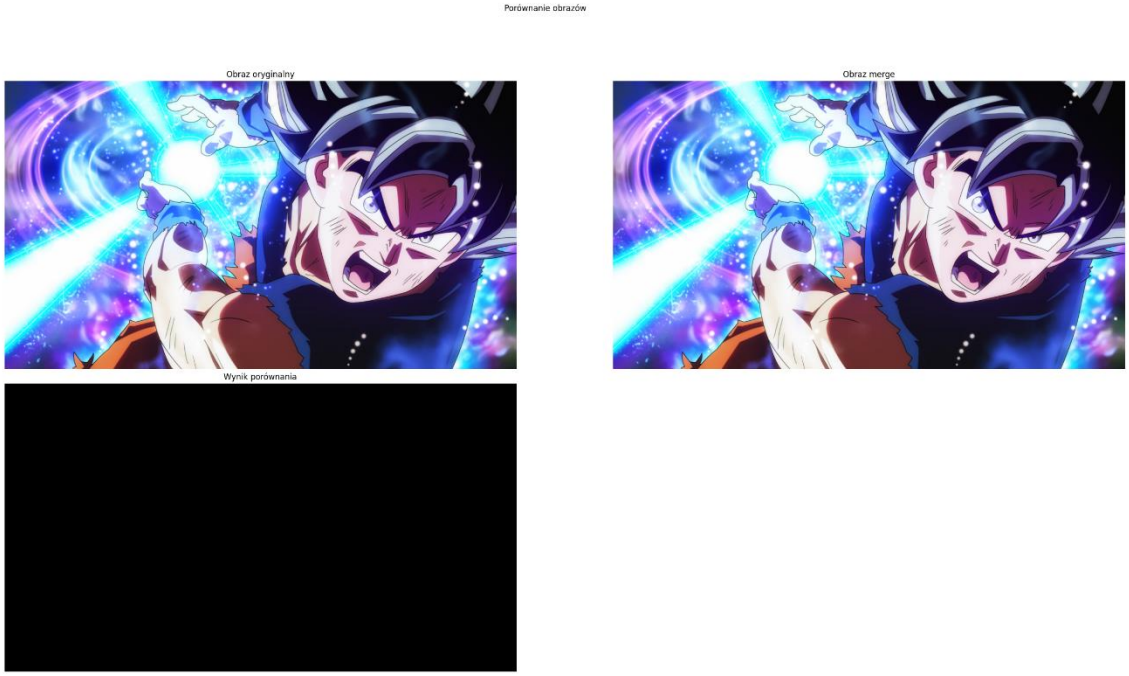
Z jakiegoś powodu metoda plt.show() generowała błąd. Problem rozwiązała dodanie dodatkowej linijki ręcznie określający backend.

```
plt.switch_backend('TkAgg')
```

Powyżej mamy oryginalny obraz wczytany jako im1, następnie pobrano każdy kanał osobno, po czym złączono ponownie tworząc im2. Następnie porównujemy obrazy za pomocą metody ImageChops.diffrences. Oba obrazy oraz efekt ich porównania umieszczamy w jednym oknie plt.

```
im1 = Image.open('obraz.jpg')

tablica_obrazu = np.array(im1)

t_r = tablica_obrazu[:, :, 0]
im_r = Image.fromarray(t_r)

t_g = tablica_obrazu[:, :, 1]
im_g = Image.fromarray(t_g)

t_b = tablica_obrazu[:, :, 2]
imi_b = Image.fromarray(t_b)

im2 = Image.merge('RGB', (im_r, im_g, imi_b))

diff1=ImageChops.difference(im1, im2)

plt.figure(figsize=(32,16))
plt.suptitle('Porównanie obrazów')

plt.subplot(2, 2, 1)
plt.title("Obraz oryginalny")
plt.imshow(im1)
plt.axis('off')

plt.subplot(2, 2, 2)
plt.title("Obraz merge")
plt.imshow(im2)
plt.axis('off')

plt.subplot(2, 2, 3)
plt.title("Wynik porównania")
plt.imshow(diff1)
plt.axis('off')

plt.subplots_adjust(wspace=0.05, hspace=0.05)
plt.savefig('fig1.png')
plt.show()
```

