Komunikacja człowiek-komputer – Sprawozdanie
Rozpoznawanie stanu gry Dobble

Dariusz Max Adamski 136674, Michalina Kmieciak 136736

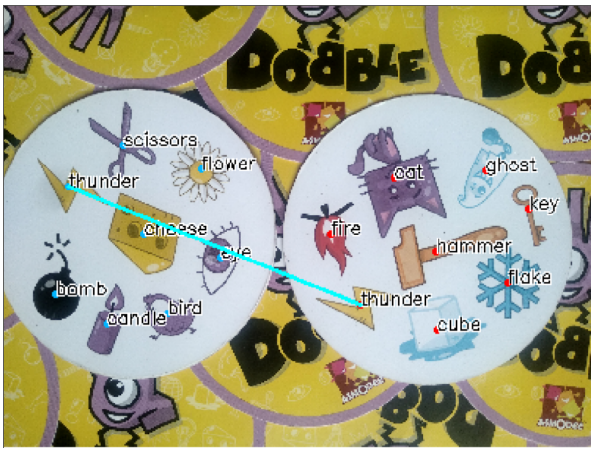
{dariusz.adamski,michalina.kmieciak}@student.put.poznan.pl

Data oddania:

Wstęp

W danym sprawozdaniu przedstawiona będzie metoda rozpoznawania obrazu na przykładzie gry Dobble. Dobble to gra rodzinna ćwicząca spostrzegawczość. Na każdej karcie w kształcie koła umieszczonych jest osiem różnych obrazków, a każde dwie karty mają jeden wspólny symbol. Celem gry jest jak najszybsze nazwanie tego symbolu.

1 Opis problemu



Rysunek 1: Przykładowe rozwiązanie problemu

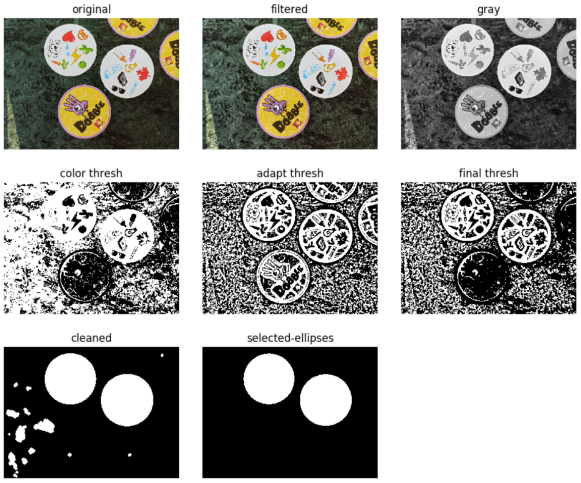
Problem polega wykryciu par obrazków między każdymi dwoma kartami.

2 Opis rozwiązania

2.1 Przetwarzanie wstępne

Na początku wczytujemy zdjęcie i proporcjonalnie zmniejszamy jego rozmiar, tak aby dłuższy bok miał długość 1000 px. Później używamy filtru bilateralnego, by zredukować szum w zdjęciu, przy zachowaniu krawędzi. Poprawiamy też kontrast używając adaptacyjnego wyrównywania histogramu CLAHE w przestrzeni kolorów Lab. Po konwersji zdjęcia na odcienie szarości, wykonujemy parę iteracji erozji i dialacji.

2.2 Segmentacja kart



Rysunek 2: Kolejne kroki przetwarzania wstępnego i segmentacji kart

Do wykrywania kart stosujemy podejście łączące zalety tresholdigu kolorów oraz Gaussowskiego adaptywnego thresholdingu.

Najpierw tworzymy maskę *wykluczającą* piksele, które na pewno nie należą do karty. W przestrzeni HSV odrzucamy bardzo nasycone i jasne kolory, bardzo ciemne kolory oraz pewne żółtawe i niebieskawe kolory, ponieważ odbierana przez oko jasność koloru w przestrzeni HSV niestety zależy też od parametru H. Ten problem można rozwiązać użyciem przestrzeni Lab, ale zyski byłyby znikome. Taką maskę odwracamy i w efekcie uzyskujemy maskę miejsc, które *mogą* być kartą.

Równolegle stosujemy Gaussowski adaptacyjny thresholding na wstępnie przetworzonym zdjęciu. Wynik mnożymy z maską otrzymaną w poprzednim kroku i wykonujemy na niej parę iteracji erozji i dialacji.

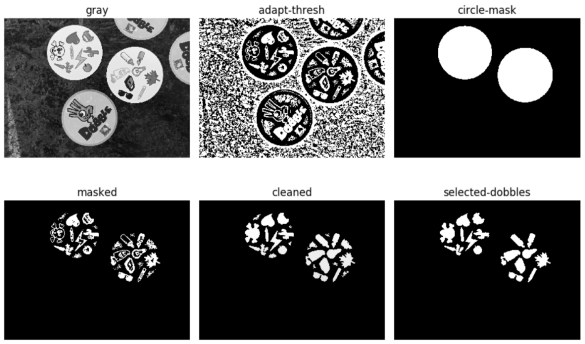
Na tej masce wykonujemy morfologiczną operację zamykania, następnie ponownie używając erozji i dialacji.

Następnie znajdujemy kontury funkcją FINDCONTOURS. Dla każdego konturu obliczamy ich mo-

menty, momenty Hu, centroid, pole i krągłość. Jeśli kontur nie jest wystarczająco krągły lub jest zbyt mały, to go ignorujemy. W słowniku zapisujemy dane wykrytych kart.

Do każdego pozostałego konturu dopasowujemy elipsę i rysujemy je na masce, która będzie użyta w wykrywaniu obrazków.

2.3 Segmentacja obrazków



Rysunek 3: Kolejne kroki segmentacji obrazków z wykrytych kart

Segmentacja obrazów przebiega podobnie do segmentacji kart.

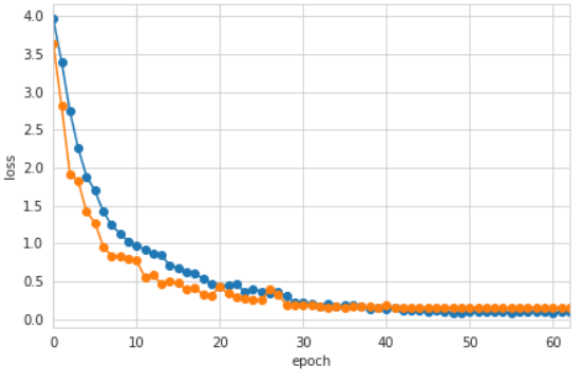
Na początku stosujemy Gaussowski adaptacyjny tresholding. Następnie używamy maski elips utworzonej w poprzednim kroku. Wykonujemy kilka iteracji erozji i dialacji oraz operację domknięcia. Usuwamy też małe obiekty.

Następnym krokiem jest wykrywanie kontur i analogicznie do wykrywania kart, obliczenie momentów, zawartości kolorystycznej kontur etc. Jeśli kontur zawiera prawie same białe/szare piksele, to go ignorujemy. W słowniku zapisujemy dane wykrytych obrazków. Zapisujemy też informację, na której karcie leży kontur.

2.4 Klasyfikacja obrazków

Aby móc klasyfikować obrazki, z uwagi na bardzo dużą ilość klas i niemożliwość łatwego odróżnienia ich klasycznymi metodami, zdecydowaliśmy się na klasyfikację przy pomocy uczenia maszynowego, a dokładniej konwolucyjnej sieci neuronowej, której dokładną architekturę zamieściliśmy na obrazku. Jako wejście przekazujemy obrazki o wymiarach 40x40x3, a na wyjściu dostajemy jedną z 59 klas.

Jako funkcję straty przyjęliśmy kategoryczną entropię krzyżową, a optymalizowaliśmy ją optymalizatorem Adam, z połowioną prędkością uczenia, przy malejących przyrostach. Przebieg uczenia przedstawiono na rysunku 4. Na pomarańczowo zaznaczona jest wartość funkcji straty dla zbioru testowego, a na niebiesko dla zbioru treningowego.



Rysunek 4: Wartość funkcji straty w zbiorze treningowym i testowym, podczas uczenia

2.4.1 Architektura klasyfikatora

1. Konwolucja 128 filtrów 3x3 (2 warstwy)
2. Max pooling 2x2 + Dropout 0.2
3. Konwolucja 256 filtrów 3x3 (2 warstwy)
4. Max pooling 2x2 + Dropout 0.2
5. Dense 256 + Dropout 0.2
6. Dense 128 + Dropout 0.2
7. Dense 59 (Softmax)

2.5 Użyte narzędzia

W programie do przetwarzania obrazów użyliśmy bibliotek OpenCV, Scikit-image oraz NumPy. Do komponentu uczącego się, użyliśmy Keras. Przy początkowych eksperymentach używaliśmy też algorytmu UMAP do redukcji wymiarowości danych i HDBSCAN do grupowania obrazków. Do wizualizacji użyliśmy biblioteki Matplotlib.

3 Metodologia

3.1 Zbiór danych

Do eksperymentu użyliśmy klasycznej wersji gry zawierającej zestaw 55 kart z 59 rodzajami obrazków oraz stworzonej przez nas bazy zdjęć z różnymi parametrami.

Postanowiliśmy podzielić ją na trzy poziomy trudności: łatwy, średni i trudny.

W kategorii łatwej znajdują się zdjęcia z jednorodnym oświetleniem, robione pod kątem 90°.

W kategorii średniej karty są położone na różnorodnych powierzniach (marmur, biała podłoga), oświetlenie i kontrast są gorsze.

W kategorii trudnej na karty dodatkowo mogą padać cienie, a w kadrze pojawiają się inne obiekty.

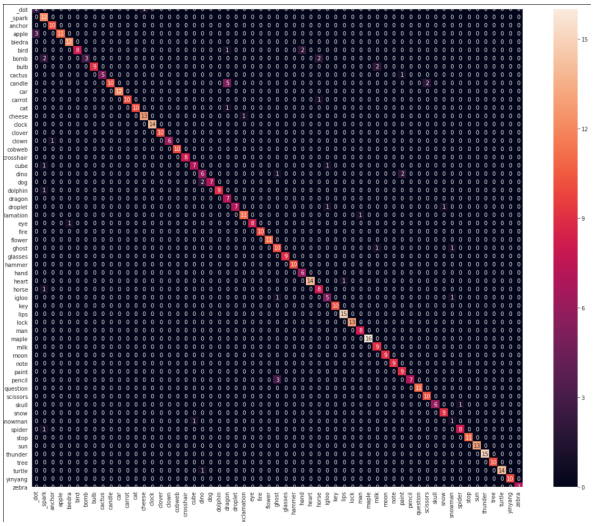
Karty mogą być różnej wielkości, być położone na bardzo trudnej powierzchni, lub być częściowo przysłonięte.

W zależności od kategorii na zdjęciu mogą być więcej niż dwie karty.

3.2 Pomiary

Do wytrenowania klasyfikatora obrazków użyliśmy 70% wydzielonych obrazków (zbiór treningowy). Dokładność była sprawdzana, na pozostałych 30% obrazków, których klasyfikator nie widział podczas trenowania (zbiór testowy).

4 Wyniki



Rysunek 5: Macierz pomyłek na zbiorze testowym

Według nas wyniki są dobre. Przy ocenie rozwiązania prosimy wziąć pod uwagę, że dla klasyfikacji przy tak małej ilości przykładów, nasz model działa dobrze. Model najczęściej myli obrazki koloru fioletowego, na przykład smok i świeczka, albo ręka i ptak. Pomyłki w zbiorze testowym są widoczne na zamieszczonej macierzy pomyłek.

Większość pomyłek programu wynika z tego, że niektóre obrazki są podobne kolorem i kształtem. Myślimy, że problem można rozwiązać modyfikacjami architektury sieci. Niestety, uczenie takich modeli zajmuje dużo czasu i nie byliśmy w stanie przetestować wielu kombinacji.

Podsumowując, nasz model uzyskał 92% dokładności (accuracy) na zbiorze testowym.

5 Możliwe ulepszenia

Program nie działa dobrze lub wcale, gdy karty nachodzą na siebie. Według nas nie jest to duży problem, ponieważ nie jest to poprawny stan gry i radzenie sobie z takimi przypadkami, zdecydowanie wykracza poza założenia projektu.

Nasza metoda ma też problem z sytuacjami, w których bardzo trudno odróżnić karty od tła, ale przy wybranej sekwencji przetwarzania nie można dużo z tym zrobić.