Sprawozdanie
Inżynieria Obrazów
Laboratorium 1

Katsiaryna Kolyshko 276708

Dr. Inż. Jan Nikodem

Zadanie 1

Wstęp Teoretyczny

Zadanie 1.

Wykonać filtr górnoprzepustowy – tzw. detektor krawędzi tzn.

dla dowolnego obrazka zastosować poniższą maskę,

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Zadanie 1 polega na tym, aby wykonać filtr górnoprzepustowy dla obrazka z maską

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$$

Najpierw powstaje pytanie czym jest filtr górnoprzepustowy.

Filtr górnoprzepustowy to technika przetwarzania obrazów, która uwydatnia wysokie częstotliwości w obrazie, czyli obszary o gwałtownych zmianach intensywności, takie jak krawędzie czy szczegóły. Nazwa "górnoprzepustowy" pochodzi z teorii sygnałów, gdzie filtr ten przepuszcza składowe sygnału o wysokich częstotliwościach, a tłumi składowe o niskich częstotliwościach.

W zadaniu też wystąpiło pojęcie maski. To jest maska konwolucji. W cyfrowym przetwarzaniu obrazów, filtry są implementowane za pomocą masek konwolucji (kerneli), które są reprezentowane jako macierze. Maska jest "przesuwana" po obrazie i w każdym punkcie wykonywana jest operacja konwolucji - suma ważona pikseli oryginalnego obrazu z odpowiednimi wagami z maski.

Maska, przedstawiona w zadaniu, jest często używana w zadaniach, gdzie musimy użyć filtr górnoprzepustowy, dlatego że suma wszystkich elementów wynosi 0, co oznacza, że jeśli obszar obrazu ma jednolity kolor (niskie częstotliwości), wynik konwolucji będzie bliski 0. Natomiast w miejscach gwałtownych zmian intensywności (wysokie częstotliwości), wynik będzie znaczący.

Jeśli suma elementów maski filtru nie jest równa 0, ma to istotny wpływ na charakterystykę filtru i otrzymany obraz wynikowy:

Zmiana jasności obrazu - Gdy suma elementów maski jest dodatnia, ogólna jasność obrazu wzrośnie, a gdy jest ujemna, obraz stanie się ciemniejszy. Jest to efekt dodawania (lub odejmowania) stałej składowej do każdego piksela.

Mieszany charakter filtru - Filtr staje się kombinacją filtru górnoprzepustowego (wykrywającego krawędzie) i dolnoprzepustowego (wpływającego na ogólną jasność obrazu). Nie jest to już "czysty" filtr górnoprzepustowy.

Uwydatnienie krawędzi z zachowaniem kontekstu - Przy sumie lekko dodatniej, krawędzie zostają wykryte, ale pewna część informacji o ogólnym poziomie jasności obszaru jest zachowana. Jest to często używane w tzw. filtrach wyostrzających (sharpening).

Parametry masek:

Rozmiar maski - Większe maski (5×5, 7×7) pozwalają na uwzględnienie większego sąsiedztwa piksela, co może być korzystne przy bardziej złożonych strukturach, ale zwiększa koszt obliczeniowy.

Wartości w masce - Determinują "siłę" filtracji i charakterystykę filtru:

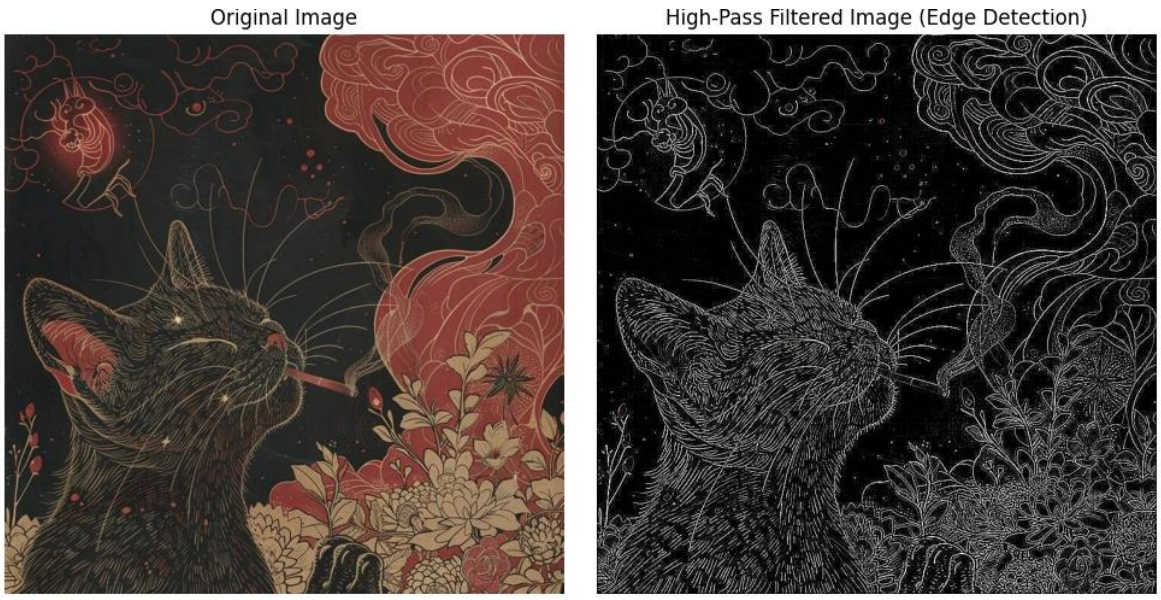
- Większa wartość centralna w stosunku do otoczenia zwiększa kontrast krawędzi
- Modyfikacja wartości negatywnych wpływa na czułość detekcji
- Suma wartości równa 0 zapewnia, że jednorodne obszary będą miały wartość 0

Kierunkowość - Niektóre maski mogą być zaprojektowane do wykrywania krawędzi w określonych kierunkach (poziome, pionowe, ukośne).

Opis działania programu i analiza wyników

Po wykonaniu zadania, otrzymaliśmy następny wykres dla maski:

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$$



Jakie wyniki możemy z tego zrobić?

Co zrobił filtr:

Ekstrakcja krawędzi: Filtr górnoprzepustowy wyizolował i wyróżnił wszystkie krawędzie i kontury na obrazie. Każda linia, krzywa i granica między różnymi elementami (futro kota, wąsy, kwiaty, dekoracyjne zawijasy) zostały zachowane, podczas gdy usunięto informacje o kolorze i cieniowaniu.

Usuwanie tła: Filtr zasadniczo wyeliminował ciemne tło, zamieniając je na całkowicie czarne, zachowując jedynie szczegóły granicy.

Wzmocnienie szczegółów: Drobne szczegóły, takie jak wąsy kota, pojedyncze pasma futra i skomplikowane wzory kwiatów, wydają się bardziej widoczne na przefiltrowanym obrazie.

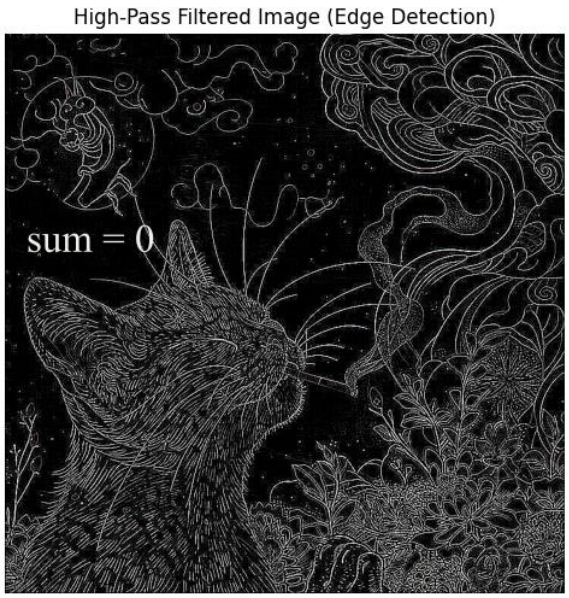
Podkreślenie tekstury: Elementy tekstury obrazu (wzory futra, szczegóły kwiatów) są teraz reprezentowane jako białe linie na czarnym tle.

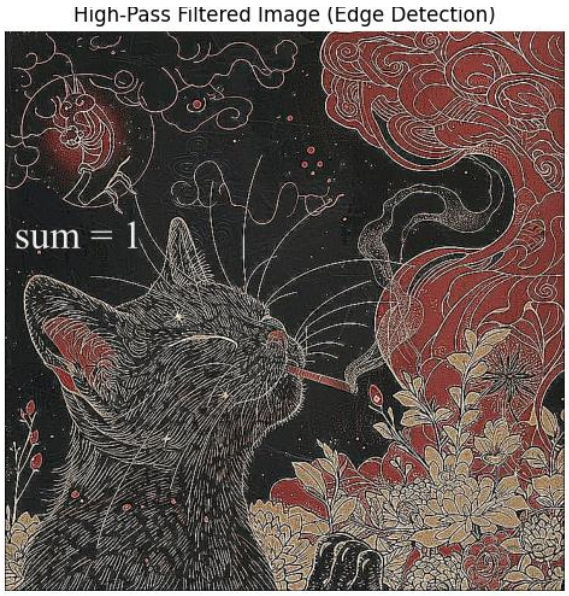
Ta transformacja pokazuje, w jaki sposób filtrowanie górnoprzepustowe izoluje składowe o wysokiej częstotliwości (szybkie zmiany wartości pikseli), które zazwyczaj odpowiadają krawędziom i drobnym szczegółom obrazu.

Dlaczego obrazek, który był kolorowany, teraz jest czarno-biały?