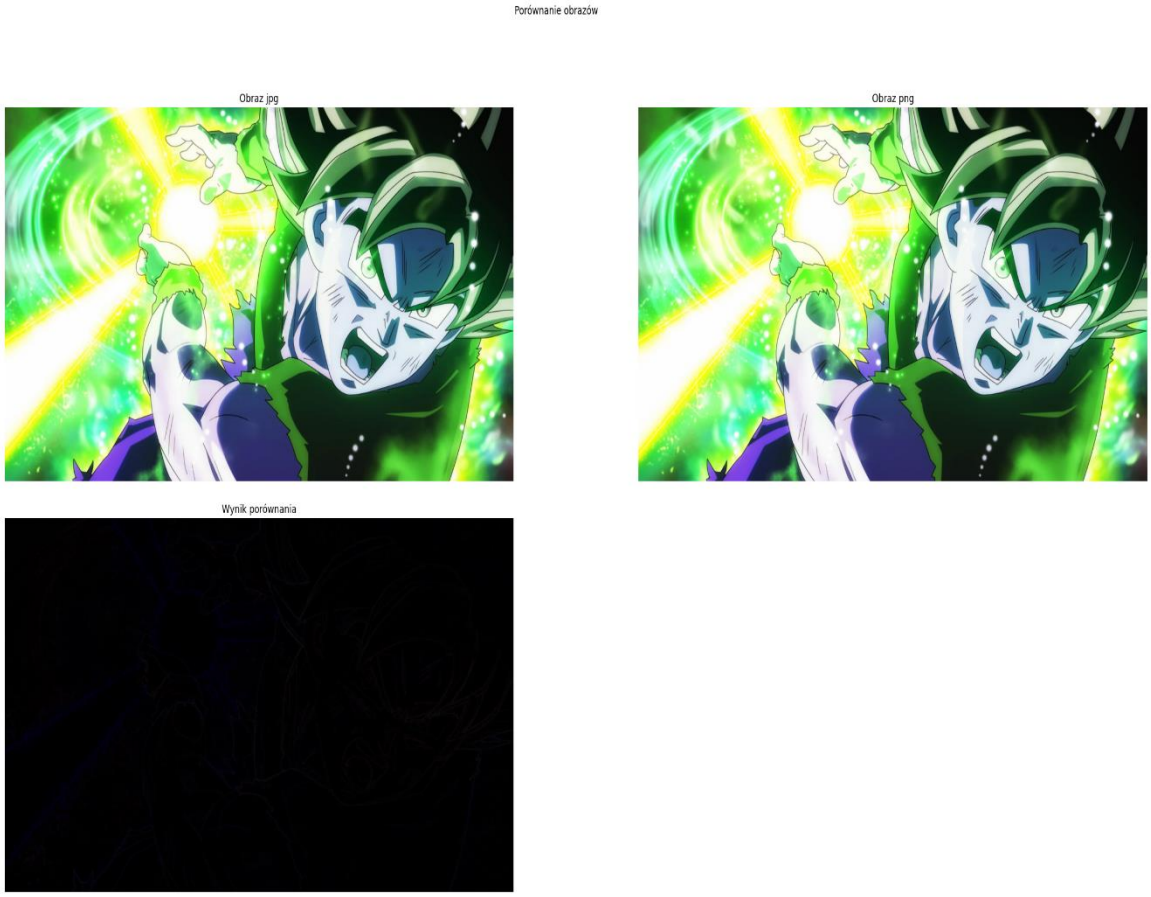



*Efekt porównania obu obrazków*

```
r, g, b = im1.split()
im3 = Image.merge('RGB', (g, b, r))
im3.save("im3.jpg")
im3.save("im3.png")
im3_jpg = Image.open("im3.jpg")
im3_png = Image.open("im3.png")
diff2 = ImageChops.difference(im3_jpg, im3_png)
diff2.show()

plt.figure(figsize=(32,16))
plt.suptitle('Porównanie obrazów')

plt.subplot(2, 2, 1)
plt.title("Obraz jpg")
plt.imshow(im3_jpg)
plt.axis('off')

plt.subplot(2, 2, 2)
plt.title("Obraz png")
plt.imshow(im3_png)
plt.axis('off')

plt.subplot(2, 2, 3)
plt.title("Wynik porównania")
plt.imshow(diff2)
plt.axis('off')

plt.subplots_adjust(wspace=0.1, hspace=0.1)
plt.savefig('fig2.png')
plt.show()
```

Pobieramy poszczególne kanały za pomocą metody split, następnie zamieniając kolejność kanałów tworzymy dwa obrazy i je zapisujemy, jeden w formacie jpg, drugi png. Następnie wczytujemy ponownie zapisane obrazy i przystępujemy do porównania. Efekty porównania umieszmy za pomocą plt na jednym oknie. Korzystając z metody show() widać wyraźnie, że oba obrazy się różnią. Z kolei na jednym oknie plt, aby takie różnice nie są widoczne na pierwszy rzut oka w tych przykładach.

