŻNICOWY



RYS. 3.2 - PORÓWNANIE HISTOGRAMU OBRAZU ORYGINALNEGO Z HISTOGRAMEM OBRAZU RÓŻNICOWEGO

```
Entropy original image: 6.8484
Entropy differential image: 5.0033
```

WYNIK 3.1 – ENTROPIA OBRAZU ORYGINALNEGO I OBRAZU RÓZNICOWEGO

- **Wygląd obrazu różnicowego bardzo oddaje jego histogram**. Na obrazie oryginalnym mono dominuje bardzo **dużo pikseli mających sąsiednie piksele o tym samym kolorze** (całe niebo, większość wody, w miejscach tych dominuje wartość piksela o **wartości 127,5** – **peak na histogramie obrazu oryginalnego**), dlatego też **różnice pomiędzy tymi pikselami będą bardzo bliskie 0**, co oddaje **peak na histogramie obrazu różnicowego**. Z **histogramu obrazu oryginalnego** można wyczytać, że występuje na obrazie oryginalnym **całkiem dużo pikseli bliskich koloru czarnego** (bliskich wartości 0), najczęściej na obrazie oryginalnym mono są to piksele pod mostem, natomiast **na obrazie różnicowym** zauważyć można **krawędzie na obrazie** – te **piksele posiadały** w obrazie oryginalnym **lewych sąsiadów o innej wartości**, świadczą o tym **na histogramie obrazu różnicowego wartości dla argumentów wokół 0**, jednak są one dużo mniej znaczące pod względem ilości w porównaniu z liczebnością różnicy pikseli wynoszącej 0.

- **Na obrazie różnicowym dominuje kolor szary** i **jego odcienie** (dlatego, że **wszystkie wartości różnicy są w bardzo bliskiej odległości od środka zakresu dopuszczalnych wartości pikseli** i po odpowiednim przeskalowaniu **środek ten odpowiada kolorowi szaremu**).

- W związku z tak **dużą dominacją koloru szarego na obrazie różnicowym**, jego **średnia informacji będzie znacznie mniejsza niż dla obrazu oryginalnego**, co oczywiście odbije się też na jego **entropię**, która będzie **znacznie mniejsza niż entropia obrazu oryginalnego mono (prawdopodobieństwa** pojedynczych **wartości pikseli** znacząco się **zwiększają)**.

IV. Współczynniki DWT

1. KOD ŹRÓDŁOWY

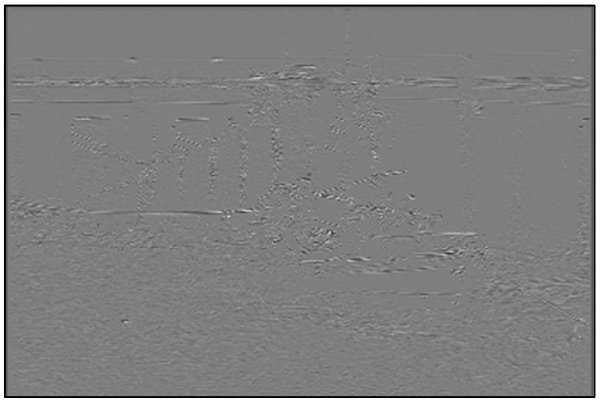
```python
def dwt(img):       # funkcja wyznaczająca pasma dwt (uproszczona) i je zwracająca
    maskL = np.array([0.02674875741080976, -0.01686411844287795, -0.07822326652898785, 0.2668641184428723,
                      0.6029490182363579, 0.2668641184428723, -0.07822326652898785, -0.01686411844287795,
                      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]
    # w filtracji górnoprzepustowej wartości ujemne, więc wynik 16-bitowy ze znakiem
    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 show_hist_bands(hist_ll, hist_lh, hist_hl, hist_hh):
    # funkcja przedstawia podane histogramy jako subplot
    fig = plt.figure()
    # zwiększenie rozmiarów okna
    fig.set_figheight(fig.get_figheight() * 2)
    fig.set_figwidth(fig.get_figwidth() * 2)
    plt.subplot(2, 2, 1)
    plt.plot(hist_ll, color="blue")
    plt.title("hist_ll")
    plt.xlim([0, 255])
    plt.subplot(2, 2, 3)
    plt.plot(np.arange(-255, 256, 1), hist_lh, color="red")
    plt.title("hist_lh")
    plt.xlim([-255, 255])
    plt.subplot(2, 2, 2)
    plt.plot(np.arange(-255, 256, 1), hist_hl, color="red")
    plt.title("hist_hl")
    plt.xlim([-255, 255])
    plt.subplot(2, 2, 4)
    plt.plot(np.arange(-255, 256, 1), hist_hh, color="red")
    plt.title("hist_hh")
    plt.xlim([-255, 255])
    plt.show()
    cv2.destroyAllWindows()

def task4():
    imageBW = cv2.imread("boat2_mono.png", cv2.IMREAD_UNCHANGED)
    # wyznaczenie pasm z użyciem transformacji DWT
    ll, lh, hl, hh = dwt(imageBW)
    # wyświetlenie pasm
    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")
    # obliczenie histogramów z odpowiednim skalowaniem i konwersją dla calcHist()
    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()
    # liczenie entropi
    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"H(LL) = {H_ll:.4f} \nH(LH) = {H_lh:.4f} \nH(HL) = {H_hl:.4f} \nH(HH) = {H_hh:.4f} \n")
    # zestawienie histogramów
    show_hist_bands(hist_ll, hist_lh, hist_hl, hist_hh)
```

