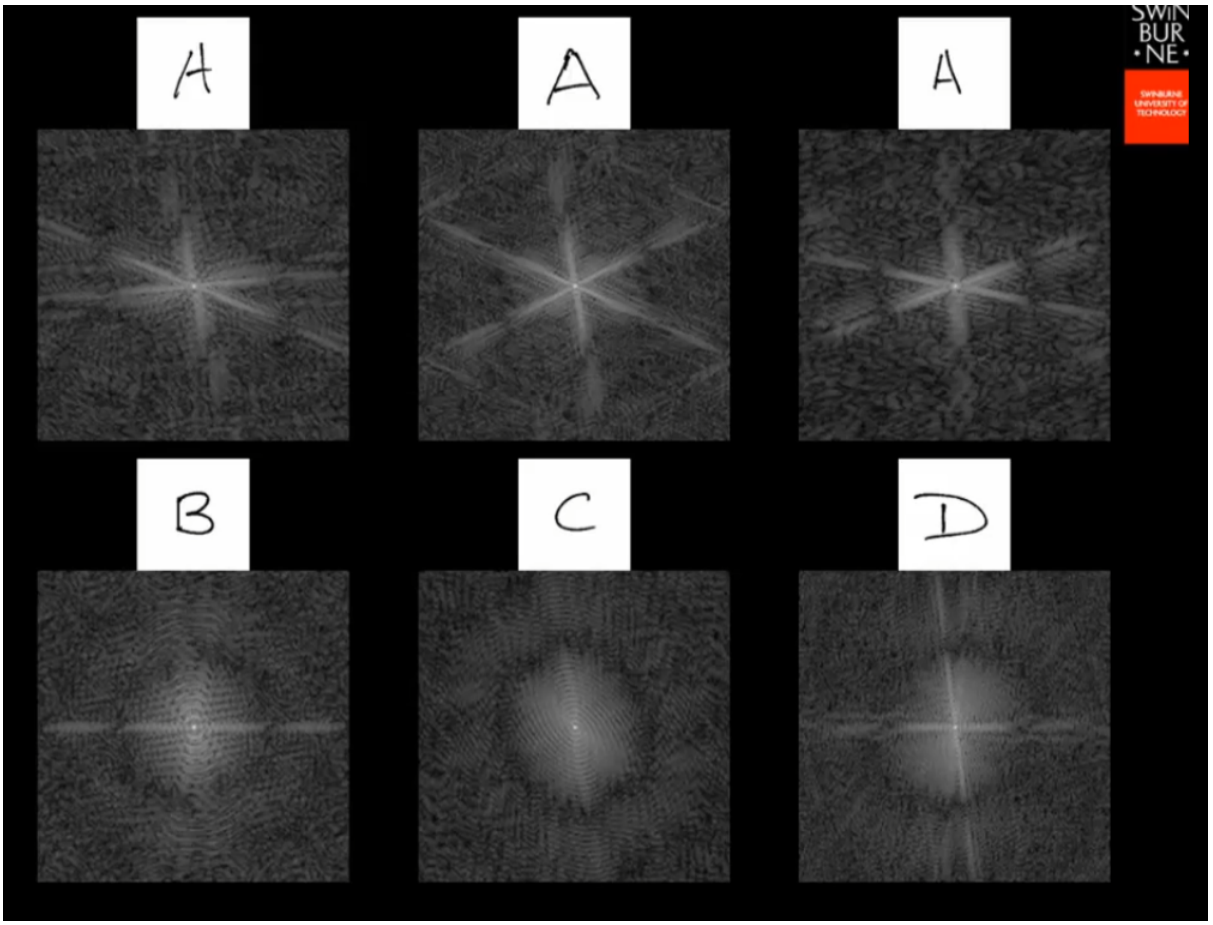
OCR

Damian Tworek

0. Dlaczego stosować transformatę Fouriera na obrazku?

The Fourier Transform is used if we want to access the geometric characteristics of a spatial domain image.
(https://homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm#:~:text=The%20Fourier%20Transform%20is%20used%20if%20we%20want%20to%20access%20the%20geometric%20characteristics%20of%20a%20spatial%20domain%20image.)



