1 Wybór problemu i danych

Zajmujemy się wyszukiwaniem i odczytywaniem numeru tablic rejestracyjnych ze zdjęć. W tym celu posługujemy się trzema zbiorami danych. Są to kolejno Car License Plate Detection, Automatic Number Plate Recognition oraz License Plate Digits Classification Dataset. Pierwsze dwa zawierają zdjęcia samochodów oraz współrzędne ich tablic rejestracyjnych. Ostatni z nich zawiera zdjęcia cyfr i liter uporządkowane w formacie *Pascal VOC*.
Sposób zbierania zdjęć samochodów oraz tworzenia adnotacji nie jest znany. Zbiory nie zawierają jednak znaczących wad. Zostały one odpowiednio połączone i należy korzystać z folderu **DANE** znajdującego się na githubie. Łączna liczba zdjęć samochodów wynosi 886 a ich rozmiar nie jest zestandaryzowany.

Rysunek 1: Elementy zbiorów Car License Plate Detection oraz Automatic Number Plate Recognition

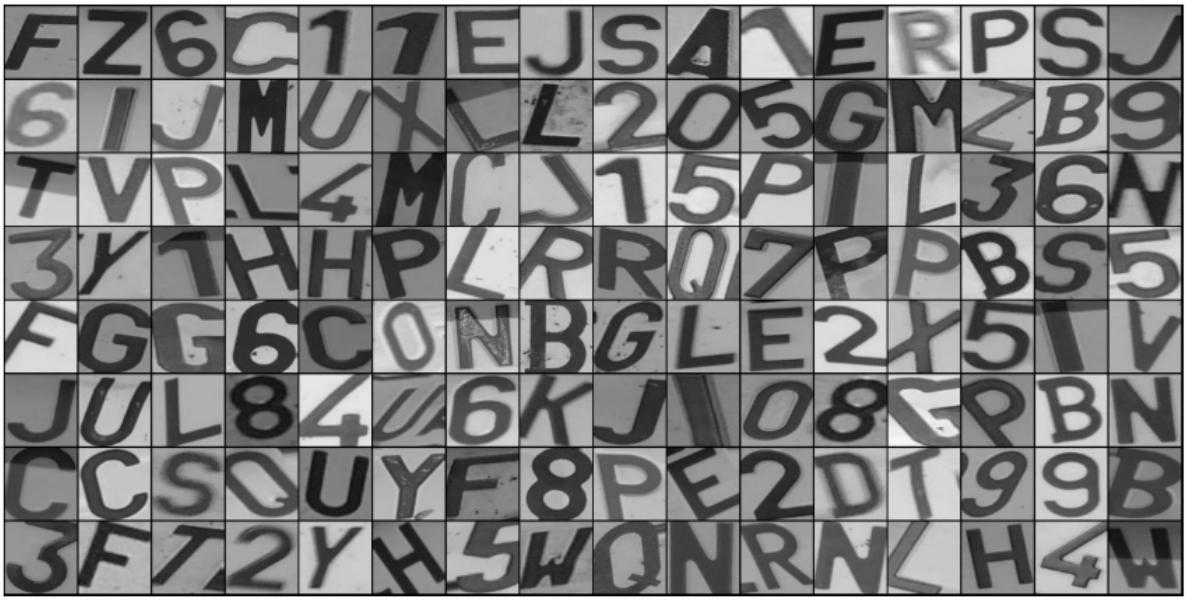
Ostatni ze zbiorów zawiera cyfry i litery pochodzące z belgijskich tablic rejestracyjnych. Podzielono je na pojedyncze znaki rozmiaru 75×100 pikseli oraz zastosowano techniki augmentacji, takie jak rotacja. Zbiór ten zawiera 35500 zdjęć wliczając augmentację.

2 Idea rozwiązania problemu

W celu rozwiązania postawionego problemu podzielimy go na trzy etapy:

- wyszukanie tablicy rejestracyjnej na zdjęciu,
- podzielenie wyszukanej tablicy na znaki,
- klasyfikację znaków.

Każdemu z etapów poświęcimy osobną sekcję.



Rysunek 2: Elementy zbioru License Plate Digits Classification Dataset

3 Budowa modelu

3.1 Wyszukanie tablicy rejestracyjnej ze zdjęcia

Znalezienie tablicy rejestracyjnej na zdjęciu jest zadaniem nietrywialnym, dlatego podejmujemy decyzję o użyciu stworzonego wcześniej modelu, który douczamy do naszych potrzeb. Korzystamy z architektury *YOLOv8*, konkretnie wersji *YOLOv8s*. Litera *s* oznacza rozmiar sieci. *YOLOv8s* jest największą siecią z tej architektury, która może zostać douczona na posiadanym przez nas sprzęcie. Po douczeniu *YOLOv8* przyjmuje na wejściu zdjęcia dowolnego rozmiaru oraz zwraca zdjęcia wykrytych tablic rejestracyjnych.

Rysunek 3: Zwracane przez *YOLOv8* obrazki wraz z detekcjami.

Model ten zawiera parametr *conf* oznaczający minimalną pewność jakiej wymagamy od zwraca-

nych predykcji. Po eksperymentacji ze zbiorem testowym ustawiamy ten parametr na poziomie 0.7. Model uczymy przez 100 epok i otrzymujemy następujące wyniki:

- box loss (complete intersection over unions loss): 0.7744
- cls loss (classification loss): 0.4018
- dfl loss (distribution focal loss): 0.9883

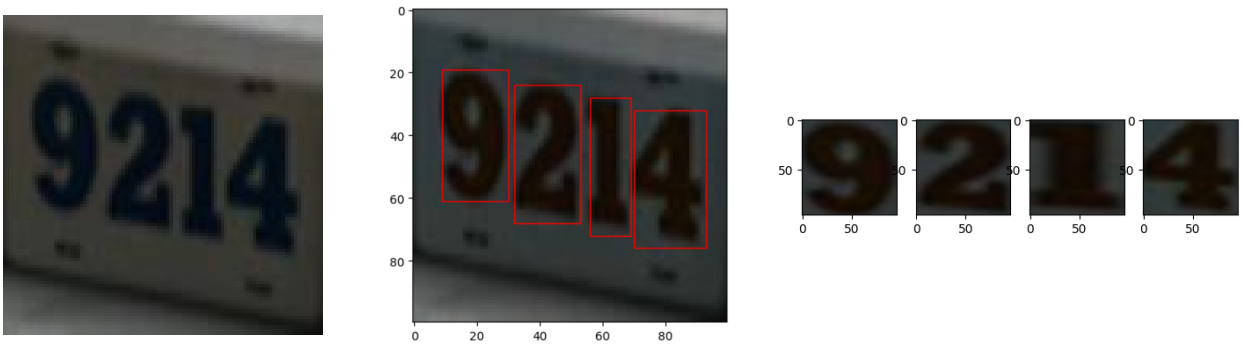
Rysunek 4: Przykłady tablic znalezionych przez nasz model

Pierwszy z błędów odpowiada za to jak blisko znajdują się predykcje modelu do prawdziwych położeń tablic rejestracyjnych. Drugi z nich odpowiada za dokładność klasyfikacji tablica/nie tablica. Może się on wydawać duży ale nie uwzględnia on narzuconego później wymogu parametru $conf = 0.7$ który znacząco pomaga w tej sprawie. Ostatni błąd nie ma większego zastosowania do naszego problemu. Ten rodzaj błędu ma na uwadze znaczące nierówności liczności klas wyszukiwanych obiektów gdy szukamy więcej niż jednego rodzaju obiektu - na przykład samochodów i rowerów w zbiorze zdjęc na których rowery są o wiele rzadsze od samochodów. W trakcie uczenia model używa odpowiednio ważonej średniej tych trzech błędów.
Ogólna dokładność modelu jest wystarczająco zadowalająca, wyszukane obszary zawierają tablice rejestracyjne. Z czterdziestu czterech przypadków tylko w jednym model zwrócił obszar niezawierający tablicy. Pięć razy nie znalazł żadnej, dwa razy znalazł więcej niż jedną.
Model został zapisany do pliku *best.pt* a implementacja, uczenie oraz cała reszta tej części jest zawarta w pliku *[Python]Szukanie.ipynb*

3.2 Podzielenie wyszukanej tablicy na znaki

Dysponując zdjęciami tablic rejestracyjnych przystępujemy do dzielenia ich na znaki, które później można będzie sklasyfikować. Ta część nie wykorzystuje sieci neuronowych. Zdjęcie tablicy rejestracyjnej w poprzedniej części zostało przeskalowane do rozmiarów 100×100 pikseli a teraz przekonwertujemy je z RGB do HSV. Następnie, dzięki funkcji *ADAPTIVE_THRES_GAUSSIAN*,

sprowadzamy zdjęcie do czarno-białego. Takie rozwiązanie pozwala poradzić sobie z lokalnie ciemniejszymi obszarami zdjęć. W kolejnym kroku szukamy kandydatów na ramki zawierające znaki. Odbywa się to poprzez szukanie zwartych grup białych pikseli (bo w naszym przypadku białe bedą znaki a czarne tło) i zamykanie ich w ramkach. Na koniec odrzucamy te z nich, które są za niskie (< 30 pikseli), za wysokie (> 95 pikseli) albo szersze niż wyższe. Takie wymogi wysokości ramek pochodzą z eksperymentacji na zbiorze tablic rejestracyjnych.