KOD 4.1 – LICZENIE DWT, WYŚWIETLENIE PASM, ICH ENTROPII I HISTOGRAMÓW

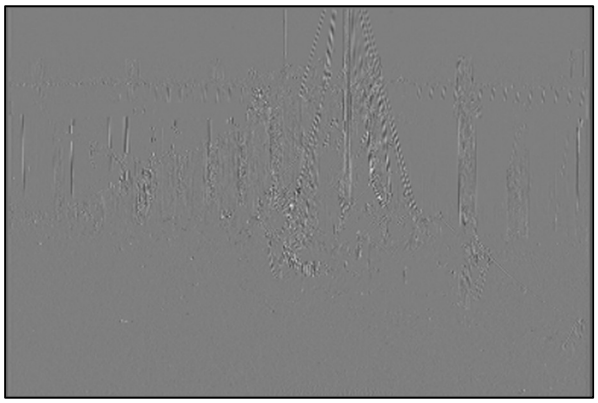
2. ROZWIĄZANIE

Na początku koniecznym było **wyznaczenie współczynników DWT** (z wykorzystaniem funkcji ze skryptu) oraz **wyświetlenie poszczególnych pasm**. Na końcu **wyznaczono histogramy** i **entropie** dla wszystkich **pasm**.

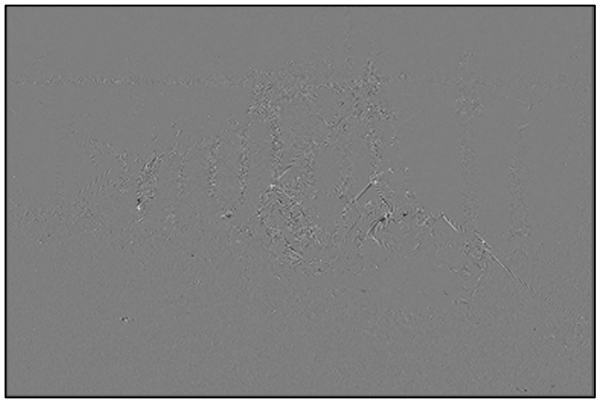
RYS 4.1 – PASMO LL



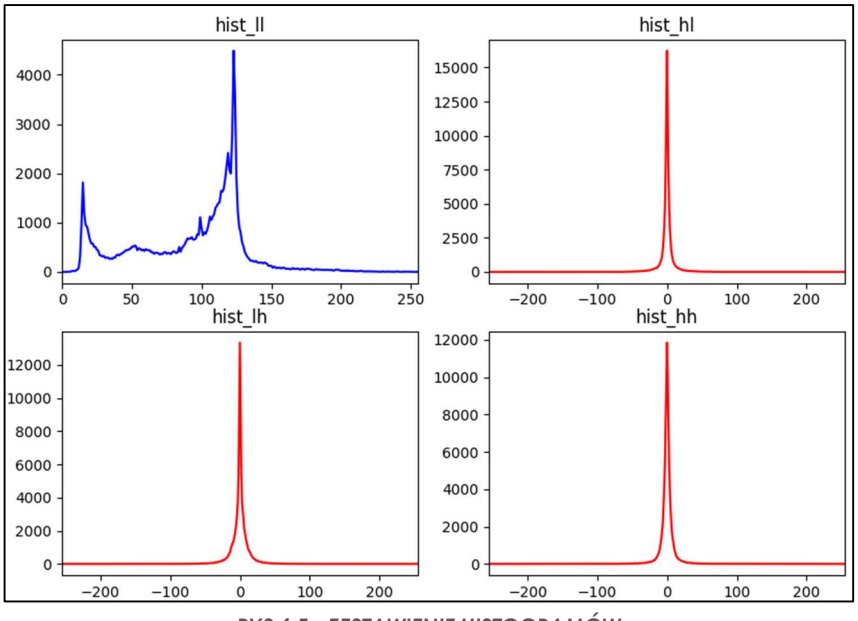
RYS 4.2 – PASMO LH



RYS 4.3 – PASMO HL



RYS 4.4 – PASMO HH



RYS 4.5 – ZESTAWIENIE HISTOGRAMÓW

```
H(LL) = 6.8753
H(LH) = 5.0958
H(HL) = 4.4986
H(HH) = 4.7002
```

WYNIK 4.1 – ZESTAWIENIE ENTROPII DLA POSZCZEGÓLNYCH PASM

- **Pasmo LL** pod względem zarówno **wyglądu**, **histogramu** jak i **entropii** najbardziej **wyróżnia się spośród wszystkich otrzymanych pasm** . Widać na podstawie tego, że **dolne pasmo zawiera najwięcej energii** i **przypomina oryginał** (ponieważ **na niskich częstotliwościach są szczegóły**). Nastąpiło tutaj **zmniejszenie rozdzielczości**, ale **wydzielono obraz podobny do oryginału**. O **podobieństwie świadczy także histogram dla pasma LL** (który jest całkiem **podobny do histogramu obrazu oryginalnego mono)**, a także **bardzo podobne do siebie entropie pasma LL i obrazu**.

- **Pasma LH, HL oraz HH są do siebie całkiem podobne**, zarówno pod względem **entropii**, **histogramu** jak i **wyglądu**. **Posiadają** one **znacznie mniejszą energię** niż w przypadku **pasma LL**. **Przypominają** one (pod względem samego **wyglądu**, **histogramu** i **entropii**) **obraz różnicowy** (przedstawione na rys.: 3.1 i 3.2). Tutaj również **dominuje kolor szary**, zgodne jest to z postaciami **histogramów** (posiadają **peak** w tym samym miejscu, **po środku** dopuszczalnych wartości). **Same histogramy dla tych pasm różnią się maksymalną wartością samego peak'u**.

