KCK - Card Detector

Jakub Wróbel - 145188
Przemysław Woźniak - 145423

23 listopada 2021

1 Temat i krótki opis rozwiązania problemu

1.1 Temat

Tematem naszego projektu jest program rozpoznający karty do gry. Zdjęcia do analizy wykonaliśmy sami używając do tego standardowej 52 - kartowej talii.

1.2 Opis rozwiązania problemu

Najpierw pobieramy zdjęcie i poddajemy je wstępnej obróbce. Następnie na podstawie odnalezionych konturów kartę wycinamy ze zdjęcia i ją obracamy. Z karty wycinamy jej symbol i kolor. W kolejnym kroku porównujemy wycięte symbole i kolory z przygotowanymi wcześniej szablonami i wybieramy ten, do którego karta najlepiej pasuje.

2 Program krok po kroku

2.1 Wczytanie i wstępna obróbka zdjęcia

Zdjęcia wczytujemy funkcją cv2.imread zarówno normalnie jak i w skali szarości. Wstępna obróbka zdjęcia polega na zastosowaniu rozmycia gaussowskiego i thresholdingu, co pozwala wyodrębnić kartę od tła.

Rysunek 1: Zdjęcie oryginalne i po obróbce

2.2 Znalezienie konturów karty

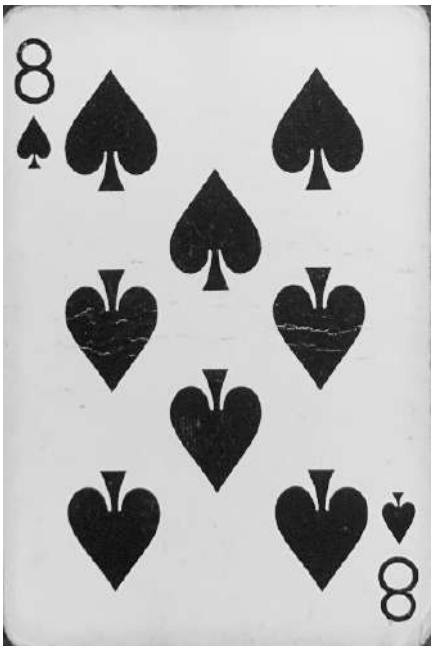
Rozpoczynamy znajdowanie kart od funkcji cv2.findContours, która zwraca nam kontury pojedynczych obiektów oraz ich hierarchię. Oczywiście w tych konturach są nie tylko nasze karty, ale również obiekty, których nie udało się usunąć thresholdingiem i rozmyciem. Poszukujemy więc kart trzema kryteriami:

- Kontur ma mniejszą powierzchnię niż maksymalna powierzchnia karty i większą niż minimalna powierzchnia karty (wyznaczyliśmy je wcześniej analizując wykonane zdjęcia)
- Kontur nie ma rodziców (nie jest w środku żadnego innego konturu)
- Kontur ma cztery rogi

Rysunek 2: Zdjęcie oryginalne z nałożonym znalezionym konturem

2.3 Dalsza obróbka zdjęcia

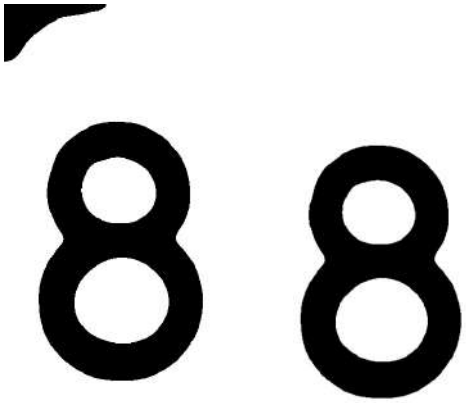
Samo znalezienie konturu jednak nie wystarczało. Przez różne perspektywy zdjęć i pozycję kart, trzeba było dostosować wycięty kontur. Przybliżamy kształt karty za pomocą funkcji cv2.approxPolyDP. Funkcją cv2.boundingRect odczytujemy wymiary naszego prostokąta (karty). W odpowiednich miejscach rozszerzamy wycięty fragment, tak, by obniżyć ryzyko wycięcia niepełnej karty ze zdjęcia. Następnym krokiem jest 'obrócenie' karty. Wykrywamy, w którą stronę jest obrócona karta poprzez stosunek jej wysokości do szerokości i w odpowiedni sposób dostosowujemy rogi. Ostatnim krokiem jest dostosowanie perspektywy. Robimy to funkcją cv2.getPerspectiveTransform, która na podstawie 4 wierzchołków oblicza macierz transformacji. Utworzoną macierz przekazujemy do funkcji cv2.warpPerspective, która transformuje wycięty przez nas fragment.