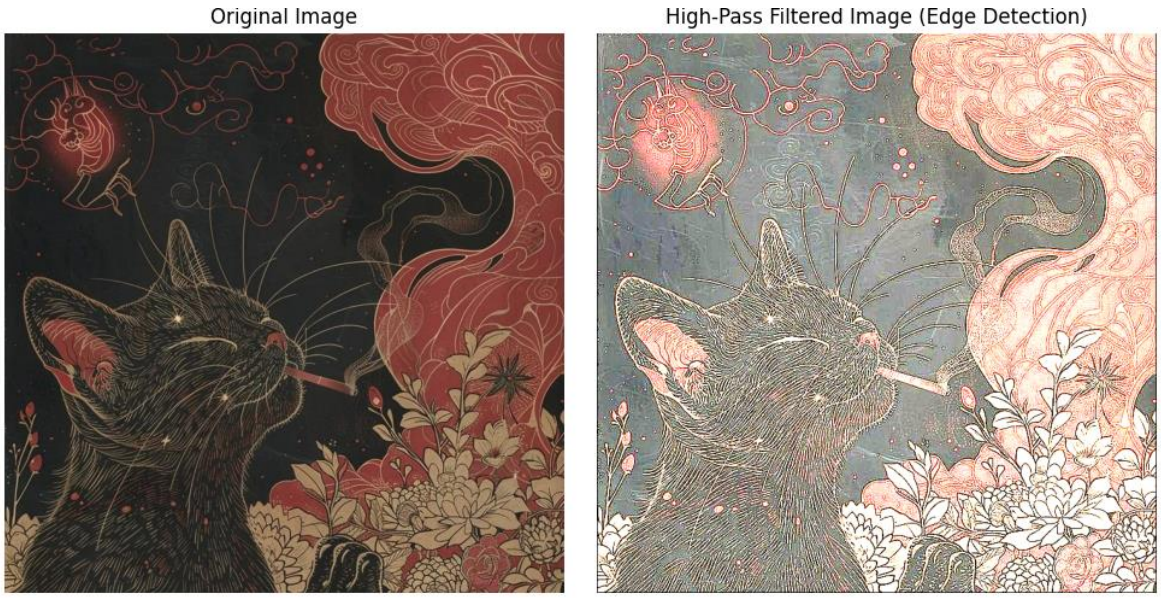
Filtry górnoprzepustowe wykrywają szybkie zmiany wartości pikseli między sąsiadującymi pikselami. Filtry te są zaprojektowane tak, aby reagować na różnice w intensywności, a nie zachowywać oryginalne wartości kolorów. Podczas stosowania filtra do obrazu kolorowego, zazwyczaj przetwarza on każdy kanał koloru (czerwony, zielony, niebieski) niezależnie, wykrywając krawędzie w każdym z nich. Wyniki są następnie często łączone w pojedynczy wynik skali szarości, w którym białe piksele reprezentują wykryte krawędzie, a czarne piksele reprezentują obszary z niewielką lub żadną zmianą

Zamienimy macierz maski i zobaczmy jak zmieni się obrazek:





Wyżej jest przedstawiona różnica pomiędzy obrazkami, gdzie suma krawędzi jest 0 a 1, a niżej, gdzie suma = 5



W przeciwieństwie do poprzedniego czarno-białego wyjścia, ten przefiltrowany obraz zachowuje kolory, chociaż wydają się zmienione. Suma dodatnia (5) tworzy ogólne rozjaśnienie obrazu. Krawędzie wydają się mieć lekko przezroczystą, świecącą jakość. Dzieje się tak, ponieważ suma dodatnia tworzy kombinację wzmocnienia krawędzi i oryginalnych informacji o obrazie.

Technicznie, gdy suma maski wynosi 5:

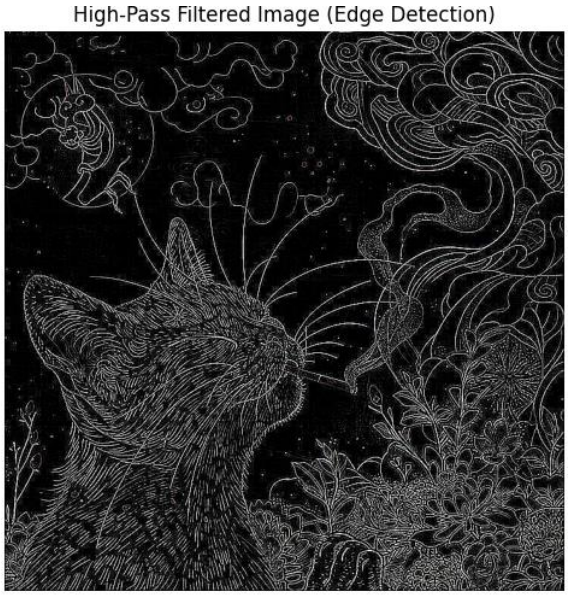
- Filtr nie jest już czysto górnoprzepustowy (wykrywanie krawędzi)
- Staje się kombinacją wzmocnienia krawędzi i wzmocnienia oryginalnego obrazu
- Każda wartość piksela jest pod wpływem zarówno jego cech krawędzi, jak i dodatkowego dodatniego odchylenia
- Tworzy to charakterystyczną „marzycielską” lub eteryczną jakość z ulepszonymi krawędziami, ale nadal zachowującą ogólną strukturę kolorów

Ten typ filtra z dodatnią sumą jest czasami nazywany filtrem „wzmocnienia krawędzi” lub „wysokiego wzmocnienia”, a nie czystym filtrem górnoprzepustowym, ponieważ łączy wykrywanie krawędzi ze wzmocnieniem oryginalnego obrazu.

Popatrzmy na wyniki działania filtru z następną maską:

```
kernel = np.array([
    [1, -1, -1],
    [1, 2, -1],
    [1, -1, -1]
])
```

Suma jest 0, ale z lewej strony mamy dodatnie jedynki, a z prawej – ujemne.



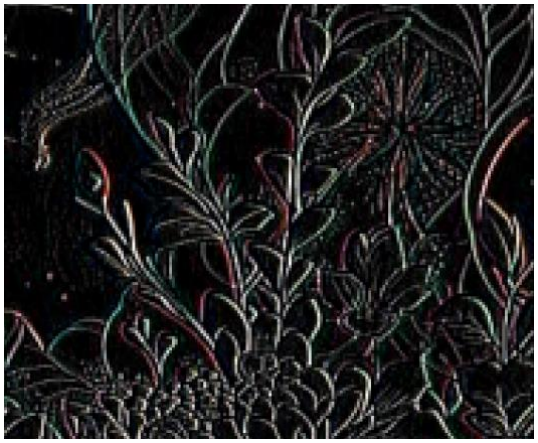
„Symetryczna”

„Asymetryczna”

Jak widać na powyższych obrazkach, obraz po lewej stronie z „asymetryczną” matrycą jest mniej widoczny i wyrazisty. Jeśli nałożymy obrazy na siebie, a pierwszy obraz będzie na asymetrycznym, możemy zobaczyć kilka różnic. Obraz z asymetryczną maską nadal ma oryginalny kolor obrazu (widoczne są czerwone i zielone piksele), a także są one lekko przesunięte w lewo w porównaniu do obrazu z symetryczną maską. Co więcej, szczegóły są znacznie bardziej widoczne na obrazie z symetryczną maską i są również czarno-białe, bez żadnych kolorów w liniach. A jednak niektóre linie są bardziej widoczne na obrazie z asymetryczną maską, jak prawa strona pedałów na kwiatach.



0% nałożenia 50% nałożenia 100% nałożenia




0% nałożenia 50% nałożenia

100% nałożenia

Modyfikując wartości w różnych pozycjach matrycy filtra górnoprzepustowego, możemy znacząco zmienić zachowanie filtra i wynikające z tego cechy obrazu. Zwiększenie wartości środkowej w stosunku do otaczających elementów wzmacnia kontrast wykrytych krawędzi, czyniąc je bardziej widocznymi. Modyfikowanie wartości po określonych stronach (lewa, prawa, góra, dół) wprowadza odchylenie kierunkowe — dodatnie wartości po lewej stronie podkreślą pionowe krawędzie z przejściami z lewej do prawej, podczas gdy dodatnie wartości na górze podkreślą poziome krawędzie z przejściami z góry do dołu. Suma wszystkich elementów macierzy odgrywa kluczową rolę: gdy jest równa zeru (jak w masce symetrycznej), filtr wytwarza prawdziwy efekt wykrywania krawędzi, a krawędzie pojawiają się jako białe linie na czarnym tle; gdy suma jest dodatnia, filtr łączy wzmocnienie krawędzi ze zwiększeniem jasności, zachowując niektóre oryginalne cechy obrazu; gdy suma jest ujemna, krawędzie są nadal wykrywane, ale cały obraz staje się ciemniejszy. Maski asymetryczne (z wartościami dodatnimi skoncentrowanymi po jednej stronie) tworzą efekt pseudo-3D lub wytłoczenia, reagując odmiennie na krawędzie w zależności od ich orientacji, w przeciwieństwie do masek symetrycznych, które reagują jednakowo na krawędzie we wszystkich kierunkach.