V. Entropia RGB oraz YUV

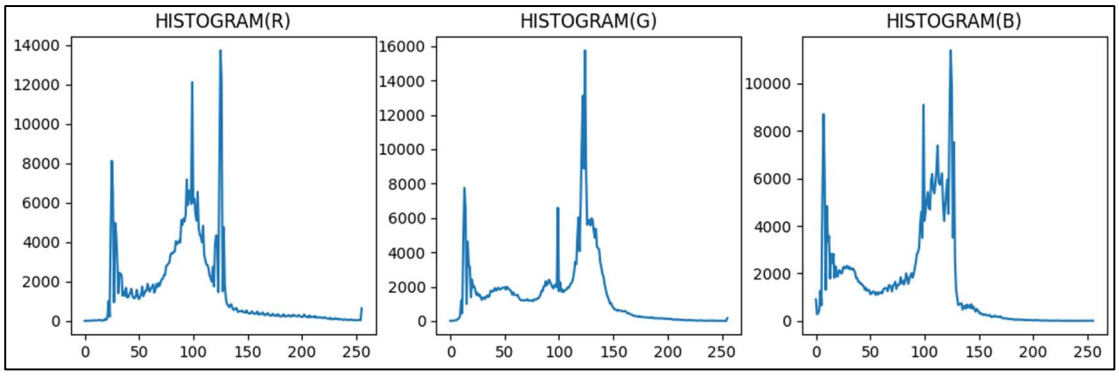
1. KOD ŹRÓDŁOWY

```python
def showHistograms(histograms, hist_names):
    # funkcja dla podanych histogramów wyświetla je obok siebie
    subplots = [131, 132, 133]
    plt.figure()
    for hist, name, sub in zip(histograms, hist_names, subplots):
        plt.subplot(sub)
        plt.plot(hist)
        plt.title(name)
    plt.show()

def task56():
    image_col = cv2.imread("boat2_col.png")
    # wyznacznie obrazów dla poszczególnych składowych
    image_R = image_col[:, :, 2]
    image_G = image_col[:, :, 1]
    image_B = image_col[:, :, 0]
    # histogramy dla każdej składowej
    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()
    # entropie dla każdej składowej
    H_R = calc_entropy(hist_R)
    H_G = calc_entropy(hist_G)
    H_B = calc_entropy(hist_B)
    print(f"H(R) = {H_R:.4f} \nH(G) = {H_G:.4f} \nH(B) = {H_B:.4f}\n")
    # konwersja do YUV
    image_YUV = cv2.cvtColor(image_col, cv2.COLOR_BGR2YUV)
    # histogramy dla składowych
    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()
    #entropie dla składowych
    H_Y = calc_entropy(hist_Y)
    H_U = calc_entropy(hist_U)
    H_V = calc_entropy(hist_V)
    print(f"H(Y) = {H_Y:.4f} \nH(U) = {H_U:.4f} \nH(V) = {H_V:.4f}\n")
    # wyświetlenie histogramów wszystkich składowych
    showHistograms([hist_R, hist_G, hist_B], ["HISTOGRAM(R)", "HISTOGRAM(G)", "HISTOGRAM(B)"])
    showHistograms([hist_Y, hist_U, hist_V], ["HISTOGRAM(Y)", "HISTOGRAM(U)", "HISTOGRAM(V)"])
```

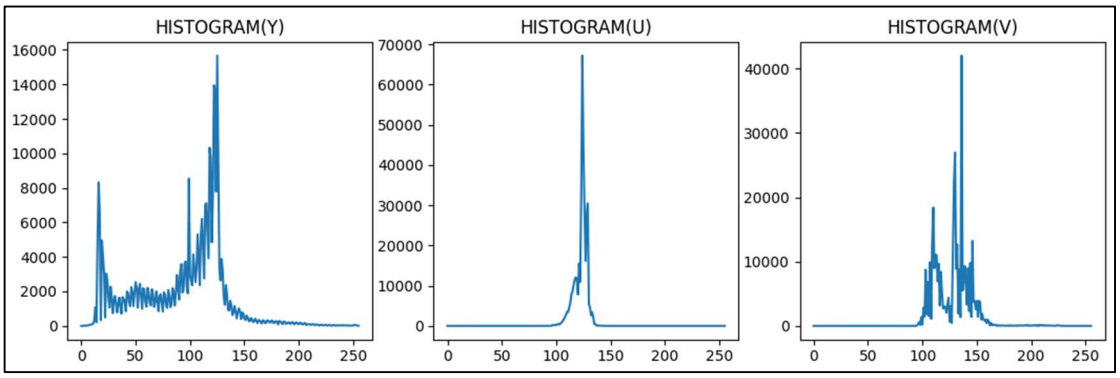
KOD 5.1 – OTRZYMANIE HISTOGRAMÓW I ENTROPII DLA SKŁADOWYCH RGB I YUV

2. ROZWIĄZANIE

Na początku koniecznym było **wyznaczenie poszczególnych składowych RGB**, wykonania na ich podstawie **histogramów** i policzenie **entropii**. W następnym kroku wykonano **przekształcenie obrazu barwnego do przestrzeni barw YUV**, wyznaczenie **składowych**, **histogramów** na ich podstawie, policzenie **entropii składowych YUV**, a na końcu **wyświetlono uzyskane entropie i histogramy**.



