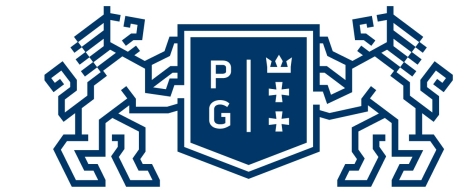


GDAŃSK UNIVERSITY
OF TECHNOLOGY

Sztuczna Inteligencja

Real-time card recognition

sprawozdanie z projektu zaliczeniowego

Aleksander Błaszkiewicz
Dominik Piotrowicz
Maciej Adryan

25 maja 2021

Streszczenie

Projekt ma na celu rozpoznawać karty pokazywane do kamery w czasie rzeczywistym.
Ze względu na znaczne ograniczenia sprzętowe ograniczyliśmy się do kart z zestawu **{9,10,J,Q,K,A}** we wszystkich **4** kolorach, tj. **{serca, pik, trefl i diament}**.

Dodatkowo zaimplementowany został *Poker Helper*, który w przypadku wykrycia **5 kart** (prawidłowej ręki gracza) informuje go o możliwych do uzyskania przez niego ułożeń.

Projekt został zrealizowany w opraciu o:

- Bibliotekę **OpenCV** do rozpoznawania i dodatkowej augmentacji poszczególnych klatek obrazu
- Bibliotekę **imutils** do przechwytywania obrazu ze streama kamerki internetowej.
- Sieć neuronową **Darknet** na której przeprowadzane było szkolenie
- framework **Yolo** w wersji 3 służące do oznaczania (labelowania danych służących do szkolenia sieci).
- platformę **Google Colab** - umożliwiające współpracę przy trenowaniu sieci oraz dostarczające platformę do trenowania (w ograniczonym czasem zakresie)

Spis treści

Rozdział 1

Pozyskanie, rekoloryzacja, augmentacja datasetu

1.1 Unifikacja

Zdjęcia wykonane zostały smartphonem, warunki światła były jednolite dzięki zastosowaniu dodatkowego źródła światła, podobieństwo i dokładność zdjęć zapewniła konstrukcja **DIY** w postaci przerobionej lampki biurkowej **FORSÅ**.

Rysunek 1.1: konstrukcja do fotografowania datasetu

Pozyskane za pomocą aplikacji do skanowania dokumentów zdjęcia zostały wyłuskane z formatu **PDF** za pomocą komendy:

```
pdfimages -png ALL.pdf ./extreacted/
```

Następnie zostały przeskalowane do jednakowych rozmiarów:

```
for FILE in *; do convert $FILE -resize 108x150 ./resized/$FILE; done
```

Oraz zrekoloryzowane (z powodu doboru złej barwy światła przy wykonywaniu zdjęć) za pomocą *batch edit* dostępnego w narzędziu **Adobe Photoshop**.

1.2 Augmentacja

Parametry programu:

- Liczba iteracji,

Parametry generatora:

- Ścieżka do tła - tło ma wymiary 500x500. Metodą prób i błędów doszliśmy do tego, że taki wymiar sprawdzał się najlepiej. Wyższe rozdzielczości znacząco podnosiły czas treningu i dawały jeszcze gorsze rezultaty. Wydaje się to być spowodowane tym, że natywna rozdzielczość yolo_v3 to 416x416
- Ścieżki wszystkich sfotografowanych kart - karty mają wymiary 150x107 i przeszły wstępną obróbkę, jaką było wyrównanie kolorów i konstrastu, przez co wszystkie mają bardzo podobne warunki oświetleniowe.

Zasada działania generatora:

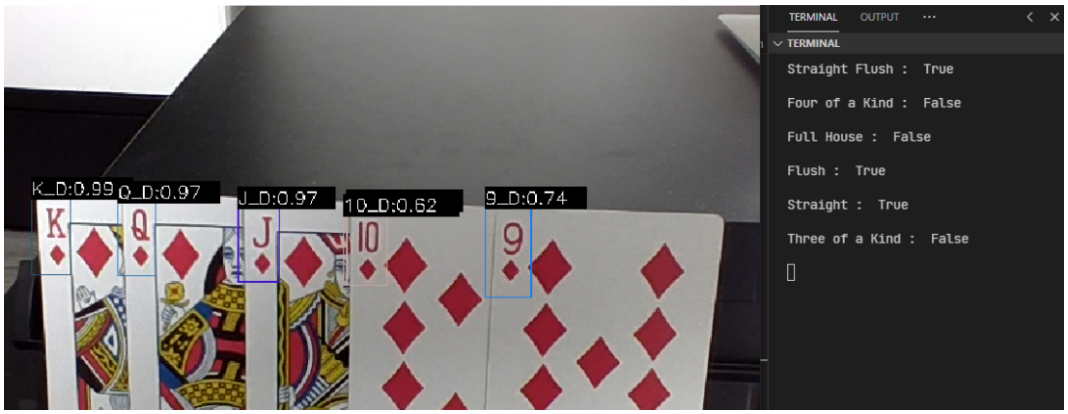
1. Załaduj tło podane jako argument
2. Te kroki powtórz 30 razy:
   - Załaduj jedno losowe zdjęcie karty
   - Określ, gdzie znajdują się bounding boxy - wyznaczone jest to procentowo, więc jeżeli zostanie załadowana karta o innych wymiarach niż 150x107, to wciąż to zadziała
   - Olabeluj bounding box jako nazwę pliku karty (karty ponazywane są według konwencji "2_H.jpg", "3_H.jpg", …)
   - Przeskaluj losowo (30-110)% zdjęcie i umieść je w losowym miejscu na tle
   - Przeskaluj i przemieść bounding boxy tak, żeby wciąż odpowiadały nałożonemu zdjęciu
   - Sprawdź, czy nałożone bounding boxy nie kolidują z jakimiś bounding boxami, które już są na obrazku. Jeżeli tak jest, to stare bounding boxy usuń
3. Dokonaj augmentacji obrazka. Składają się na to transformacje geometryczne oraz zmienianie parametrów obrazu.
   Geometryczne:
   - Skalowanie - oddzielne skalowanie na x i y, żeby zasymulować zniekształcenia obrazu - wartości (1, 1.2)
   - Przemieszczenie o dany procent (-0.1, 0.1)
   - Obrócenie o stopnie (-10, 10)

   Operacje na obrazie:
   - Zmiana kontrastu liniowego (0.75, 1.5)
   - Nałożenie rozmycia gausowskiego o parametrze sigma=(0, 0.5) na połowę zdjęć
   - Przemnożenie wszystkich pikseli na obrazku względem przyciemnienia/rozjaśnienia przez wartość (0.8, 1.2), przy czym transformacja ta na 20% zdjęć odbywa się niezależnie kanałowo, przez co zmieniana jest ogólna barwa zdjęcia (czasem karty są lekko różowe, czy też zielone)

   Warto zaznaczyć, że operacje te wykonywane są w losowym porządku. Dodam też, że wszystkie operacje geometryczne aktualizują pozycje wcześniej wyznaczonych bounding boxów.
4. Zapisz zdjęcie w formacie .jpg, a opisy klas wyciągnięte z labeli bounding boxów zapisz w pliku .txt o tej samej nazwie.

1.3 Działanie

Przykładowy wynik działania dla 5 kart (poprawnej ręki):



Rysunek 1.2: przykład działania

Rozdział 2

Trening

Sieć trenowaliśmy na platformie Google Colab, jako że dostaje się tam za darmo mocną kartę graficzną, przez co względnie szybko da się wytrenować sieć. Skorzystaliśmy z YOLO_V3 i notebooka w colabie, który szkolił sieć korzystając z pre-trenowanej sieci darknet53.conv.74. Aby wyszkolić model, należało zmienić parametry:

Batches - zalecana wartość to (liczba klas * **2000**), jednak my dla **24** klas ustawiliśmy tę wartość na **8000**, ponieważ trening trwałby zbyt długo. Nie należy jednak ustawiać tej wartości na losową, bo od niej zależy nie tylko długość treningu, ale także learning rate, który zmienia się od **0** do **0.001** według domyślnych ustawień sieci i zbytnia modyfikacja parametru batches może zakłócić naukę.

Classes - parametr ten należy ustawić na liczbę klas, czyli w naszym wypadku **24**

Filters - według wzoru ustaliliśmy go na (liczba klas + 3) * 5

Początkowo trenowaliśmy sieć na innym notatniku i dosyć często Google Colab zabierał nam dostęp do GPU i resetował cały runtime, przez co sieć trzeba było trenować od początku. Po pewnym czasie znaleźliśmy jednak sposób na późniejsze doszkalanie sieci z zapisanego punktu. Domyślny trening zapisuje wyniki wagi sieci co 100 iteracji na dysku google połączonym z Colabem.