Pierwsze zadanie demonstruje zastosowanie filtrów górnoprzepustowych przy użyciu różnych masek splotowych i ich wpływ na przetwarzanie obrazu. Symetryczna maska z wartością środkową 8 otoczona wartościami -1 generuje klasyczny efekt wykrywania krawędzi, podkreślając granice, a jednocześnie tłumiąc jednolite obszary, co skutkuje białymi krawędziami na czarnym tle. Po zmodyfikowaniu w celu utworzenia asymetrycznej maski z dodatnimi wartościami skoncentrowanymi po jednej stronie filtr wprowadza odchylenie kierunkowe, podkreślając krawędzie o określonych orientacjach i tworząc pseudo-3D wytłoczony wygląd. Zmiana sumy elementów macierzy również znacząco wpływa na wyniki: suma zerowa utrzymuje prawdziwe wykrywanie krawędzi, suma dodatnia łączy wzmocnienie krawędzi z ogólnym wzrostem jasności (zachowując pewne informacje o kolorze). Te zmiany ilustrują, jak subtelne zmiany w strukturze maski filtra mogą radykalnie zmienić cechy wizualne przetworzonego obrazu, jednocześnie wykonując wzmocnienie o wysokiej częstotliwości.

Kod

Funkcja high_pass_filter(image_path)

Ta funkcja aplikuje filtr górnoprzepustowy do obrazu, uwydatniając krawędzie i szczegóły. Funkcja:

- Przyjmuje ścieżkę do pliku obrazu jako parametr wejściowy
- Przetwarza obraz za pomocą konwolucji z jądrem górnoprzepustowym
- Wyświetla zarówno oryginalny, jak i przefiltrowany obraz
- Zwraca tablicę przefiltrowanego obrazu

Wczytywanie obrazu i automatyczne przetwarzanie

- img = Image.open(image_path): Wczytuje obraz z podanej ścieżki
- img_array = np.array(img): Konwertuje obraz na tablicę NumPy do przetwarzania
- Kod automatycznie wykrywa typ obrazu (kolorowy lub czarno-biały) na podstawie kształtu tablicy

```
    # Read the image (PIL handles the image type automatically)
    img = Image.open(image_path)

    # Convert image to numpy array for processing
    img_array = np.array(img)
```

Definicja maski:

```
# Define the high-pass filter kernel (Sobel operator for better edge detection)
kernel = np.array([
    [-1, -1, -1],
    [-1, 8, -1],
    [-1, -1, -1]
])
```

Pobieranie rozmiaru foto:

```
    # Get image dimensions
    if len(img_array.shape) == 3:  # Color image (has channels)
        height, width, channels = img_array.shape
    else:  # Grayscale image
        height, width = img_array.shape
        # Expand dimensions to make processing uniform
        img_array = np.expand_dims(img_array, axis=2)
        channels = 1
```

Konwulsja:

```
    # Apply the filter using convolution
    for c in range(channels):
        for i in range(1, height - 1):
            for j in range(1, width - 1):
                # Extract the 3x3 neighborhood around the pixel
                neighborhood = img_array[i - 1:i + 2, j - 1:j + 2, c if channels > 1 else 0]
                # Apply the kernel
                value = np.sum(neighborhood * kernel)
                # Clip values to valid range
                filtered_img[i, j, c if channels > 1 else 0] = np.clip(value, a_min: 0, a_max: 255)
```

Kod wykonuje konwolucję w ujednolicony sposób: tymczasowo dodaje wymiar kanału do obrazów czarno-białych dla spójnego przetwarzania, po czym iteruje przez każdy piksel obrazu (z wyjątkiem krawędzi), wyodrębniając sąsiedztwo 3×3 wokół każdego piksela, następnie aplikuje jądro filtru przez pomnożenie wartości sąsiednich pikseli przez wagi jądra, a na końcu sumuje wyniki i ogranicza wartości do prawidłowego zakresu [0, 255].

Konwolucja to operacja matematyczna, w której jądro filtru (mała macierz wag) jest przesuwane po obrazie i dla każdej pozycji obliczana jest suma iloczynów wartości jądra i odpowiadających im pikseli obrazu. W przetwarzaniu obrazów konwolucja pozwala na wykrywanie krawędzi, rozmycie, wyostrzenie oraz inne transformacje, zależnie od użytego jądra filtru.

```
    # Remove the extra dimension for grayscale images
    if channels == 1:
        filtered_img = filtered_img[:, :, 0]
        img_array = img_array[:, :, 0]
```

Gdy kod wykryje obraz w skali szarości (który ma 2 wymiary: wysokość i szerokość), tymczasowo dodaje trzeci wymiar (wymiar kanału), aby przetwarzanie było jednolite z obrazami kolorowymi. Po zakończeniu całego przetwarzania kod musi usunąć ten dodatkowy wymiar, który został dodany w celach przetwarzania.

Jest to konieczne, ponieważ chcemy zwrócić obraz w jego oryginalnym formacie, a matplotlib oczekuje, że obrazy w skali szarości będą tablicami 2D (bez wymiaru kanału) podczas korzystania z cmap='gray'

Wyświetlenie obrazów do i po filtrowaniu:

```
    # Display the original and filtered images
    plt.figure(figsize=(10, 5))

    plt.subplot(121)
    plt.imshow(img_array, cmap='gray' if channels == 1 else None)
    plt.title('Original Image')
    plt.axis('off')

    plt.subplot(122)
    plt.imshow(filtered_img, cmap='gray' if channels == 1 else None)
    plt.title('High-Pass Filtered Image (Edge Detection)')
    plt.axis('off')

    plt.tight_layout()
    plt.show()

    return filtered_img
```

Zadanie 2

Wstęp Teoretyczny

Zadanie 2.

Przekształcić kolory obrazu tzn.

- dla dowolnego obrazka kolorowego RGB [0-255;0-255;0-255]
  dokonać konwersji na format zmiennoprzecinkowy RGB [0-1.0;0-1.0;0-1.0]

- wyznaczyć nowe wartości według poniższego wzoru

$$\begin{bmatrix} R_{new} \\ G_{new} \\ B_{new} \end{bmatrix} = \begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.689 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

jeżeli któraś z nowych wartości przekroczy **1.0**, należy ją przyjąć jako **1.0**.

Zadanie 2 dotyczy przekształcenia przestrzeni barw obrazu. W przetwarzaniu obrazów, transformacja kolorów jest istotnym elementem pracy z obrazami cyfrowymi, pozwalającym na zmianę charakterystyki barwnej czy wzmocnienie określonych cech wizualnych.

W tym zadaniu zajmujemy się specyficznym rodzajem transformacji kolorystycznej - konwersją standardowej przestrzeni RGB (gdzie wartości kanałów mieszczą się w zakresie 0-255) na format zmiennoprzecinkowy RGB (o zakresie 0-1), a następnie zastosowaniem macierzy transformacji do uzyskania nowych wartości kolorów. Macierz transformacji 3×3 zawiera współczynniki, które określają, w jakim stopniu każda składowa oryginalnego koloru (R, G, B) wpływa na wartość nowej składowej (R_{new}, G_{new}, B_{new}).

Taka transformacja pozwala na precyzyjną modyfikację barw obrazu poprzez zdefiniowanie konkretnych zależności między kanałami. Może być używana do korekcji kolorów, symulacji różnych warunków oświetleniowych, kompensacji wadliwych czujników czy artystycznej modyfikacji obrazu. Ponieważ niektóre wartości po transformacji mogą przekroczyć dozwolony zakres (1.0), zadanie przewiduje ograniczenie (clipping) wartości przekraczających ten próg.

Po wykonaniu zadania powinniśmy zobaczyć obraz z przekształconą kolorystyką, gdzie nowe barwy będą rezultatem zastosowania podanej macierzy transformacji. W zależności od użytej macierzy, możemy spodziewać się różnych efektów wizualnych - od subtelnych zmian tonalnych po znaczące modyfikacje kolorystyczne.

Opis działania programu i analiza wyników

Wyniki dla wartości RGB z zadania (Zółty):

Wartości macierzy:

```
25      # Define the transformation matrix
26      transformation_matrix = np.array([
27          [0.393, 0.769, 0.189],
28          [0.349, 0.686, 0.168],
29          [0.272, 0.534, 0.131]
30      ])
```

Jak widać na zdjęciu, program faktycznie zmienił kolor zdjęcia, sprawiając, że wygląda ono „staro”, przypominając starą fotografię, zmieniając główny kolor na odcień żółtego.

Wyniki dla wartości RGB dowolne (Czerwone):

Wartości macierzy:

```
#    transformation_matrix = np.array([
#        [0.840, 0.370, 0.160],  # Strong red output
#        [0.120, 0.530, 0.140],  # Reduced green
#        [0.180, 0.270, 0.350]  # Moderate blue for wine-like depth
#    ])
```

Tak jak na powyższym obrazku, zmieniliśmy kolor podstawowy na czerwony wino, nawet jeśli mieliśmy już czerwone kolory na obrazku, to uczyniło je bardziej żywymi i zmieniło odcień czerwieni na chmurach i uszach. Zmieniając wartości macierzy, możemy manipulować kolorem filtra z żółtego na czerwony i na dowolny kolor, jaki chcemy.

Wyniki dla wartości RGB z zadania (żółty filtr) pokazały, że transformacja macierzowa skutecznie zmieniła kolorystykę obrazu, nadając mu charakterystyczny żółtawy odcień przypominający stare fotografie, przy jednoczesnym zachowaniu wszystkich szczegółów i struktury oryginalnego zdjęcia. Zastosowanie alternatywnej macierzy transformacji (czerwony filtr) doprowadziło do wzmocnienia czerwonych tonów, szczególnie widocznych na obszarach, które już zawierały czerwone elementy, takich jak uszy kota czy chmury. Eksperymenty potwierdziły, że manipulując wartościami macierzy transformacji kolorów, możemy precyzyjnie kontrolować efekt końcowy i uzyskiwać szeroki zakres modyfikacji kolorystycznych - od subtelnych zmian tonalnych po wyraźne artystyczne przekształcenia. Ograniczenie (clipping) wartości przekraczających 1.0 zapewniło, że obraz pozostał w prawidłowym zakresie wartości RGB, niezależnie od zastosowanych współczynników transformacji.

