

---

# **Przetwarzanie Cyfrowe Obrazów**

rozpoznawanie logo sieci sklepów Kaufland

Domański Piotr 293102

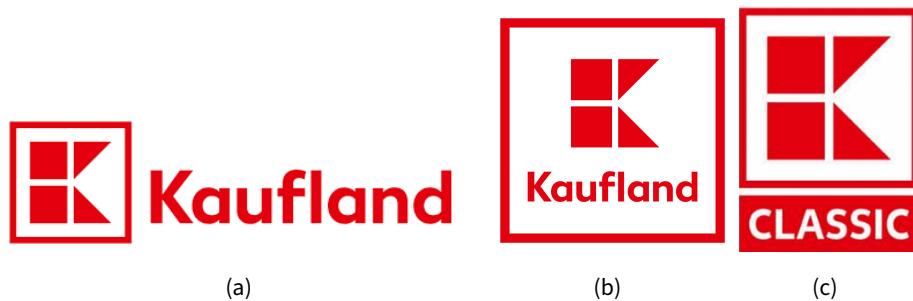
2022-01-13

## Zadanie realizowane w ramach projektu

W ramach projektu realizowana była implementacja procedur wstępnego przetwarzania, segmentacji, wyznaczania cech oraz identyfikacji obiektów na obrazach cyfrowych. Powstały w ramach projektu program powinien poprawnie rozpoznawać wybrane obiekty dla reprezentatywnego zestawu obrazów wejściowych. W trakcie projektu należało przetestować wybrane algorytmy i ocenić ich praktyczną przydatność.

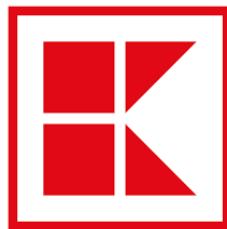
### Rozpoznawany obiekt - logo Kaufland

Logo sieci sklepów Kaufland można znaleźć nie tylko w samym sklepie, ale również na produktach produkowanych w ramach ich marki: Kaufland Classic. Warto zaznaczyć, że logo umieszczane na szyldach sklepowych różni się od loga umieszczonego na produktach. Porównanie różnych wersji widoczne jest na rysunku 1.



**Rysunek 1:** Różne wersje logo Kaufland

Ze względu na różne wersje logo Kaufland, konieczne jest rozpoznawanie jedynie ich części wspólnej przedstawionej na rysunku 2.



**Rysunek 2:** Część wspólna różnych wersji logo Kaufland

Część wspólna różnych wersji logo Kaufland jest kształtu kwadratu i składa się z:

1. czerwonej ramki
2. dwóch czerwonych kwadratów wewnętrz ramki
3. dwóch czerwonych trójkątów wewnętrz ramki, będących wzajemnym odbiciem w pionie

Warto zaznaczyć, że powierzchnia pomiędzy wymienionymi elementami nie jest biała, a transparentna i na niektórych produktach marki ma inny kolor niż biały. Dodatkowo, wewnętrz ramki może znajdować się tekst "Kaufland". Ze względu na te rozbieżności, powierzchnia ta nie jest rozpoznawana w ramach projektu.

## Akwizycja obrazu

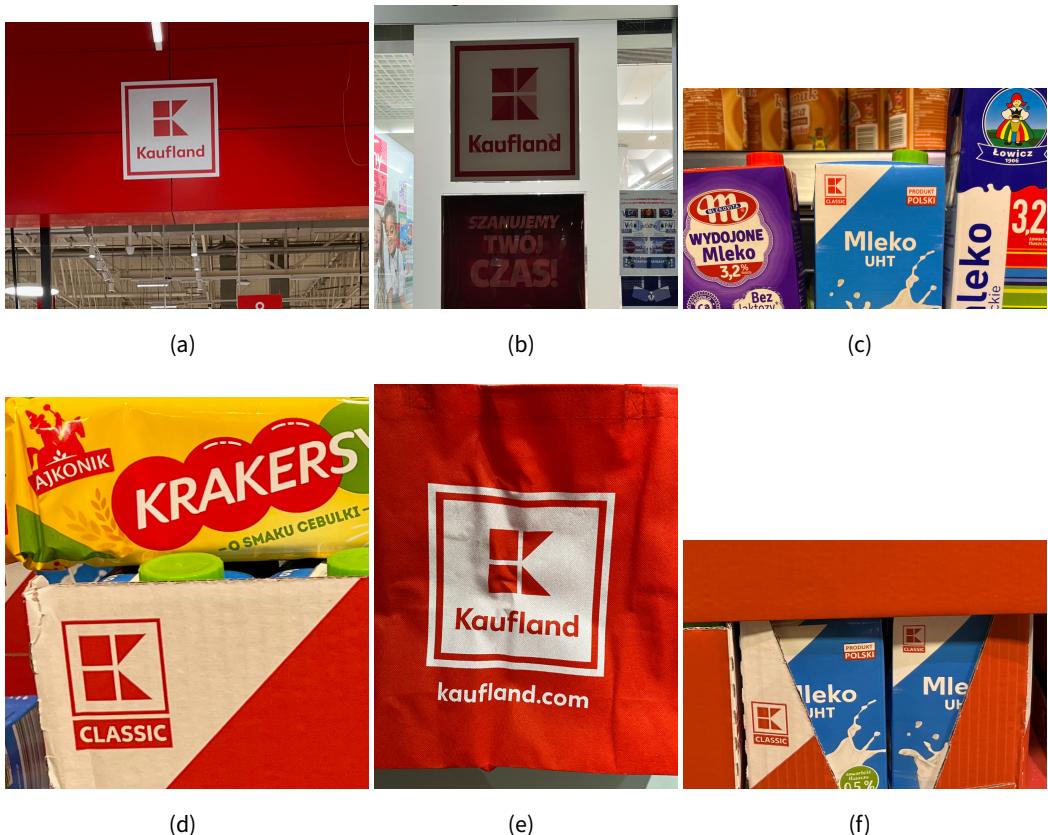
Logo Kaufland zostało przeze mnie wybrane jako obiekt rozpoznawany w ramach projektu, ponieważ właśnie w tej sieci sklepów najczęściej wykonuję zakupy, dzięki czemu wszystkie zdjęcia analizowanego zbioru danych zostały wykonane przeze mnie osobiście. Zbiór danych składa się z 6 zdjęć. Zdjęcia zostały wykonane w różnych warunkach oświetleniowych przy pomocy telefonu komórkowego, natomiast na żadnym z nich nie ma wyraźnych zakłóceń, są ogólnie dobrej jakości. Zbiór danych został przedstawiony na rysunku 3.

Pierwsze dwa zdjęcia zbioru danych przedstawiają logo Kaufland na szyldach sklepowych, pierwsze stanowi przypadek prosty natomiast drugie ma stanowić przypadek trudny do analizy, ponieważ szyld znajduje się na szklanych drzwiach i jest zakluczyony przez logo umieszczone po drugiej ich stronie.

Kolejne dwa zdjęcia przedstawiają logo znajdujące się na produktach marki Kaufland Classic z widocznymi produktami innych marek i ich logami. Specjalnie umieszczono na nich produkty innych firm z czerwonymi logami.

Piąte zdjęcie przedstawia logo sklepu Kaufland na materiałowej reklamówce.

Ostatnie zdjęcie przedstawia dwa produkty Kaufland Classic obok siebie, co ma za zadanie zweryfikować możliwość rozpoznania wielu log na jednym zdjęciu.



**Rysunek 3:** Zbiór testowy

## Algorytm wykrywania logo Kaufland

Algorytm do wykrywania logo Kaufland można podzielić na poniższe fazy:

1. wstępne przetwarzanie, zmiana rozmiaru i poprawa jakości obrazu
2. zmiana przestrzeni barw na HSV - rozdzielenie informacji o kolorze, nasyceniu i jasności
3. segmentacji - wydzielenie obszarów
4. wyznaczenie cech dla każdego obszaru
5. identyfikacja kształtów na podstawie ich opisów
6. rozpoznanie logo jako złożenie wielu kształtów

## Użyte narzędzia

Projekt zaimplementowano w języku programowania Python z wykorzystaniem pakietu OpenCV jedynie w ramach dozwolonych funkcji: wczytania i wyświetlenia obrazu oraz rysowania prostokąta do

wyrysowania bounding box.

Ponieważ język Python jest wolniejszy od C++, implementowane algorytmy były silnie optymalizowane w celu maksymalizacji wykorzystania biblioteki do obliczeń numerycznych Numpy.

Dodatkowo, przy prezentacji wyników pośrednich t.j. segmentacji wykorzystano bibliotekę Matplotlib. Pozwala ona na interakcję z wyświetlonym obrazem, np. powiększaniem obszarów co znacznie ułatwiło analizę na etapie implementacji.

Wykorzystano najnowszą dostępną wersję języka Python - 3.11.

## Wstępne przetwarzanie

Wstępne przetwarzanie ma na celu poprawę sprawności działania algorytmu poprzez poprawę jakości obrazu oraz zmniejszenie jego rozmiarów.

### Zmiana rozmiaru

Pierwszym etapem wstępnego przetwarzania jest przeskalowanie obrazu do mniejszych rozmiarów w celu zmniejszenia złożoności obliczeniowej powiązanej z analizowaniem pojedynczych pikseli.

W tym celu zaimplementowano metodę interpolacji najbliższym sąsiadem. W programie domyślnie zdjęcia przetwarzane w wielkości dwa razy mniejszej niż oryginał.



**Rysunek 4:** Obraz oryginalny vs. przeskalowany metodą interpolacji najbliższym sąsiadem

### Poprawa jakości

Do poprawy jakości zostały zaimplementowane:

1. filtr gaussa
2. filtr medianowy
3. wyrównywanie histogramu

Ponieważ jakoś zdjęć ze zbioru danych jest bardzo dobra, wykorzystanie tych metod jest opcjonalne. Wyrównywanie histogramu okazało się najbardziej pomocne przy zdjęciach wykonanych na terenie sklepu wykonanych przy silnym sztucznym świetle.

Filtry gaussa i medianowy są użyteczne przy przetwarzaniu w pełnej rozdzielczości, bez skalowania. Pozwalają one na dokładniejszą segmentację obiektów, co uniemożliwia delikatny szum np. dla zdjęcia 2. Natomiast zastosowanie ich przy skali 0.5 prowadzi do nie rozpoznania małych logo, ponieważ łączy ze sobą ich składowe elementy rozdzielone momentami jedynie jednym pikselem.

Wyrównywanie histogramu zostało zaimplementowane tak, aby była możliwość zastosowania dla dowolnego systemu barw, nie tylko HSV. Przyjmuje ona index kanału lub kanałów, dla których ma zachodzić wyrównywanie histogramu.

```
def equallize_histogram(  
    image: np.ndarray, color_index: Union[Tuple[int], List[int], int] = 0  
) -> np.ndarray:  
    """equallize_histogram.  
  
    :param image:  
    :type image: np.ndarray  
    :param color_index:  
    :type color_index: Union[Tuple[int], List[int], int]  
    :rtype: np.ndarray  
    """  
    values = image[..., color_index]  
  
    histogram = create_histogram(values)  
    item_count = image.shape[0] * image.shape[1]  
  
    look_up_table = create_lut(histogram, item_count)  
  
    equallized = image.copy()  
  
    for i, row in enumerate(values):  
        for j, pixel in enumerate(row):  
            mean_value = np.round(pixel.mean())  
  
            if mean_value == 0:  
                continue
```

```

new_mean_value = look_up_table[int(mean_value)]
equallized[i, j, color_index] = np.clip(
    np.round(pixel.astype(np.float32) / mean_value *
→ new_mean_value), 0, 255
)
return equallized

```

### Zmiana przestrzeni barw

W ramach projektu zaimplementowałem konwersję z przestrzeni barw BGR do HSV. Wynika to z charakteru segmentacji: interesują nas bowiem elementy o zadanym kolorze, dlatego możliwość prostej analizy koloru jest bardzo istotna. Ponieważ efekty nakładane są na oryginalny obraz, konwersja odwrotna nie była implementowana.

Przestrzeń HSV reprezentowana jest w programie w zakresie 0-255 dla każdej składowej, dzięki czemu możliwy jest prosty zapis do pliku obrazu w formacie HSV i jego prezentacja.



**Rysunek 5:** Porównanie tego samego zdjęcia w przestrzeni barw BGR i HSV

Działanie algorytmu zostało porównane z konwersją przy użyciu biblioteki OpenCV dając wyniki z zakresu błędu przybliżenia wartości zmiennoprzecinkowych do wartości stałych.

```

def convert_bgr_to_hsv(image: np.ndarray) -> np.ndarray:
    """convert_bgr_to_hsv.

```

*Algorithm inspired by wikipedia formulas for HSV  
[https://en.wikipedia.org/wiki/HSL\\_and\\_HSV](https://en.wikipedia.org/wiki/HSL_and_HSV)*

```
:param image:
:type image: np.ndarray
:rtype: np.ndarray
"""
result = np.zeros_like(image)
image = image / 255
R, G, B = 2, 1, 0
r = image[..., R]
b = image[..., B]
g = image[..., G]

vmax = image.max(2)
vmin = image.min(2)
diff = vmax - vmin

cmax = image.argmax(2)
non_zeros = diff != 0.0

where = (cmax == R) & non_zeros
result[where, 0] = np.floor(
    (60.0 * np.mod(((g[where] - b[where]) / diff[where]), 6.0)) / 2.0
)

where = (cmax == G) & non_zeros
result[where, 0] = np.floor(
    (60.0 * (((b[where] - r[where]) / diff[where]) + 2.0)) / 2.0
)

where = (cmax == B) & non_zeros
result[where, 0] = np.floor(
    (60.0 * (((r[where] - g[where]) / diff[where]) + 4.0)) / 2.0
)

where = vmax != 0
result[where, 1] = diff[where] / vmax[where] * 255

result[..., 2] = vmax * 255

return result
```

## Segmentacja

Poprawna segmentacja obrazu jest podstawą wykrywania logo. W programie została użyta segmentacja przez progowanie wartości H, S i V.

Ponieważ kolor czerwony leży w przestrzeni barw w okolicach wartości granicznej składowej H (wartości z zakresu 0-20 oraz 170-255 przy 256 stopniowej skali Hue), wykorzystano dwie rozdzielne maski dla wartości H z zakresu 0-20 i 170-255 oraz je połączono. Warto zaznaczyć, że ta maska jest delikatnie rozszerzona o wartości z zakresu różu i pomarańczowego, co było wymagane dla paru przypadków z danych testowych - na obrazku pierwszym obwódka loga kafuland jest delikatnie różowa, a na piątym ciemne elementy są postrzegane jako ciemny brązowy.

Przy segmentacji wykorzystano również składowe nasycenia i jasności. Poniższa tabela prezentuje wartości progowe obu masek, a rysunek 6 porównanie obrazu wsadowego do otrzymanej maski.

maska	H min	H max	S min	S max	V min	V max
czerowny1	0	20	145	255	40	255
czerwony2	170	255	145	255	40	255



**Rysunek 6:** Porównanie zdjęcia i otrzymanej maski binarnej dla koloru czerwonego

Następnie zastosowano zmodyfikowany algorytm flood fill do rozdzielenia fragmentów maski na segmenty, wykorzystywany do wypełniania zamkniętych obszarów. Algorytm ten dla kolejnych białych pikseli maski binarnej metodą przeszukiwania wszerz koloruje na losowy, unikalny dla maski kolor wszystkie sąsiadujące z nim piksele.

```
def flood_fill(
    mask: np.ndarray,
```

```
) -> Tuple[np.ndarray, List[np.ndarray]]:  
    """flood_fill.  
  
    :param mask:  
    :type mask: np.ndarray  
    :rtype: Tuple[np.ndarray, np.ndarray]  
    """  
  
    segmnets_mask = np.zeros((*mask.shape[:2], 3), dtype=np.ubyte)  
    segmnets_mask[np.where(mask == 255)] = (255, 255, 255)  
    colors = []  
    checked = set()  
  
while True:  
    # find next non flooded segment  
    available_segments = np.stack(  
        np.where((segmnets_mask == (255, 255, 255)).all(2))  
    ).swapaxes(0, 1)  
  
    if available_segments.shape[0] == 0:  
        break  
  
    if len(colors) >= 255 * 255 * 255:  
        raise Exception("Too many segments, no more colors available  
        → for them")  
  
    chosen =  
→     available_segments[np.random.choice(available_segments.shape[0])]  
  
    while (color := tuple(np.random.randint(0, 255, 3))) in colors:  
        pass  
  
    colors.append(color)  
    point_queue = [chosen]  
  
while point_queue:  
    chosen = point_queue.pop()  
  
    checked.add(tuple(chosen))  
  
try:  
    if mask[tuple(chosen)] == 255:  
        segmnets_mask[tuple(chosen)] = color  
        for direction in (
```

```

        (1, 1),
        (1, 0),
        (1, -1),
        (0, 1),
        (0, -1),
        (-1, 1),
        (-1, 0),
        (-1, -1),
    ):

        to_check = chosen + direction
        # don't check same point multiple times
        if tuple(to_check) not in checked:
            point_queue.append(to_check)

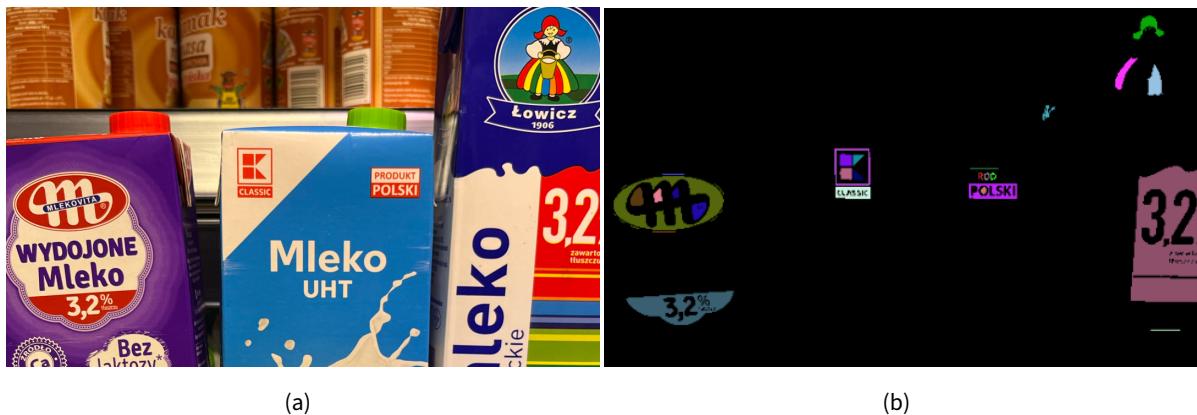
    except IndexError:
        continue

return segmnets_mask, colors

```

Kolejnym etapem segmentacji jest usunięcie zbyt małych i zbyt dużych obszarów. Tutaj usunięto obszary bardzo małe (poniżej 0.01% obszaru całego zdjęcia) oraz bardzo duże (powyżej 9% wielkości zdjęcia, nawet jak logo zajmuje cały obszar zdjęcia żaden jego element nie będzie miał tak dużej powierzchni).

Wynik podziału na segmenty przedstawiony został na rysunku 7.



**Rysunek 7:** Porównanie zdjęcia i otrzymanych segmentów

## Wyznaczanie cech

W celu wyznaczenia cech segmentów skorzystano z momentów centralnych oraz współczynnika Blaira-Blissa. Momenty geometryczne centralne oraz współczynnik Blaira-Blissa cechują się niezmiennością niezależnie od dokonanej translacji co jest szczególnie istotne ze względu na dwa różne trójkąty w logo Kaufland, gdzie jeden jest odbiciem drugiego w pionie, oraz pod kątem wykrywania logo Kaufland niezależnie od jego orientacji.

Dla każdego momentu obliczane są wartości niezmienniczych momentów M1-M6 oraz W4 (współczynnik Blaira-Blissa).

```
def calculate_moments(self) -> None:
    """calculate_moments.

    Calculates M1 to M7

    :rtype: None
    """
    self.moment_0_0 = self.find_moment(0, 0)
    self.moment_1_0 = self.find_moment(1, 0)
    self.moment_0_1 = self.find_moment(0, 1)

    self.i_center = self.moment_1_0 / self.moment_0_0
    self.j_center = self.moment_0_1 / self.moment_0_0

    central_moment_1_1 = self.find_central_moment(1, 1)
    central_moment_0_2 = self.find_central_moment(0, 2)
    central_moment_2_0 = self.find_central_moment(2, 0)
    central_moment_1_2 = self.find_central_moment(1, 2)
    central_moment_2_1 = self.find_central_moment(2, 1)
    central_moment_3_0 = self.find_central_moment(3, 0)
    central_moment_0_3 = self.find_central_moment(0, 3)

    M1 = (central_moment_2_0 + central_moment_0_2) /
    ← np.power(self.moment_0_0, 2)
    M2 = (
        np.power((central_moment_2_0 - central_moment_0_2), 2)
        + 4 * np.power(central_moment_1_1, 2)
    ) / np.power(self.moment_0_0, 4)
    M3 = (
        np.power((central_moment_3_0 - 3 * central_moment_1_2), 2)
        + np.power((3 * central_moment_2_1 - central_moment_0_3), 2)
    ) / np.power(self.moment_0_0, 5)
```

```

M4 = (
    np.power((central_moment_3_0 + central_moment_1_2), 2)
    + np.power((central_moment_2_1 + central_moment_0_3), 2)
) / np.power(self.moment_0_0, 5)

M5 = (
    (central_moment_3_0 - 3 * central_moment_1_2)
    * (central_moment_3_0 + central_moment_1_2)
    *
    (np.power((central_moment_3_0 + central_moment_1_2), 2)
     - 3 * np.power((central_moment_2_1 + central_moment_0_3),
                    2))
)
+ (3 * central_moment_2_1 - central_moment_0_3)
* (central_moment_2_1 + central_moment_0_3)
*
( 3 * np.power((central_moment_3_0 + central_moment_1_2), 2)
  - np.power((central_moment_2_1 + central_moment_0_3), 2)
)
) / np.power(self.moment_0_0, 10)

M6 = (
    (central_moment_2_0 - central_moment_0_2)
    *
    (
        (
            np.power((central_moment_3_0 + central_moment_1_2), 2)
            - np.power((central_moment_2_1 + central_moment_0_3),
                       2)
        )
    )
    +
    4
    *
    central_moment_1_1
    *
    (central_moment_3_0 + central_moment_1_2)
    *
    (central_moment_2_1 + central_moment_0_3)
) / np.power(self.moment_0_0, 7)

M7 = (
    central_moment_2_0 * central_moment_0_2 -
    np.power(central_moment_1_1, 2)
) / np.power(self.moment_0_0, 4)

W4 = self.area / np.sqrt(
    2
    *
    np.pi
    *
    np.power((self.where - (self.j_center, self.i_center)), (2,
      2)).sum()
)

```

kształt	M1	M2	M3	M4	M5	M6	M7	W4
trójkąt	min	0.2	0.0072	0.0028	0.0001	-384.0	-0.000285	0.0078
	max	0.231	0.016	0.0063	0.00031	660.432	1.68798e-05	0.0096
kwadrat	min	0.164	2.149e-05	0.0	0.0	-0.05	-4.513e-07	0.0066
	max	0.17115	0.001381	4.042e-05	8.95e-07	0.002	6.242e-07	0.007
ramka	min	1.38	0.00045	-0.0167	-0.0075	-201e9	-46230.5029	0.47
	max	2.304	0.1105	0.0182	0.01536	272345e9	25.18	1.33

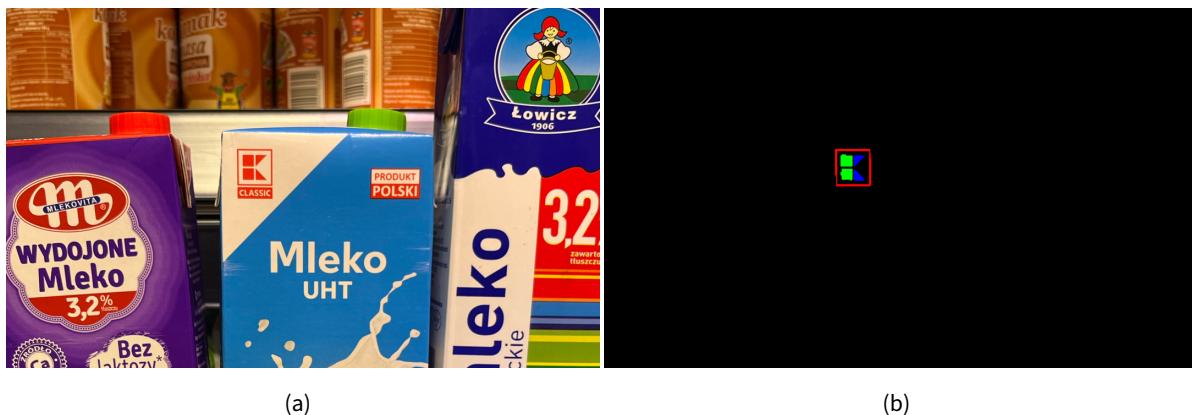
**Tabela 2:** wyznaczone zakresy cech dla poszczególnych kształtów

)

```
self.central_moments = np.array((M1, M2, M3, M4, M5, M6, M7, W4))
```

### Indentyfikacja kształtów

Na zbiorze danych wyznaczono wartości graniczne dla wyznaczonych cech pozwalające na identyfikację: trójkątów, kwadratów oraz ramki. Wartości te przedstawiono w tabeli 2. Na rysunku 8 przedstawiono wyniki identyfikacji zadanych kształtów.

**Rysunek 8:** Porównanie zdjęcia i zidentyfikowanych kształtów. Kolorem czerwonym oznaczono ramkę, zielonym kwadraty a niebieskim trójkąty

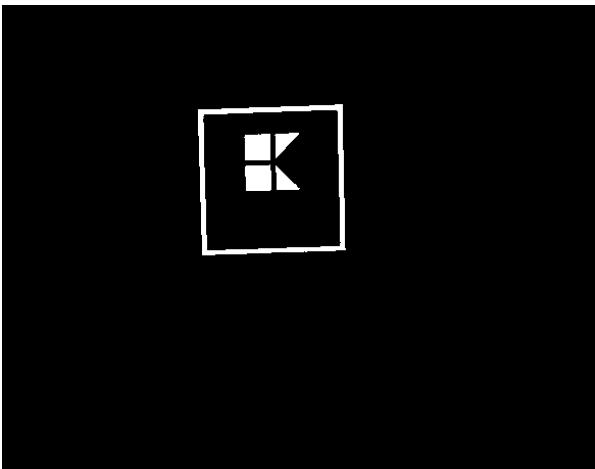
### Rozpoznanie logo

Po identyfikacji kształtów, ostatnim elementem jest rozpoznanie logo. W tym celu wystarczy znaleźć ramkę, wewnętrznej której są dokładnie 2 kwadraty i 2 trójkąty. Efekt końcowy można zapisać w postaci

oryginalnego zdjęcia z naniesionym bounding boxem lub maski binarnej. Na rysunkach 9, 10 i 11 przedstawiono wszystkie zdjęcia ze zbioru danych oraz wyniki dla nich otrzymane.



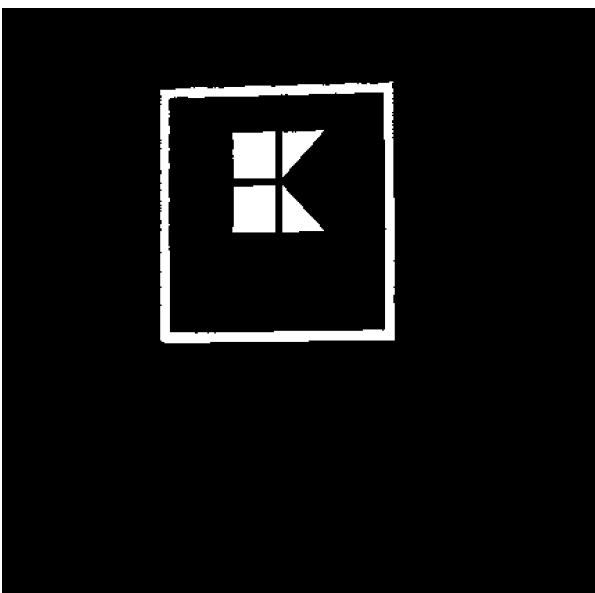
(a)



(b)



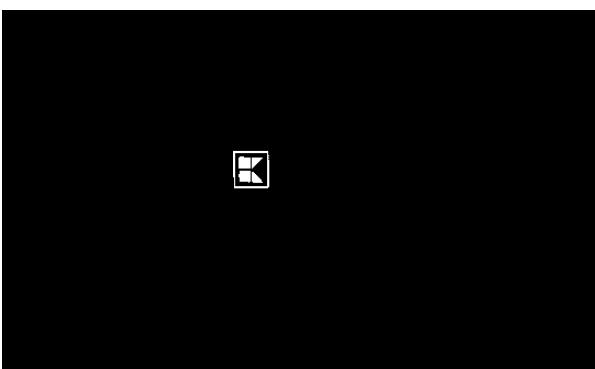
(c)



(d)



(e)



(f)

**Rysunek 9:** Porównanie zdjęć ze zbioru danych oraz otrzymanych dla nich wyników



**Rysunek 10:** Porównanie zdjęć ze zbioru danych oraz otrzymanych dla nich wyników, ciąg dalszy



**Rysunek 11:** Porównanie zdjęć ze zbioru danych oraz otrzymanych dla nich wyników, ciąg dalszy

## Efekt i wnioski

Program zaimplementowany w ramach projektu pozwala na dowolne sterowanie użytymi algorytmami, np. użycie zadanych metod poprawy jakości, sterowanie skalą w jakiej zdjęcie jest przetwarzanie, minimalną i maksymalną wielkością segmentów, pozwala na wyświetlanie efektów na każdym kroku i wybór efektu końcowego (maska czy bounding box). Udało się poprawnie rozpoznać logo dla każdego zdjęcia ze zbioru danych.

Możliwym usprawnieniem byłoby rozpoznawanie dodatkowo napisów - "Kaufland" wewnątrz ramki lub po jej prawej stronie oraz "Classic" pod ramką. Sprowadzałoby się to do wyznaczenia cech dla każdej litery oraz modyfikacji rozpoznawania logo tak, aby pod uwagę brany był również napis.