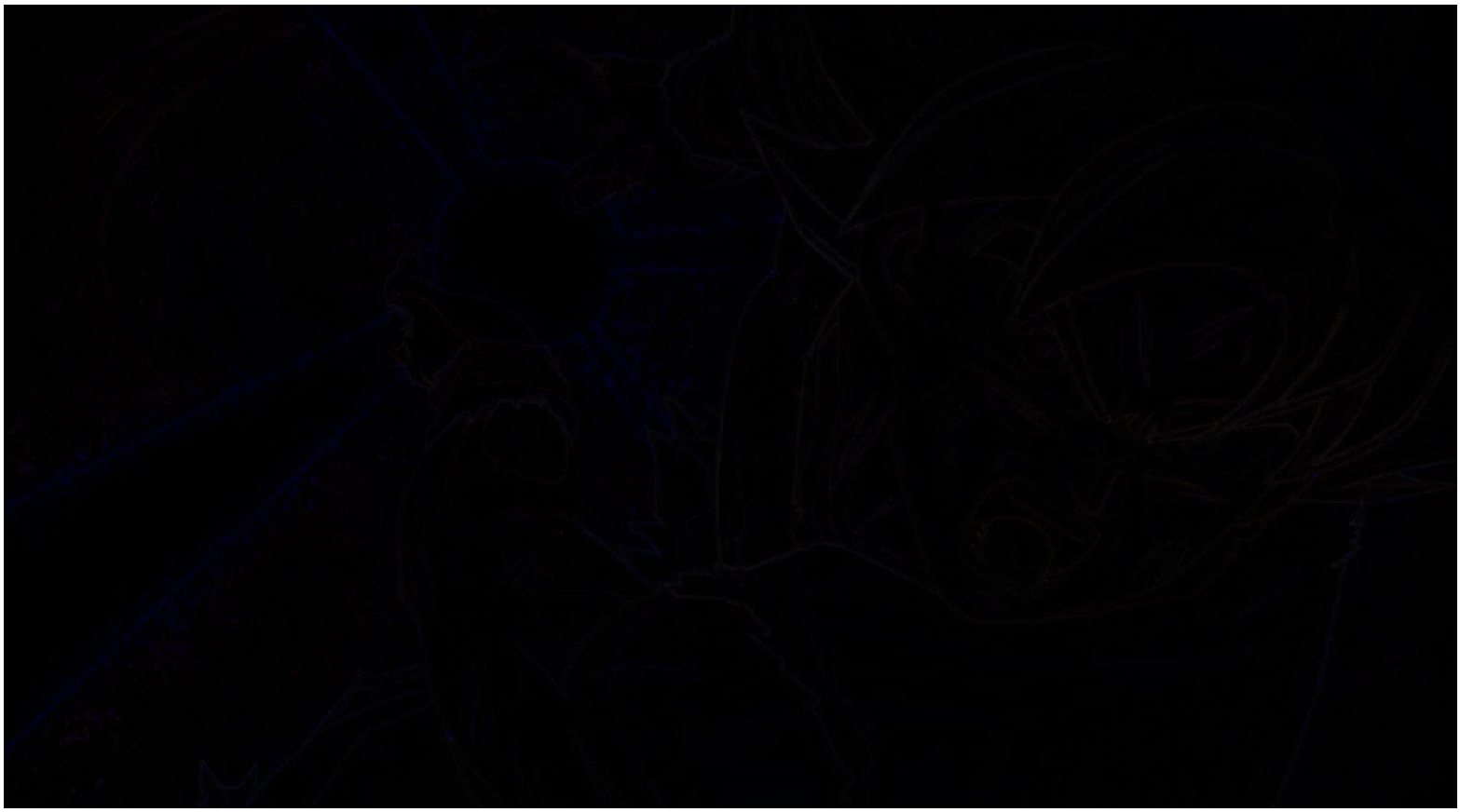


Pomimo, że na pierwszy rzut oka nie widać, to efekt porównania dwóch tych obrazów w różnych formatach (jpg, bmp), nie jest czarny. W takim przypadku wystarczy tylko przybliżyć, i widać różnice, jednak nie zawsze się tak da, czasem różnice są tak małe, że warto zastosować inne metody. Poniżej widzimy statystyki powyższego obrazu.

```
extrema  [(0, 44), (0, 39), (0, 137)]
count  [2073600, 2073600, 2073600]
mean  [2.10850887345679, 1.48931905864197 53, 3.0510923032407407]
median  [1, 1, 2]
stddev  [2.751075044975117, 2.0343182475479398, 4.438120284956337]
```

