AIPO - Laboratorium 2

Marcin TORGiren Fabrykowski

10 marca 2013

1 Teoria

1.1 Filtr rozmywający

Filtr splotowy uśredniający (dolnoprzepustowy) - działanie tego filtru polega na ustawieniu wartości danego piksela na podstawie wartości pikseli znajdujących się w najbliższym otoczeniu tego piksela. Wynikiem jest uzyskanie rozmycia obrazu wejściowego.

1.2 Filtr wyostrzający

Filtr splotowy wyostrzający (górnoprzepustowy) - służy do wzmocnienia szczegółów o dużej częstotliwości. Wynikiem jest poprawa ostrości oraz kontrastu obrazu, jednak wzmacnia również szumy. Często stosowane po silnej filtracji uśredniającej, aby przywrócić ostrość obrazu.

1.3 Przetwarzany obraz

Obraz źródłowy znajduje się na rys. 1

2 Analiza obrazu

2.1 Skalowanie

Na rys 2 przedstawiono skalowanie w dół w skali 0.4, natomiast na rys 3 skalowanie w górę w skali 2.5.

2.2 Progowanie

Na rysunku 4 przedstawiono progowanie z użytym globalną wartością progowania. Na rysunkach 5, 6, 7, 8, 9 przedstawiono progowanie localne z otoczeniami odpowiednio: 5, 11, 15, 21, 25.

Na rysunkach 10, 11, 12 na których przedstawiono progowanie mieszane dla otoczenia 15 i odchyleniu średniej odpowiednio: 15, 25, 35.

2.3 Filtry splotowe

Na rysunkach 13, 14, 15 przedstawiono filtrowanie obrazu następującymi filtrami rozmywającymi:

1	1	1
1	1	1
1	1	1

,

1	1	1
1	2	1
1	1	1

,

1	2	1
2	4	2
1	2	1

Na rysunkach 16, 17, 18 przedstawiono filtrowanie obrazu następującymi filtrami wyostrzającymi:

0	-1	0
-1	5	-1
0	-1	0

,

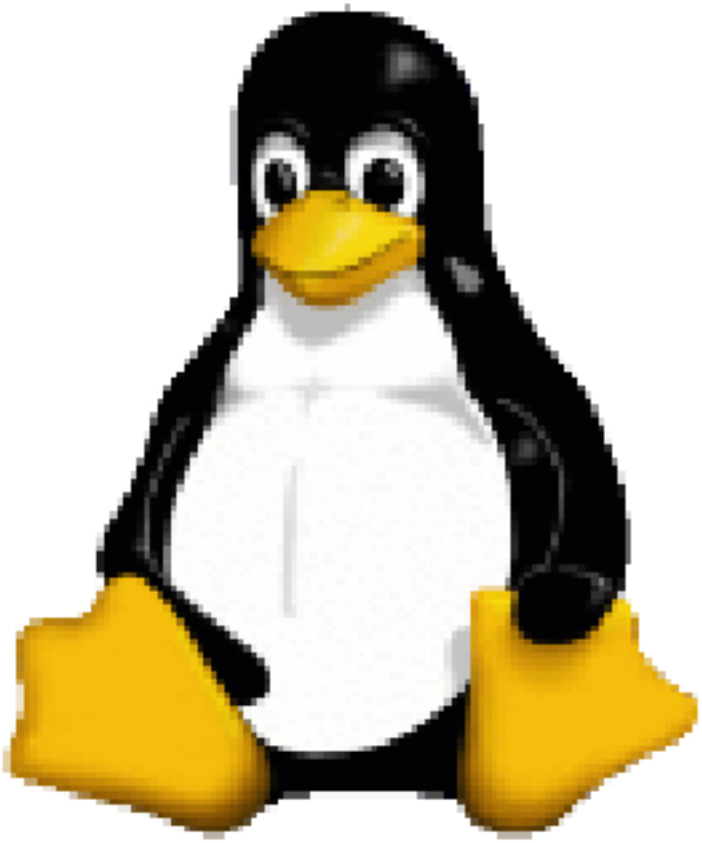
-1	-1	-1
-1	9	-1
-1	-1	-1

,

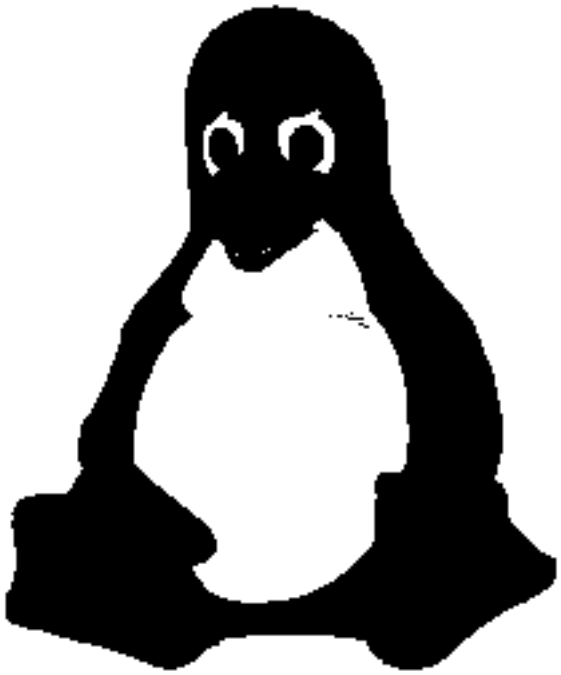
1	-2	1
-2	5	-2
1	-2	1

Rysunek 1: Obraz źródłowy

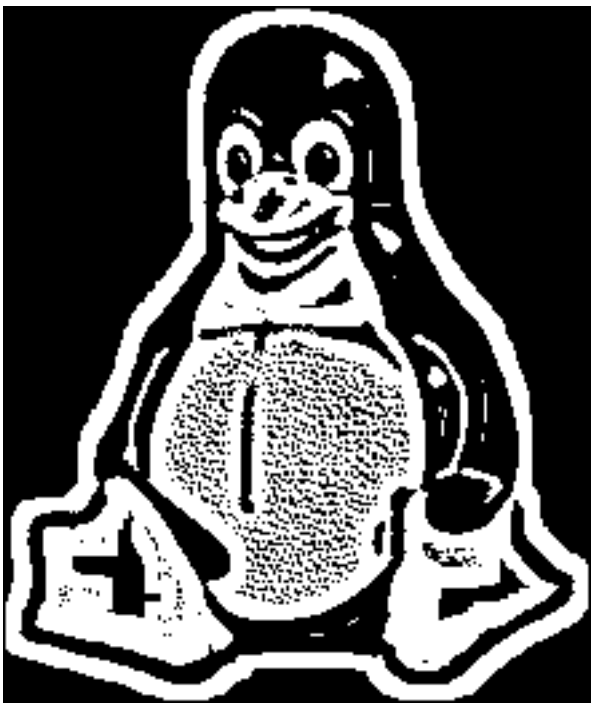
Rysunek 2: Skalowanie w dół - 0.4



