Wprowadzenie do Grafiki Maszynowej NS
Marcin Głód
164004
grupa II

# Laboratorium 6 – sprawozdanie

**Zadanie 1.**

Pozyskałem parametry filtra BLUR , oraz zastosowałem go do mojego obrazu za pomocą poniższego kodu:

```
 12 # print(ImageFilter.BLUR.filterargs)
 13 # ((5, 5), 16, 0, (1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0,
1, 1, 1, 1, 1, 1))
 14 # shrimp_kernel_blur = shrimp.filter(ImageFilter.Kernel(size=(5,5),
scale=16, offset=0, kernel=(1,     1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
0, 0, 0, 1, 1, 1, 1, 1, 1)))
 15 # shrimp_kernel_blur.show()
 16 # shrimp_kernel_blur.save("shrimp_k_bl.png")
 17 # shrimp_diff = ImageChops.difference(shrimp_blur, shrimp_kernel_blur)
```

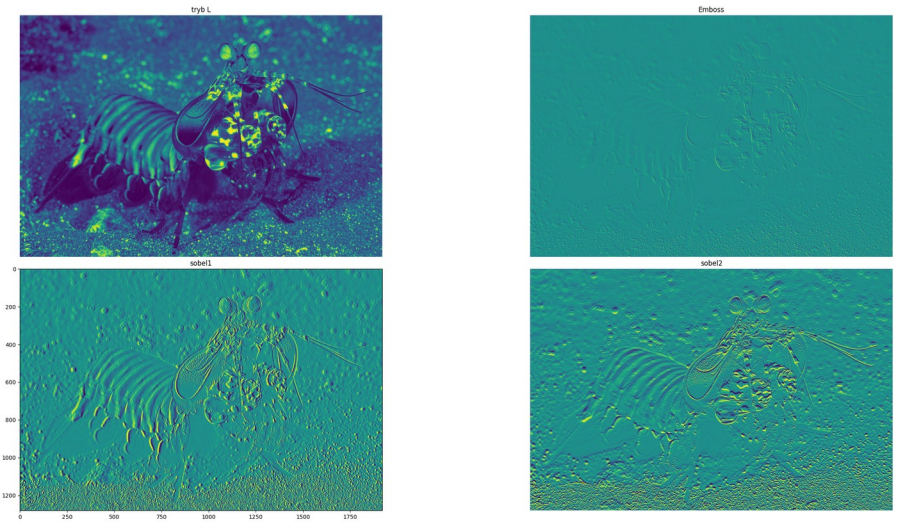
Oraz porównałem je ze sobą przy użyciu *ImageChops.difference()*
Uzyskałem w ten sposób następujące obrazy (złożone za pomocą MatPlotLib'a):


**Zadanie 2.**

Przy użyciu poniższego kodu utworzyłem obrazy powstałe po przepuszczeniu krewetki przez filtr Emboss ze zmienioną matrycą filtra:

```
 14 # shlimp = shrimp.convert('L')
 13 # shlimp_emboss = shlimp.filter(ImageFilter.EMBOSS)
 11 # print(ImageFilter.EMBOSS.filterargs)
 10 #
  9 # ImageFilter.EMBOSS.filterargs = ((3,3), 1, 128, (-1, 0, 1, -2, 0, 2, -1,
0, 1))
  8 # shlimp_sobel1 = shlimp.filter(ImageFilter.EMBOSS)
  5 # ImageFilter.EMBOSS.filterargs = ((3,3), 1, 128, (-1, -2, -1, 0, 0, 0, 1,
2, 1))
  4 # shlimp_sobel2 = shlimp.filter(ImageFilter.EMBOSS)
```

Uzyskałem w ten sposób następujące obrazy:




Matryce z których korzystamy w tym filtrze wyglądają następująco:

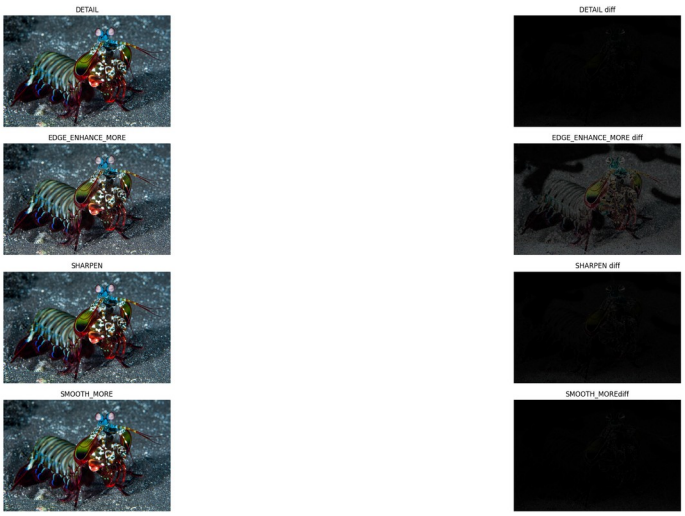
| Emboss | Sobel1 | Sobel2 |
|---|---|---|
| -1 0 0 | -1 0 1 | -1 -2 -1 |
| 0 1 0 | -2 0 2 | 0 0 0 |
| 0 0 0 | -1 0 1 | 1 2 1 |

Filtry Sobel1 i Sobel2 tworzą wyraźniejszy obraz, ponieważ posiadają większe różnice wartości w matrycy, co wyraźniej „odciska się” na wyjściowym obrazku.
Wizualnie metoda ta podkreśla krawędzie obrazka w taki sposób, że zaczyna przypominać płaskorzeźbę. Przestawienie wartości pomiędzy Sobel1, a Sobel2 wygląda, jakby na identyczną płaskorzeźbę światło padało z innego kąta.

**Zadanie 3.**
Przy użyciu poniższego kodu utworzyłem obrazy będące przefiltrowaną wersją wejściowego obrazka krewetki, a także utworzyłem pliki będące różnicą pomiędzy plikiem wejściowym, a przefiltrowanym obrazem.

```
14 # shrimp_detail = shrimp.filter(ImageFilter.DETAIL)
13 # detail_diff = ImageChops.difference(shrimp, shrimp_detail)
11 # shrimp_edge = shrimp.filter(ImageFilter.EDGE_ENHANCE_MORE)
10 # edge_diff = ImageChops.difference(shrimp, shrimp_edge)
 8 # shrimp_sharpen = shrimp.filter(ImageFilter.SHARPEN)
 7 # sharpen_diff = ImageChops.difference(shrimp, shrimp_sharpen)
 5 # shrimp_smooth = shrimp.filter(ImageFilter.SMOOTH_MORE)
 4 # smooth_diff = ImageChops.difference(shrimp, shrimp_smooth)
```

Wynik w postaci zestawienia wykresów z MatPlotLib’a:




**Zadanie 5.**
Przy użyciu poniższego kodu utworzyłem wyrównany obraz zamieszczony poniżej jako equalized.png:

```
# toof = Image.open("zeby.png")
# print(toof.mode)
# teef = toof.convert('L')
# teefqualized = ImageOps.equalize(teef, mask = None)
# teefqualized.save("equalized.png")
```

Obraz wejściowy:

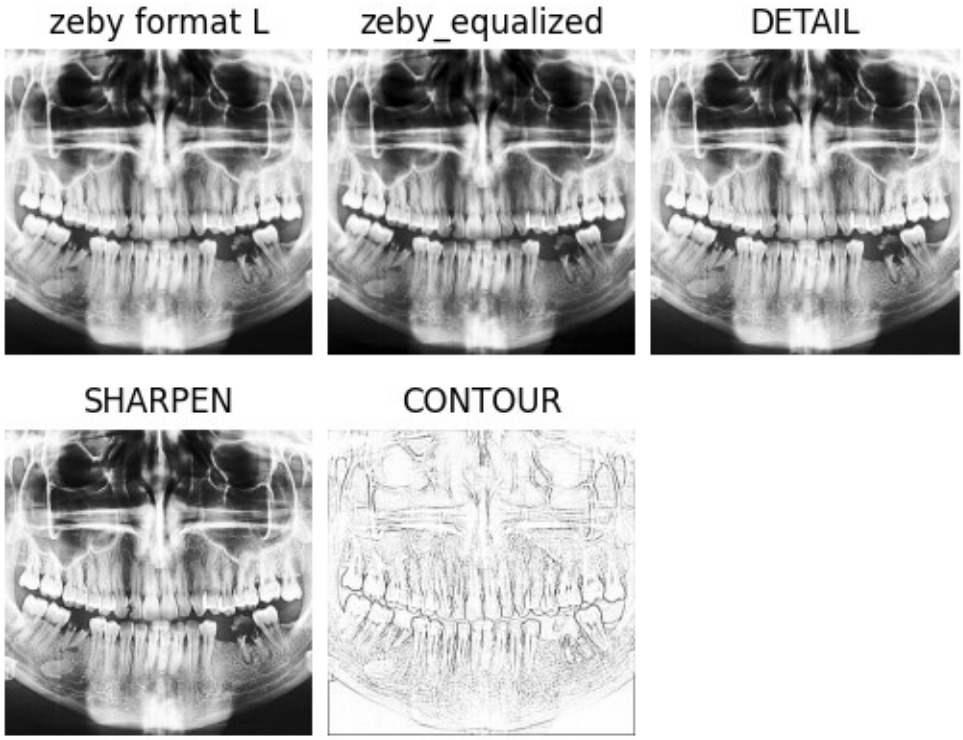
obraz po konwersji do trybu L i zastosowaniu wyrównania:

**Zadanie 6.**

Kod z powyższego zadania uzupełniłem o wywołanie następujących funkcji:

```
  8 # teef_detail = teef.filter(ImageFilter.DETAIL)
  7 # teef_sharpen = teef.filter(ImageFilter.SHARPEN)
  6 # teef_contour = teef.filter(ImageFilter.CONTOUR)
```

Uzyskałem w ten sposób następujące obrazy:




Według mnie, w kontekście przedstawionego obrazu wejściowego najlepiej zadziałał filtr *equalize*, ponieważ w największym stopniu uwidocznił słabo uwydatnione szczegóły tkwiące w jasnych szarościach.

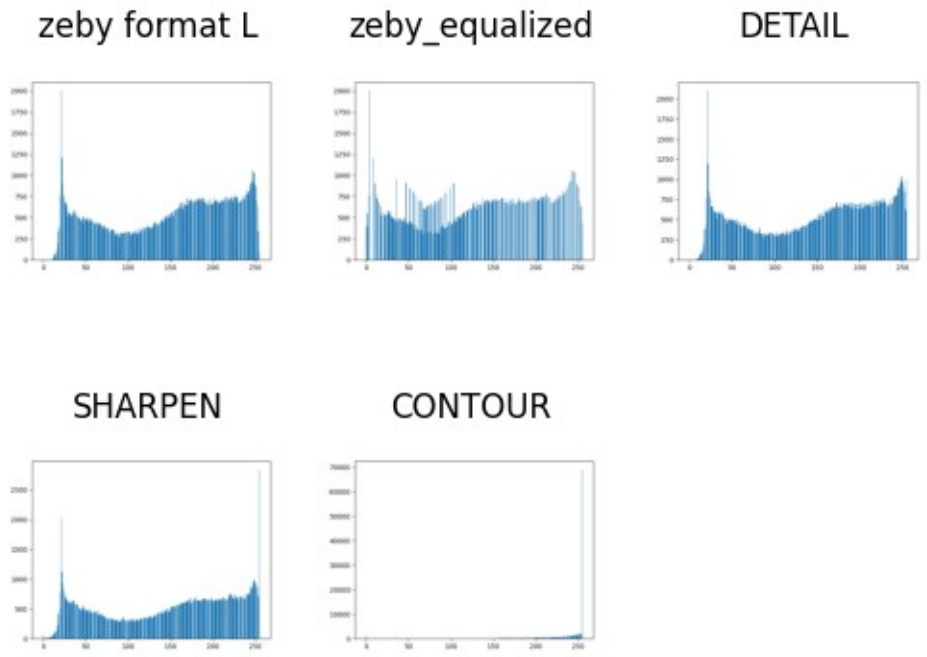
**b)**

przy użyciu poniższego kodu wygenerowałem histogramy obrazów z poprzedniego zadania:

```
10 # def get_histogram(im, name):
  9 #     plt.clf()
  8 #     hist = im.histogram()
  7 #     p = 0
  6 #     print(hist[p])
  5 #     plt.bar(range(256), hist[:])
  4 #     plt.savefig(f"{name}.png")
  2 # histo_teef = get_histogram(teef,"histo_teef")
  1 # histo_teefqualized = get_histogram(teefqualized,
"histo_teefqualized")
178 # histo_teef_detail =  get_histogram(teef_detail,
"histo_teef_detail")
  1 # histo_teef_sharpen = get_histogram(teef_sharpen,
"histo_teef_sharpen")
```

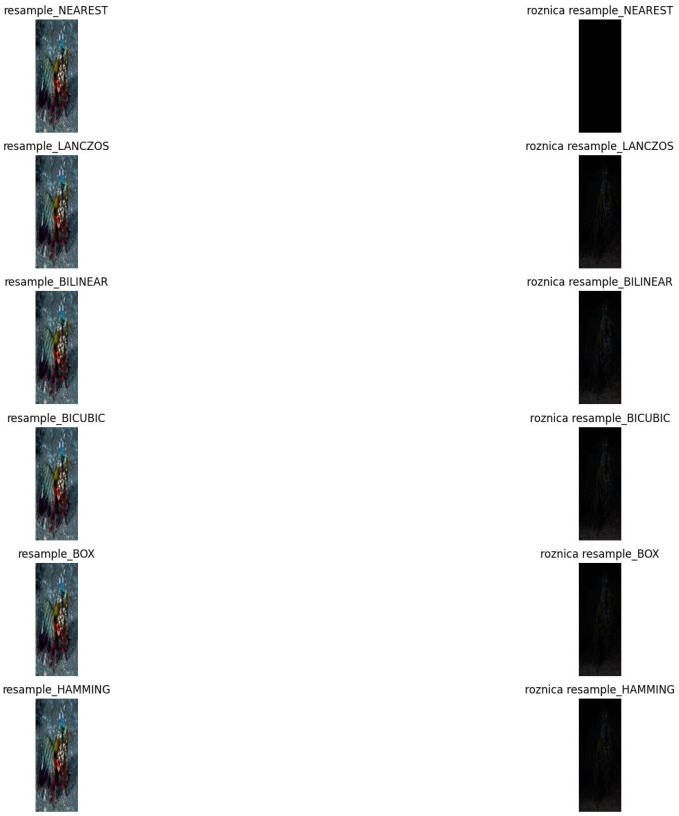
```
  2 # histo_teef_contour = get_histogram(teef_contour,
"histo_teef_contour")
```