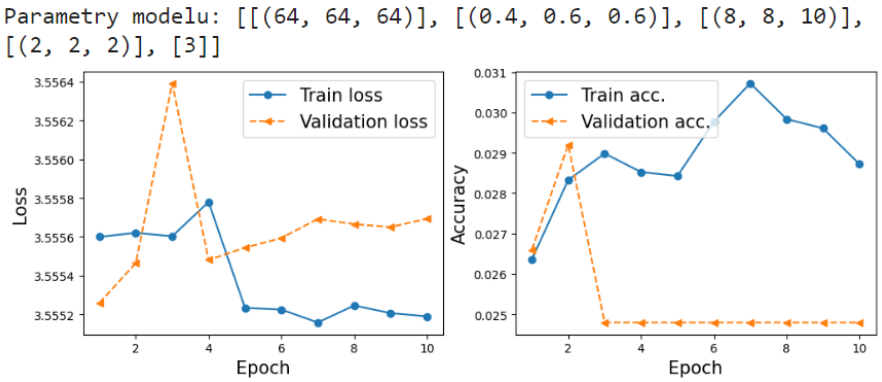


Rysunek 5: Przykład działania funkcji do dzielenia tablicy

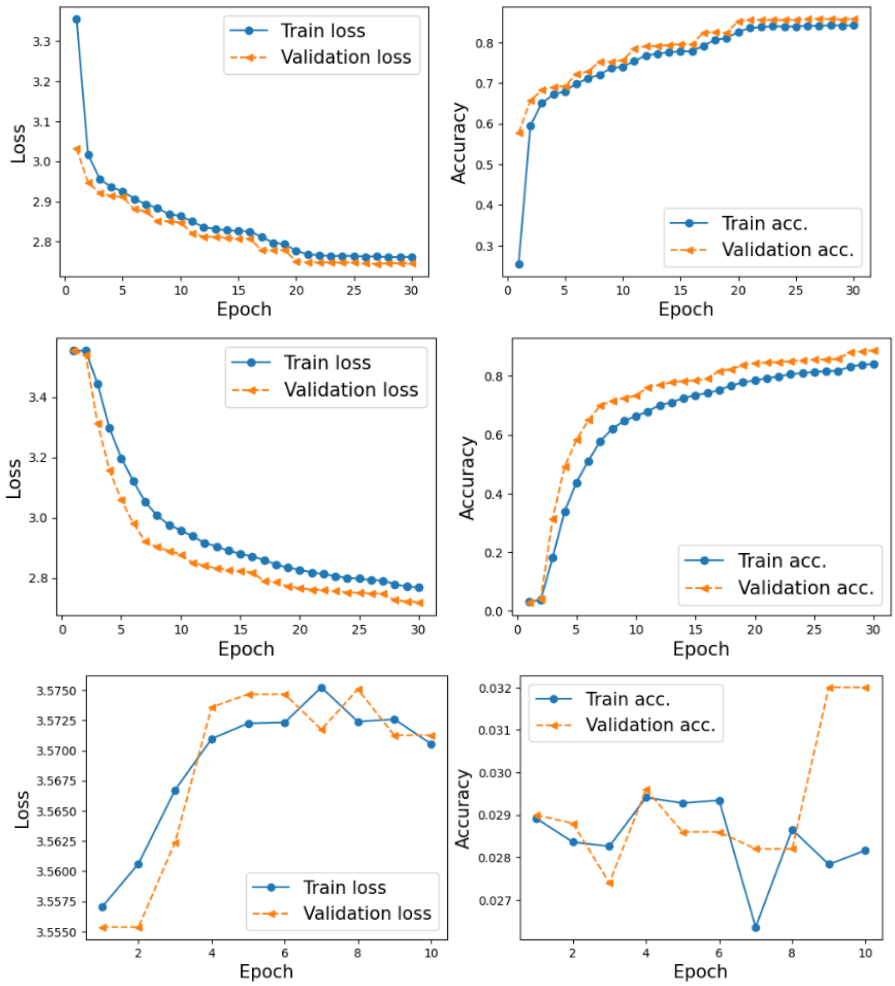
Otrzymane w ten sposób obrazki są przeskalowywane do rozmiaru 96×96, łączone w jeden obrazek o wymiarach $96 \times 96N$ dla N znalezionych ramek oraz zapisywane. Implementacja tej części jest zawarta w pliku *[Python]Dzielenie*

3.3 Klasyfikacja znaków

Problem klasyfikacji znaków rozwiązujemy przy pomocy sieci konwolucyjnych. Przeszukujemy wiele różnych modeli z różnymi parametrami. Generalnym wnioskiem jest fakt, że duża wartość parametru tempa uczenia nie przynosi akceptowalnych rezultatów. W większości sprawdzanych przypadków jedynie tempo uczenia poniżej wartości 0.01 powodowało utworzenie modelu o dokładności przekraczającej 10%. Stosujemy również metodę randomizacji polegającą na przeszukiwaniu modeli z losowo zainicjalizowanymi parametrami z wybranego zbioru. Próbujemy dodać więcej warstw konwolucyjnych i liniowych oraz dokonujemy podobnego przeszukania. Finalnie otrzymujemy model o dokładności zbliżonej do 99.9%, którą uznajemy za dostatecznie wysoką. Model zapisujemy do pliku *model.pt* a jego architekturę można zobaczyć na końcu pliku *[Python]Klasyfikacja*. Jest to model o czterech warstwach konwolucyjnych, dwóch warstwach liniowych oraz, co ciekawe, bez funkcji aktywacji na końcu.



Rysunek 6: Przykładowy zrandomizowany model. Wektory odpowiadają kolejno za: liczbę neuronów w każdej z warstw liniowych, wartości parametru *dropout* dla tych warstw, parametr *out_channel* dla warstw konwolucyjnych, *padding* w każdej z tych warstw. Ostatnia liczba to *kernel_size* dla *MaxPool*.



Rysunek 7: Przykłady znalezionych modeli

4 Ewaluacja modelu

Ponieważ nie dysponujemy zbiorem danych złożonym jednocześnie ze zdjęć, lokacji tablic rejestracyjnych na tych zdjęciach oraz ich numerów, dokładność modelu została ręcznie sprawdzona na niewielkiej próbce dziesięciu znacznie różniących się między sobą przykładów. Model ma problem z generalizacją na nieznane dane, niepochodzące ze zbioru *License Plate Digits Classification Dataset*. Liczba pomyłek wahała się między dwa a cztery dla jednej tablicy.
Przyczyną tego błędu są znaczące różnice wizualne pomiędzy zbiorem użytym do uczenia modelu klasyfikacji a otrzymanymi spoza niego zdjęciami znaków. Pewnym heurystycznym oszacowaniem błędu mógłby być iloczyn błędów poszczególnych części. Jednak takie kryterium wprowadza w błąd ponieważ nie mamy tutaj niezależnych kroków - jeśli model *YOLOv8* nie znajdzie tablicy to w dalszych krokach nie będzie czego dzielić.

Rysunek 8: Przykład działania modelu

Jeśli do kryterium oceny modelu podejdziemy zero-jedynkowo, tzn. model się nie pomylił jeśli dokładnie znalazł tablicę i bezbłędnie ją odczytał, to ma on praktycznie zerową dokładność. Jeśli do tego problemu podejdziemy inaczej to model wydaje się akceptowalny chociaż wymagający poprawy, głównie lepszego zbioru uczącego dla klasyfikatora znaków. Tablice w większości są wyszukiwane poprawnie jak opisano wcześniej, dzielenie również działa z dokładnością przekraczającą 95% (dokładnością w sensie następującym: model dokładnie znalazł wszystkie znaki tablicy) a sam

klasyfikator ma na nowych, nieznanych wcześniej danych dokładność w przedziale $[40\%, 60\%]$ w zależności od użytych zdjęć.

5 Replikowalność wyników

Do przeprowadzenia całej operacji wystarczą pliki *model.pt*, *best.pt* oraz *[Python]Całość*. Dodatkowo można wybrać dowolne własne zdjęcie zawierające samochód i przetestować działanie całości projektu. Należy w pliku *[Python]Całosć* ustawić ścieżkę **Parent_dir** na folder w którym znajdują się pozostałe dwa pliki oraz wybrane zdjęcie, które należy nazwać "**Test_image.jpg**". Następnie należy przeklikać cały notebook w takiej kolejności jaka jest w notatniku.
Dzielenie tablic nie wymaga replikacji, jest to deterministyczna funkcja. Dodatkowo cała jej implementacja jest powtórzona w pliku *[Python]Całość.ipynb*. Uczenie modelu *YOLOv8* można przeprowadzić w pliku *[Python] Szukanie.ipynb* wskazując odpowiednią ścieżkę do folderu *DANE* na początku notatnika. Przestrzegamy jednak przed próbami uczenia tej sieci na słabszym sprzęcie. Uczenie klasyfikatorów znaków może zostać w sposób analogiczny przeprowadzone w pliku *[Python]Klasyfikacja.ipynb*

Rysunek 9: Grupa po oddaniu projektu