Rysunek 3: Zdjęcie po dalszej obróbce

2.4 Wycięcie symbolu i koloru karty

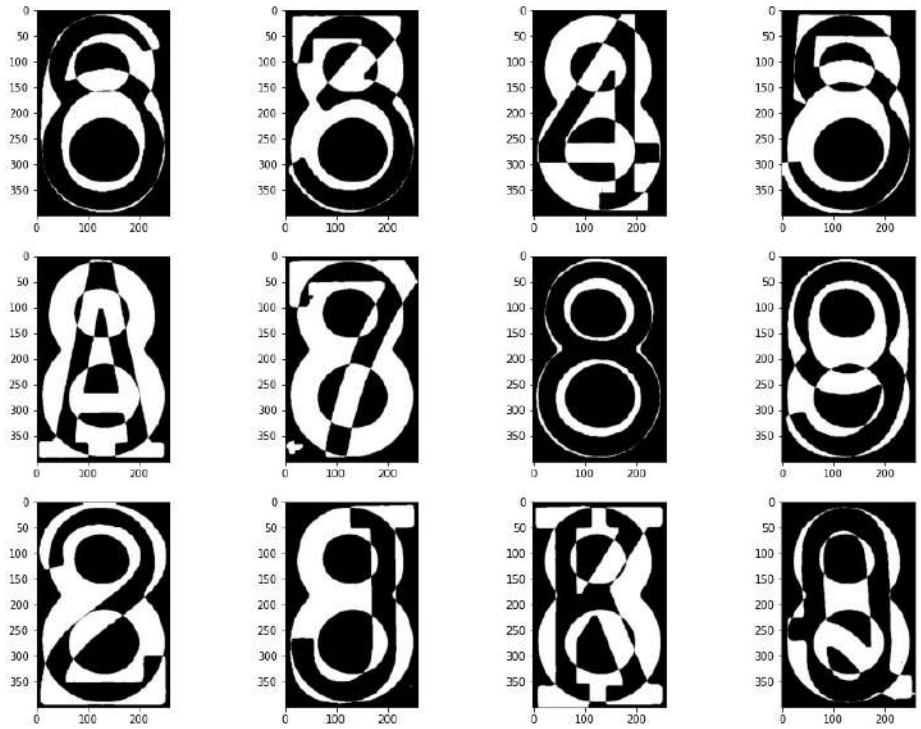
Teraz musimy z wyciętej karty wyodrębnić interesujący nas fragment. Lewy górny róg karty posiada potrzebne nam informacje. Dostosowaliśmy wcześniej rozdzielczość, więc możemy 'na sztywno' podać rozmiar fragmentu zawierającego symbol karty i fragmentu zawierającego jej kolor. Z tych fragmentów wycinamy pozostałości po obróbce (niedoskonałości z 'dalszej obróbki') i tak otrzymujemy gotowe symbole i kolory.



Rysunek 4: Wycięty fragment i fragment po wycięciu niedoskonałości

2.5 Porównanie wyciętych fragmentów z szablonami

Przygotowane wcześniej szablony odczytujemy i stosujemy na nich thresholding. Następnie fragmenty i szablony porównujemy binarnie. Na końcu wystarczy policzyć, które zdjęcia najlepiej się na siebie nałożyły (mają najmniej białego, pustego miejsca) i możemy wywnioskować jaka karta widoczna była na oryginalnym zdjęciu.



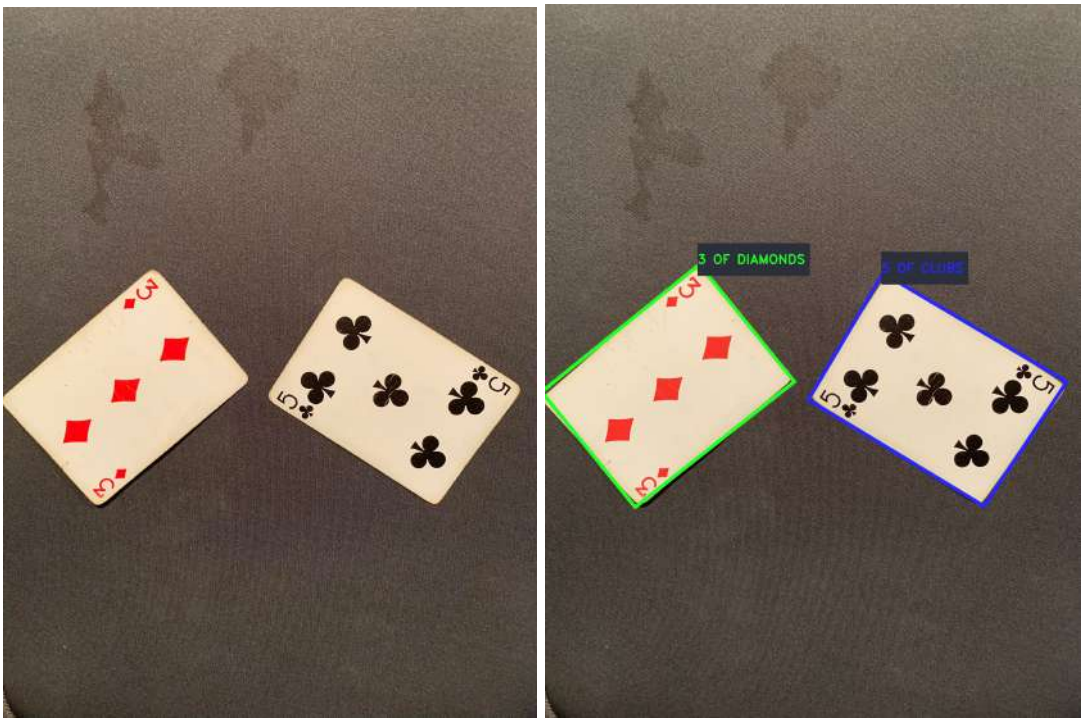
Rysunek 5: Porównywanie fragmentów z szablonami

Rysunek 6: Ostateczny werdykt

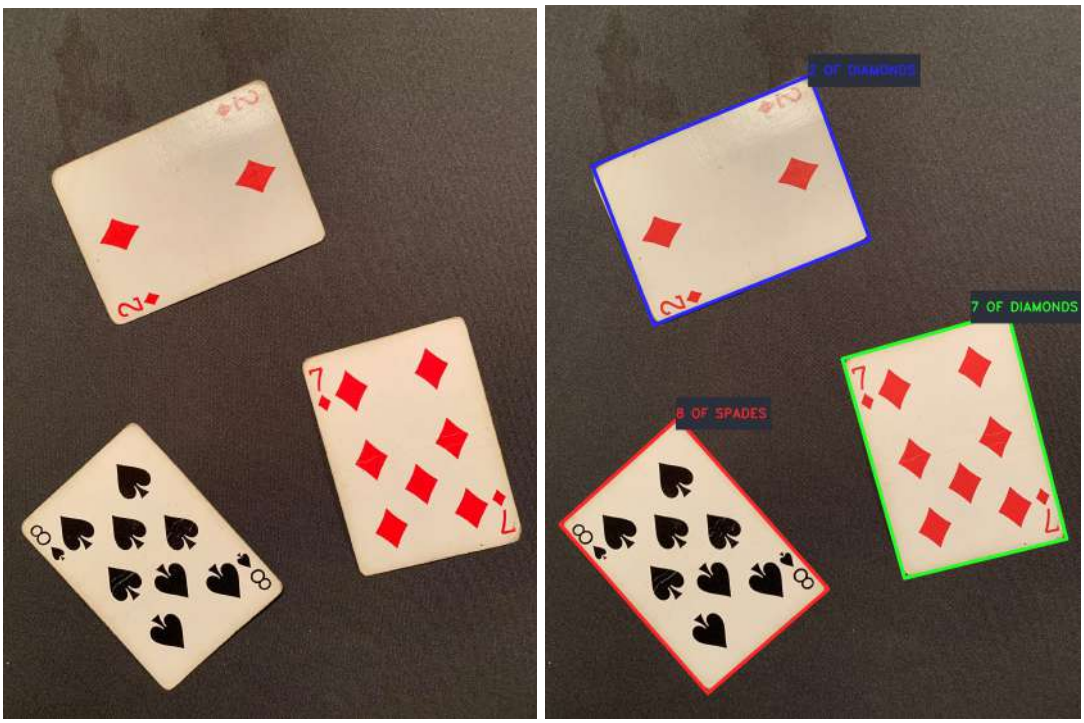
3 Wyniki

3.1 Zdjęcia łatwe

Rysunek 7: INPUT - OUTPUT



Rysunek 8: INPUT - OUTPUT



Rysunek 9: INPUT - OUTPUT

Rysunek 10: INPUT - OUTPUT

Rysunek 11: INPUT - OUTPUT

Rysunek 12: INPUT - OUTPUT

Rysunek 13: INPUT - OUTPUT

Rysunek 14: INPUT - OUTPUT

Rysunek 15: INPUT - OUTPUT

Rysunek 16: INPUT - OUTPUT

3.2 Zdjęcia średnie

Rysunek 17: INPUT - OUTPUT

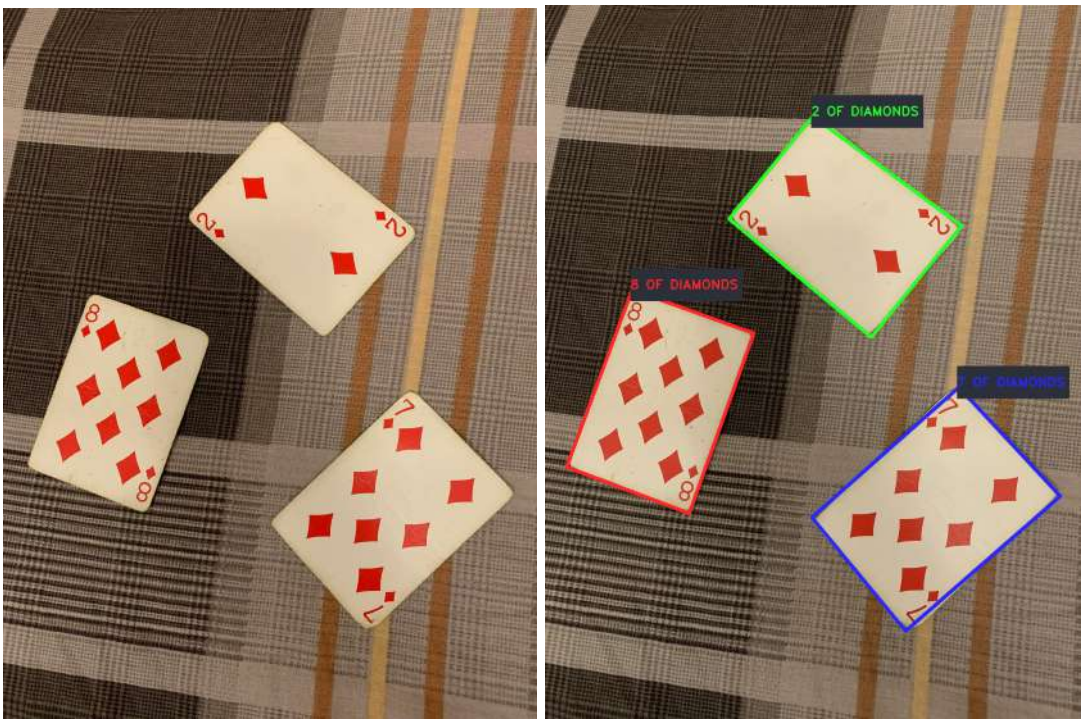
Rysunek 18: INPUT - OUTPUT

Rysunek 19: INPUT - OUTPUT

Rysunek 20: INPUT - OUTPUT

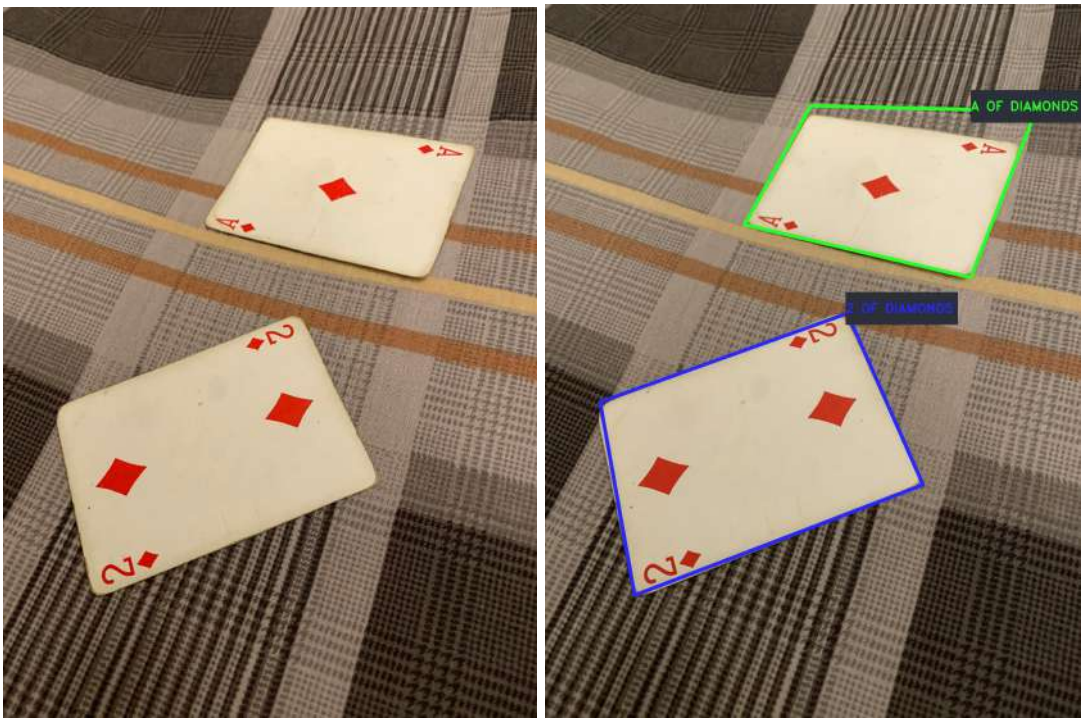
Rysunek 21: INPUT - OUTPUT

Rysunek 22: INPUT - OUTPUT



Rysunek 23: INPUT - OUTPUT

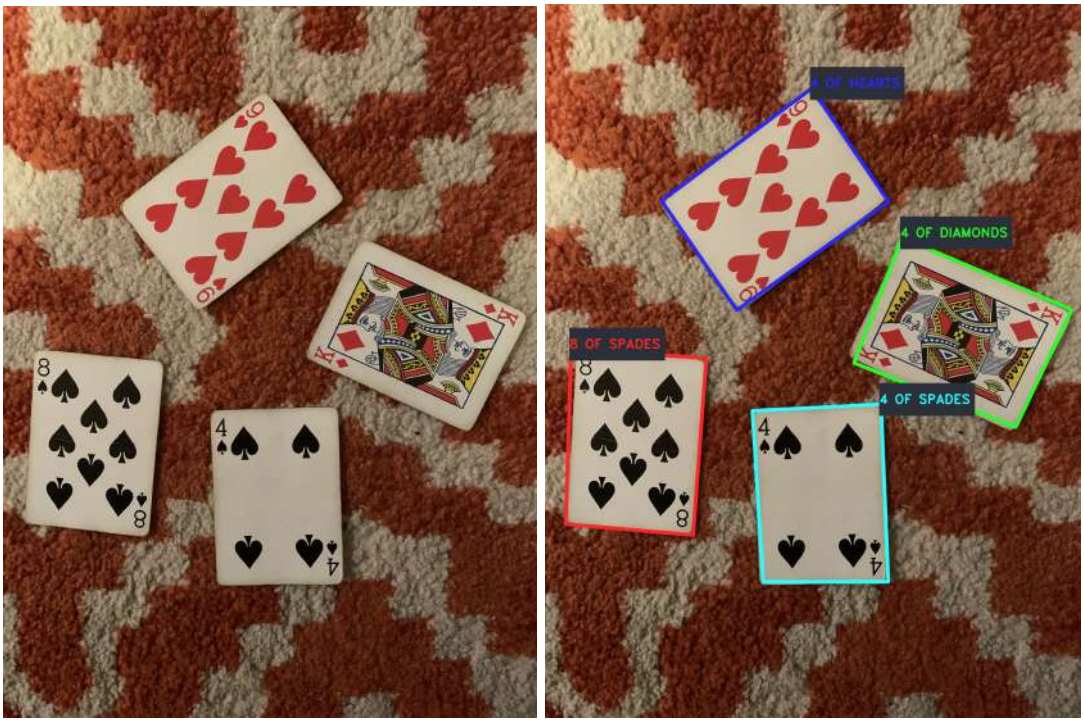
Rysunek 24: INPUT - OUTPUT



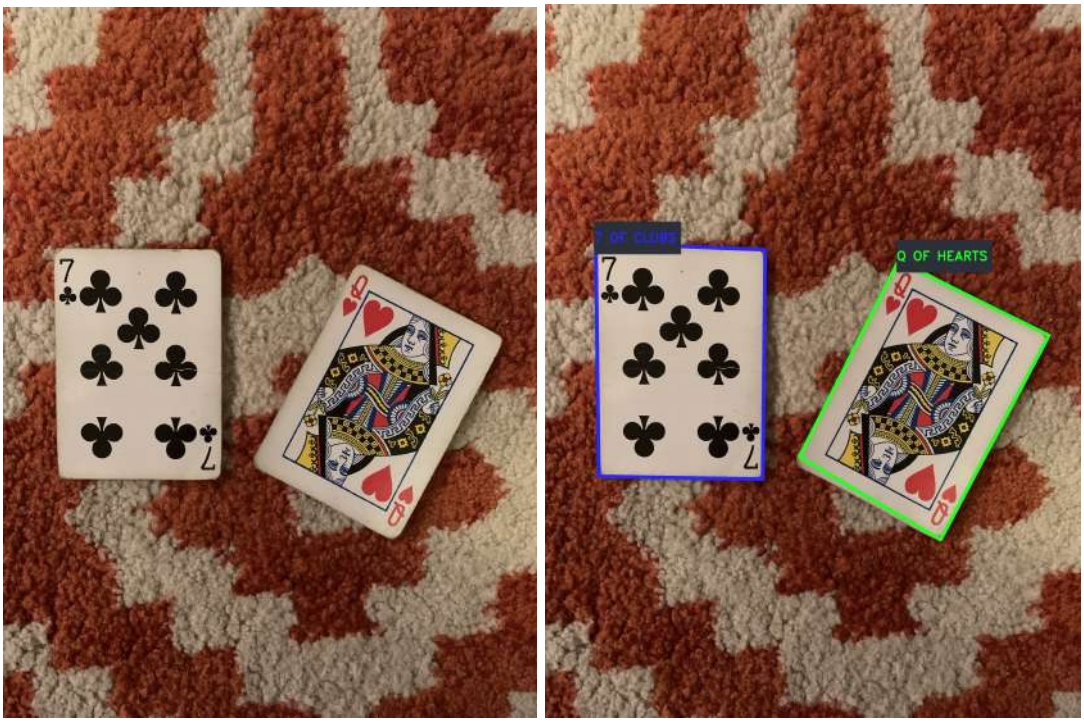
Rysunek 25: INPUT - OUTPUT

Rysunek 26: INPUT - OUTPUT

3.3 Zdjęcia trudne



Rysunek 27: INPUT - OUTPUT

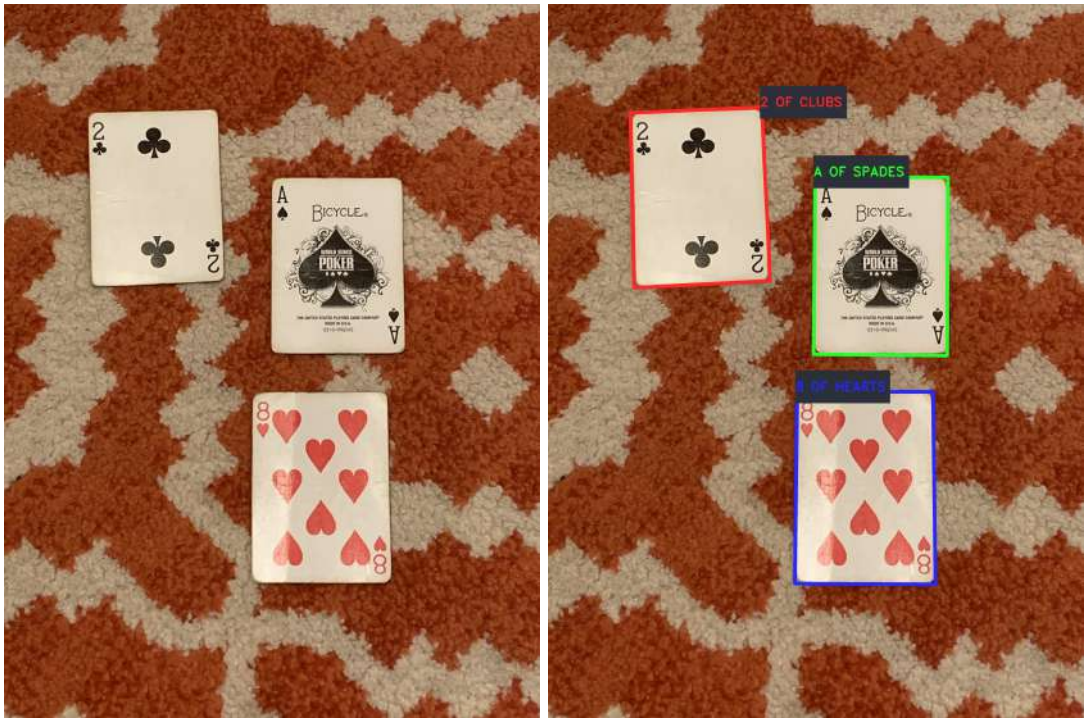


Rysunek 28: INPUT - OUTPUT

Rysunek 29: INPUT - OUTPUT

Rysunek 30: INPUT - OUTPUT

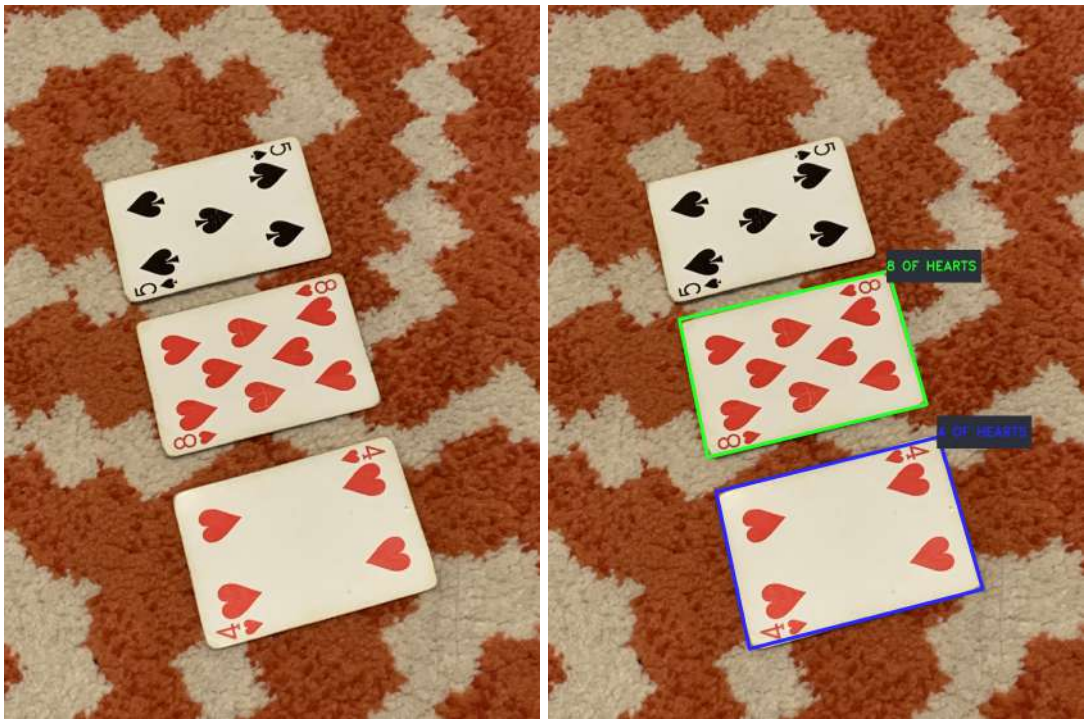
Rysunek 31: INPUT - OUTPUT



Rysunek 32: INPUT - OUTPUT

Rysunek 33: INPUT - OUTPUT

Rysunek 34: INPUT - OUTPUT



Rysunek 35: INPUT - OUTPUT

Rysunek 36: INPUT - OUTPUT

4 Podsumowanie wyników

W przypadku zdjęć, z którymi program sobie nie poradził przyczyn mogło być kilka, ale były to najprawdopodobniej złe oświetlenie i niesprzyjające tło. Biała karta leżąca na białej części dywanu często nie została wykrywana. Tak samo jeżeli część kart była oświetlona inaczej niż inna, to ustalone przez nas progowanie nie dawało sobie rady z częścią z nich. Solucją tego problemu mogłoby być dynamiczne ustalanie parametrów progowania, tak by dla każdej kolejnej karty działało ono zgoła inaczej.