Extrema (wartości maksymalne i minimalne):

- Extrema to zakres wartości pikseli w obrazie RGB w poszczególnych kanałach. Na przykład, w pierwszym kanale (czerwonym) wartości wahają się od 0 (najniższa jasność) do 44 (najwyższa jasność). W drugim kanale (zielonym) wartości wynoszą od 0 do 39, a w trzecim kanale (niebieskim) od 0 do 137.

Count (ilość pikseli):

- Liczba pikseli w każdym z kanałów wynosi 2073600. To jest całkowita liczba pikseli w obrazie w danym kanale.

Mean (średnia):

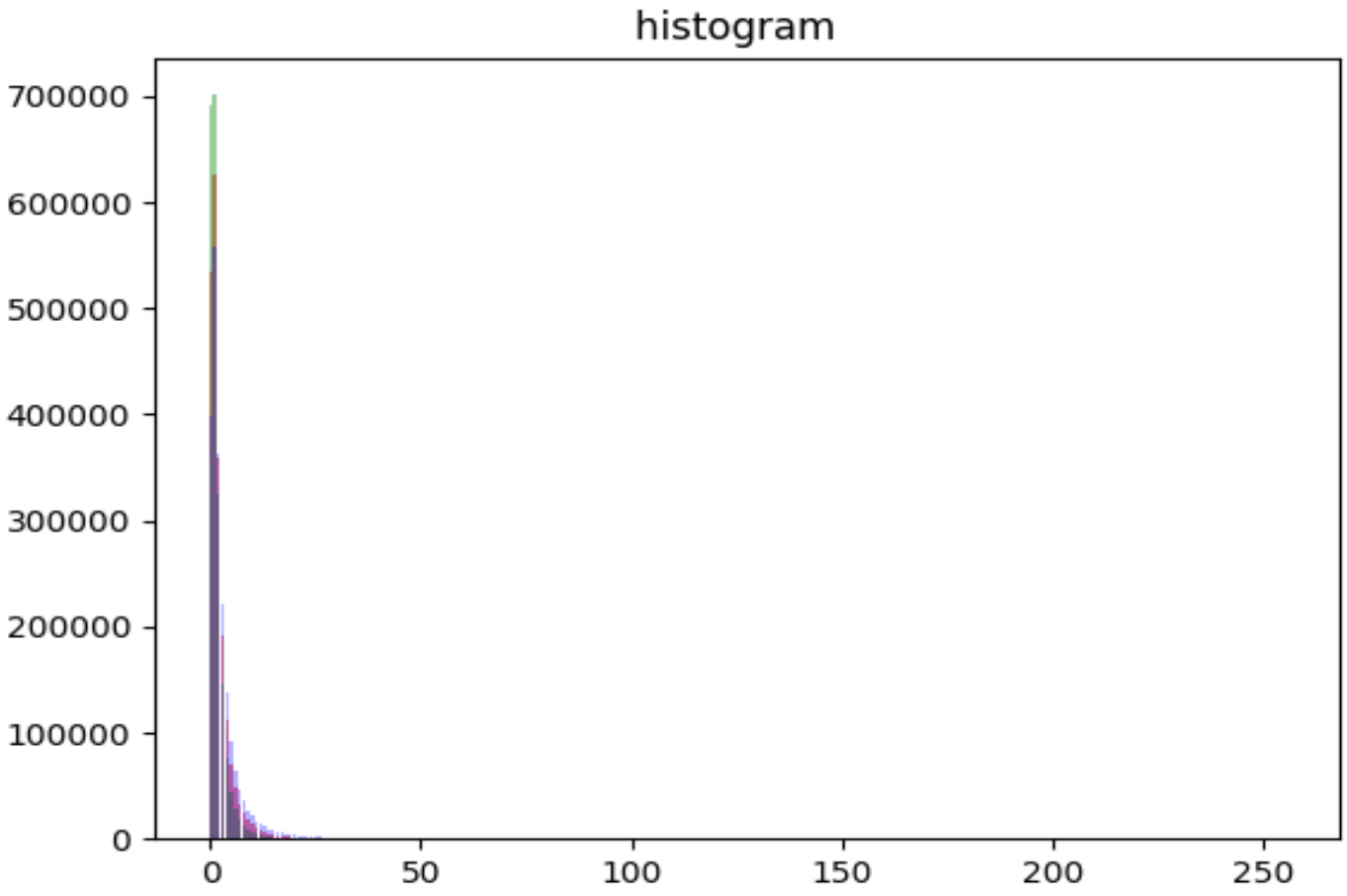
- Średnia wartość pikseli w każdym z kanałów wynosi około 2.1 (w pierwszym kanale), 1.49 (w drugim kanale) i 3.05 (w trzecim kanale). To daje ogólny pogląd na jasność kanałów, gdzie wyższa średnia oznacza jaśniejszy kanał.