RYS 5.1 – HISTOGRAMY DLA SKŁADOWYCH RGB



RYS 5.2 – HISTOGRAMY DLA SKŁADOWYCH YUV

```
H(R) = 6.9182
H(G) = 7.0234
H(B) = 6.9199

H(Y) = 6.8474
H(U) = 4.2524
H(V) = 5.4024
```

WYNIK 5.1 – ENTROPIE DLA POSZCZEGÓLNYCH SKŁADOWYCH RGB I YUV

- Jeśli chodzi o **histogramy dla składowych RGB**, **żaden z nich nie wyróżnia się na tle pozostałych**. Świadczy o tym również **entropia dla poszczególnych składowych**, składowe są do siebie bardzo zbliżone pod jej względem, tzn. że **każda ze składowych posiada bardzo podobną średnią informację niesioną przez siebie** i **w każdej ze składowych skupiona jest podobna energia**.

- **Różnice można zauważyć natomiast w przypadku składowych YUV**, gdzie **histogram dla składowej Y jest znacznie bardziej rozszerzony niż histogramy dla składowych U i V** (gdzie występuje skupienie w okolicy środkowej dopuszczalnej wartości pikseli). Podobnie jest w przypadku **entropii** – **widać znaczącą różnicę między entropia dla poszczególnych składowych**. **Najmniej energii skupionej jest w składowej V**, **więcej w U**, **a najwięcej w składowej Y (składowej luminancji)**. **Właściwość** ta jest **wykorzystywana w kompresji**. Z powodu, że mamy **zróżnicowaną energię w poszczególnych składowych**, **składowe,** w których skupione jest **mniej energii**, mogą być **reprezentowane i obserwowane na mniejszej liczbie bitów**.