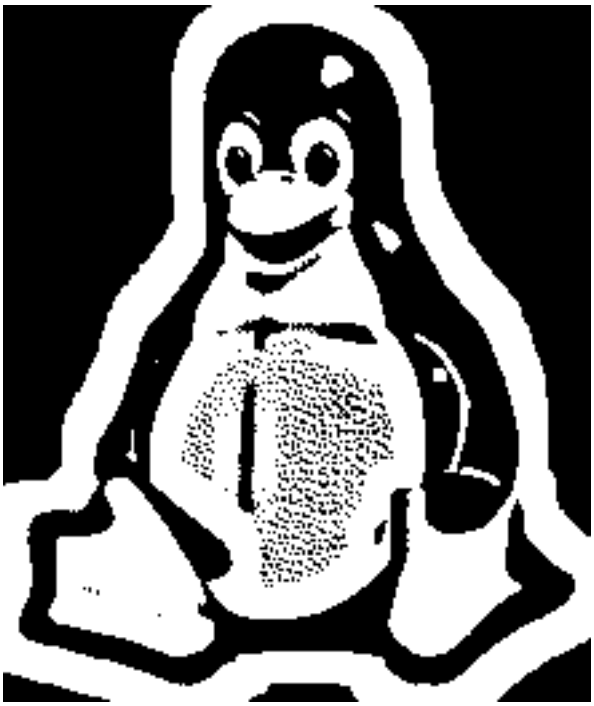
Rysunek 3: Skalowanie w górę - 2.5



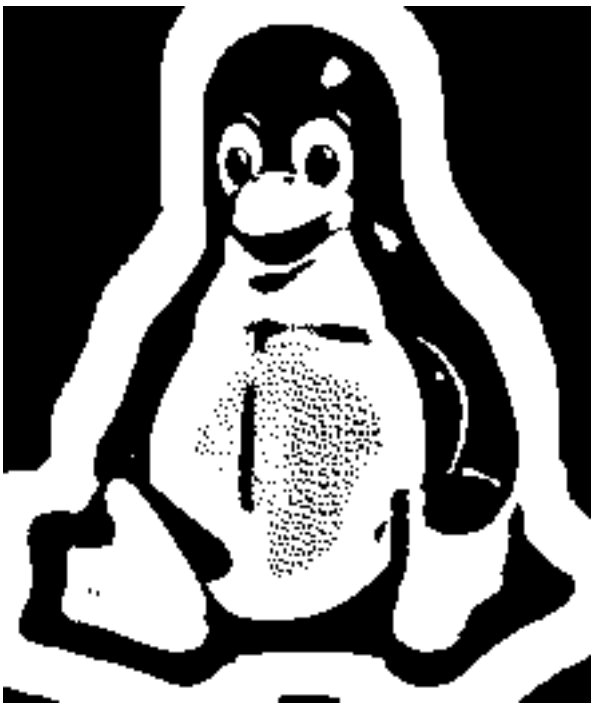
Rysunek 4: Progowanie globalne



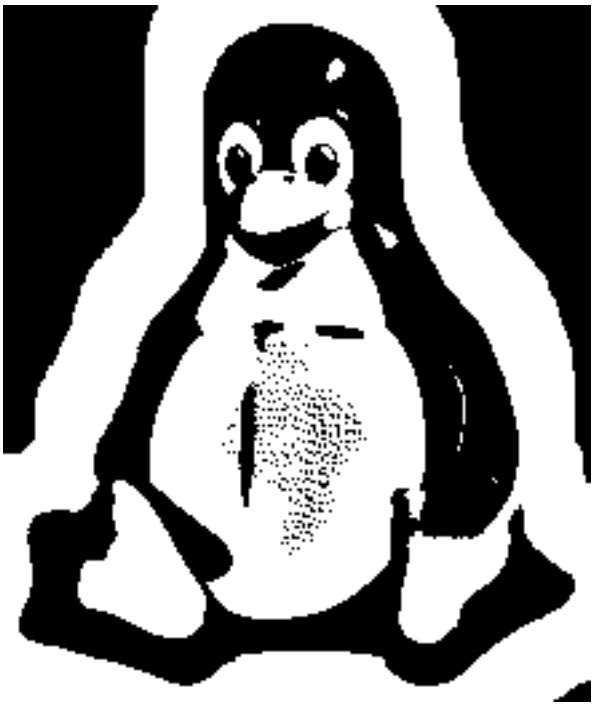
Rysunek 5: Progowanie lokalne, otoczenie 5



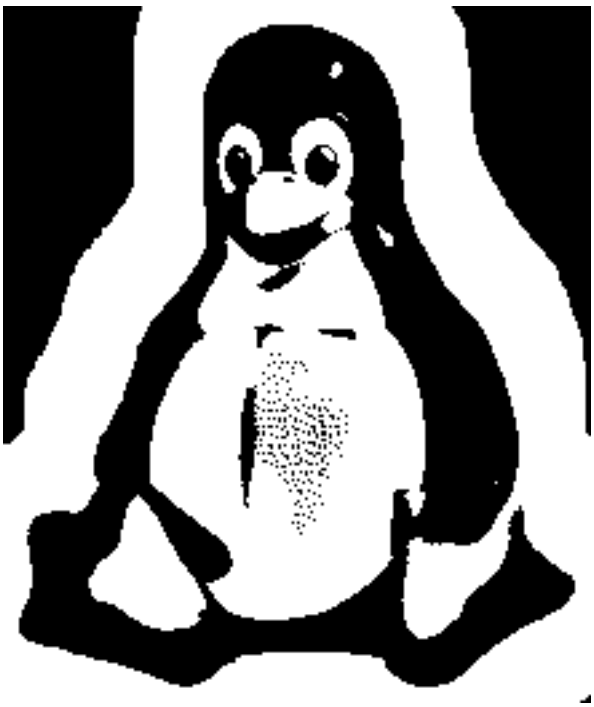
Rysunek 6: Progowanie lokalne, otoczenie 11



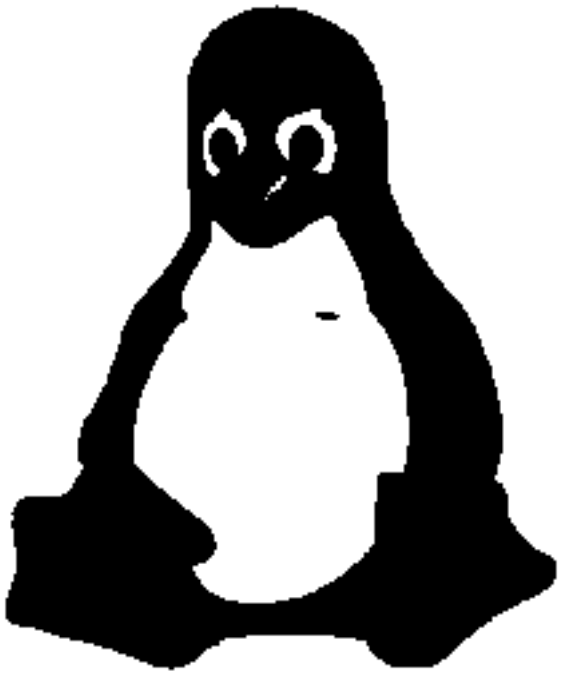
Rysunek 7: Progowanie lokalne, otoczenie 15



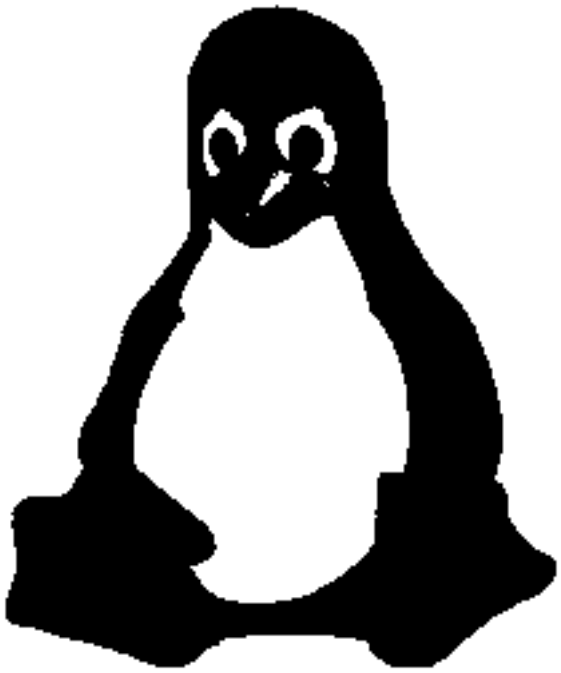
Rysunek 8: Progowanie lokalne, otoczenie 21



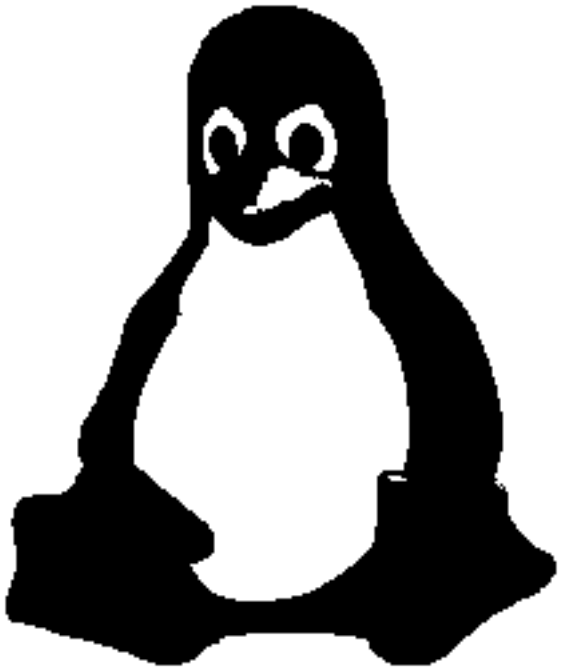
Rysunek 9: Progowanie lokalne, otoczenie 25



Rysunek 10: Progowanie mieszane, odchylenie 15



Rysunek 11: Progowanie mieszane, odchylenie 25

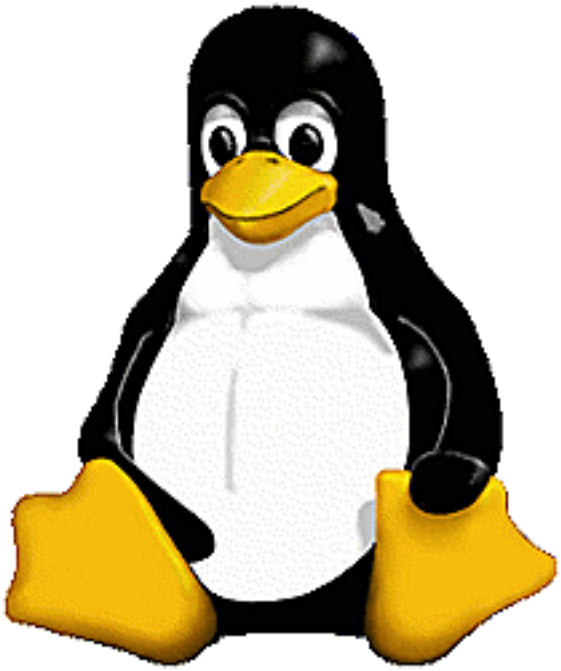


Rysunek 12: Progowanie mieszane, odchylenie 35

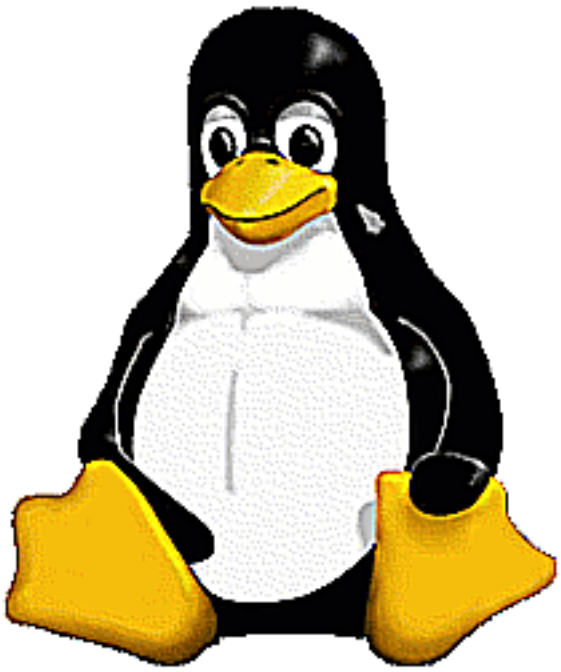
Rysunek 13: Filtrowanie rozmywające 1

Rysunek 14: Filtrowanie rozmywające 2

Rysunek 15: Filtrowanie rozmywające 3



Rysunek 16: Filtrowanie wyostrzające 1



Rysunek 17: Filtrowanie wyostrzające 2

3 Kod programu

```
#*-* coding: utf8 *-*
"""Modul zawierajacy klasy uzywane do przetwarzania obrazu"""
import numpy as np
from scipy import misc
import itertools


class NoImageError(Exception):
    """Wyjatek sygnalizujacy probe operowania na niewczytanym obrazie"""
    pass


class NoSuchMethodError(Exception):
    """Wyjatek sygnalizujacy podanie zlej metody operacji"""
    pass


class FilterSizeError(Exception):
    """Wyjatek sygnalizujacy bledny format filtru"""
    pass


class ImageAnal:
    """Klasa przetwarzajaca obrazy"""

    def image_loaded(fn):
        """dekorator.
        Sprawdza czy zostal zaladowany obraz"""
        def wrapped(self, *args, **kwargs):
            if self.__image is None:
                raise NoImageError()
            return fn(self, *args, **kwargs)
        return wrapped

    def __init__(self, path=None):
        """Konstruktor obiektu ImageAnal"""
        self.__image = None
        if path:
            self.load_image(path)

    def load_image(self, path):
        """Wczytuje obraz z pliku <path>"""
        self.__image = misc.imread(path)

    @image_loaded
    def negative(self):
        """Tworzy negatyw obrazu"""
```

```
        self.__image = 255 - self.__image

    @image_loaded
    def grayscale(self, method=1):
        """Konwertuje do odcieni szarosci.
        method:
        1 (default) wykorzystuje metode wartosci sredniej kolorow
        2 wykorzystuje wzor 0.3*R+0.59*G+0.11*B
        Obsluga tylko formatu RGB"""
        if method == 1:
            self.__grayscale1()
        elif method == 2:
            self.__grayscale2()
        else:
            raise NoSuchMethodError()

#     @image_loaded
#     def convert(self, fmt):
#         self.__image = self.__image.convert(fmt)
#         """Konwertuje obraz do zadanego formatu"""

    @image_loaded
    def normalize(self):
        data = self.__image
        R = data[:, 0]
        G = data[:, 1]
        B = data[:, 2]
        R = (R - R.min()) * 255 / R.max()
        G = (G - G.min()) * 255 / G.max()
        B = (B - B.min()) * 255 / B.max()

        data[:, 0] = R
        data[:, 1] = G
        data[:, 2] = B

        self.__image = data

    @image_loaded
    def scale(self, factor):
        if factor < 1:
            self.__scale_down(factor)
        else:
            self.__scale_up(factor)

    @image_loaded
    def progowanie(self, method="global", otoczenie=5, odchylenie=15):
        """Przeprowadza progowanie obrazka.
        metody:
        global - progowanie globalne
```

```
            local - progowanie lokalne
            mixed - progowanie mieszane

            parametry:
            otoczenie = rozmiar otoczenia pixela
            odchylenie - stopien ochylenia od sredniej"""
            self.__grayscale1()
            if method == "global":
                self.__progowanie_globalne()
            elif method == "local":
                self.__progowanie_lokalne(otoczenie=otoczenie)
            elif method == "mixed":
                self.__progowanie_mieszane(otoczenie=otoczenie, odchylenie=odchylenie)
        @image_loaded
        def splot(self, filter):
            filter = np.array(filter, dtype=np.int8)
            if filter.shape != (3,3):
                raise(FilterSizeError)
            data = self.__image
            new = self.__expand(data, 1)
            new = np.array(new, dtype=np.int32)
#           new = np.array(new, dtype=np.uint8)
#           print (filter[0,0] * new[:-2,:-2])[160,130]
#           print (filter[0,1] * new[:-2,1:-1])[160,130]
#           print (filter[0,2] * new[:-2,2:])[160,130]
#           print (filter[1,0] * new[1:-1,:-2])[160,130]
#           print (filter[1,1] * new[1:-1,1:-1])[160,130]
#           print (filter[1,2] * new[1:-1,2:])[160,130]
#           print (filter[2,0] * new[2:,:-2])[160,130]
#           print (filter[2,1] * new[2:,1:-1])[160,130]
#           print (filter[2,2] * new[2:,2:])[160,130]
            new = (filter[0,0] * new[:-2,:-2] + filter[0,1] * new[:-2,1:-1] +\
                   filter[0,2] * new[:-2,2:] + filter[1,0] * new[1:-1,:-2] +\
                   filter[1,1] * new[1:-1,1:-1] + filter[1,2] * new[1:-1,2:] +\
                   filter[2,0] * new[2:,:-2] + filter[2,1] * new[2:,1:-1] +\
                   filter[2,2] * new[2:,2:])
            new = new / (filter.sum())
            new -= 255
            new = new * (new<0)
            new += 255
            new = new * (new>0)
            data = np.array(new, dtype=np.uint8)
            self.__image = data
#           self.normalize()


        @image_loaded
        def save(self, path):
            """Zapisuje obraz do pliku"""
```

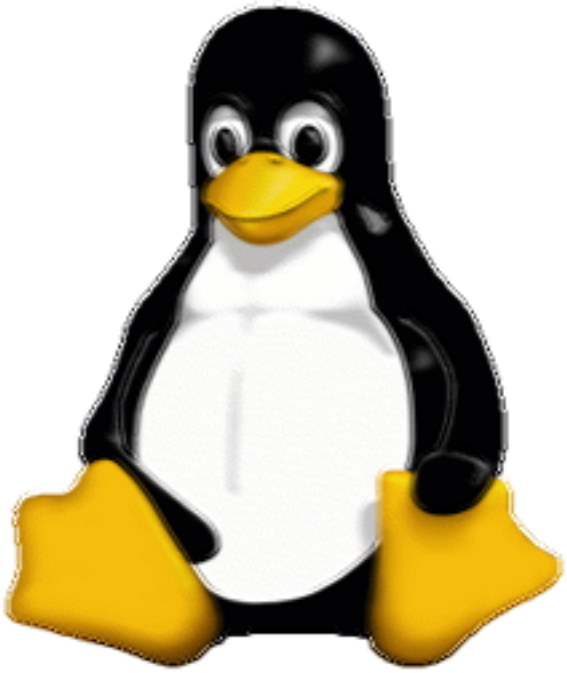
```
        self.__clear_alpha()
        misc.imsave(path, self.__image)

    def __grayscale1(self):
        """Konwersja do skali szarosci"""
        data = self.__image
#         data[:,:] = 3 * (data[:,:].mean())
#         x = [4 * (int(x.mean()),) for x in data]
        size = data.shape
        new = np.array(data, dtype=np.uint32)
        new[:,:,0] += data[:,:,1]
        new[:,:,0] += data[:,:,2]
        new[:,:,0] /=3
        data[:,:,1] = data[:,:,2] = data[:,:,0] = new[:,:,0]

        self.__image = data

    def __scale_down(self, factor):
        factor = (int)(factor ** (-1))
        data = self.__image
        data = np.array(data[::factor, ::factor, :])
        self.__image = data

    def __scale_up(self, factor):
        data = self.__image
        new = np.zeros((data.shape[0] * factor, data.shape[1] * factor, data.shape
        for x in xrange(data.shape[0]):
            for y in xrange(data.shape[1]):
                new[x * factor:(x + 1) * factor, y * factor:(y + 1) * factor, :] =
        self.__image = new

    def __progowanie_globalne(self, *args, **kwargs):
        data = self.__image
        mean = self.__prog_globalny()
#         mean = data[:, :, 0].mean()
        data = (data > mean) * 255.
        self.__image = data

    def __progowanie_lokalne(self, otoczenie=5, *argx, **kwargs):
        data = self.__image
        prog = self.__prog_lokalny( otoczenie)
        data = (data > prog) * 255
        self.__image = data

    def __progowanie_mieszane(self, otoczenie, odchylenie):
        data = self.__image
        prog = self.__prog_mieszany(otoczenie, odchylenie)
        data = (data > prog) * 255
        self.__image = data
```

```
    def __prog_globalny(self):
        data = self.__image
        return data[:,:,0].mean()

    def __prog_lokalny(self,otoczenie):
        data = self.__image
        new = self.__expand(data, otoczenie)
        prog = np.zeros(data.shape)
#         for x in xrange(otoczenie, new.shape[0] - otoczenie):
#             for y in xrange(otoczenie, new.shape[1] - otoczenie):
#                 prog[x - otoczenie, y - otoczenie] = new[x - otoczenie: x + otocze
        for d in itertools.product(np.arange(0,2*otoczenie+1), repeat=2):
            prog[:,:] += new[d[0]:new.shape[0]-2*otoczenie+d[0],
                             d[1]:new.shape[1]-2*otoczenie+d[1]]

        prog /= (2*otoczenie+1)**2
#         print prog
        return prog

    def __prog_mieszany(self, otoczenie, odchylenie):
        glob = self.__prog_globalny()
        prog = self.__prog_lokalny(otoczenie)
        prog -= (glob + odchylenie)
        prog = prog * (prog>0)
        prog -= 2 * odchylenie
        prog = prog * (prog<0)
        prog += (glob + odchylenie)
        return prog

    def __expand(self, src, otoczenie):
        data = src.copy()
        left = data[:, 0, :]
        right = data[:, -1, :]
        for i in xrange(otoczenie - 1):
            left = np.column_stack((left, data[:, 0, :]))
            right = np.column_stack((right, data[:, -1, :]))
        left = left.reshape((data.shape[0], -1, data.shape[2]))
        right = right.reshape((data.shape[0], -1, data.shape[2]))
        data = np.column_stack((left, data, right))
        top = data[0, :, :]
        bottom = data[-1, :, :]
        for i in xrange(otoczenie - 1):
            top = np.column_stack((top, data[0, :, :]))
            bottom = np.column_stack((bottom, data[-1, :, :]))
        top = top.reshape((-1, data.shape[1], data.shape[2]))
        bottom = bottom.reshape((-1, data.shape[1], data.shape[2]))
        data = np.vstack((top, data, bottom))
        return data
```

```
def __clear_alpha(self):
    print "clear alpha"
    if self.__image.shape[2] == 4:
        self.__image[:, :, 3] = 255
```

4 Wnioski

Jakoś skalowanie w górę można poprawić poprzez obliczanie wartości średnich dla nowych pixeli zamiast kopiować jeden pixel wielokrotnie.
Najlepszy wynik progowowanie dała metoda mieszana z odchyleniem 35. Usuwa ona wszystkie niepotrzebne detale i pozostawia najważniejsze elementy obrazu, tj. kształt pingwina, a usuwa drobne przebłyski na smokingu i stopach.
Wszystkie 3 filtry rozmywające dają podobny efekt wizualny, oraz nie różnią się zbytnio złożonością obliczeniową, dlatego uważam że wszystkie one są równorzędne.
Moim zdaniem filtr wyostrzający 2 daje najlepsze wyniki. Można to zauważyć np. przy uwydatnieniu drobnego cienia świetlnego na rękach który mógłby zostać niezauważony na oryginalnym obrazku. Podobnie jest z prawą brwią pingwina - praktycznie niezauważalna na oryginalnym obrazku zostaje wyraźnie przedstawiona przy filtrze nr. 2



Rysunek 18: Filtrowanie wyostrzające 3