Median (mediana):

- Mediana w pierwszym i drugim kanale wynosi 1, a w trzecim kanale wynosi 2. Mediana jest wartością, która dzieli dane na pół, więc to informacja o centralnej jasności kanałów.

Stddev (odchylenie standardowe):

- Odchylenie standardowe wskazuje, jak rozproszone są wartości pikseli w danym kanale. Wartości odchylenia standardowego wskazują na to, że piksele w pierwszym i drugim kanale są mniej rozproszone (blisko średniej), podczas gdy w trzecim kanale są bardziej rozproszone (większa zmienność).



Histogram jest graficznym narzędziem do analizy rozkładu danych, które pomaga zrozumieć, jak często dane występują w określonych przedziałach. W kontekście przetwarzania obrazów, histogram jest często używany do analizy rozkładu jasności lub kolorów pikseli na obrazie. Powyżej mamy histogram stworzony z obrazu diff.png, a poniżej funkcje wykonujące powyższe czynności.

```
def statystyki(im):
    s = stat.Stat(im)
    print("extrema ", s.extrema)  # max i min
    print("count ", s.count)  # zlicza
    print("mean ", s.mean)  # srednia
    print("median ", s.median)  # mediana
    print("stddev ", s.stddev)  # odchylenie standardowe

def rysuj_histogram_RGB(obraz):
    hist = obraz.histogram()
    plt.title("histogram  ")
    plt.bar(range(256), hist[:256], color='r', alpha=0.5)
    plt.bar(range(256), hist[256:2 * 256], color='g', alpha=0.4)
    plt.bar(range(256), hist[2 * 256:], color='b', alpha=0.3)
    plt.savefig("histogram1.png")
    plt.show()
```

```python
def zlicz_roznice_srednia_RGB(obraz, wsp): # wsp - współczynnik określający dokładność oceny
    t_obraz = np.asarray(obraz)
    h, w, d = t_obraz.shape
    zlicz = 0
    for i in range(h):
        for j in range(w):
                if np.mean(t_obraz[i, j, :]) > wsp:
                    zlicz = zlicz + 1
    procent = zlicz/(h*w)
    return zlicz, procent

def zlicz_roznice_suma_RGB(obraz, wsp): # wsp - współczynnik określający dokładność oceny
    t_obraz = np.asarray(obraz)
    h, w, d = t_obraz.shape
    zlicz = 0
    for i in range(h):
        for j in range(w):
                if sum(t_obraz[i, j, :]) > wsp:
                    zlicz = zlicz + 1
    procent = zlicz/(h*w)
    return zlicz, procent
```

