Odczytywanie numerów tablic rejestracyjnych ze zdjęć

Jakub Pięta, Piotr Białecki, Tomasz Synowiec, Julia Sowula

7 czerwca 2024

1 Wstęp

W ramach projektu zajmowaliśmy się problemem odczytywania numeru tablicy rejestracyjnej ze zdjęcia. Danymi wejściowymi będą fotografie na których znajdują się samochody, najczęściej z widocznymi tablicami rejestracyjnymi. Wyjściem z modelu jest ciąg znaków oznaczający numer tablicy.

Rysunek 1: Przykładowe zdjęcia użyte w ramach projektu

W trakcie rozwiązywania postawionego problemu użyliśmy więcej niż jednego zbioru. Pełna lista zbiorów danych którymi się zajmowaliśmy znajduje się na końcu wraz z linkami pod którymi można je znaleźć. Użyte zbiory jednak można podzielić na dwie kategorie.

1.1 Zbiory użyte do wykrywania tablic

Pierwszą kategorią użytych zbiorów danych są te które zawierają pary plików w formatach (zdjęcie, adnotacja). Pierwszy element tej pary jest oczywisty, drugi to plik w formacie *.xml* w którym znajdują się informacje dotyczące pozycji tablicy rejestracyjnej na zdjęciu. Jest to tak zwany format *Pascal VOC* o którym więcej można dowiedzieć się pod tym linkiem.

1.2 Zbiory użyte do klasyfikacji znaków na tablicy rejestracyjnej

Drugim rodzajem użytych zbiorów były takie które zawierały pochodzące z różnych źródeł zdjęcia trzydziestu pięciu znaków wraz z ich klasą. Przez znaki rozumiemy cyfry od 0 do 9 oraz litery alfabetu łacińskiego.

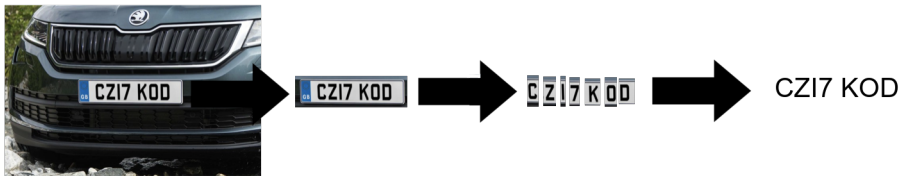
1.3 Uwaga na temat oceny dokładności modelu

Nie dysponujemy zbiorem złożonym jednocześnie ze zdjęć samochodów, pozycji tablic rejestracyjnych oraz numerów tych tablic. Stworzenie takiego zbioru zajęłoby znaczą ilość czasu. Dlatego właśnie ustalenie dokładności modelu przeprowadziliśmy poprzez ręczne sprawdzenie od początku do końca tego, czy modele poprawnie rozwiązały problem znalezienia i odczytania tablicy. Pliki z odpowiednimi zdjęciami znajdują się w folderze **CNN_PREDICTIONS**.

2 Zarys rozwiązania problemu

Nasze podejście do problemu koncentruje się na podzieleniu zadania na mniejsze zadania które możemy z większą bądź mniejszą łatwością rozwiązać. Postanowiliśmy podzielić zadanie na następujące kroki:

1. Znajdowanie tablicy na zdjęciu i zapisanie jej w osobnym pliku graficznym.
2. Podzielenie znalezionej tablicy na obszary zawierające jeden znak.
3. Stworzenie klasyfikatora który rozpozna każdy znak i złączenie otrzymanych klasyfikacji w numer tablicy.



Rysunek 2: Cel który chcemy osiągnąć zbudowanym modelem

Każdy z etapów potraktowaliśmy jako osobne zadanie do którego użyliśmy innych narzędzi.

2.1 Lokalizowanie tablicy na zdjęciu

Znalezienie tablicy na zdjęciu jest zadaniem o nietrywialnej trudności. Pierwszą myślą było przeszukanie zdjęcia w sposób intuicyjny sposób - z pomocą okna o pewnym rozmiarze, które przechodziłoby po wszystkich możliwych pozycjach na obrazie. Dla każdej pozycji odbywałaby się klasyfikacja binarna - czy okno zawiera tablice czy też nie. Następnie wybieralibyśmy okno co do którego model zwrócił najwyższy wynik, który byłby traktowany jako pewność modelu do swojej predykcji. Z takiego podejścia jednak szybko zrezygnowaliśmy z paru powodów:

- Nie każda tablica ma te same proporcje. Tablice samochodów osobowych mają istotnie różne proporcje od tablic rejestracyjnych pojazdów jednośladowych.

- Dobranie rozmiaru okna byłoby niezwykle problematyczne, nawet jeśli zdjęcia będą zestandaryzowane co do swojego rozmiaru. Tablica może znajdować się tuż przed kamerą ale może też znajdować się w oddali.