Kod

Program wczyta foto:

```
8       # Read the image
9       img = Image.open(image_path)
```

Konwertuj do tablicy

```
14      # Convert image to numpy array
15      img_array = np.array(img)
```

Linijka kodu img_array = np.array(img) konwertuje obraz z formatu PIL (Python Imaging Library) na tablicę NumPy. Wykonujemy tę konwersję, ponieważ NumPy oferuje wydajne operacje matematyczne na tablicach, których potrzebujemy do przeprowadzenia konwolucji.

Normalizacja wartości pikseli

```
17      # Normalize values to 0-1 range
18      img_norm = img_array.astype(float) / 255.0
```

Linijka kodu img_norm = img_array.astype(float) / 255.0 normalizuje wartości pikseli z zakresu 0-255 do zakresu 0-1. Jest to niezbędne do precyzyjnych operacji matematycznych na kolorach, zwłaszcza przy transformacjach macierzowych. Normalizacja ułatwia późniejsze przycinanie wartości do poprawnego zakresu i zapewnia stabilność numeryczną podczas obliczeń.

Definicji matryc

```
20      # Define the transformation matrix
21  #    transformation_matrix = np.array([
22  #        [0.393, 0.769, 0.189],
23  #        [0.349, 0.686, 0.168],
24  #        [0.272, 0.534, 0.131]
25  #    ])
26
27      transformation_matrix = np.array([
28          [0.840, 0.370, 0.160],  # Strong red output
29          [0.120, 0.530, 0.140],  # Reduced green
30          [0.180, 0.270, 0.350]  # Moderate blue for wine-like depth
31      ])
```

Na obrazu są macierzy kolorów żółty i czerwony. Zmieniając wartości macierzy będzie zmieniało kolor ostatecznego foto.

Transformacja foto

```
    # Apply transformation
    # Reshape for matrix multiplication
    height, width, channels = img_norm.shape
    pixels = img_norm.reshape((height * width, channels))

    # Apply transformation to each pixel
    transformed_pixels = np.dot(pixels, transformation_matrix.T)

    # Clip values to ensure they're in the 0-1 range
    transformed_pixels = np.clip(transformed_pixels, a_min: 0.0, a_max: 1.0)

    # Reshape back to image format
    transformed_img = transformed_pixels.reshape((height, width, channels))

    # Convert back to 0-255 range for display
    transformed_img_display = (transformed_img * 255).astype(np.uint8)

```

Najpierw program przekształca obraz z trójwymiarowej tablicy (wysokość, szerokość, kanały) na dwuwymiarową tablicę pikseli, aby umożliwić mnożenie macierzowe. Następnie aplikuje macierz transformacji do każdego piksela za pomocą operacji np.dot, co zmienia wartości RGB. Wartości są przycinane do zakresu 0-1, aby uniknąć nieprawidłowych kolorów. Na końcu, przekształcona tablica jest przywracana do oryginalnego kształtu obrazu i konwertowana z powrotem do zakresu 0-255 dla poprawnego wyświetlania.

Zadanie 3

Wstęp Teoretyczny

Zadanie 3.

Dokonać konwersji obrazu RGB do modelu barw YCbCr tzn.

- dla dowolnego obrazka kolorowego RGB [0-255;0-255;0-255] dokonać konwersji według poniższego wzoru (YCrCb)

$$\begin{bmatrix} Y \\ Cr \\ Cb \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.229 & 0.587 & 0.114 \\ 0.500 & -0.418 & -0.082 \\ -0.168 & -0.331 & 0.500 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- wyświetlić oryginalny obraz, składowe Y, Cb, Cr w odcieniach szarości oraz obraz po konwersji odwrotnej.

Zadanie 3 dotyczy konwersji obrazu pomiędzy różnymi przestrzeniami barw, ze szczególnym uwzględnieniem przestrzeni RGB oraz YCbCr. Przestrzenie barw to matematyczne modele reprezentacji kolorów, które umożliwiają zapisywanie i przetwarzanie informacji o kolorach w sposób dostosowany do różnych zastosowań i urządzeń.

Przestrzeń YCbCr to model reprezentacji kolorów, który rozdziela informację o jasności (luminancja - składowa Y) od informacji o kolorze (chrominancja - składowe Cb i Cr). W przeciwieństwie do RGB, gdzie wszystkie trzy kanały (czerwony, zielony, niebieski) zawierają zarówno informacje o jasności, jak i kolorze, YCbCr rozdziela te informacje, co ma istotne zalety w wielu zastosowaniach.

Składowe YCbCr oznaczają:

- **Y** (Luminancja) - odpowiada za jasność obrazu i zawiera większość informacji o szczegółach. Jest to ważona suma kanałów RGB, przy czym kanał zielony ma największy wkład, ponieważ ludzkie oko jest najbardziej wrażliwe na ten kolor.
- **Cb** (Chrominancja niebieska) - reprezentuje różnicę między składową niebieską a wartością odniesienia, wskazując na ilość "niebieskiego" koloru.
- **Cr** (Chrominancja czerwona) - reprezentuje różnicę między składową czerwoną a wartością odniesienia, wskazując na ilość "czerwonego" koloru.

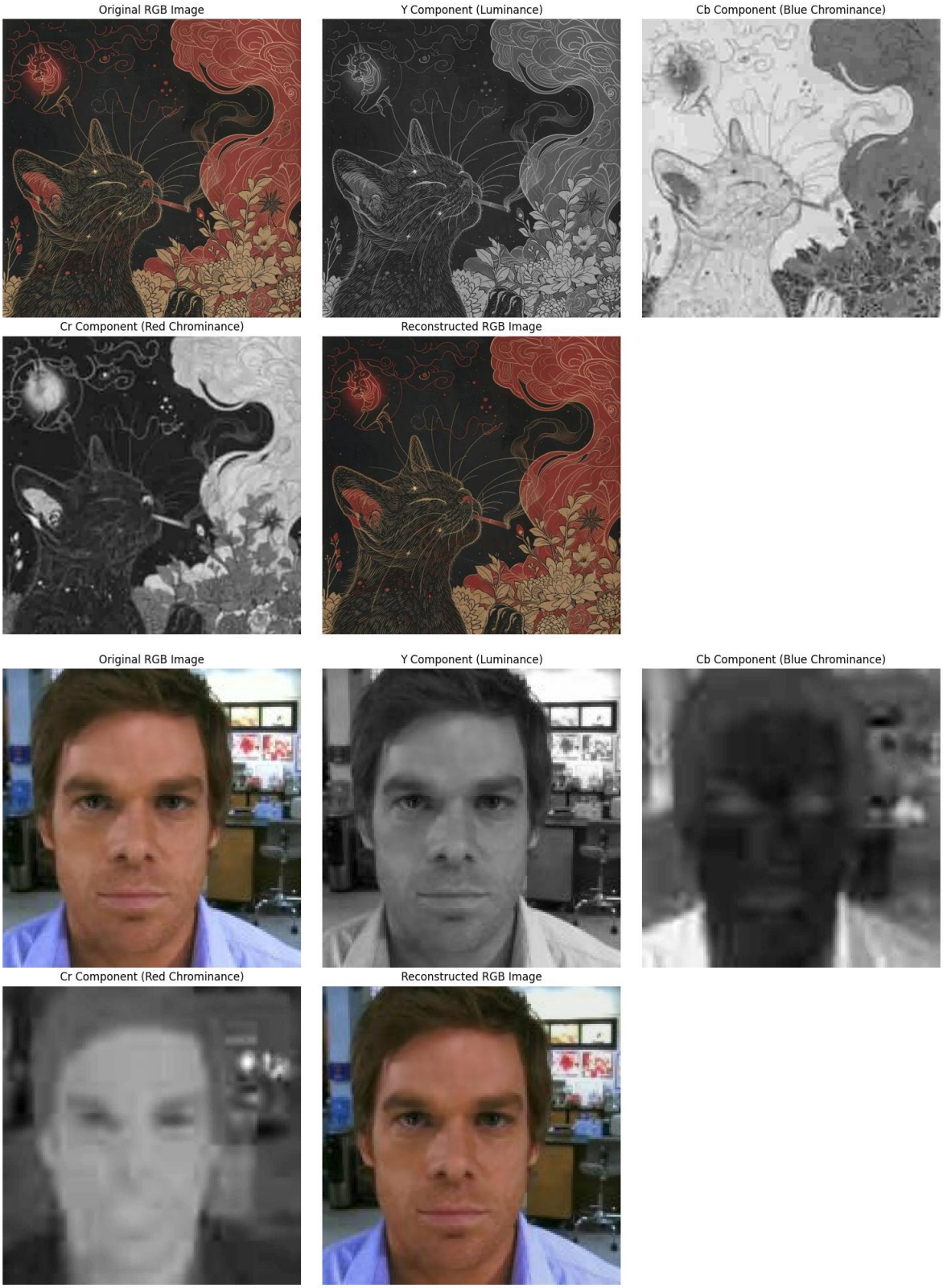
Konwersja z RGB do YCbCr jest procesem liniowym, realizowanym za pomocą operacji macierzowych. Dla standardowych wartości RGB (0-255), wzory konwersji są następujące:

Y = 0.299R + 0.587G + 0.114B Cb = 128 - 0.168736R - 0.331264G + 0.5B Cr = 128 + 0.5R - 0.418688G - 0.081312B

Jedną z głównych zalet przestrzeni YCbCr jest możliwość kompresji informacji o kolorze bez znaczącego wpływu na postrzeganą jakość obrazu. Ludzkie oko jest bardziej wrażliwe na zmiany jasności niż koloru, dlatego w wielu systemach kompresji (np. JPEG, MPEG) składowe chrominancji (Cb i Cr) są próbkowane z mniejszą rozdzielczością niż luminancja (Y), co pozwala na znaczne zmniejszenie ilości danych przy minimalnym pogorszeniu jakości postrzeganej przez człowieka.

Opis działania programu i analiza wyników

Na przedstawionych obrazach widzimy proces dekompozycji i rekonstrukcji obrazów RGB w przestrzeni YCbCr. Dla obu zestawów obrazów (artystycznego obrazu kota oraz zdjęcia mężczyzny) przeprowadzono konwersję z przestrzeni RGB do YCbCr, a następnie zrekonstruowano obraz RGB z poszczególnych składowych



Analiza składowych YCbCr

1. **Składowa Y (Luminancja)**:
    - Jest przedstawiona jako obraz w skali szarości, zawierający informacje o jasności obrazu.

- Wyraźnie widoczne są wszystkie szczegóły strukturalne obrazu, kontury i granice obiektów.
- W przypadku zdjęcia mężczyzny (Dekster), doskonale widać rysy twarzy, a w przypadku obrazu kota - wszystkie detale futra, kwiatów i konturów.
- Jest to jedyna składowa, która niesie większość informacji o kształtach i teksturach.

2. **Składowa Cb (Chrominancja niebieska)**:

    - Reprezentowana jako obraz w skali szarości, gdzie jaśniejsze obszary odpowiadają większej zawartości koloru niebieskiego.
    - W przypadku kota, widać znaczący kontrast między jasnymi (niebieskimi) elementami tła a ciemniejszymi obszarami, gdzie dominuje czerwień.
    - Na zdjęciu mężczyzny, niebieska koszula jest wyraźnie jaśniejsza w składowej Cb.

3. **Składowa Cr (Chrominancja czerwona)**:

    - Również przedstawiona jako obraz w skali szarości, gdzie jaśniejsze obszary oznaczają większą zawartość koloru czerwonego.
    - W obrazie kota, czerwone kwiaty i elementy są widoczne jako jasne obszary.
    - W przypadku zdjęcia mężczyzny, skóra twarzy (zawierająca odcienie czerwieni) jest jaśniejsza niż niebieska koszula.

Proces rekonstrukcji

Rekonstrukcja obrazu RGB z trzech składowych YCbCr polega na zastosowaniu odwrotnej transformacji matematycznej. Każdy kanał RGB jest odtwarzany na podstawie kombinacji wartości Y, Cb i Cr według określonych wzorów:

$R = Y + 1.402 \times (Cr - 128)$
$G = Y - 0.344136 \times (Cb - 128) - 0.714136 \times (Cr - 128)$
$B = Y + 1.772 \times (Cb - 128)$

Analizując obrazy oryginalne i zrekonstruowane, można zauważyć:

- W obu przypadkach zrekonstruowane obrazy są wizualnie niemal identyczne z oryginałami.
- Nie widać znaczących różnic w kolorach, szczegółach czy ogólnej jakości.

Dlaczego składowe są czarno-białe?

Składowe Y, Cb i Cr są przedstawione jako obrazy czarno-białe (w skali szarości), ponieważ każda z nich reprezentuje pojedynczą wartość liczbową dla każdego piksela:

- **Y** zawiera informację wyłącznie o jasności (od czerni do bieli).

- **Cb** i **Cr** reprezentują odchylenia od wartości neutralnej w kierunku odpowiednio niebieskiego i czerwonego koloru. Wartość neutralna (128 w skali 0-255) jest reprezentowana jako szary, wartości wyższe są jaśniejsze, a niższe ciemniejsze.

Ta reprezentacja pozwala na wizualizację informacji zawartej w każdej składowej, mimo że same w sobie nie są one kolorowe. Dopiero połączenie wszystkich trzech składowych według odpowiednich wzorów matematycznych pozwala odtworzyć pełny kolor RGB.

Eksperyment potwierdza, że przestrzeń YCbCr skutecznie rozdziela informację o jasności od informacji o kolorze. Składowa Y zawiera większość informacji strukturalnych, podczas gdy Cb i Cr zawierają informacje o kolorze. Ta właściwość jest szeroko wykorzystywana w kompresji obrazów i wideo (np. JPEG, MPEG), gdzie składowe chrominancji mogą być próbkowane z mniejszą rozdzielczością niż luminancja, co pozwala na znaczne zmniejszenie ilości danych przy minimalnym wpływie na jakość postrzeganą przez człowieka.

Kod

Normalizacja pikseli i pobieranie RGB kanałów

```
17      # Normalize RGB values to 0-1 range
18      rgb_norm = img_array.astype(float) / 255.0
19
20      # Get individual RGB channels
21      R = rgb_norm[:, :, 0]
22      G = rgb_norm[:, :, 1]
23      B = rgb_norm[:, :, 2]
```

Ten fragment kodu przygotowuje wartości RGB obrazu do konwersji kolorów. Najpierw normalizuje wartości pikseli z zakresu 0-255 do zakresu 0-1 poprzez konwersję na typ zmiennoprzecinkowy i dzielenie przez 255.0. Następnie wyodrębnia poszczególne kanały kolorów (czerwony, zielony i niebieski) do oddzielnych tablic dwuwymiarowych. Takie rozdzielenie kanałów jest niezbędne do przeprowadzenia konwersji z przestrzeni RGB do YCbCr, gdzie każdy kanał YCbCr jest obliczany jako liniowa kombinacja kanałów RGB.

Obliczanie macierzy YCrCb i jej implementacja

```
25      # Define the YCbCr conversion matrix from the task
26      # Y  = 0.229*R + 0.587*G + 0.114*B + 0
27      # Cr = 0.500*R - 0.418*G - 0.082*B + 128
28      # Cb = -0.168*R - 0.331*G + 0.500*B + 128
29
30      # Calculate YCbCr channels
31      Y = 0.229 * R + 0.587 * G + 0.114 * B
32      Cr = 0.500 * R - 0.418 * G - 0.082 * B + 128 / 255.0  # Normalize the offset
33      Cb = -0.168 * R - 0.331 * G + 0.500 * B + 128 / 255.0  # Normalize the offset
```

Fragment realizuje konwersję z przestrzeni barw RGB do YCbCr według określonych wzorów matematycznych. Składowa Y (luminancja) reprezentuje jasność obrazu i jest obliczana jako ważona suma kanałów RGB, gdzie największy wpływ ma kanał zielony. Składowe Cr i Cb reprezentują. Do obu składowych chrominancji dodawany jest offset 128 (znormalizowany do zakresu 0-1 przez podzielenie przez 255), aby umożliwić reprezentację wartości ujemnych w standardowym formacie obrazu.

Wracanie do obrazu RGB

```
35      # Combine into YCbCr image (just for storage)
36      ycbcr = np.stack( arrays: (Y, Cb, Cr), axis=2)
37
38      # Convert back to RGB
39      # Define inverse matrix for YCbCr to RGB conversion
40      # R = Y + 1.402*(Cr-128/255)
41      # G = Y - 0.344*(Cb-128/255) - 0.714*(Cr-128/255)
42      # B = Y + 1.772*(Cb-128/255)
43
44      R_conv = Y + 1.402 * (Cr - 128 / 255.0)
45      G_conv = Y - 0.344 * (Cb - 128 / 255.0) - 0.714 * (Cr - 128 / 255.0)
46      B_conv = Y + 1.772 * (Cb - 128 / 255.0)
47
48      # Clip values to ensure they're in the 0-1 range
49      R_conv = np.clip(R_conv,  a_min: 0.0,  a_max: 1.0)
50      G_conv = np.clip(G_conv,  a_min: 0.0,  a_max: 1.0)
51      B_conv = np.clip(B_conv,  a_min: 0.0,  a_max: 1.0)
```

Fragment łączy obliczone kanały Y, Cb i Cr w jedną trójwymiarową tablicę reprezentującą obraz w przestrzeni YCbCr, używając funkcji np.stack. Następnie wykonuje konwersję odwrotną - z YCbCr z powrotem do RGB - stosując odpowiednie wzory matematyczne. Po przekształceniu wartości są przycinane funkcją np.clip do zakresu 0-1, aby zapewnić, że wynikowe kolory są w prawidłowym zakresie i nie wystąpią zniekształcenia przy wyświetlaniu obrazu. Następnie, zbiera elementy i tworzy nowe foto RGB

```
53      # Combine back to RGB
54      rgb_reconstructed = np.stack( arrays: (R_conv, G_conv, B_conv), axis=2)
```

Ten fragment konwertuje znormalizowane wartości pikseli z zakresu 0-1 z powrotem do standardowego zakresu 0-255 używanego w obrazach cyfrowych i zmienia typ danych na 8-bitowe liczby całkowite bez znaku (uint8), aby umożliwić poprawne wyświetlenie obrazu.

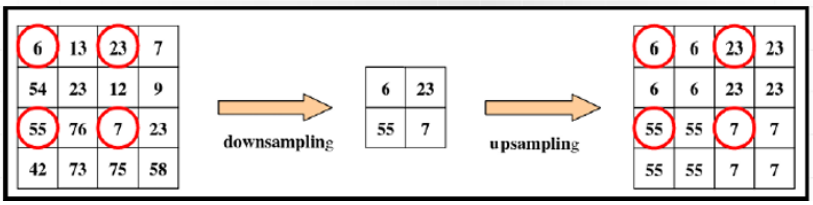
```
56      # Convert back to 0-255 range for display
57      rgb_reconstructed_display = (rgb_reconstructed * 255).astype(np.uint8)
```

Zadanie 4

Wstęp Teoretyczny

Zadanie 4.

Wykonać symulację transmisji obrazu w systemie DVB tzn.
- przeprowadzić konwersję obrazu z modelu RGB do YCbCr (zadanie 3),
- na kanałach Cb i Cr zrealizować operację downsamplingu,
- przeprowadzić operację upsamplingu na macierzach kanałów Cb i Cr,
- złożyć obraz RGB z otrzymanych macierzy Cb i Cr oraz Y,
- wyświetlić obraz oryginalny i po transmisji oraz poszczególne składowe (YCbCr) w odcieniach szarości.



Opis działania programu i analiza wyników

Zadanie 4 dotyczy symulacji transmisji obrazu w systemie DVB (Digital Video Broadcasting), w którym kluczową rolę odgrywa optymalizacja przesyłu danych przy zachowaniu odpowiedniej jakości obrazu. System ten wykorzystuje przestrzeń barw YCbCr, która skutecznie rozdziela informację o jasności (Y) od informacji o kolorze (Cb i Cr).

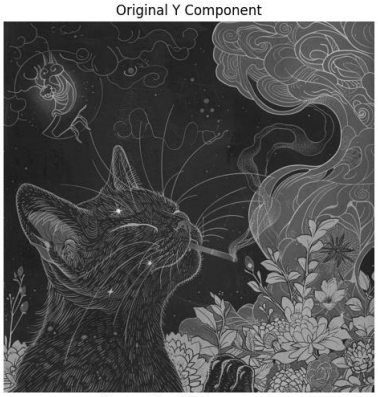
Istotą tego zadania jest implementacja techniki kompresji stosowanej w rzeczywistych systemach transmisji obrazu - downsamplingu (podpróbkowania) kanałów chrominancji. Technika ta opiera się na właściwości ludzkiego oka, które jest bardziej wrażliwe na zmiany jasności niż koloru. Dzięki temu możliwe jest zmniejszenie rozdzielczości składowych kolorystycznych (Cb i Cr) bez znaczącego wpływu na postrzeganą jakość obrazu.

Po stronie odbiornika konieczne jest przeprowadzenie procesu upsamplingu (nadpróbkowania), który rekonstruuje pełnowymiarowe macierze Cb i Cr na podstawie przesłanych danych o zredukowanej rozdzielczości.

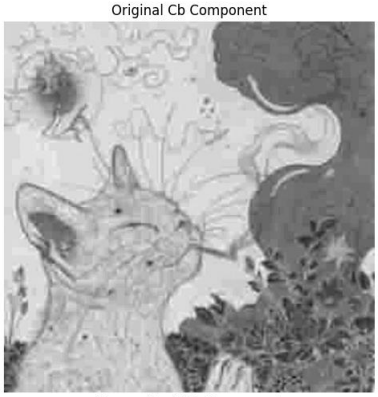
Porównanie obrazu oryginalnego z obrazem po transmisji pozwoli zaobserwować wpływ tych operacji na ostateczną jakość wizualną, a analiza poszczególnych składowych YCbCr pomoże zrozumieć, w jaki sposób informacje o obrazie są kodowane i przetwarzane w systemach transmisji cyfrowej.

Original RGB Image

Original Y Component

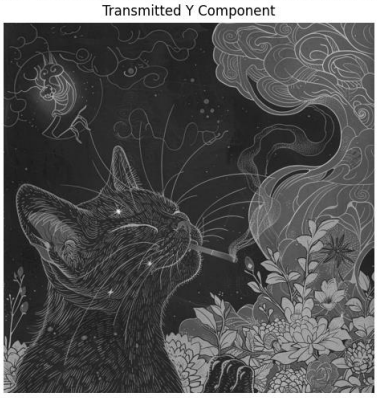


Original Cb Component

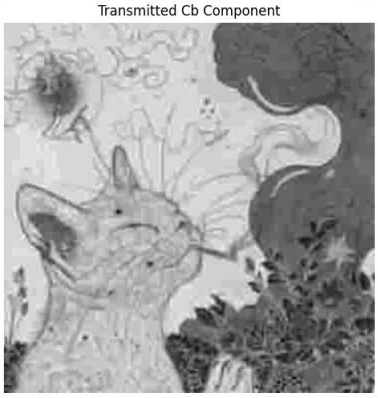


After DVB Transmission

Transmitted Y Component

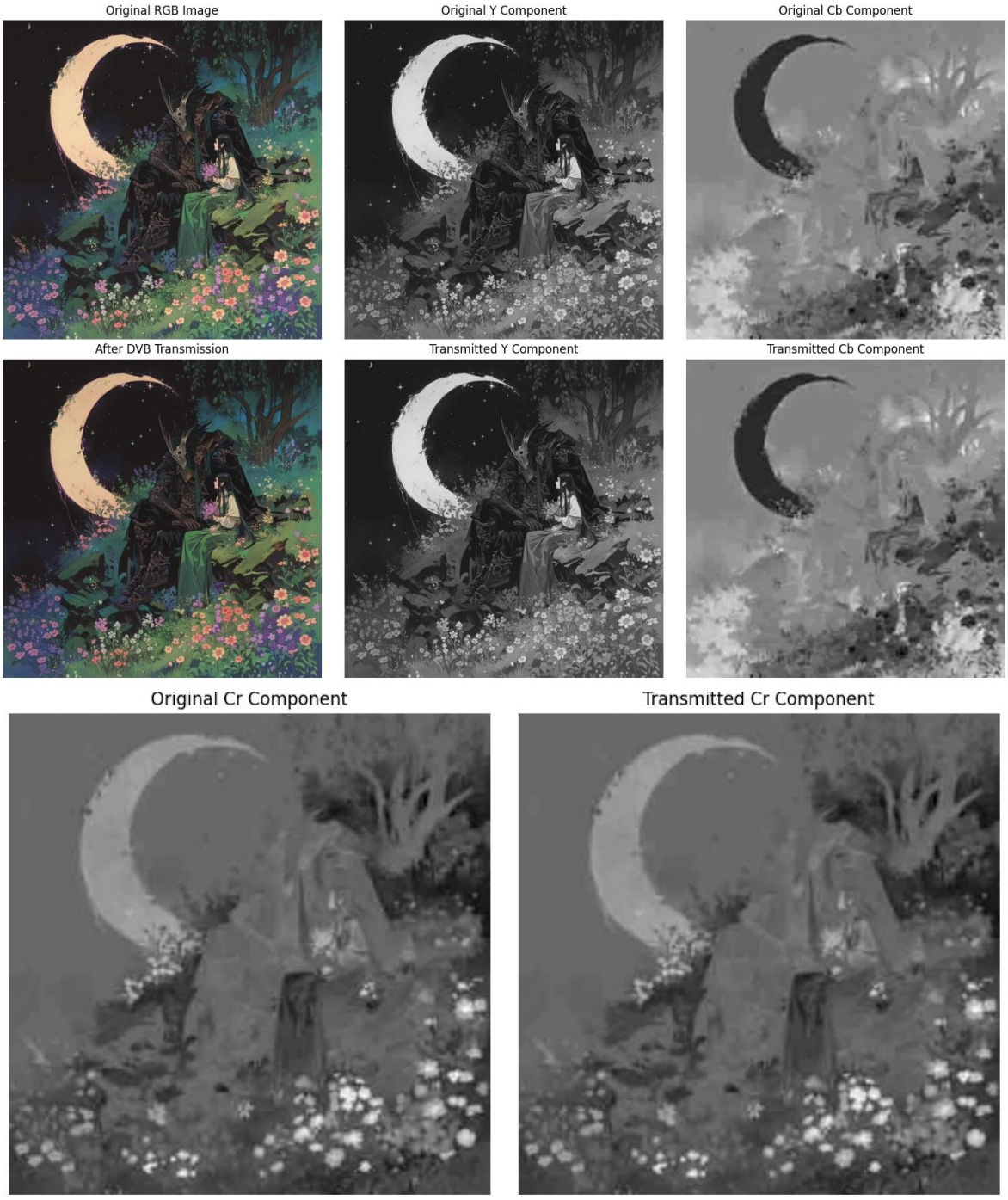


Transmitted Cb Component



Original Cr Component

Transmitted Cr Component



W ramach zadania 4 przeprowadzono symulację transmisji obrazu w systemie DVB, obejmującą proces konwersji do przestrzeni YCbCr, operacje downsamplingu i upsamplingu kanałów chrominancji oraz rekonstrukcję obrazu RGB.

Porównanie wyników:

Składowa Y (luminancja) w przeprowadzonej symulacji transmisji DVB została zachowana bez modyfikacji rozdzielczości, co potwierdza porównanie oryginalnej i transmitowanej wersji - brak widocznych różnic między nimi. W przeciwieństwie do kanałów chrominancji, kanał luminancji nie podlega operacji downsamplingu, zachowując pełną rozdzielczość i wszystkie szczegóły strukturalne obrazu. Dzięki temu kluczowe informacje o konturach, kształtach i

teksturach zostają w pełni zachowane w procesie transmisji, co jest zgodne z technicznymi założeniami systemu DVB.

Składowe Cb i Cr (chrominancja) po przeprowadzeniu pełnego cyklu operacji downsamplingu i upsamplingu wykazują wysoką jakość rekonstrukcji bez wyraźnych artefaktów czy utraty informacji wizualnej. Zastosowana redukcja rozdzielczości kanałów chrominancji w procesie downsamplingu, a następnie ich odtworzenie poprzez upsampling z wykorzystaniem metod interpolacji, daje rezultaty technicznie satysfakcjonujące. Efektywność tego procesu wynika z charakterystyki sygnałów chrominancji, które zwykle cechują się niższą częstotliwością przestrzenną i większą ciągłością w obrębie obszarów obrazu, co sprawia, że algorytmy interpolacyjne mogą skutecznie rekonstruować pełnowymiarowe macierze Cb i Cr na podstawie zredukowanych danych.