Sieć osiąga bardzo zadowalające efekty po około 1500 iteracjach, co w naszym wypadku zajęło **6** godzin. Szacowany czas szkolenia sieci dla pełnej talii kart, czyli **52** klas to **360 godzin** na googlowskiej karcie *Tesla K80*, czyli jest to praktycznie niemożliwe do wytrenowania na naszych kartach.

Rozdział 3

Rozpoznawanie kart

3.1 Rozpoznawanie

Za rozpoznawanie kart widocznych na steamie video z kamerki odpowiedzialna jest instancja klasy CardRecognizer, a dokładnie jej metoda lookForCards.

```
frame, card_collection = card_recognizer.lookForCards()
```

Obiekt klasy CardRecognizer odczytuje obraz z kamerki internetowej za pomocą biblioteki OpenCV, następnie na podstawie pobranej z niej klatki (frame) przy pomocy funkcji blob z biblioteki OpenCV generuje, z przeskalowanej współczynnikiem 1/255 klatki, dane które przekazywane są do instancji sieci neuronowej z klasy NN.

```
def lookForCards(self, camera_stream=WebcamVideoStream(src=0).start(),
                 input_width=224, input_height=224):
    frame = camera_stream.read()
    blob = cv.dnn.blobFromImage(frame, 1/255, (input_width, input_height), [0, 0,
    ↪  0], 1, crop=False)
    self.network.net.setInput(blob)
    outs = network.net.forward(self.getLayerNames(network.net))
    frame, collection = self.processOutput(
        frame, outs), self.card_collection
    cv.imshow(self.window_name, frame)
    return frame, collection
```

3.2 Przetwarzanie wyniku

Zwrócone przez funkcję wyniki (rozpoznania obiektów) następnie są filtrowane celem doboru najlepszych możliwych dopasowań rozpoznanych elementów, odpowiedzialna za to jest metoda processOutput, dodatkowo metoda ta umieszcza na klatce bounding boxy dookoła rozpoznanych przez sieć obiektów.

```
def processOutput(self, frame, outs):
    frame_height, frame_width = frame.shape[0], frame.shape[1]
    class_ids, confidences, boxes = [],[],[]
    for out in outs:
        for detection in out:
            scores = detection[5:]
            classId = np.argmax(scores)
            confidence = scores[classId]
            if confidence > self.confidence_threshold:
            [...]
```

Pod sam koniec rozpoznane karty zostają zsetowane (celem uniknięcia duplikatów w przypadku rozpoznania obydwu narożników), posortowane w zależności od tego ile razy pojawiły się one w ciągu ostatnich x rozpoznań klatek (celem uniknięcia fałszywych, jednoklatkowych rozpoznań wynikających z niedostatecznego dotrenowania sieci).

```
found_cards = list(sorted(set([self.classes[class_ids[i]]
              for i in range(len(class_ids))])))
self.card_collection.append(found_cards)
```

Docelowo wektor historii kart ograniczony jest do ostatnich y elementów, przekazany zostaje do globalnej funkcji normalizeCardCollection która sortuje karty w zależności od częstości ich ostatnich wystąpień.

```
def normalizeCardCollection(collection):
    flat_list = list(itertools.chain(*collection))
    sorted_by_occurences = Counter(flat_list).most_common(5)
    global cards
    figures = [cards[sorted_by_occurences[i][0]]
               for i in range(len(sorted_by_occurences))]
    return figures
```

3.3 Wykrywanie ułożenia

Na koniec kolekcja zostaje przekazana do funkcji check, która rozpoznaje możliwe do otrzymania kombinacje kart.

```
result = check(card_collection)
```

Rozdział 4

Kombinacje kart

Program na podstawie pięciu ostatnich kart wykrywa najlepszą możliwą rękę pokerową. Jest on w stanie wykryć ułożenia takie jak:

- Straight Flush,
- Four of a Kind
- Full House
- Flush
- Straight
- Three of a Kind

Dla użytkownika dostępny jest interfejs w postaci 6 funkcji sprawdzających czy dana pięcioelementowa lista kart jest którymś z powyższych ułożeń, a także jedna funkcja sprawdzająca po kolei wszystkie ręce. Dostępne funkcje:

```
Card = dict[str, str]
straight_flush(cards:  list[Card]) -> bool
four_kind(cards: list[Card]) -> bool
full_house(cards: list[Card]) -> bool
flush(cards: list[Card]) -> bool
straight(cards: list[Card]) -> bool
three_kind(cards: list[Card]) -> bool
check(cards: list[Card]) -> None
```