Musielibyśmy zatem dokonywać przeszukiwania dla wielu możliwych rozmiarów okien i dla wielu możliwych proporcji szerokości do wysokości. Zdecydowaliśmy zatem posłużyć istniejącymi już modelami które rozwiązują ten problem w równie intuicyjny sposób. Naszą uwagę przykuła architektura *YOLO (You Only Look Once)*. Tu coś o tym jak to działa, nie wiem xD .
Wybraliśmy wersję *YOLOv8s*. Litera *s* oznacza jej rozmiar, jest to druga od końca pod względem wielkości sieć *YOLOv8*, zawiera 11.2mln parametrów. Taki wybór był podyktowany głownie tym, że na żadnym posiadanym przez nas sprzęcie nie było realnej szansy użyć większych wersji.
Sieć *YOLOv8* nie przewiduje tablic rejestracyjnych, douczyliśmy ją na naszych zbiorach danych. Ustawiliśmy również minimalną wartość pewności modelu co do predykcji na 0.75. Zatem model może wykryć tablice ale nie będzie co do swojej predykcji wystarczająco pewny abyśmy ją zaakceptowali. Model potrzebował względnie małej liczby przykładów do nauki, całość przykładów zdjęć z pozycjami tablic nie przekraczała tysiąca. Mimo to otrzymane dla zbioru testowego predykcje były bardzo dokładne. W związku z brakiem potrzeby poprawy postanowiliśmy, że w tym wypadku zbiór walidacyjny będzie również zbiorem testowym.

Kilka zwróconych przez model tablic można zobaczyć poniżej

Rysunek 3: Przykłady tablic znalezionych przez nasz model

Te tablice posłużą jako przykłady na których będziemy rozwiązywać kolejny etap problemu, czyli dzielenie tablicy na znaki.

3 Podzielenie tablicy

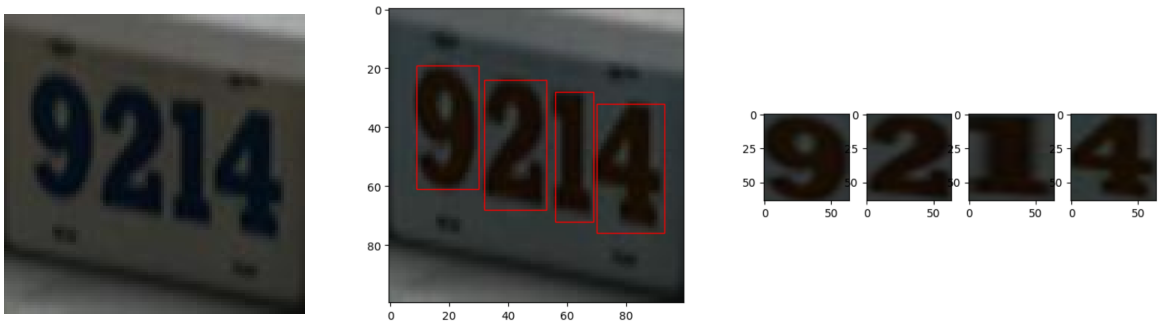
Rozwiązanie problemu podzielenia tablicy na znaki jest problemem o którego rozwiązaniu myśleliśmy na dwa sposoby:

- Można spróbować manipulacji kontrastem na zdjęciu, korzystając z tego, że znaki na tablicy rejestracyjnej co do zasady różnią się istotnie kolorem od tła tablicy. Takie rozwiązanie nie wymaga użycia sieci neuronowych a jedynie napisania dostatecznie dobrze przemyślanej funkcji.

- Jeśli powyższe rozwiązanie się nie powiedzie, to ręcznie można stworzyć pliki ze współrzędnymi boxów zawierającymi znaki. Następnie takie dane mogą zostać użyte do nauki

kolejnej sieci neuronowej, przykładowo kolejnego modelu z architekturą $YOLOv8$. Jednak wymaga to więcej pracy z konieczności ręcznego stworzenia odpowiednich danych. Okazało się, że nie ma potrzeby uciekać się do takich rozwiązań.

Pierwsze z podanych rozwiązań okazało się wyjątkowo skuteczne. Kluczowy okazał się *adaptive tresholding*[1]. W uproszczeniu jest to funkcja która sprowadza obrazek do czarno-białego ale robi to lokalnie, tak aby ciemniejsze obszary obrazka nie zakłócały wyniku który powinien zostać otrzymany w innej części obrazka. Następnie z pomocą funkcji *findContours* szukamy kandydatów na boxy zawierające znaki. W ostatnim kroku odrzucamy oczywiście niepoprawne propozycje boxów - takie które są w złej proporcji albo są za małe lub za duże.



Rysunek 4: Przykład działania funkcji do dzielenia tablicy

Obrazki z tablicami mają znormalizowany rozmiar - 100x100 pikseli. Zdecydowaliśmy, że każdy box musi spełniać następujące warunki:

- Box musi być wyższy niż szerszy
- Maksymalna wysokość boxa to 95 pikseli
- Minimalna wysokość boxa to 30 pikseli

Takie parametry, dobrane w drodze eksperymentów metodą prób i błędów, wydają się być rozsądne i działają bardzo dobrze.

[1]Szczegóły działania funkcji *https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html*

4 Stworzenie klasyfikatora znaków

Stworzenie klasyfikatora jest najprostszą częścią tego projektu. Mimo to postaraliśmy się o jak znalezienie jak najdokładniejszego modelu przy jednoczesnym zachowaniu jego małego rozmiaru.

5 Future Work

Literatura

[1] A. C. Melissinos and J. Napolitano, *Experiments in Modern Physics*, (Academic Press, New York, 2003).

[2] N. Cyr, M. Têtu, and M. Breton, IEEE Trans. Instrum. Meas. **42**, 640 (1993).

[3] *Expected value*, available at `http://en.wikipedia.org/wiki/Expected_value`.