Zrekonstruowany obraz RGB wykazuje wysoką wierność względem oryginału, zachowując szczegółowość oraz charakterystykę barwną mimo zastosowanego procesu kompresji kanałów chrominancji. Analiza techniczna potwierdza skuteczność zastosowanego schematu podpróbkowania, który pozwala zredukować ilość przesyłanych danych o około 50% (przy standardzie 4:2:0) bez wprowadzania dostrzegalnych strat jakościowych. Dokładna rekonstrukcja drobnych szczegółów kolorystycznych (widocznych na kwiatach i innych elementach obrazów testowych) świadczy o efektywności algorytmów interpolacyjnych zastosowanych w procesie upsamplingu. Proces ten stanowi podstawę współczesnych systemów kompresji obrazu, umożliwiając znaczną redukcję zapotrzebowania na przepustowość przy zachowaniu wysokiej jakości wizualnej transmitowanych obrazów.

Przeprowadzona symulacja transmisji DVB działa zgodnie z oczekiwaniami i potwierdza skuteczność zastosowanej strategii kompresji. Operacje downsamplingu i upsamplingu kanałów chrominancji pozwalają na znaczną redukcję ilości przesyłanych danych przy zachowaniu wysokiej jakości wizualnej obrazu.

Ta technika jest fundamentem wielu nowoczesnych systemów kompresji obrazu i wideo, w tym standardów JPEG, MPEG i różnych wariantów systemów DVB. Wyniki pokazują, dlaczego ta metoda jest tak powszechnie stosowana w systemach transmisji cyfrowej - oferuje optymalny kompromis między efektywnością wykorzystania pasma a jakością obrazu.

Kod

W tym programie musimy wykonać kilka kroków, aby program działał prawidłowo, dokładnie te kilka kroków:

Krok 1: Konwersja obrazu RGB do przestrzeni kolorów YCbCr, gdzie Y reprezentuje jasność (luminancję), a Cb i Cr reprezentują informacje o kolorze (chrominancję). (Jest opisany w zadaniu 3)

Krok 2: Podpróbkowanie kanałów chrominancji (Cb i Cr) poprzez zachowanie tylko co drugiego piksela w obu wymiarach, co symuluje kompresję stosowaną w transmisji DVB.

Krok 3: Nadpróbkowanie kanałów chrominancji przy użyciu interpolacji najbliższego sąsiada, powielając każdy piksel, aby wypełnić blok 2x2 w oryginalnej rozdzielczości.

Krok 4: Rekonstrukcja obrazu RGB z kanału Y oraz nadpróbkowanych kanałów Cb i Cr przy użyciu odwrotnych równań transformacji kolorów.

Krok 5: Wyświetlenie oryginalnego i zrekonstruowanego obrazu wraz z poszczególnymi składowymi kolorów, aby zwizualizować efekty symulowanej transmisji.

Krok 2:

```
    # Step 2: Perform downsampling on Cb and Cr channels
    # Downsampling by factor of 2 in both dimensions (keeping every other pixel)
    # This simulates the chroma subsampling in DVB
    height, width = Y.shape
    Cb_downsampled = Cb[::2, ::2]
    Cr_downsampled = Cr[::2, ::2]
```

Ten fragment kodu wykonuje podpróbkowanie (downsampling) składowych chrominancji Cb i Cr. Najpierw pobierane są wymiary obrazu z kanału luminancji Y. Następnie, używając notacji slice z krokiem 2 ([::2, ::2]), kod wybiera co drugi piksel w pionie i poziomie z oryginalnych kanałów Cb i Cr. Ta operacja efektywnie zmniejsza rozdzielczość informacji o kolorze o połowę w każdym wymiarze, co jest standardową praktyką w kompresji obrazu używanej w cyfrowej transmisji wideo (DVB). Technika ta wykorzystuje fakt, że ludzkie oko jest bardziej wrażliwe na zmiany jasności niż koloru, co pozwala zaoszczędzić przepustowość bez znacznego pogorszenia postrzeganej jakości obrazu.

Krok 3:

```
    # Step 3: Perform upsampling on Cb and Cr channels
    # Simple nearest-neighbor upsampling
    Cb_upsampled = np.zeros_like(Cb)
    Cr_upsampled = np.zeros_like(Cr)

    # Replicate each downsampled pixel to a 2x2 block in the upsampled version
    for i in range(Cb_downsampled.shape[0]):
        for j in range(Cb_downsampled.shape[1]):
            i2 = i * 2
            j2 = j * 2

            # Handle edge cases for odd dimensions
            if i2 < height and j2 < width:
                Cb_upsampled[i2, j2] = Cb_downsampled[i, j]

                if j2 + 1 < width:
                    Cb_upsampled[i2, j2 + 1] = Cb_downsampled[i, j]

                if i2 + 1 < height:
                    Cb_upsampled[i2 + 1, j2] = Cb_downsampled[i, j]

                if i2 + 1 < height and j2 + 1 < width:
                    Cb_upsampled[i2 + 1, j2 + 1] = Cb_downsampled[i, j]

    # Same for Cr
```

Ten fragment kodu wykonuje nadpróbkowanie (upsampling) uprzednio zmniejszonych kanałów chrominancji. Używana jest metoda interpolacji najbliższego sąsiada, gdzie każdy piksel z wersji podpróbkowanej jest powielany do bloku 2x2 w wersji nadpróbkowanej. Kod tworzy najpierw puste tablice o oryginalnych wymiarach, a następnie w pętli przechodzi przez

każdy piksel podpróbkowanego obrazu, umieszczając jego wartość w odpowiednich czterech pozycjach w obrazie nadpróbkowanym. Dodatkowo kod obsługuje przypadki brzegowe, gdy wymiary obrazu są nieparzyste. Robi on to dla Cb i Cr.

Krok 4:

```
    # Step 4: Reconstruct RGB from Y and upsampled Cb, Cr
    R_reconstructed = Y + 1.402 * (Cr_upsampled - 128 / 255.0)
    G_reconstructed = Y - 0.344 * (Cb_upsampled - 128 / 255.0) - 0.714 * (Cr_upsampled - 128 / 255.0)
    B_reconstructed = Y + 1.772 * (Cb_upsampled - 128 / 255.0)

    # Clip values to ensure they're in the 0-1 range
    R_reconstructed = np.clip(R_reconstructed, a_min: 0.0, a_max: 1.0)
    G_reconstructed = np.clip(G_reconstructed, a_min: 0.0, a_max: 1.0)
    B_reconstructed = np.clip(B_reconstructed, a_min: 0.0, a_max: 1.0)
```

Ten fragment kodu wykonuje rekonstrukcję kanałów RGB z kanału luminancji Y oraz nadpróbkowanych kanałów chrominancji Cb i Cr. Stosowane są odwrotne równania transformacji kolorów, aby odzyskać wartości R, G i B. Po przekształceniu, wartości są przycinane funkcją np.clip do zakresu 0-1, aby zapewnić, że wynikowe kolory są w prawidłowym zakresie i nie wystąpią zniekształcenia przy wyświetlaniu obrazu.

Zadanie 5

Wstęp Teoretyczny

Zadanie 5.

Wyznaczyć różnicę między obrazkami z poprzedniego zadania (4) tzn.. obliczyć błąd średniokwadratowy (ang. *Mean Square Error*),

$$MSE = \frac{1}{m} \cdot \frac{1}{n} \cdot \sum_{i=1}^{3} \sum_{j=1}^{n} \left(X_{ij} - \hat{X}_{ij} \right)^2$$

gdzie:

n – liczba pikseli obrazu,

m – liczba kanałów (trzy w modelu RGB),

Xij – wartość *j*-tego koloru *i*-tego piksela w obrazie wejściowym,

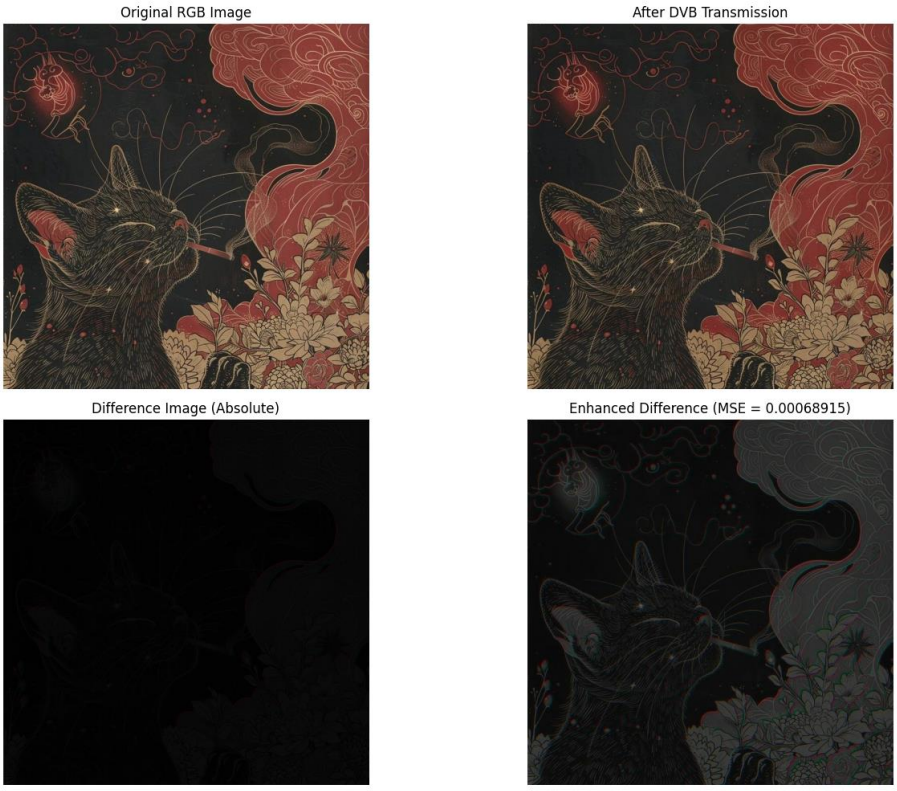
^*Xij* – wartość *j*-tego koloru *i*-tego piksela w obrazie wyjściowym;

Zadanie 5 koncentruje się na ilościowej ocenie jakości obrazu po przeprowadzeniu operacji przetwarzania z poprzedniego zadania. Kluczowym aspektem systemów transmisji i kompresji obrazów jest możliwość obiektywnej oceny stopnia degradacji wprowadzanej przez zastosowane algorytmy.