Powyższe funkcje pozwolą zobaczyć jak obraz się od obrazu czarnego. Nie wiem w jakim zakresie widzi ludzkie oko, ale np. zmiana wartości o 1-2 na pojedynczych pikselach sprawi, że powstały obraz będzie inny, chociaż na „oko” oba mogą być takie same. Powyższe funkcje wywołałem w pętli z różnymi współczynnikami, zaczynając od 0 i zwiększając o 5, aż do 20. Poniżej kod wywołujący te funkcje, a także wyniki jakie wyszły w zależności od współczynników. Jako, że w „print” opisałem dokładnie co aktualnie się wyświetla, pozwolę sobie nie komentować tego ponownie. Jedynie dodałem metodę time() z biblioteki time, aby można było sprawdzić czas działania w sekundach.

```
liczba niepasujących pikseli z funkcji zlicz_roznice_suma_RGB()
 wywołanej z współczynnikiem 0: 1963700
procent niepasujących pikseli z funkcji zlicz_roznice_suma_RGB()
 wywołanej z współczynnikiem 0: 0.9470003858024691
liczba niepasujących pikseli z funkcji zlicz_roznice_srednia_RGB()
 wywołanej z współczynnikiem 0: 1963700
procent niepasujących pikseli z funkcji zlicz_roznice_srednia_RGB()
 wywołanej z współczynnikiem 0: 0.9470003858024691
Pętla nr. 1: Czas: 13.52
********************************************************************************
 *************
liczba niepasujących pikseli z funkcji zlicz_roznice_suma_RGB()
 wywołanej z współczynnikiem 5: 834656
procent niepasujących pikseli z funkcji zlicz_roznice_suma_RGB()
 wywołanej z współczynnikiem 5: 0.40251543209876545
liczba niepasujących pikseli z funkcji zlicz_roznice_srednia_RGB()
 wywołanej z współczynnikiem 5: 188525
procent niepasujących pikseli z funkcji zlicz_roznice_srednia_RGB()
 wywołanej z współczynnikiem 5: 0.09091676311728394
Pętla nr. 2: Czas: 13.5
********************************************************************************
 *************
liczba niepasujących pikseli z funkcji zlicz_roznice_suma_RGB()
 wywołanej z współczynnikiem 10: 357316
procent niepasujących pikseli z funkcji zlicz_roznice_suma_RGB()
 wywołanej z współczynnikiem 10: 0.1723167438271605
liczba niepasujących pikseli z funkcji zlicz_roznice_srednia_RGB()
 wywołanej z współczynnikiem 10: 38321
procent niepasujących pikseli z funkcji zlicz_roznice_srednia_RGB()
 wywołanej z współczynnikiem 10: 0.018480420524691357
Pętla nr. 3: Czas: 13.51
********************************************************************************
 *************
liczba niepasujących pikseli z funkcji zlicz_roznice_suma_RGB()
 wywołanej z współczynnikiem 15: 188525
procent niepasujących pikseli z funkcji zlicz_roznice_suma_RGB()
 wywołanej z współczynnikiem 15: 0.09091676311728394
liczba niepasujących pikseli z funkcji zlicz_roznice_srednia_RGB()
 wywołanej z współczynnikiem 15: 6988
procent niepasujących pikseli z funkcji zlicz_roznice_srednia_RGB()
 wywołanej z współczynnikiem 15: 0.0033699845679012346
Pętla nr. 4: Czas: 13.51
********************************************************************************
 *************
liczba niepasujących pikseli z funkcji zlicz_roznice_suma_RGB()
 wywołanej z współczynnikiem 20: 107869
procent niepasujących pikseli z funkcji zlicz_roznice_suma_RGB()
 wywołanej z współczynnikiem 20: 0.052020158179012344
liczba niepasujących pikseli z funkcji zlicz_roznice_srednia_RGB()
 wywołanej z współczynnikiem 20: 837
procent niepasujących pikseli z funkcji zlicz_roznice_srednia_RGB()
 wywołanej z współczynnikiem 20: 0.00040364583333333333
Pętla nr. 5: Czas: 13.52
```

```python
image = Image.open("obraz.jpg")
print("Obraz oryginalny")
statystyki(image)

print('*' * 100)

for i in range(1, 6):
    image.save(f"obraz{i}.jpg")
    image = Image.open(f"obraz{i}.jpg")

image4 = Image.open("obraz4.jpg")
image5 = Image.open("obraz5.jpg")

print("Obraz po 4 zapisach")
statystyki(image4)

print('*' * 100)
print("Obraz po 5 zapisach")
statystyki(image5)
```

Obraz w formacie JPG jest kompresją stratną, co oznacza, że część danych jest trwale usuwana w procesie kompresji. Mimo że po wielu kompresjach może się wydawać, że obraz jest już wystarczająco "dobry", wciąż zachodzi utrata informacji, co oznacza, że nie można przywrócić pełnej jakości oryginalnego obrazu.