VI. Koder PNG a koder JPEG

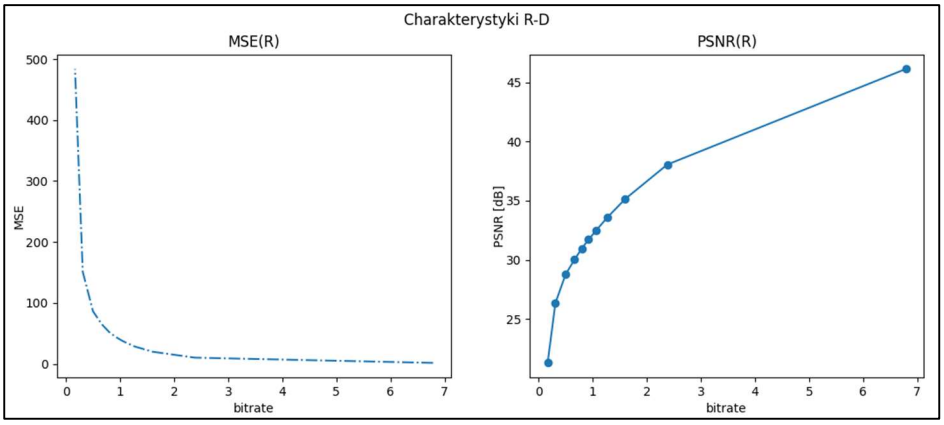
1. KOD ŹRÓDŁOWY

```python
def calc_mse_psnr(img1, img2):
    # Funkcja liczy MSE i PSNR dla podanych obrazów, zał: piksele z przedziału [0, 255]
    imax = 255. ** 2  # maksymalna wartość sygnału -> 255
    # Istotne wartości ujemne, dlatego img1 konwertowany do np.float64 (liczby rzeczywiste)
    mse = ((img1.astype(np.float64) - img2) ** 2).sum() / img1.size
    psnr = 10.0 * np.log10(imax / mse)
    return mse, psnr

def task7():
    image_col = cv2.imread("boat2_col.png", cv2.IMREAD_UNCHANGED)
    xx = []  # tablica na bitrate
    ym = []  # tablica na MSE
    yp = []  # tablica na PSNR
    for quality in [100, 90, 80, 70, 60, 50, 40, 30, 20, 10, 0]:
        out_file_name = f"out_image_q{quality:03d}.jpg"
        # Zapis do pliku w formacie .jpg z ustaloną jakością
        cv2.imwrite(out_file_name, image_col, (cv2.IMWRITE_JPEG_QUALITY, quality))
        # Odczyt skompresowanego obrazu, policzenie bitrate'u i PSNR
        image_compressed = cv2.imread(out_file_name, cv2.IMREAD_UNCHANGED)
        bitrate = 8 * os.stat(out_file_name).st_size / (image_col.shape[0] * image_col.shape[1])
        mse, psnr = calc_mse_psnr(image_col, image_compressed)
        xx.append(bitrate)
        ym.append(mse)
        yp.append(psnr)
    # narysowanie wykresów
    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)
    plt.subplot(1, 2, 2)
    plt.plot(xx, yp, "-o")
    plt.title("PSNR(R)")
    plt.xlabel("bitrate")
    plt.ylabel("PSNR [dB]", labelpad=0)
    plt.show()
    # liczenie stopnia kompresji
    size_png = os.stat("boat2_col.png").st_size
    size_jpg_100 = os.stat("out_image_q100.jpg").st_size
    size_jpg_10 = os.stat("out_image_q010.jpg").st_size
    print("~~STOPNIE KOMPRESJI:")
    print(f"Stopień kompresji przy quality=100: {size_png / size_jpg_100:.4f}")
    print(f"Stopień kompresji przy quality=10: {size_png / size_jpg_10:.4f}")
    # liczenie przepływności
    print("~~PRZEPŁYWNOŚCI:")
    image_col = cv2.imread("boat2_col.png", cv2.IMREAD_UNCHANGED)
    bitrate_col = 8 * os.stat("boat2_col.png").st_size / (image_col.shape[0] * image_col.shape[1])
    image_bw = cv2.imread("boat2_mono.png", cv2.IMREAD_UNCHANGED)
    bitrate_bw = 8 * os.stat("boat2_mono.png").st_size / (image_bw.shape[0] * image_bw.shape[1])
    image_jpg = cv2.imread("out_image_q100.jpg", cv2.IMREAD_UNCHANGED)
    bitrate_jpg = 8 * os.stat("out_image_q100.jpg").st_size / (image_jpg.shape[0] * image_jpg.shape[1])
    print(f"Przepływność dla RGB PNG: {bitrate_col} bpp")
    print(f"Przepływność dla MONO PNG: {bitrate_bw} bpp")
    print(f"Przepływność dla RGB JPG: {bitrate_jpg} bpp")
```

KOD 6.1 – WYZNACZONA ZALEŻNOŚĆ ZNIEKSZTAŁCENIA D OD PRZEPŁYWNOŚCI R DLA KODERA JPEG

2. ROZWIĄZANIE

W tym zadaniu konieczne jest wyznaczenie **zależności zniekształcenia D od przepływności R** w postaci **krzywej R-D dla kodera JPEG** z miarami zniekształceń **PSNR** oraz **MSE**. W tym celu zostały dobrane tak parametry **quality**, żeby **uzyskane wykresy krzywej były gładkie**. Następnie **zobrazowano wykresy** tej zależności (dla każdej miary osobno), a także **oceniono subiektywnie obrazy zrekonstruowane**.