Wygenerowane histogramy:




**zadanie 7.**

**Zadanie 8.**



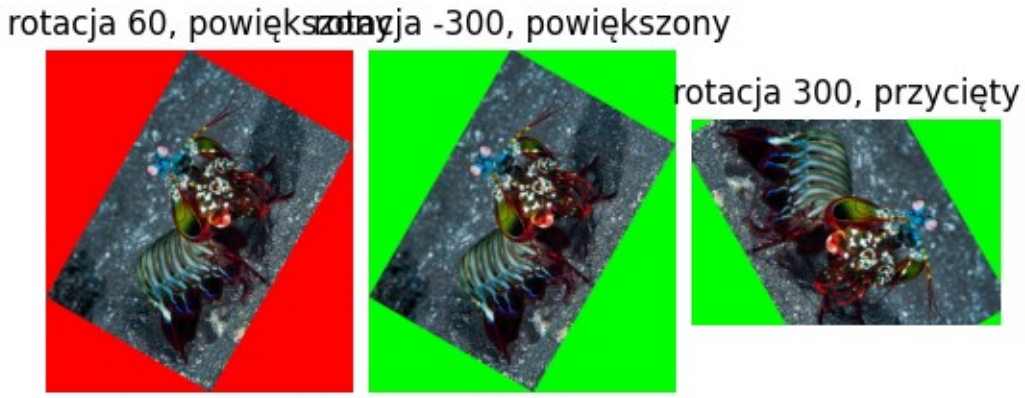

Za pomoca poniższego kodu pomniejszyłem jeden z obrazów z poprzedniego zadania, następnie przywróciłem do wejściowego rozmiaru:

```
244 # h,w = shrimp.size
  1 # plt.clf()
  2 # shrimp_R = shrimp.resize((int(w*s_w), int(h*s_h)), 3)
  3 # shrimp_B = shrimp_R.resize((1920,1280),3)
  8 # diff=ImageChops.difference(shrimp,shrimp_B)
  9 # # diff.show()
```

Uzyskałem w ten sposób:

Po przeskalowaniu wyraźnie widać, że została utracona spora częśc zawartych w obrazie danych i obraz po powiększeniu utracił znaczną część szczegółów.

**Zadanie 9.**




Powyższe obrazy zostały utworzone przy pomocy następującego kodu:

```
 21 rotato60 = shrimp.rotate(60, expand=1, fillcolor=(255,0,0))
 20 rotato_300 = shrimp.rotate(-300, expand=1, fillcolor=(0, 300, 0))
 19 rotato300 = shrimp.rotate(300, expand=0, fillcolor=(0, 300, 0))
 14 plt.subplot(1,3,1)
 13 plt.title("rotacja 60, powiększony")
 12 plt.imshow(rotato60)
 11 plt.axis('off')
 10 plt.subplot(1,3,2)
  9 plt.title("rotacja -300, powiększony")
  8 plt.imshow(rotato_300)
  7 plt.axis('off')
  6 plt.subplot(1,3,3)
  5 plt.title("rotacja 300, przycięty")
  4 plt.imshow(rotato300)
  3 plt.axis('off')
  2 plt.subplots_adjust(wspace=0.05, hspace=0.15)
  1 plt.savefig("fig7.png")
297 plt.show()
```

Obraz z rotacją w lewo o 60st wydaje się być identyczny do obrazu o rotacji w prawo o 300st ze względu na to, że suma ich rotacji jest pełnym obrotem.

**Zadanie 10.**

Efekt identyczny do metody TRANSPOSE() oraz TRANSVERSE() możemy uzyskać poprzez odpowiednie dobranie parametrów obrotu, oraz funkcję FLIP_LEFT_RIGHT(), które w sumie są w stanie wygenerować nam obraz w identycznym odiciu lustrzanym, co w wyniku kżycia ww. metod.

Pamiętać musimy jedynie, by odwrócić kierunek obrotu: zostanie on „poprawiony” w momencie nakładania funkcji FLIP_LEFT_RIGHT().