(https://www.youtube.com/watch?v=gwaYwRwY6PU)

Widzimy, że litery A tworzą charakterystyczny kształt, różniący się od pozostałych liter.

Idea

Dostajemy na wejściu obrazek i wyciągamy z niego napis.

Podejście

Początkowa próba:

```
x = np.fft.fft2(self.img)
y = np.fft.fft2(np.rot90(pattern, 2), x.shape)
res = np.multiply(x, y)
corr = np.abs(np.fft.ifft2(res)).astype(float)
corr[corr < thresh * np.max(corr)] = 0
corr[corr > upper_thresh * np.max(corr)] = 0
corr[corr != 0] = 255
```

Jednakże odcinanie po największej wartości z macierzy `corr` nie wykrywało poprawnie liter, więc wpadłem na inny pomysł…

Dla każdego szukanego patternu zapisuję jego wartość największa z macierzy korelacji, która jest wyznaczana w następujący sposób:

```
pattern = self.pattern_imgs[l]
x = np.fft.fft2(pattern)
y = np.fft.fft2(np.rot90(pattern, 2), x.shape)
res = np.multiply(x, y)
corr = np.abs(np.fft.ifft2(res)).astype(float)
scores[l] = np.max(corr)
```

Następnie dla wejściowego obrazka, z którego chce wydobyć tekst, stosuję podobną procedurę, czyli

```
x = np.fft.fft2(self.img)
y = np.fft.fft2(np.rot90(pattern, 2), x.shape)
res = np.multiply(x, y)
corr = np.abs(np.fft.ifft2(res)).astype(float)
corr[corr < thresh * self.scores[key]] = 0
corr[corr > upper_thresh * self.scores[key]] = 0
corr[corr != 0] = 255
```

gdzie,

```
thresh ~= 0.93
upper_thresh ~= 1.07
```

Miarą podobieństwa jest wymnożenie macierzy element po elemencie.

Następnie tam, gdzie macierz corr ma wartość 255, zapisuję jej pozycję i zamazuję obszar, w którym ją wykryło na czarno.

1. Kolejność liter

```
self.patterns = ['1', '2', '8', '4', '5', '6', '7', '3', '9', '0', 'g',
'z', 'x', 'b', 'm', 'h', 'd', 'y', 'v',
                 'a', 's', 'p', 'q', 'o', 'e', 'c', 'f', 'n', 'j',
                 'k', 'l',
                 'w', 'r', 't', 'u', 'i', 'comma', 'question',
'exclamation', 'dot']
```

uzyskana metodą prób i błędów, takie nasunęły się spostrzeżenia:

```
"""
c zamazalo b
v zamazalo n i m
c zamazalo n
b zamzało g
g poprawnie działa dla rotacji o 180 stopni
n zamazało h
c zamzało d
v zamazało y
c zamazało o i trochę p
3 zamazała 8
"""
```

2. Gdybyśmy nie odwracali patternu o 180 stopni to moja implementacja OCR się myli znacznie, przy prostym tekście wykrywa:

qhfb mbyhk
br mh
mhmfrha

zamiast:

hello mownit
ocrtest
multiline

3. Testy

Do testów użyłem metryki Jaro Distance:

Jaro similarity [edit]

The Jaro similarity sim_j of two given strings s_1 and s_2 is

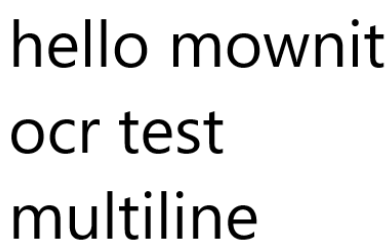
$$sim_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

Where:

- $|s_i|$ is the length of the string s_i;
- m is the number of *matching characters* (see below);
- t is the number of *transpositions* (see below).

Dla czcionki SegoeUI:

input:

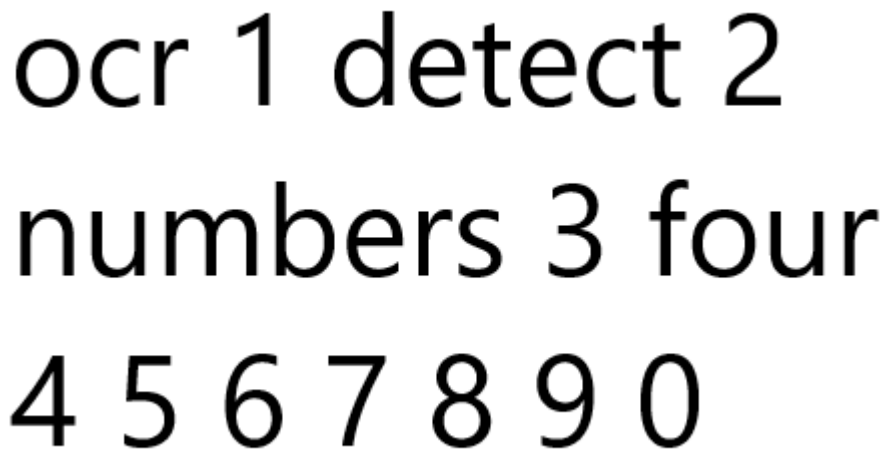


output:

hello mownit

ocrtest
muitiline

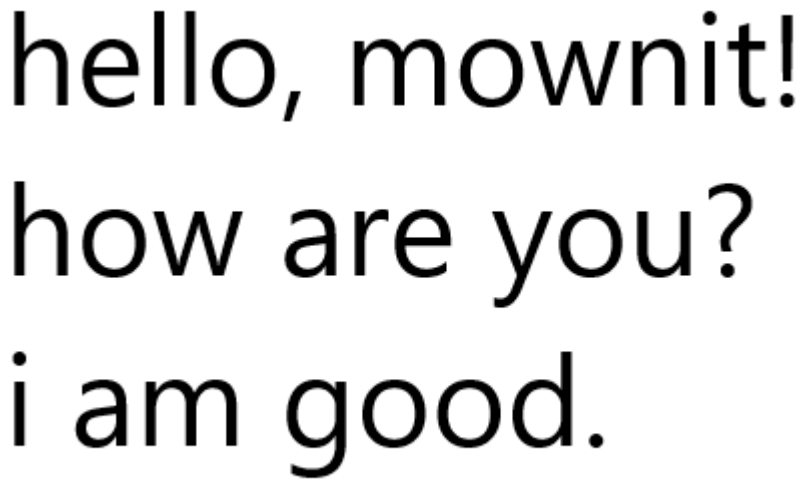
Jaro similarity:
0.9558892596712396



output:
ocr1 detect2
numbers3four
4567890

Jaro similarity:
0.9025756884854013

(źle spacje wykryło)



output:

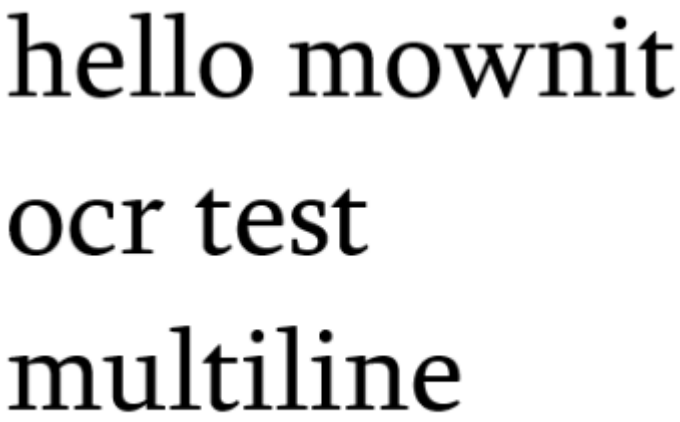
hello, mownit,!
howareyou.
iamgood.

jargo similarity:

0.398578811369509

Jargo similarity takie niskie, bo źle spacje wykrywa, a ta metryka traktuje to jako błąd, ale subiektywnie to jest dobra predykcja.

Dla czcionki Sylfaen

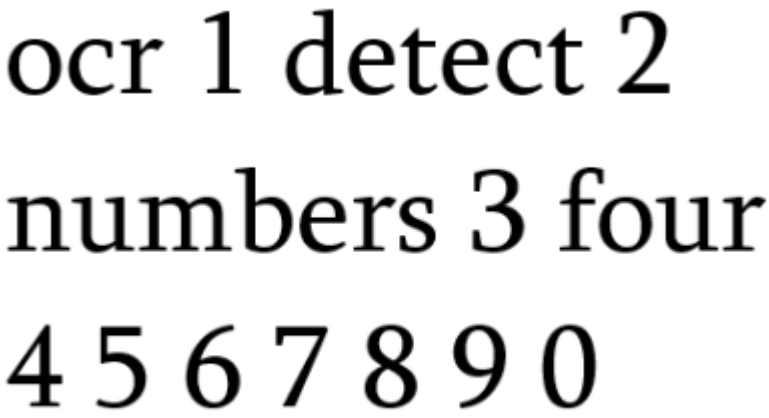


output:

heilo mcvnit
ocrtest
multiline

score:

0.8742731979291118



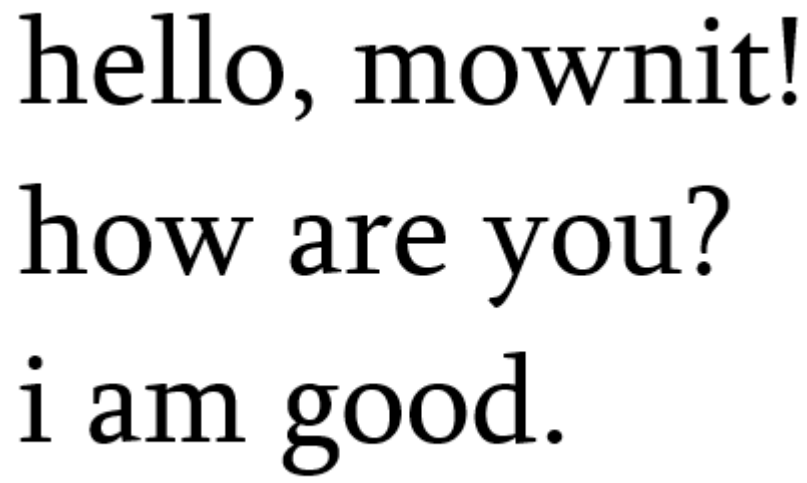
output:

ocr1detect2
numbers3four
4567890

score:

0.9046276720695325

test3:



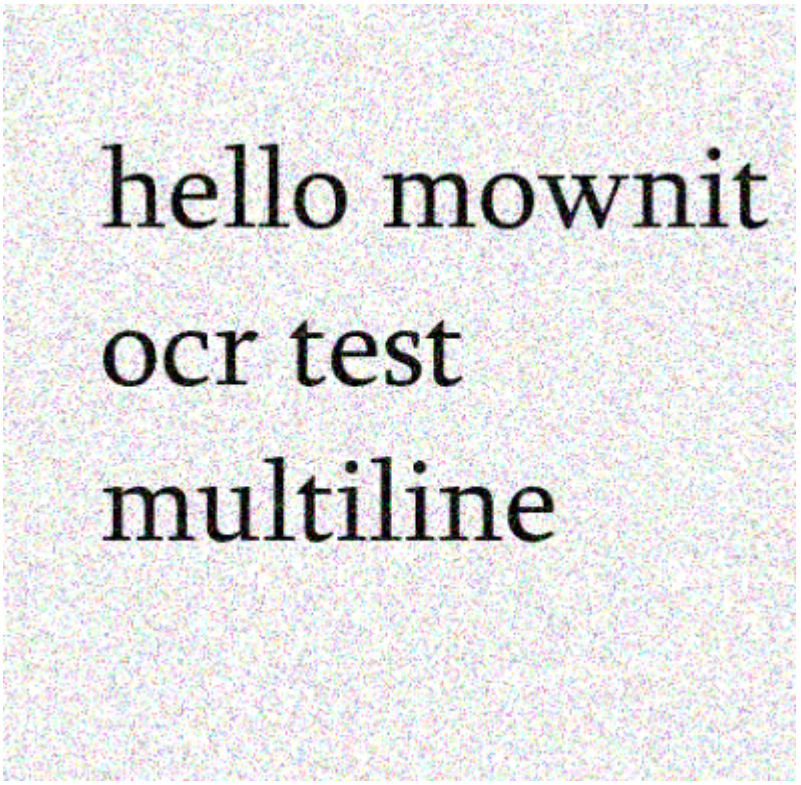
output:

heilo, mcvni,t!
hcv areyou.
iamgood.

score:

0.734038030771243

Test zakłóconego obrazka (40%):



output:

hel!o m vnit
ocrtest
multline

0.8726495726495727

score:

0.8726495726495727

Po zastosowaniu SVD score nie uległ zmianie, więc przetestowałem funkcję openCV do odszumiania obrazów:

```
cv2.fastNlMeansDenoising(img, None, 10, 7, 21),

ale tak samo, jak w przypadku SVD nie wpłynęło to na wynik.
```

Korekcja kątu nachylenia tekstu

- Próbowałem trzech różnych metod, ale żadna z nich nie działa dobrze. najlepiej działa `correct_skew(self, image, delta=10, limit=6),` gdzie delta to liczba iteracji a limit to maksymalny kąt wychylenia, ale rezultaty nie były zadowalające, bo czasem źle wyprostuje, dlatego zrezygnowałem z tej korekty.

Program zwraca liczbę wystąpień każdej litery

output:
g detected 1 times
z detected 0 times
x detected 1 times
b detected 0 times
…

Redukcja szumu na obrazie wejściowym

Do tego celu użyłem SVD

```
def get_img_approx(self, img, k):
    U, s, V = np.linalg.svd(img)
    S = np.zeros((img.shape[0], img.shape[1]))
    S[:img.shape[0], :img.shape[0]] = np.diag(s)
    n_component = k
    S = S[:, :n_component]
    VT = V[:n_component, :]
    A = U.dot(S.dot(VT))
    return A
```

oraz algorytmu openCV

```
cv2.fastNlMeansDenoising
```

Wnioski:

- można zastosować podejście opisane w sprawozdaniu do wykrywania prostych tekstów, jeśli zrobimy następujące założenia:

  znamy czcionkę
  wielkość liter,
  małe litery alfabetu angielskiego,
  brak odchylenia tekstu w pionie o jakiś kąt

  Jednakże trzeba mieć na uwadze, że w wielu przypadkach algorytm może się pomylić.

- Zastosowanie w praktyce opisanego podejścia jest trudne, bo trzeba być odpowiedzialnym za wiele hiperparametrów i decyzji, takich jak:

  lower_threshold, upper_threshold,
  kolejność wykrywania patternów,
  korekta kąta nachylenia,
  redukowanie szumów

- Alternatywnym podejściem byłoby zastosowanie dla każdego patternu własnego thresholdu, dla którego ma być zidentyfikowany.