RYS. 6.1 – CHARAKTERYSTYKI R-D DLA MIARY MSE ORAZ PSNR

RYS 6.2 – OBRAZ W WYNIKU KOMPRESJI JPEG DLA QUALITY=100

RYS 6.3 – OBRAZ W WYNIKU KOMPRESJI JPEG DLA QUALITY=10

Można zauważyć, że **wraz ze zmniejszaniem się jakości kompresji** (i maleniem rozmiaru pliku kompresowanego), **maleje jakość obrazów wyjściowych** (dla jakości **100** obraz jest **doskonałej jakości**, jednak dla jakości **10** na obrazie widać **zniekształcenia** w postaci m.in. **„zpikselowanych" fragmentów np. nieba**). Jeśli chodzi o wykresy to widać, że **wraz ze zwiększaniem się przepływności w bitach na piksel** (zwiększaniem jakości kompresji), **logarytmicznie wzrasta szczytowy stosunek sygnału do szumu PSNR** (skala decybelowa), natomiast **błąd średniokwadratowy MSE maleje wymiernie** (ze względu na coraz większe rozmiary obrazu w bajtach). Dodatkowo została stworzona tabela zestawiająca subiektywne oceny obrazów zrekonstruowanych do zakresów ich przepływności.

BITRATE [BITS PER PIXEL]	SUBIEKTYWNIE JAKOŚĆ
≤ 0.3	*JAKOŚĆ NIEAKCEPTOWALNA*
$(0.3,0.5\rangle$	*JAKOŚĆ ZŁA*
$(0.5,0.8\rangle$	*JAKOŚĆ ŚREDNIA*
$(0.8,1.2\rangle$	*JAKOŚĆ DOBRA*
$(1.2,2.4\rangle$	*JAKOŚĆ BARDZO DOBRA*
> 2.4	*JAKOŚĆ DOSKONAŁA*

TABELA 6.1 – ZESTAWIENIE SUBIEKTYWNEJ OCENY JAKOŚCI DLA DANYCH PRZEPŁYWNOŚCI OBRAZÓW SKOMPRESOWANYCH

Na końcu zostały zestawione **stopnie kompresji JPEG** z **jakością 100** oraz z **jakością 10**, a także **porównane przepływności dla obrazów**: **oryginalnego kolorowego kompresowanego PNG**, **oryginalnego monochromatycznego PNG**, a także **kolorowego kompresowanego JPEG z jakością 100**. Sam **stopień kompresji** liczony był z użyciem wzoru: $\boldsymbol{CR} = \frac{\boldsymbol{RozmiarPrzedSkompresowaniem}}{\boldsymbol{RozmiarPoSkompresowaniu}}$.

```
~~STOPNIE KOMPRESJI:
Stopień kompresji przy quality=100: 1.9093
Stopień kompresji przy quality=10: 41.8653

~~PRZEPŁYWNOŚCI:
Przepływność dla RGB PNG: 12.9577 bpp
Przepływność dla MONO PNG: 5.0293 bpp
Przepływność dla RGB JPG: 6.7866 bpp
```

WYNIK 6.1 – WYLICZONY STOPIEŃ KOMPRESJI DLA JAKOŚCI 10 ORAZ 100 WRAZ Z OBLICZONYMI PRZEPŁYWNOŚCIAMI

Jak widać, **kompresowanie przy użyciu kodera JPEG z jakością = 10**, gwarantuje **40 razy większy stopień kompresji** niż w przypadku użyciu **kodera JPEG z jakością = 100**, jednak **należy pamiętać** o tym, że **niska jakość** może znacząco wpłynąć na **liczbę zniekształceń w obrazie wyjściowym**. Użycie kodera JPEG z najlepszą jakością zagwarantowało dwukrotnie lepszy stopień kompresji niż użycie samego kodera PNG. Mówi o tym również fakt, że **przepływność bitowa dla obrazu barwnego skompresowanego koderem PNG jest znacznie większa od przepływności dla obrazu skompresowanego koderem JPEG** (niemal dwukrotnie większa). Jeśli chodzi o **przepływność bitową dla obrazu monochromatycznego**, to jest ona **najmniejsza spośród wszystkich trzech**, co wynika z **tylko jednej składowej opisującej kolor**, jednak nie jest ona znacznie mniejsza niż dla kolorowego obrazu skompresowanego koderem JPEG.