

# Sprawozdanie - Poker Recognizer

Justyna Frączek 145307

Filip Ciesielski 145257

Michał Ciesielski 145325

7 grudnia 2021

## Spis treści

<b>1</b>	<b>Wstęp</b>	<b>2</b>
1.1	Efekt końcowy:	2
<b>2</b>	<b>Proces wykrywania kart</b>	<b>3</b>
2.1	Wykrywanie krawędzi kart	3
2.2	Wykrywanie kart na podstawie wykrytych krawędzi	6
2.3	Metoda identyfikacji rang i kolorów kart	9
<b>3</b>	<b>Opis algorytmu znajdującego najlepszą możliwą pokerową kombinację kart</b>	<b>10</b>
<b>4</b>	<b>Wykonane testy</b>	<b>10</b>
<b>5</b>	<b>Wnioski</b>	<b>13</b>

# 1 Wstęp

Celem projektu było zaimplementowanie metod rozpoznawania kart do gry na zdjęciu, oraz zaimplementowanie algorytmu znajdującego najlepszą możliwą pokerową kombinację kart z tych przedstawionych na zdjęciu. Kod źródłowy projektu oraz instrukcja uruchamiania znajduje się pod tym linkiem [https://github.com/filipciesielski7/Poker\\_Recognizer](https://github.com/filipciesielski7/Poker_Recognizer)

## 1.1 Efekt końcowy:



## 2 Proces wykrywania kart

W naszym programie w celu obróbki obrazu wykorzystaliśmy wbudowane funkcje, które oferuje biblioteka OpenCV.

### 2.1 Wykrywanie krawędzi kart

Na początku po wczytaniu zdjęcia, skupiamy się na wykrywaniu krawędzi kart. W tym celu posługując się metodą `preprocess_image(image)` przygotowujemy zdjęcie do wykrycia krawędzi - poddajemy zdjęcie:

- **procesowi wyszarzania** - używamy metody `cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)`



- **procesowi blurowania** - używamy metody `cv2.GaussianBlur(grayed, (5, 5), 0)`



- **procesowi progowania** - poziom progowania jest zmienny, dobierany w zależności od światła (pikseli odcieni szarości pobieranych z wyszczególnego zdjęcia, które przechowujemy w zmiennej `bkg_level`).



`BACKGROUND_THRESHOLD` to granica dla tła, która jest u nas stała i wynosi 60.

Poziom progowania uzyskujemy przez dodanie zmiennych `bkg_level`) i `BACKGROUND_THRESHOLD` otrzymując w ten sposób zmienną `thresh_level`.

W celu zwrócenia przetworzonego progowo obrazu wykorzystujemy funkcję `cv2.threshold(blurred, thresh_level, 255, cv2.THRESH_BINARY)`, gdzie `blurred` to obraz poddany wcześniejszemu procesowi blurowania.

Gdy zdjęcie jest już poddane odpowiedniej obróbce, posługujemy się metodą `find_cards(pre_process)`, w której znajdujemy krawędzie kart.

Krawędzie znajdujemy za pomocą metody `cv2.findContours(thresh_image, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)`.

Po zastosowaniu metody `cv2.findContours` przechodzimy przez znalezione przez nią kontury w celu zidentyfikowania, czy są to faktycznie kontury naszych kart.

Identyfikujemy kontur karty na podstawie:

- Mniejszego pola niż maksymalne dopuszczalne - `CARDS_MAX_AREA` (ma stałą wartość wynoszącą 120000) i większego pola niż minimalne dopuszczalne - `CARDS_MIN_AREA` (ma stałą wartość wynoszącą 25000). Pole uzyskujemy za pomocą metody `cv2.contourArea`.
- Braku przodków - sprawdzanie, czy karta nie zawiera się w innym konturze
- Ilości rogów - sprawdzanie, czy karta ma cztery rogi (używamy metody `cv2.approxPolyDP`)

Metoda `find_cards` zwraca uzyskane krawędzie kart w postaci tablicy.

## 2.2 Wykrywanie kart na podstawie wykrytych krawędzi

Dla każdego znalezionego konturu karty, wycinamy po kolej znalezioną kartę przy jednoczesnej zmianie perspektywy, aby zdjęcia oddawały widok karty z góry, nawet w sytuacji zrobienia zdjęcia pod kątem.

W celu znalezienia wysokości i szerokości karty posługujemy się metodą `cv2.boundingRect(contour)`.

Spłaszczamy kartę oraz zmieniamy jej wymiary do 200x300 za pomocą metody `flattener(image, points, w, h)`. Metoda ta wykrywa również orientację karty i sprowadza obrazek do perspektywy top-down.

Przykładowa wycięta karta ze zdjęcia:



Następnie wycinamy róg karty z rangą oraz kolorem, jednocześnie czterokrotne go przybliżając:

```
Qcorner = qCard.warp[0:CARDS_HEIGHT, 0:CARDS_WIDTH]
```

```
Qcorner_zoom = cv2.resize(Qcorner, (0, 0), fx=4, fy=4)
```

CARDS\_HEIGHT oraz CARDS\_WIDTH to stałe, które wynoszą odpowiednio 84 i 32.

Wycięty róg karty:



Następnie znajdujemy prostokąt ograniczający dla największego konturu w górnej i dolnej części przybliżonego wcześniej wyciętego rogu w celu zidentyfikowania rangi oraz koloru karty, po wcześniejszym zastosowaniu odpowiedniego poziomu progowania.

Tak wygląda róg karty podzielony na dwie części:



## 2.3 Metoda identyfikacji rang i kolorów kart

W celu identyfikacji rang i kolorów naszych kart posługujemy się gotowym zbiorem zdjęć rang i kolorów, do którego przymierzamy wycięte rangi i kolory z rogów naszych kart.

Wzorcowe zdjęcia rang znajdują w tablicy `train_ranks`, a zdjęcia kolorów w tablicy `train_colors`.

Porównanie ma miejsce w metodzie `match_card(cards[k], train_ranks, train_colors)`.

Do porównania naszych obrazów z wzorcami posługujemy się metodą `cv2.absdiff()`.

Sprawdzamy tym samym, ile białych pikseli otrzymamy poprzez nałożenie obrazów na siebie. Im mniej białych pikseli, tym nasz wycięty obraz rangi lub koloru jest bardziej zbliżony do wzorcowego obrazu, z którym jest aktualnie porównywany.

Tym sposobem przypisanie naszej karty do odpowiedniej rangi i koloru nastąpi dla tego zdjęcia wzorcowego, z którym otrzymamy najmniejszą ilość białych pikseli - przechowywana jest ona w zmiennych `rank_diff` dla rangi i `color_diff` dla koloru.

Jeśli są one tymczasowo najlepsze, czyli dają najmniejszy wynik, to jest on przechowywany w zmiennych odpowiednio `best_rank_match_diff` i `best_color_match_diff`.

Od razu również wtedy do zmiennych `best_rank_name` i `best_color_name` przypisujemy odpowiednią nazwę rangi i koloru.

Funkcja `match_card` zwróci nam te cztery zmienne dla każdej naszej karty.

### **3 Opis algorytmu znajdującego najlepszą możliwą pokerową kombinację kart**

Dodatkowo oprócz algorytmu rozpoznawania kart, zaimplementowaliśmy algorytm znajdujący najlepszą możliwą pokerową kombinację pięciu kart z siedmiu przedstawionych na zdjęciu (pięć ze stołu i dwie z ręki gracza).

Algorytm najpierw wyznacza wszystkie możliwe kombinacje pięciu kart, a następnie dla każdej z nich sprawdza po kolej, czy spełnia ona warunki poszczególnych pokerowych kombinacji, rozpoczynając od najmocniejszej (poker królewski), a kończąc na najsłabszej (najwyższa karta).

W przypadku gdy dana kombinacja zostaje zidentyfikowana, zostaje przypisana do danej piątki kart.

Ostatnim etapem jest wybór takiej piątki, dla której przypisana kombinacja jest najmocniejsza, i wskazanie jej na obrazie wynikowym poprzez zaznaczenie konturów kart na zielono. Kontury pozostałych kart zaznaczamy na czerwono.

### **4 Wykonane testy**

Do przeprowadzenia eksperymentów wykorzystaliśmy 70 własno robionych zdjęć układów kart.

Obrazy podzieliliśmy na dwie grupy. Połowa naszych zdjęć była robiona w tak zwanych dobrych warunkach, a połowa w niekorzystnych.

Do niekorzystnych warunków zaliczyliśmy następujące scenariusze:

- Zdjęcie robione pod ostrym kątem

Nie odczytano poprawnie wszystkich kart



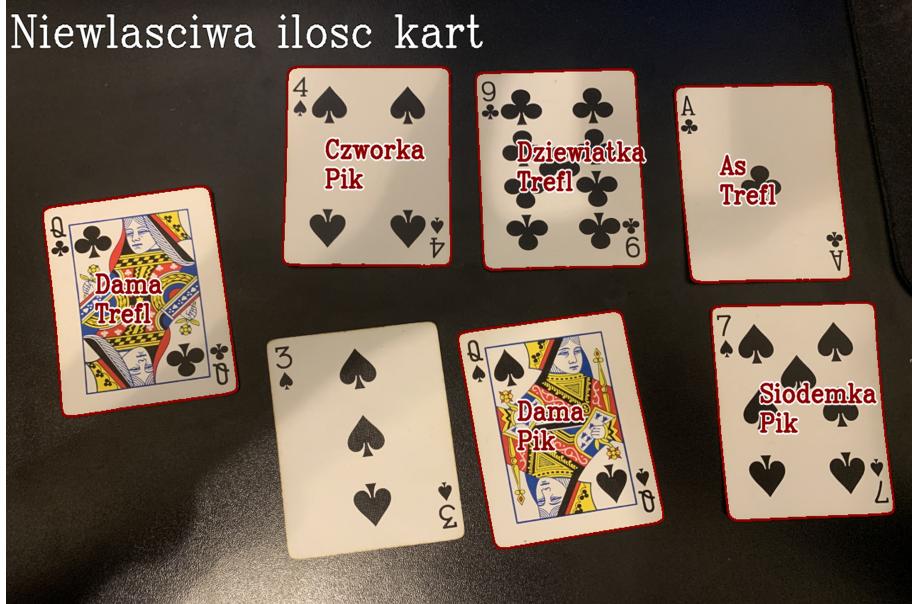
- Zdjęcie robione na bardzo jasnym tle

Nie znaleziono konturów kart

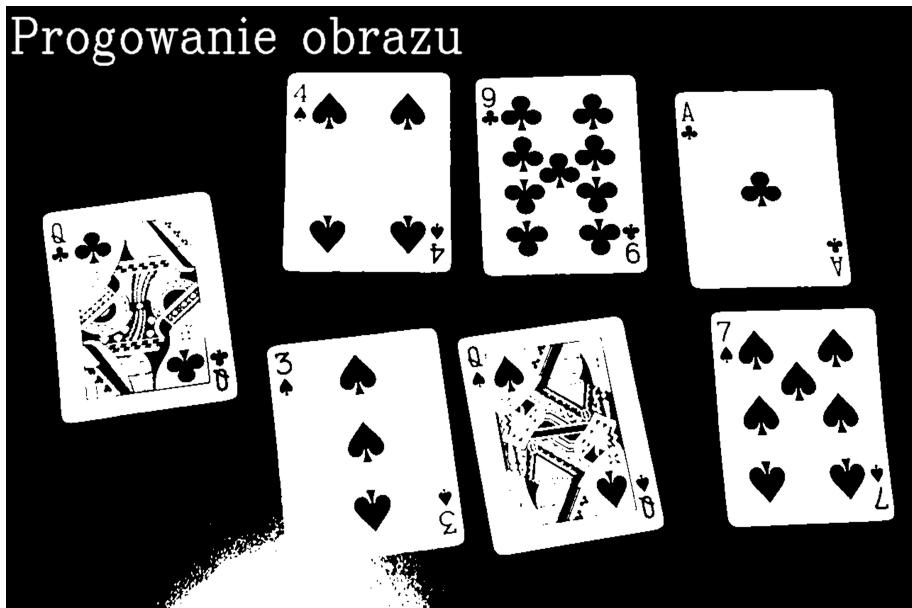


- Zdjęcie robione w bardzo niekorzystnym świetle (np. spore przeświecenie).

Niewlasciwa ilosc kart



Progowanie obrazu



Każdy scenariusz został uwzględniony w przeprowadzonych przez nas eksperimentach.

Dla wykonanych przez nas testów w dobrych warunkach, tylko jedno z 35 zdjęć nie zostało w pełni poprawnie odczytane. W przypadku niekorzystnych warunków (przedstawionych powyżej) takich niepowodzeń było już o wiele więcej, a skuteczność algorytmu wyniosła około 51%.

## 5 Wnioski

Po przeprowadzonych eksperymentach oraz ogólnego naszego wrażenia dotyczącego działania aplikacji, okazuje się, że metody machine learningu nie były konieczne do zbudowania solidnego rozwiązania. Oczywiście zostawia to pewną furtkę do rozwoju naszego algorytmu poprzez dobór odpowiednich parametrów na podstawie wytrenowania większego zbioru danych. Natomiast przy pomocy przeciętnego aparatu oraz kart ułożonych odpowiednio na ciemnym blacie (np. w kasynie) aplikacja spełnia oczekiwane założenia funkcjonalności.