Opis projektu

W tym pliku chciałbym pokrótce przedstawić tok rozumowania, który doprowadził mnie do rozwiązania zadania oraz odpowiedzieć na pytania postawione w treści zadania.

Rozwiązanie zadania zacząłem od analizy danych (folder EDA) z której wyciągnąłem informacje na temat tego jak dane są ukształtowane, jak należy przeprowadzić preprocessing i jak podejść do problemu. Dodatkowo mogłem już zauważyć pierwsze problemy na jakie mogę napotkać (bramka pod kątem).

Potem zbudowałem modele. Uznałem, że zbuduję dwa modele – jeden który powie czy robot widzi bramkę (a jeżeli nie to po której jego stronie się znajduje np. model przewiduje, że bramka jest na prawo od robota, więc robot skręca w prawo i po raz kolejny robi zdjęcie, żeby wykryć bramkę), a drugi który, gdy widać całą bramkę, poda jej dokładną lokalizację (koordynaty). Uznałem, że dzięki rozbiciu zadania na dwa mniejsze problemy rezultat końcowy będzie lepszy. Uznałem też, że najlepiej będzie jak inputem do modelu będzie zdjęcie o wielkości 200 x 125 pikseli (16 razy mniejszy od oryginalnego zdjęcia). Dzięki temu zachowamy proporcję.

W czasie testowania modeli implementowałem metody augmentacji danych i zmieniałem parametry modeli, żeby uzyskać jak najlepszy rezultat.

Cały kod starałem się pisać zgodnie ze standardem PEP 8. Wszystkie funkcje starałem się jak najlepiej udokumentować, żeby każdy mógł je bez problemu zrozumieć.

Zalety, wady i optymalne warunki rozwiązania

Zalety mojego rozwiązania:

1. Podzielenie zadania na mniejsze problemy – większa szansa na osiągnięcie lepszych rezultatów
2. Zaimplementowanie 3 metod augmentacji danych – poprawiły one proces nauki
3. Możliwość nauki na GPU (CUDA) – szybszy proces nauki
4. Podczas trenowania został użyty callback Early Stopping co pozwoliło na przerwanie nauki modelu w momencie gdy overfittował on do danych treningowych
5. Użycie TensorBoard co pozwala na łatwą wizualizację kosztów.
6. Zaimplementowana wizualizacja pozwala na łatwe zobaczenie wyników
7. Użycie argparse – cały projekt można odpalić w konsoli, nie ma potrzeby uruchamiania IDE do odpalenia projektu, można w łatwy sposób ustawić parametry funkcji
8. Dokumentacja – została zrobiona analiza danych dzięki której widać jakie mamy dane i z jakim problemem mamy do czynienia, jest dostępny plik w którym jest przedstawiona analiza wyników (Results analysis.pdf), jest dostępne czytelne README
9. Styl pisania kodu – kod został napisany zgodnie ze standardem PEP 8, został użyty type hinting, każda funkcja została udokumentowana – nikt nie powinien mieć problemu ze zrozumieniem co się dzieje w programie

Wady mojego rozwiązania:

1. Często konwertuję wartości float na int celem uproszczenia zadania. Jednak w ten sposób w nieodwracalny sposób tracimy dane. Nie jest to bardzo szkodliwe (bardzo lekkie przesunięcie w lewo koordynatów bramki), ale warto o tym pamiętać.

2. Akceptowanie wszystkich zdjęć / brak odrzucania złych zdjęć – mimo dokonanej analizy danych model przyjmuje wszystkie zdjęcia do których miałem dostęp. Nie analizowałem tego czy na pewno wszystkie zdjęcia się nadają (a jeżeli nie to jakie i dlaczego)

Optymalne warunki rozwiązania:

1. Większy dataset, który jest odpowiednio zróżnicowany (warto pamiętać o tym, że model klasyfikacyjny przewiduje lokalizację bramki, więc dobrze by było gdyby było więcej bramek położonych na lewo, prawo lub w dół).
2. Dobrze by było gdyby zamiast obecnie użytej notacji w plikach .txt podawać następujące parametry:
    a. Kod określający położenie bramki
    b. Koordynaty lewego górnego rogu bramki
    c. Koordynaty prawego dolnego rogu bramki

    Ułatwiło by to preprocessing i dzięki temu sam model byłby mniej narażony na ewentualne złe koordynaty z powodu bramki pod zbyt dużym kątem.