Jeśli będziesz porównywać nowy obraz JPG z obrazem JPG, który był wielokrotnie kompresowany i otwierany, to nawet jeśli różnica w jakości będzie trudna do zauważenia wizualnie, nadal będzie istniała różnica w jakości i zawartości pikseli. Obrazy te będą różniły się od siebie, choć może być to trudne do wykrycia. Różnica między obrazem oryginalnym i 5 jest łatwiejsza do zauważenia, niż miedzy 4 i 5. Z każdym zapisem strata jest mniejsza, ale nadal występuje.

```
Obraz oryginalny
extrema  [(0, 255), (2, 255), (2, 255)]
count  [2073600, 2073600, 2073600]
mean  [123.46471547067901, 131.59052131558641, 175.23089023919752]
median  [110, 124, 225]
stddev  [82.81019047668228, 91.2390489018285, 87.99118879980365]
****************************************************************************************************
Obraz po 4 zapisach
extrema  [(0, 255), (0, 255), (0, 255)]
count  [2073600, 2073600, 2073600]
mean  [123.45539496527778, 131.51743875385802, 174.5012471064815]
median  [111, 125, 224]
stddev  [82.60693210408309, 91.08213376053995, 87.17725769736623]
****************************************************************************************************
Obraz po 5 zapisach
extrema  [(0, 255), (0, 255), (0, 255)]
count  [2073600, 2073600, 2073600]
mean  [123.45401523919753, 131.51487654320988, 174.4857638888889]
median  [111, 125, 224]
stddev  [82.60072099782246, 91.07655490170089, 87.16522808853688]
```

Korzystając z funkcji z poprzednich lab, porównałem różnice w pojedynczych pikselach najpierw obraz oryginalny z piątym:

```
Ilość różnic w r: 1478473
Ilość różnic w g: 1189473
Ilość różnic w b: 964206
```

Następnie obraz 4 i 5:

```
Ilość różnic w r: 29306
Ilość różnic w g: 20277
Ilość różnic w b: 7065
```

Na poniższym obrazie zakodowano inny obraz. Gołym okiem jest całkowicie nie widoczny, bo różni się o 1-2 wartości na pikselu.

Poniższy kod korzysta z dwóch obrazów, oryginały i zakodowanego. W pierwszej kolejności zamienia oba obrazy na tablicę RGB. Następnie tworzymy pustą tablicę w formacie uint8 w tym przypadku wypełnionej jedynkami, wypełnienie jej zerami nie zrobi różnicy. Następnie przechodzimy przez cała tablicę, granice range() ustalamy za pomocą metody shape(). W tym przypadku zakładamy, że obie tablice są równe. Jeśli w danych współrzędnych piksele są równe, w odpowiadających współrzędnych na tablicy odkodowanej zamienia piksel na wartość 0, a kiedy znajdzie różnice, zamienia go na 255. Następnie zamienia tablicę na obraz, i zwraca go jako wynik działania funkcji.

```python
def odkoduj(image1, image2):
    tab_image1 = np.array(image1)
    tab_image2 = np.array(image2)
    odkodowana = np.ones(tab_image2.shape, dtype=np.uint8)

    for i in range(tab_image1.shape[0]):
        for j in range(tab_image2.shape[1]):
            if np.array_equal(tab_image1[i][j], tab_image2[i][j]):
                odkodowana[i][j] = 0
            else:
                odkodowana[i][j] = 255
    reultat = Image.fromarray(odkodowana)
    return reultat
```

Teraz wystarczy już tylko wywołać funkcję i zapisać rezultat jako kod2.bmp

```python
zakodowany = Image.open("zakodowany2.bmp")
oryginal = Image.open("jesien.jpg")

odkoduj(zakodowany, oryginal).save("kod2.bmp")
```

A o to rezultat odkodowania: