Metody Obliczeniowe w Nauce i Technice

Laboratorium 9

Wojciech Łącki

SPIS TREŚCI

ZADANIE 1 – ANALIZA OBRAZÓW

1. Wczytaj obraz przedstawiający tekst, dokonaj odwrócenia kolorów (czarne tło), a następnie jego transformacji do domeny częstotliwościowej za pomocą DFT. Przedstaw wartości fazy i modułu współczynników Fouriera (osobno).

Kod, który wywołuje odpowiednie funkcje, aby otrzymać zamierzony efekt z zadania został przedstawiony poniżej.

```
image, image_rotated = prepare_image_data(analyzed_image)
pattern, pattern_rotated = prepare_image_data(pattern_image)
pattern_width, pattern_height = pattern.size
image_dft = fft2(image_rotated)
show_amplitude_and_phase(image_dft)
```

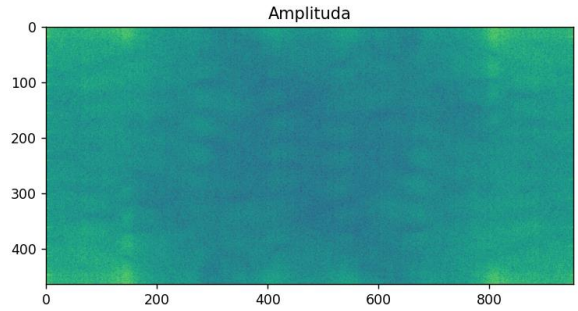
Funkcja przekształcająca otrzymany obraz dokonująca odpowiednich przekształceń obrazu, w tym zamiana kolorów i obrót. Dla tekstu zamieniamy kolory, natomiast dla ławicy nie (wtedy lepsze dpasowania).

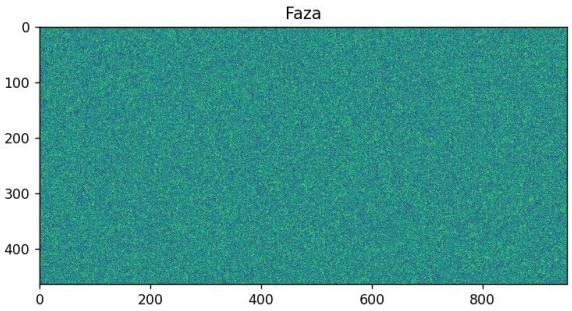
```
def prepare_image_data(analyzed_image):
    image = Image.open(analyzed_image)
    image_grey = ImageOps.grayscale(image)
    image_invert = ImageOps.invert(image_grey)
    image_rotated = np.swapaxes(np.asarray(image_invert), 0, 1) # for text
    # image_rotated = np.swapaxes(np.asarray(image_grey), 0, 1) # for fishes
    return image, image_rotated
```

Natomiast faza i współczynniki można wyświetlić dzięki poniższej funkcji.

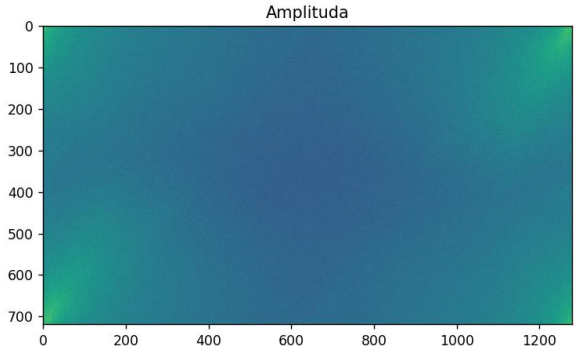
```
def show_amplitude_and_phase(image_dft):
    image_back = np.swapaxes(np.array(image_dft), 0, 1)
    fig, ax = plt.subplots(1, 2)
    ax[0].imshow(np.log(abs(image_back)))
    ax[0].set_title('Amplituda')
    ax[1].imshow(np.angle(image_back))
    ax[1].set_title('Faza')
    plt.show()
```

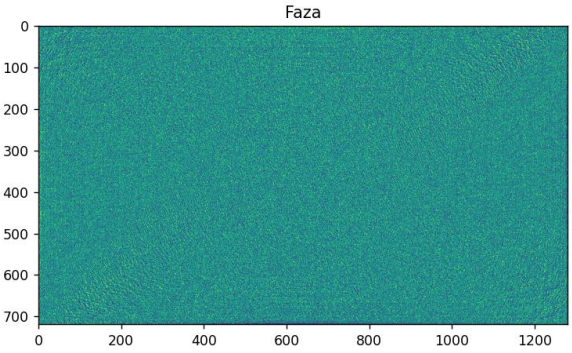
Wyniki uzyskane dla teksty.





Wyniki uzyskane dla ryb.





2. Wybierz przykładowego reprezentanta grupy - wytnij z dużego zdjęcia wzorzec obiektu np. literę *e* lub charakterystyczny fragment ryby z ławicy (imcrop).

Dla tekstu została wycięta litera „e”.

Natomiast dla ławicy ryb fragment ryby.

3. Wykorzystując transformacje Fouriera oblicz korelacje między wybranym wzorcem, a całym obrazem.

```
c = np.real(ifft2(image_dft * fft2(np.rot90(pattern_rotated, k=2), s=image.size)))
```

4. Przedstaw otrzymany obraz wyjściowy oraz punkty, w których wykryto wystąpienie wzorca (nałóż je na obraz oryginalny).

Aby nałożyć wzorce na obraz, to najpierw trzeba wyfiltrować te współrzędne, w których należy je umieścić. Żeby to zrobić jednym z parametrów programu jest współczynnik, który mówi nam od jakiego pułapu zaczyna dana współrzędna być akceptowana.

```
filter_value = coefficient * np.max(c)
image_loaded = image.load()
matches = np.argwhere(c >= filter_value)
```

Nałożenie wzorca wykonuje poniższy kod.

```
print("Liczba wystąpień wzorca:", len(matches))
for startX, startY in matches:
    for x in range(pattern_width):
        for y in range(pattern_height):
            fill_strategy(image, image_loaded, startX, startY, x, y, pattern_width, pattern_height)
show_image_with_patterns(image)
```

Pokazanie efektu nakładania wzorca.

```
def show_image_with_patterns(image):
    fig, ax = plt.subplots(1, 1)
    ax.imshow(image)
    ax.set_title("Wzorzec nałożone na obraz")
    plt.show()
```

Zostały napisane różne strategie zaznaczania obszarów, w których wykryto wzorzec.
Pierwsza wypełnia całe miejsce.

```
def fill_space(image, image_loaded, start_x, start_y, x, y, pattern_width, pattern_height):
    image_loaded[start_x - x, start_y - y] = (255, 0, 0)
```

Druga tylko ramkę.

```
def fill_frame(image, image_loaded, start_x, start_y, x, y, pattern_width, pattern_height):
    if x == 0 or y == 0 or x == pattern_width - 1 or y == pattern_height - 1:
        image_loaded[start_x - x, start_y - y] = (255, 0, 0)
```

Trzecia została stworzona tylko dla tekstu, żeby litery zmieniły kolor.

```
def condition_fill_space(image, image_loaded, start_x, start_y, x, y, pattern_width, pattern_height):
    r, g, b = image.getpixel((start_x - x, start_y - y))
    if (r, g, b) != (255, 255, 255):
        image_loaded[start_x - x, start_y - y] = (255, 0, 0)
```

5. Znajdź liczbę wystąpień wzorca. Testy przeprowadź na dwóch zbiorach danych.

Dla tekstu otrzymaliśmy następującą liczbę wystąpień (współczynnik równy 0.9).

```
Liczba wystąpień wzorca: 43
```

Dla ławicy było ich trochę więcej (współczynnik równy 0.75). Należy jednak zauważyć, że wyniki się pokrywają i nie jest to wykrywane, przynajmniej w tym zadaniu (przy OCR jest to już uwzględniane). Różnych wyników można zaobserwować 16, tak wynika patrząc na zbiory wyników.

```
Liczba wystąpień wzorca: 3284
```

ZADANIE 2 – OCR

Napisz program przekształcający obraz w tekst, przyjmując następujące założenia:

1. Na obrazie znajduje się tekst złożony z małych liter alfabetu łacińskiego oraz cyfr.
2. Na obrazie występuje jeden typ i rozmiar czcionki.
3. Weź pod uwagę czcionki szeryfowe i bezszeryfowe.
4. W tekście występują znaki przestankowe: .,?!.
5. Tekst składa się z wielu linii.
6. Tekst może być obrócony (krzywo zeskanowany w całości).
7. Program powinien zwracać tekst wraz ze znakami białymi i znakami nowych linii.
8. Program może raportować procent poprawnie rozpoznanych liter dla pre-definiowanych obrazów testowych.
9. Program powinien dodatkowo zliczać liczbę wystąpień każdej litery.
10. Należy zastosować operacje splotu i DFT albo inne metody (klasyfikacja).
11. Należy dokonać redukcji szumu na obrazie wejściowym.

Na początek została napisana funkcja do obracania obrazu w razie potrzeby.

```
def repair_image(filename):
    img_before = cv2.imread(filename)
    img_before = cv2.bitwise_not(img_before)
    # cv2.imshow("Before", img_before)
    # key = cv2.waitKey(0)
    img_gray = cv2.cvtColor(img_before, cv2.COLOR_BGR2GRAY)
    img_edges = cv2.Canny(img_gray, 100, 100, apertureSize=3)
    lines = cv2.HoughLinesP(img_edges, 1, math.pi / 180.0, 100, minLineLength=20, maxLineGap=5)
    angles = []
    for [[x1, y1, x2, y2]] in lines:
        # cv2.line(img_before, (x1, y1), (x2, y2), (255, 0, 0), 3)
        angle = math.degrees(math.atan2(y2 - y1, x2 - x1))
        angles.append(angle)
    # cv2.imshow("Detected lines", img_before)
    # key = cv2.waitKey(0)
    median_angle = np.median(angles)
    print("Angle: " + str(median_angle))
    if median_angle == 0:
        print("No need to rotate image")
        return filename
    print("Rotating image...")
    img_rotated = ndimage.rotate(img_before, median_angle)
    img_rotated = cv2.bitwise_not(img_rotated)

    arr = filename.split(".")
    rotated_filename = arr[0] + "_rotated.png"
    cv2.imwrite(rotated_filename, img_rotated)
    return rotated_filename
```

Następnie powstał początek głównej funkcji, która czyta wszystkie czcionki z folderu.

```
fonts, space_widths, line_heights = [], [], []
available_fonts = os.listdir("fonts")
for fontname in available_fonts:
    font = ImageFont.truetype("fonts/" + fontname, size)
    fonts.append(font)
    space_widths.append(font.getsize(" ")[0] * 0.9)
    line_heights.append(font.getsize(chars)[1])
space_width = sum(space_widths) / len(space_widths)
font_height = sum(line_heights) / len(line_heights)
```

Przygotowujemy nasze dane z obrazu.

```
image, image_rotated = prepare_image_data(analyzed_image)
```

Funkcja ta ma też możliwość redukcji szumów.

```
def reduce_noise(data):
    height, width = data.shape
    u, s, vh = scipy.sparse.linalg.svds(scipy.sparse.linalg.aslinearoperator(np.array(data, dtype="float32")),
                                        k=min(height - int(height * 0.9) - 1, width - int(width * 0.9) - 1))
    return u @ np.diag(s) @ vh


def prepare_image_data(analyzed_image):
    image = Image.open(analyzed_image)
    image_grey = ImageOps.grayscale(image)
    image_invert = ImageOps.invert(image_grey)
    image_rotated = np.swapaxes(np.asarray(image_invert), 0, 1)
    image_rotated = reduce_noise(image_rotated)_# comment this line if you dont want to reduce noise
    return image, image_rotated
```

Wyszukujemy wszystkie pasujące znaki, z wcześniej zadeklarowanych.

```
all_matches = []
for index, font in enumerate(fonts):
    for letter in chars:
        print("Checking char " + letter + " from font " + str(index))
        letter_array = load_letter(letter, font, font_height)
        # cannot delete columns filled with zeros because better to leave it
        # letter_array = letter_array[:, ~np.all(letter_array == 0, axis=0)]
        # cannot delete rows filled with zeros because all letters has to have the same height
        # letter_array = letter_array[~np.all(letter_array == 0, axis=1)]
        pattern_height, pattern_width = letter_array.shape
        pattern_rotated = np.swapaxes(np.asarray(letter_array), 0, 1)
        c = np.real(ifft2(fft2(image_rotated) * fft2(np.rot90(pattern_rotated, k=2), s=image.size)))
        maxi = np.max(c)
        possible_coefficient = special.get(letter, -1)
        if possible_coefficient != -1:
            coefficient = possible_coefficient
        filter_value = coefficient * maxi
        matches = np.argwhere(c >= filter_value)
        for start_x, start_y in matches:
            ratio = c[start_x, start_y] / maxi
            if start_y - pattern_height > 0 and start_x - pattern_width > 0:
                # (y_start, x_start), (height, width), ratio, letter, font_index
                all_matches.append(((start_y, start_x), letter_array.shape, ratio, letter, index))
```

Ładowanie danej litery z określonej czcionki odbywa się przy pomocy poniższego kodu.

```
def load_letter(letter, font, font_height):
    letter_im = PIL.Image.new('L', (font.getsize(letter)[0], font_height), 0)
    letter_draw = PIL.ImageDraw.Draw(letter_im)
    letter_draw.text((0, 0), letter, font=font, fill=255)
    return np.asarray(letter_im)
```

Dla niektórych liter zostały zadeklarowane specjalne warunki dopasowania.

```
special = {
    "z": 1,
    ".": 1,
    ",": 1,
    "?": 1,
    "!": 1
}
```

Następnie zebrane dane są sortowane po współczynniku dopasowania.

```
all_matches.sort(key=lambda tup: tup[2], reverse=True)
considerable_matches = []
for i in range(len(all_matches)):
    ok = True
    r1 = all_matches[i][0]
    l1 = (r1[0] - all_matches[i][1][0] + 1, r1[1] - all_matches[i][1][1] + 1)
    for j in range(len(considerable_matches)):
        r2 = considerable_matches[j][0]
        l2 = (r2[0] - considerable_matches[j][1][0] + 1, r2[1] - considerable_matches[j][1][1] + 1)
        if do_overlap(l1, r1, l2, r2):
            common_surface = surface((max(l1[0], l2[0]), max(l1[1], l2[1])), (min(r1[0], r2[0]), min(r1[1], r2[1])))
            rect1_surface = surface(l1, r1)
            rect2_surface = surface(l2, r2)
            total_surface = rect1_surface + rect2_surface - common_surface
            ratio = common_surface / total_surface
            if ratio > 0.1:
                if all_matches[i][2] == considerable_matches[j][2]:
                    old_letter_font_index = considerable_matches[j][4]
                    new_letter_font_index = all_matches[i][4]
                    if check_better_new_letter(considerable_matches[j][3], fonts[old_letter_font_index],
                                               all_matches[i][3], fonts[new_letter_font_index], font_height):
                        considerable_matches[j] = all_matches[i]
                ok = False
                break
    if ok:
        considerable_matches.append(all_matches[i])
```

Znaki są sprawdzane między sobą, czy nie następują kolizje.

```
def do_overlap(l1, r1, l2, r2):
    # If one rectangle is on left side of other
    if l1[1] >= r2[1] or l2[1] >= r1[1]:
        return False
    # If one rectangle is above other
    if l2[0] >= r1[0] or l1[0] >= r2[0]:
        return False
    return True


def surface(l1, r1):
    width = r1[1] - l1[1]
    height = r1[0] - l1[0]
    return width * height
```

W przypadku kolizji są sprawdzane pewne warunki:

- Czy wspólna część („ratio”) jest większa od pewnej stałej („0.1”)?
- Czy współczynniki dopasowania są równe?
- Czy możliwy znak ma więcej pixeli w kolorze czarnym?

Jeśli on zajdą, to znak, który był wcześniej zostaje zastąpiony nowym.

```
def check_better_new_letter(old_letter, old_letter_font, new_letter, new_letter_font, font_height):
    old_letter_array = load_letter(old_letter, old_letter_font, font_height)
    old_letter_similar = np.argwhere(old_letter_array > 0)
    new_letter_array = load_letter(new_letter, new_letter_font, font_height)
    new_letter_similar = np.argwhere(new_letter_array > 0)
    return len(new_letter_similar) > len(old_letter_similar)
```

Następnie następuje redukcja podobnych poziomów wysokości, aby później łatwiej było odtworzyć oczekiwane rozwiązanie. Po wykonaniu poniższego kodu wszystkie znaki na podobnym poziomie będą mieć zmienioną wysokość na jedną z wartości, które zostały wcześniej znalezione.

```
detected_height_levels = find_height_levels(considerable_matches)
detected_height_levels.sort()
height_levels = reduce_height_levels(detected_height_levels, font_height)
change_letters_levels(height_levels, considerable_matches)
```

```
def find_height_levels(considerable_matches):
    height_levels = []
    for match in considerable_matches:
        if match[0][0] not in height_levels:
            height_levels.append(match[0][0])
    return height_levels


def reduce_height_levels(detected_height_levels, serif_font_height):
    height_levels = [detected_height_levels[0]]
    for i in range(1, len(detected_height_levels)):
        if detected_height_levels[i] - height_levels[-1] <= serif_font_height / 2:
            height_levels[-1] = detected_height_levels[i]
        else:
            height_levels.append(detected_height_levels[i])
    return height_levels
```

```
def change_letters_levels(height_levels, considerable_matches):
    for i, match in enumerate(considerable_matches):
        best_height = height_levels[find_closest(match[0][0], height_levels)]
        if best_height != match[0][0]:
            new_match = ((best_height, match[0][1]), match[1], match[2], match[3], match[4])
            considerable_matches[i] = new_match


def find_closest(value, heights):
    best = abs(value - heights[0])
    index = 0
    for i in range(1, len(heights)):
        possible = abs(value - heights[i])
        if possible < best:
            best = possible
            index = i
    return index
```

Następnie znaki są sortowane po współrzędnych „x”, a później po współrzędnych „y”. W kolejnym kroku zliczane są wystąpienia wszystkich rozpatrywanych liter.

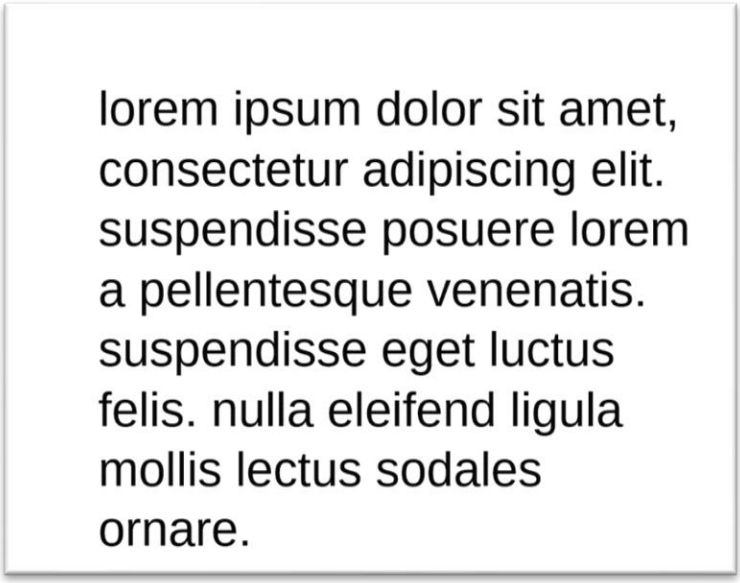
```
considerable_matches.sort(key=lambda tup: tup[0][1] - tup[1][1])  # x sort
considerable_matches.sort(key=lambda tup: tup[0][0] - tup[1][0])  # y sort
print()
chars_occur = {}
for letter in chars:
    chars_occur[letter] = 0
for i in range(len(considerable_matches)):
    current_char = considerable_matches[i][3]
    occurs = chars_occur.get(current_char, -1)
    chars_occur[current_char] = occurs + 1
for letter, occurs in chars_occur.items():
    print(letter, ":", occurs)
```

Na sam koniec tekst jest wypisywany do konsoli, a poszczególne znaki są zaznaczane na obrazie.

```
image_loaded = image.load()
for i in range(len(considerable_matches)):
    current_char = considerable_matches[i][3]
    start_y, start_x = considerable_matches[i][0]
    pattern_height, pattern_width = considerable_matches[i][1]
    for x in range(pattern_width):
        for y in range(pattern_height):
            fill_strategy(image_loaded, start_x, start_y, x, y, pattern_width, pattern_height)
    if considerable_matches[i][0][0] - considerable_matches[i - 1][0][0] > font_height / 2:
        print("")
    space_between_chars = (considerable_matches[i][0][1] - considerable_matches[i][1][1]) - \
                          considerable_matches[i - 1][0][1]
    if space_between_chars > space_width:
        step = space_width
        while space_between_chars > step:
            print(" ", end="")
            step += space_width
    print(current_char, end="")
print()
show_image_with_patterns(image)
```

```
def show_image_with_patterns(image):
    fig, ax = plt.subplots(1, 1)
    ax.imshow(image)
    ax.set_title("Wzorzec nałożone na obraz")
    plt.show()
```

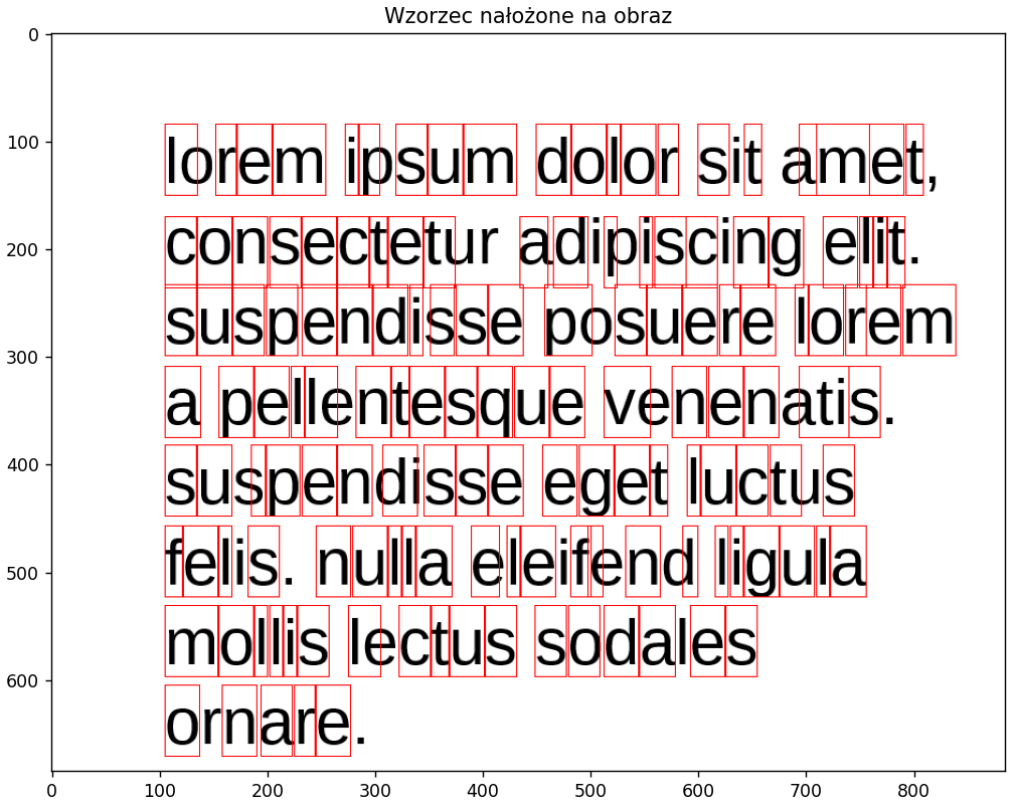
Zostało wykonanych kilka testów. **Nazleży zaznaczyć, że tekst był pisany w wordzie w dwóch czcionkach „Liberation Sans” oraz „Liberation Serif” w rozmiarze 36, a następnie były robione screeny tych tekstów. Współczynnik w testach był ustawiony na 0.94.**
Najpierw dla czcionki bezszeryfowej i bez obrotu.



Wyniki były następujące.
Dla tego tekstu nie trzeba było wykonać obrotu.

```
No need to rotate image
```

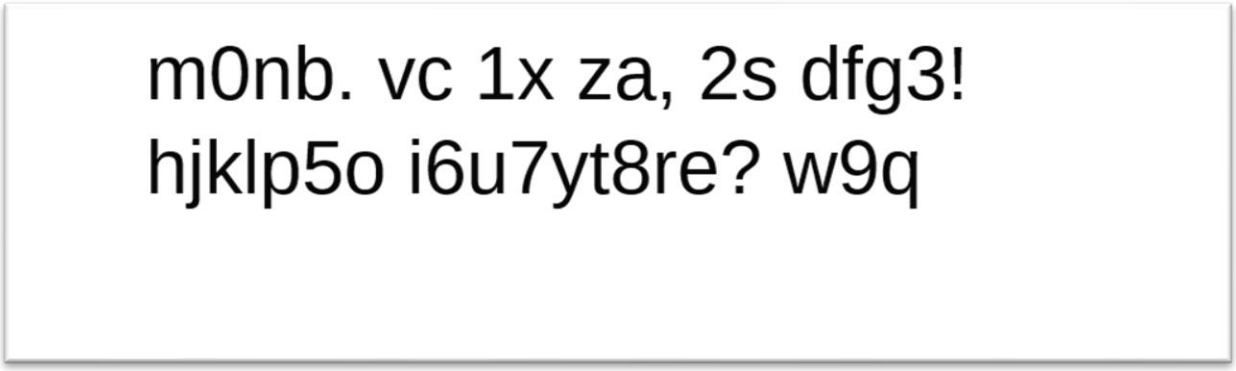
Zaznaczone miejsca, w których znaleziono jakieś dopasowanie.



A tekst uzyskany jest następujący.

```
k rem irsum doior st  ,met
cohsecteh    zdl ischg eiit
suspehdisse w suere iorem
a peik htesque w hen ms
su ,pen1sse eget lucb s
feis  huiia eief. h j iiguia
moiiis k ctus scdaes
o hzre
```

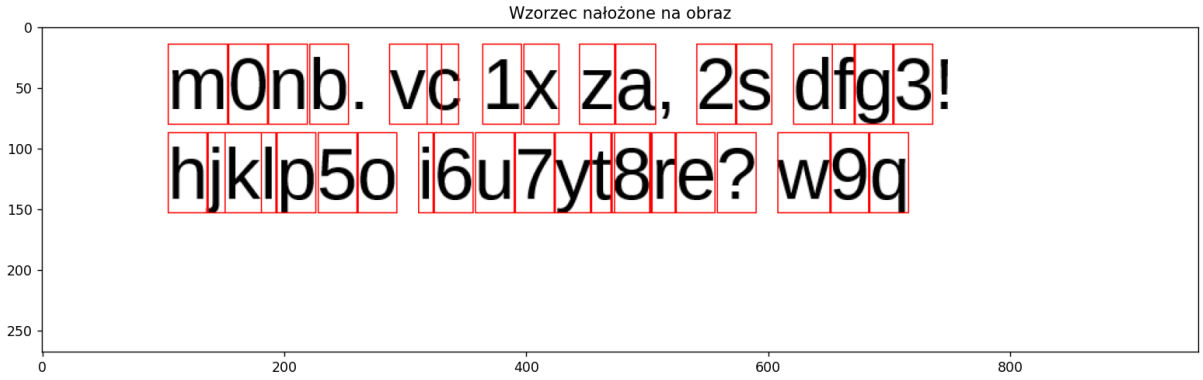
Kolejny test czcionki bezszeryfowej i bez obrotu.



Wyniki były następujące.
Dla tego tekstu nie trzeba było wykonać obrotu.

```
No need to rotate image
```

Zaznaczone miejsca, w których znaleziono jakieś dopasowanie.



A tekst uzyskany jest następujący.

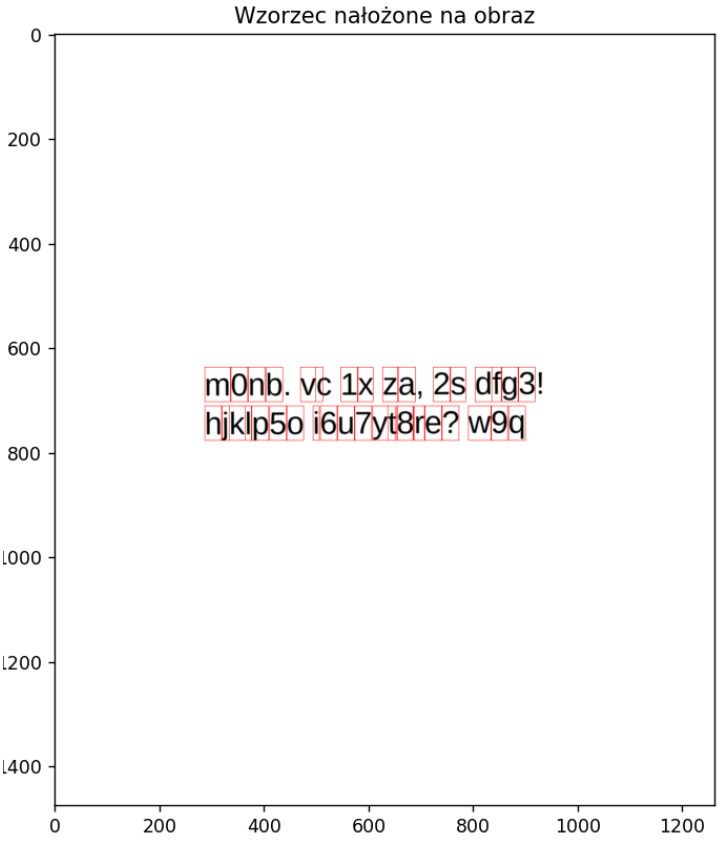
```
m0nb  wc 1x za  2s dfg3
hjkip5o i6u7yt88re? w9q
```

Teraz wykonano test czcionki bezszeryfowej z obrotem.

Program dokonał obrotu obrazu.

```
Angle: -57.03544476060773
Rotating image...
```

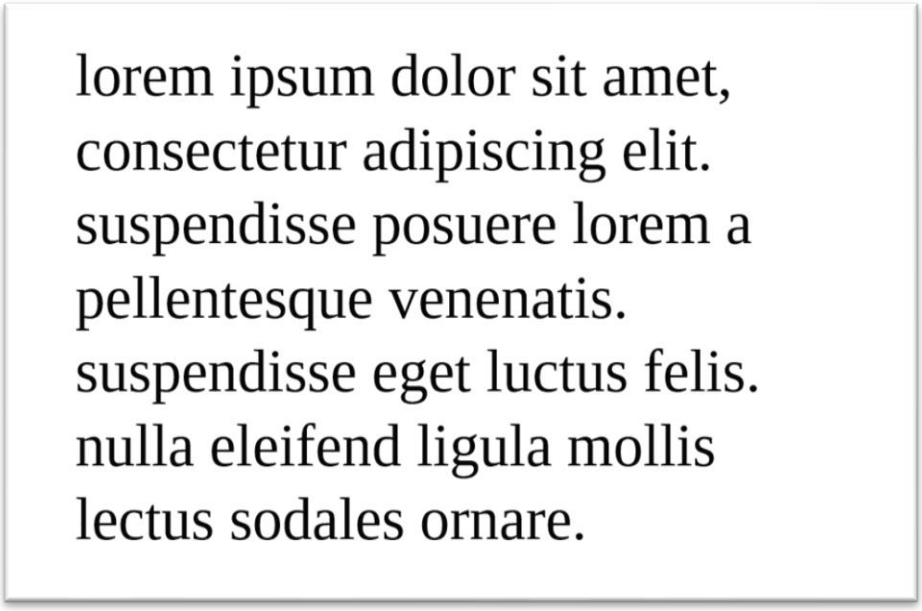
Wyniki były następujące.
Zaznaczone miejsca, w których znaleziono jakieś dopasowanie.



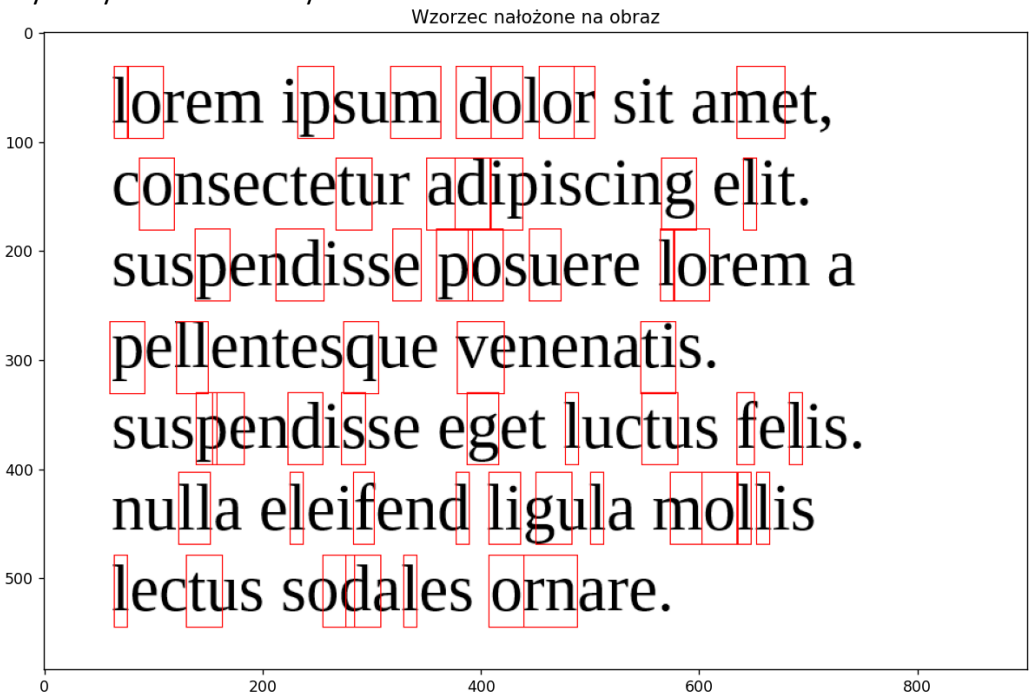
A tekst uzyskany jest następujący.

```
m0nb  wv   1x za  2s dfg3
hjkip5o i6u7y.88re? w9q
```

Podobne testy wykonano dla czcionki szeryfowej bez obrotu.



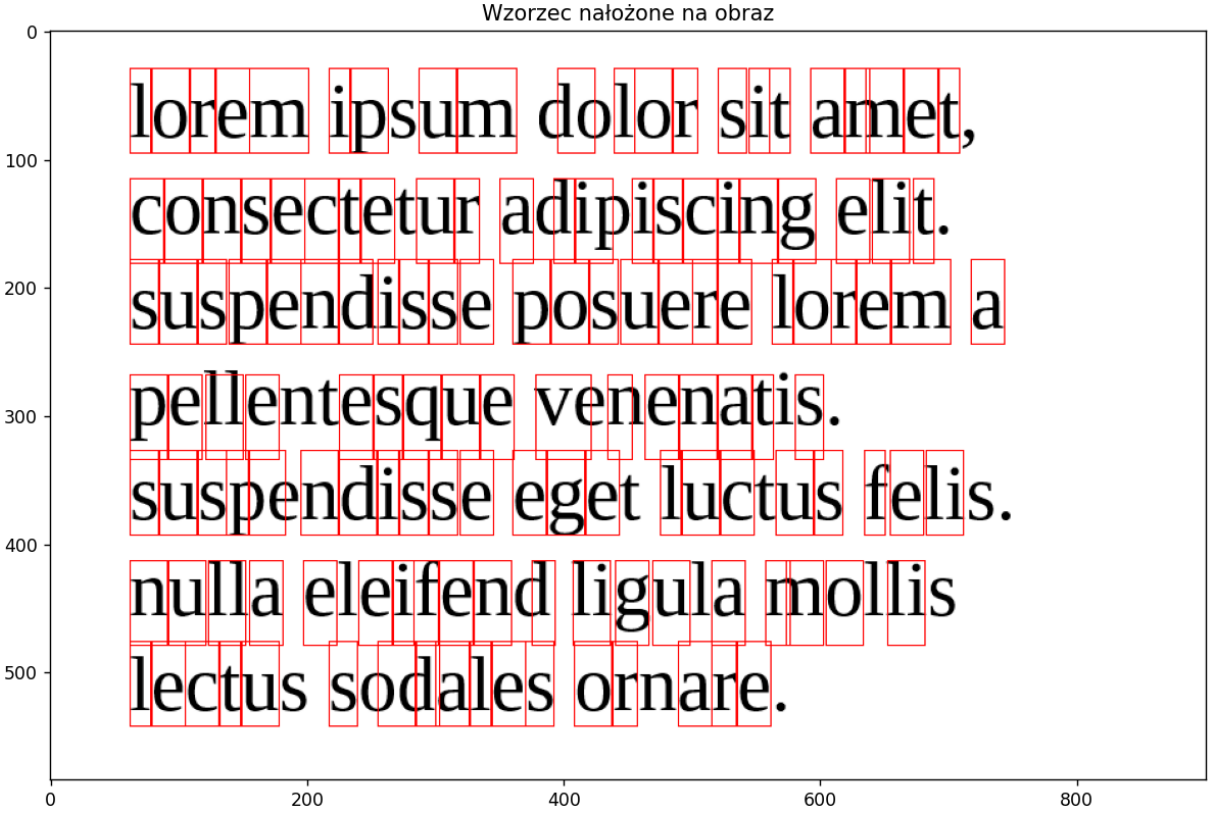
Wynik był strasznie marny.



Dlatego program ten wywołano jedynie z czcionką szeryfową. Wynik był o wiele lepszy.
Wyniki były następujące.
Dla tego tekstu nie trzeba było wykonać obrotu.

```
No need to rotate image
```

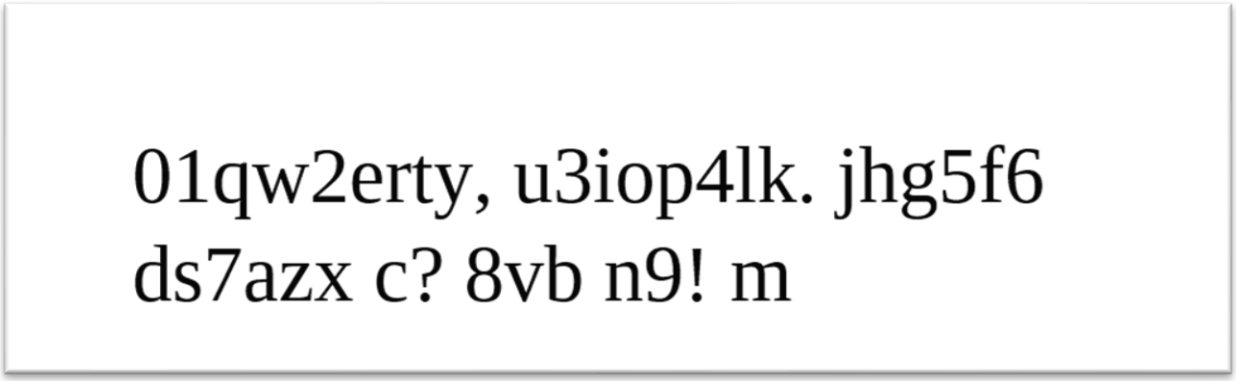
Zaznaczone miejsca, w których znaleziono jakieś dopasowanie.



A tekst uzyskany jest następujący.

```
lorem ip um  k lor sit arnet
consecte ur a i4 iscin3 eht
suspencisse posuere lorem a
pe2e   es9ue wrenat s
susjxndisse ege  luc us ieb
nu6a e eifen jh?u a rno b
lectu  s xjales or  are
```

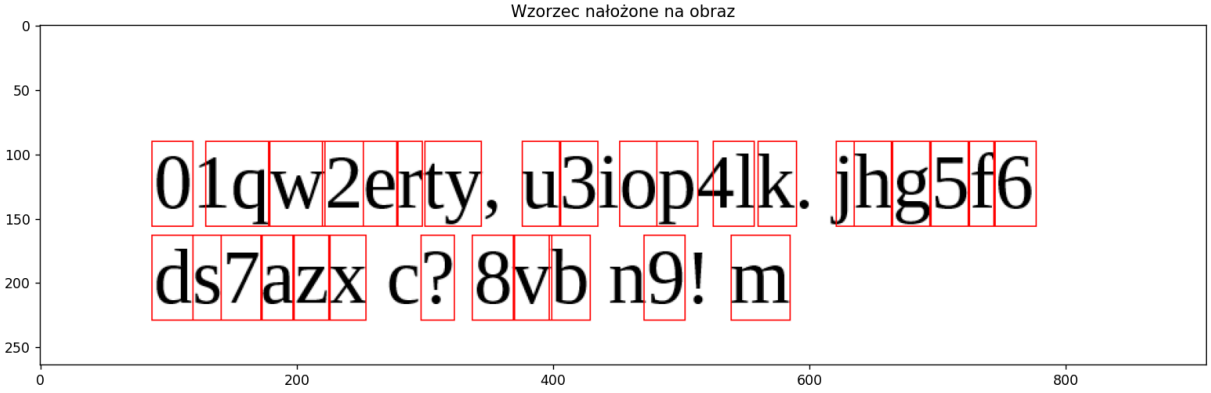
Kolejny test dla czcionki szeryfowej bez obrotu.



Wyniki były następujące.
Dla tego tekstu nie trzeba było wykonać obrotu.

```
No need to rotate image
```

Zaznaczone miejsca, w których znaleziono jakieś dopasowanie.



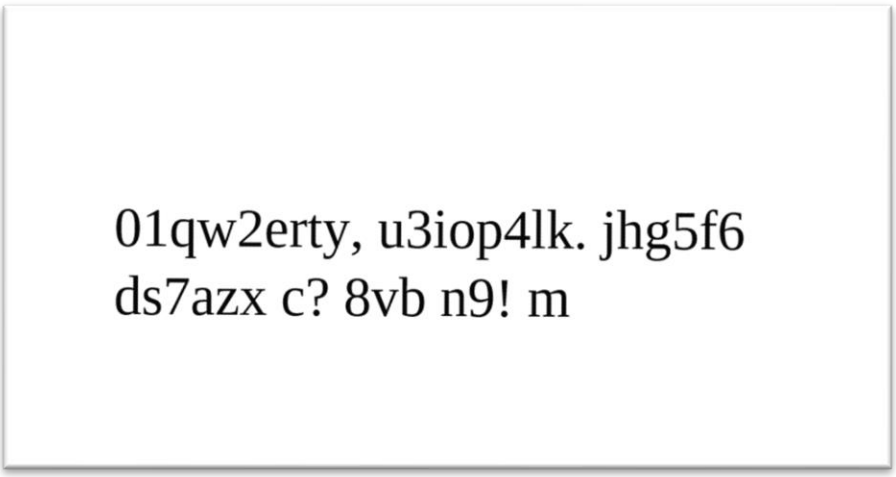
A tekst uzyskany jest następujący.

```
0mw2erw  u3 opgk  jhg5f6
ds7azx  ?8vb  9  m
```

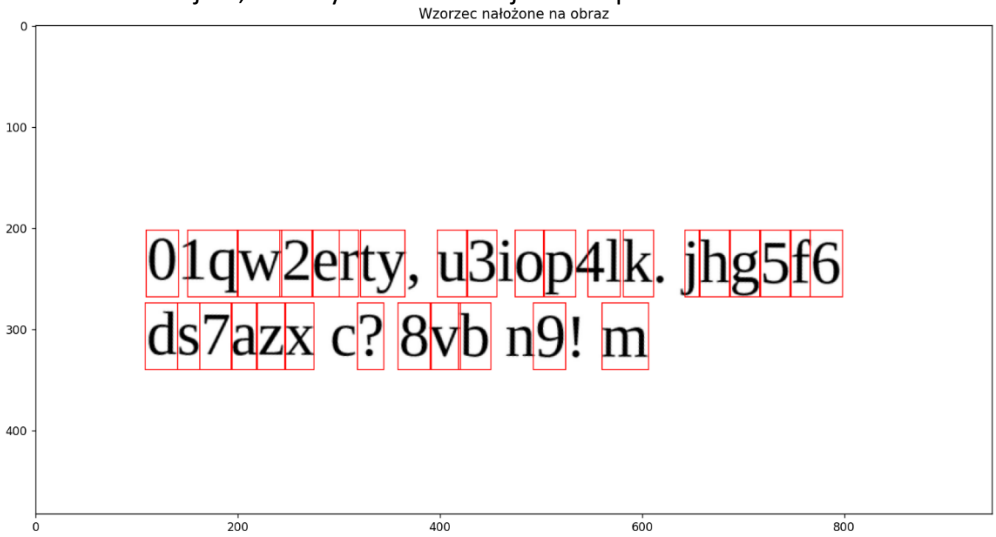
Teraz dokonano obrotu tekstu.

Program dokonał obrotu.

```
Angle: 14.502374590303004
Rotating image...
```



Wyniki były następujące.
Zaznaczone miejsca, w których znaleziono jakieś dopasowanie.



A tekst uzyskany jest następujący.

```
0mw2erw  u3 opgk  jhg5f6
ds7azx  ?8vb  9  m
```