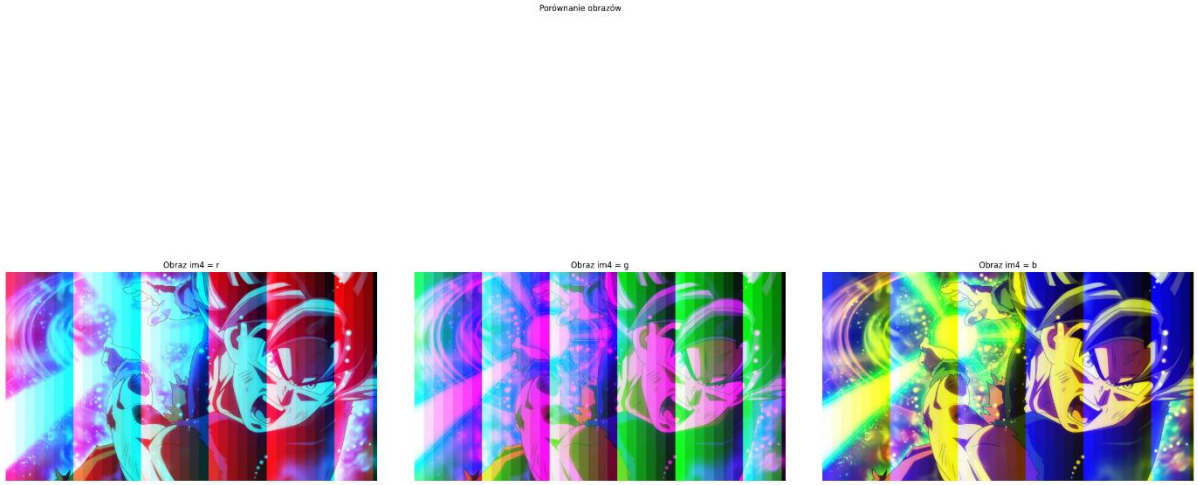
*Efekt porównania pokazany za pomocą metody show()*




*Obraz jpg, png i efekt ich porówniania*

```
im4 = rysuj_szare_paski(h=1080, w=1920, grubosc=50)

im4_r = Image.merge("RGB", (im4, g, b))
im4_g = Image.merge("RGB", (r, im4, b))
im4_b = Image.merge("RGB", (r, g, im4))

plt.figure(figsize=(32,16))
plt.suptitle('Porównanie obrazów')

plt.subplot(1, 3, 1)
plt.title("Obraz im4 = r")
plt.imshow(im4_r)
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title("Obraz im4 = g")
plt.imshow(im4_g)
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title("Obraz im4 = b")
plt.imshow(im4_b)
plt.axis('off')

plt.subplots_adjust(wspace=0.1, hspace=0.1)
plt.savefig('fig3.png')
```

Stworzono obraz w odcieniach szarości w formie pasków, kod wykorzystany z lab2. Następnie wynik funkcji podstawiono kolejno jako kanał r, g i b. Następnie wynik zapisano na jednej grafice za pomocą plt.