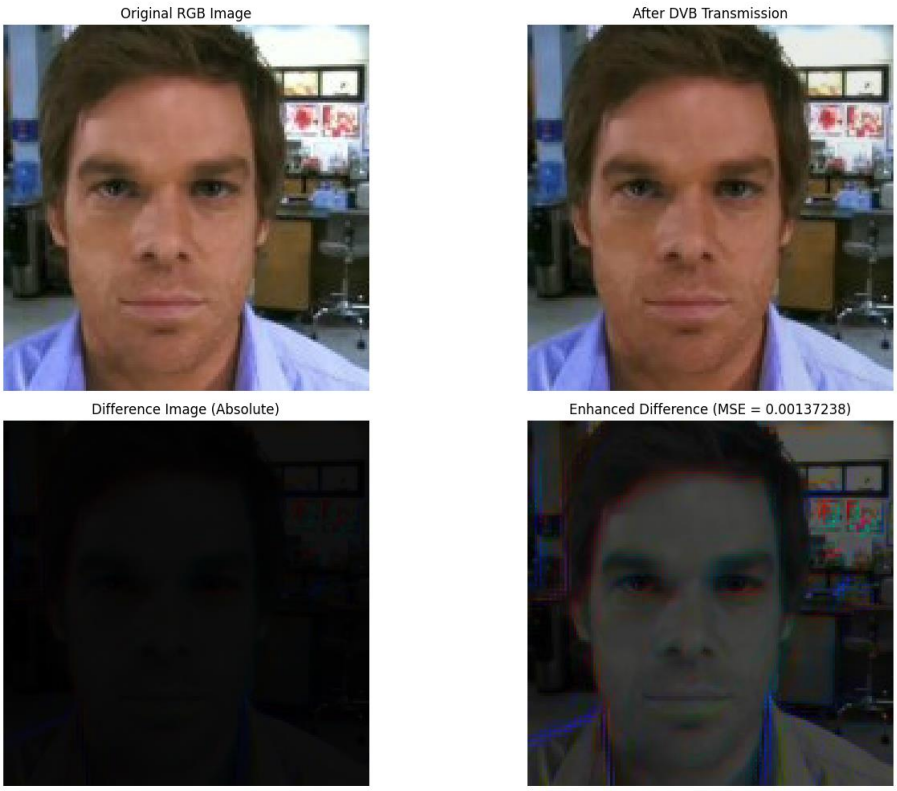
Błąd średniokwadratowy (ang. Mean Square Error, MSE) stanowi jedną z najpopularniejszych metryk stosowanych do ilościowej oceny podobieństwa między obrazami. Jest to miara określająca średnią kwadratów różnic między odpowiadającymi sobie pikselami w obrazie oryginalnym i przetworzonym.

W przypadku obrazów kolorowych, MSE oblicza się osobno dla każdego kanału kolorystycznego (R, G, B), a następnie uśrednia się wyniki. Wartość MSE bliska zeru wskazuje na wysokie podobieństwo między obrazami, podczas gdy większe wartości świadczą o znaczących różnicach.

Opis działania programu i analiza wyników



Na obrazie "Enhanced Difference" kota widoczne są delikatne artefakty kompresji. Wartość MSE wynosząca 0.00068915 jest wyjątkowo niska, co świadczy o minimalnych stratach. Obraz "Difference Image (Absolute)" jest niemal całkowicie czarny, potwierdzając że bezwzględne różnice między pikselami są bardzo małe, bliskie zeru. Natomiast obraz "Enhanced Difference" celowo wzmacnia te niewielkie różnice, uwidaczniając miejsca występowania artefaktów kompresji.



Obliczona wartość błędu średniokwadratowego (MSE) wynosi 0.00137238, co wskazuje na niewielkie różnice między obrazem oryginalnym a przetworzonym. Mimo zastosowania redukcji danych poprzez downsampling kanałów chrominancji, system DVB zapewnia wysoką wierność odwzorowania.

Na obrazie "Enhanced Difference" Dextera widoczne są wyraźne artefakty, szczególnie w obszarze koszuli, która ma zauważalny niebieski odcień. Jest to typowy efekt niedoskonałej rekonstrukcji informacji kolorystycznej podczas upsamplingu kanałów chrominancji. Obraz "Difference Image (Absolute)" pozostaje bardzo ciemny, ponieważ przedstawia bezwzględne różnice między pikselami obrazu oryginalnego i przetworzonego, które pomimo widocznych różnic kolorystycznych, są wartościowo niewielkie.

Analiza obu przypadków potwierdza założenia inżynierskie systemów DVB, które wykorzystują mniejszą wrażliwość ludzkiego oka na zmiany w chrominancji niż na zmiany w luminancji. Obrazy różnicowe (Absolute) są niemal całkowicie czarne, co wskazuje na wysoką skuteczność algorytmów kompresji, natomiast obrazy "Enhanced Difference" wzmacniają te różnice, aby uwidocznić artefakty kompresji, które w rzeczywistości są ledwo dostrzegalne. Mimo widocznych różnic na obrazach wzmacniających, w praktycznym zastosowaniu różnice te pozostają na akceptowalnym poziomie, co potwierdza efektywność kompromisu między stopniem kompresji a jakością obrazu w cyfrowych systemach transmisji.

Kod

Kod do zadania 5 jest bardzo podobny do zadania 4, bo są związane pomiędzy sobą i zadanie 5 jest rozszerzeniem zadania 4. Niżej będą opisane tylko nowe funkcje, które nie występowali w zadaniu 4

```
    # Step 5: Calculate the Mean Square Error (MSE)
    # m = number of channels (3 for RGB)
    # n = number of pixels (height * width)

    # Calculate the squared difference between original and reconstructed image for each channel
    diff_R = (R - R_reconstructed) ** 2
    diff_G = (G - G_reconstructed) ** 2
    diff_B = (B - B_reconstructed) ** 2

    # Calculate MSE according to the formula
    m = 3  # Number of channels (RGB)
    n = height * width  # Number of pixels

    mse = (1 / m) * (1 / n) * (np.sum(diff_R) + np.sum(diff_G) + np.sum(diff_B))
```

Ten fragment kodu oblicza błąd średniokwadratowy (MSE) między oryginalnym a zrekonstruowanym obrazem. Najpierw obliczane są kwadraty różnic między oryginalnymi a zrekonstruowanymi wartościami dla każdego kanału RGB, a następnie stosowana jest formuła MSE, gdzie m to liczba kanałów (3 dla RGB), a n to liczba pikseli (wysokość × szerokość). MSE jest ważnym parametrem ilościowym, który pozwala ocenić stopień degradacji obrazu po procesie kompresji.

Dodatkowo, kod tworzy wizualizacje różnic między oryginalnym a przetworzonym obrazem, zarówno w wersji podstawowej, jak i wzmocnionej (pomnożonej przez współczynnik), co pozwala na lepszą ocenę wizualną wpływu kompresji na jakość obrazu.

Podsumowanie

Laboratorium obejmowało szereg kluczowych technik przetwarzania obrazów cyfrowych i ich praktyczne zastosowania. Zadania koncentrowały się na filtrach górnoprzepustowych z różnymi maskami konwolucyjnymi, które znajdują zastosowanie w wykrywaniu krawędzi, uwydatnianiu szczegółów i analizie obrazów. Badania wykazały, jak zmiany w strukturze maski (symetria, asymetria, wartości sumy) wpływają na charakterystykę wyjściową obrazu. Transformacje przestrzeni barw, w tym konwersje między RGB a YCbCr, stanowią fundament nowoczesnych systemów kompresji obrazu i wideo, takich jak standardy JPEG czy MPEG.

Symulacja systemu transmisji DVB pozwoliła przeanalizować technikę podpróbkowania (downsamplingu) i nadpróbkowania (upsamplingu) kanałów chrominancji, powszechnie stosowaną w systemach telewizji cyfrowej i streamingu wideo. Ta metoda kompresji opiera się na fizjologicznych właściwościach ludzkiego wzroku, który wykazuje większą wrażliwość na zmiany jasności niż koloru. Analiza ilościowa jakości obrazu po kompresji za pomocą błędu średniokwadratowego (MSE) wykazała niskie wartości (0,00068915 i 0,00137238), co świadczy o wysokiej skuteczności zastosowanych algorytmów kompresji przy zachowaniu akceptowalnej jakości wizualnej. Eksperymenty uwidoczniły kompromisy między stopniem kompresji a jakością obrazu, niezbędne w projektowaniu efektywnych systemów przetwarzania i transmisji multimediów.