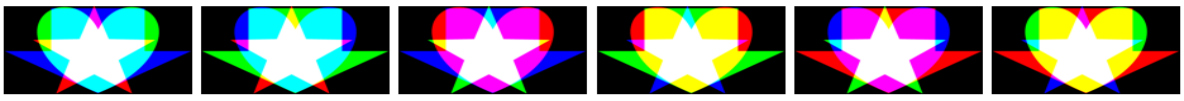



*Efekt uzyskany przez podstawienie im4 w kanały kolejno r, g i b*

```
gwiazda = Image.open("gwiazda.bmp").convert('L')
serce = Image.open("serce.bmp").convert('L')
strzalka = Image.open("strzalka.bmp").convert('L')

images = [gwiazda, serce, strzalka]
rgb_permutations = list(permutations(images, 3))

fig = plt.figure(figsize=(12, 8))
fig.suptitle('Permutacje obrazów')

for indeks, permutation in enumerate(rgb_permutations):
    r, g, b = permutation
    permuted_rgb = Image.merge("RGB", (r, g, b))

    ax = fig.add_subplot(1, 6, indeks + 1)
    ax.imshow(permuted_rgb)
    ax.axis('off')

plt.subplots_adjust(wspace=0.05, hspace=0.05)

plt.savefig('fig4.png')

plt.show()
```

Wczytujemy stworzone w paincie obrazy w kształtach gwiazdy, serca i strzałki. Białe kształty na czarnym tle. Następnie dostosowujemy je tak, aby każdy z nich był jednym kanałem rgb. Tworzymy z ich permutacji i umieszczamy w jednej grafice za pomocą plt. Do stworzenia permutacji korzystam z biblioteki itertools i metody permutation().

```
from itertools import permutations
```



```python
def pozyskaj_r(image):
    r = image[:, :, 0]
    return r

def pozyskaj_g(image):
    g = image[:, :, 1]
    return g

def pozyskaj_b(image):
    b = image[:, :, 2]
    return b

def porownaj(image1, image2):
    tab_image1 = np.array(image1)
    tab_image2 = np.array(image2)

    counter_r = 0
    counter_g = 0
    counter_b = 0

    if tab_image1.shape != tab_image2.shape:
        print("Tablice obrazów różnią się rozmiarem")
    else:
        for i in range(tab_image1.shape[0]):
            for j in range(tab_image2.shape[1]):
                if pozyskaj_r(tab_image1)[i][j] == pozyskaj_r(tab_image2)[i][j]:
                    continue
                else:
                    counter_r += 1
                if pozyskaj_g(tab_image1)[i][j] == pozyskaj_g(tab_image2)[i][j]:
                    continue
                else:
                    counter_g += 1
                if pozyskaj_b(tab_image1)[i][j] == pozyskaj_b(tab_image2)[i][j]:
                    continue
                else:
                    counter_b += 1

        print(f"Ilość różnic w r: {counter_r}")
        print(f"Ilość różnic w g: {counter_g}")
        print(f"Ilość różnic w b: {counter_b}")
```

Najbardziej dokładnym sposobem aby porównać oba obrazy jest rozdzielić je na poszczególne kanały r, g i b, a następnie porównać je piksel po pikselu. Nie jest to efektywna metoda, a kiedy chce się wszystkie różnice wypisać działanie programu czy zebrane dane mogą być ogromne. Przy rozdzielczości 1920x1080 każdy kanał ma 2 073 600 pikseli, a powyższy kod porównuje każdy piksel na 3 kanałach czyli dokonuje ponad 6 milionów porównań. Zamiast pritnować różnice, ograniczyłem się jedynie do ich wyliczenia ile jest różnych pikseli. Ponadto, zacząłem od porównania samej wielkości tych obrazów.