Politechn ika Warszawska

Wydział Mechaniczny Energetyki i Lotnictwa

Automatyka i Robotyka

Krzysztof Miśków

Nr. indeksu 304470

System detekcji dziur drogowych w obrazach

Praca Przejściowa Inżynierska

Spis treści

1. Cel

Celem pracy było stworzenie systemu detekcji dziur drogowych w obrazach z kamer drogowych używając popularnych sieci neuronowych. Następnie przetestowanie programu w warunkach rzeczywistych i ocenienie działania zastosowanych modeli. Perfekcyjny model byłby idealną synergią pomiędzy dokładnością, precyzją oraz szybkością wykrywania utrudnień na drodze, gdyż program, aby być użytecznym, musi zwrócić informację jak najszybciej, aby kierowca mógł zareagować zanim samochód najedzie na dziurę.

2. Tworzenie Programu

2.1 Dane

Do wytrenowania sieci zastosowane zostało w sumie 1379 obrazów podzielonych na trzy zbiory: treningową, testową i walidacyjną, które odpowiadały za wytrenowanie sieci oraz ocenę jej skuteczności. Do każdego z obrazów przyporządkowana została adnotacja z informacją o położeniu poszukiwanych obiektów na obrazie. Przypisy były zapisane w odpowiednim formacie do pracy z sieciami Faster RCNN oraz YOLOv5, każdy z nich zawierał 5 liczb oznaczających kolejno: klasę, współrzędne górnego lewego wierzchołka oraz współrzędne dolnego prawego wierzchołka, na każdą dziurę drogową znajdującą się na obrazku. Koordynaty były odpowiednio wyskalowane względem rozmiaru obrazu.

2.2 Transfer Learning

Ze względu na stosunkowo mały zasób danych oraz w celu uzyskania lepszych wyników zdecydowałem się skorzystać z wcześniej wytrenowanych modeli na zbiorze danych COCO (Common Objects in Context), który jest ogromnym zestawem obrazów z oznaczonymi obiektami w różnych warunkach. Metoda transfer learningu polega na wykorzystaniu parametrów zdobytych przez sieć neuronową podczas nauki rozwiązywania zadanego zadania w celu rozwiązania zadania pokrewnego.

2.3 Wykrywanie obiektów w obrazach

Najbardziej powszechne metody wykrywania obiektów w obrazach dzielą się na trzy części. Pierwszym krokiem jest wyodrębnienie z obrazu fragmentów, w których najprawdopodobniej znajduję się jakiś obiekt. Następnie stosując algorytm HOG (Histogram ukierunkowanych gradientów) uzyskujemy informacje na temat obiektów przedstawionych na zadanym regionie. Ostatnim etapem jest określenie, na podstawie wcześniej pozyskanej wiedzy, jakiego typu klasy jest wykryty obiekt.

2.4 Faster R-CNN

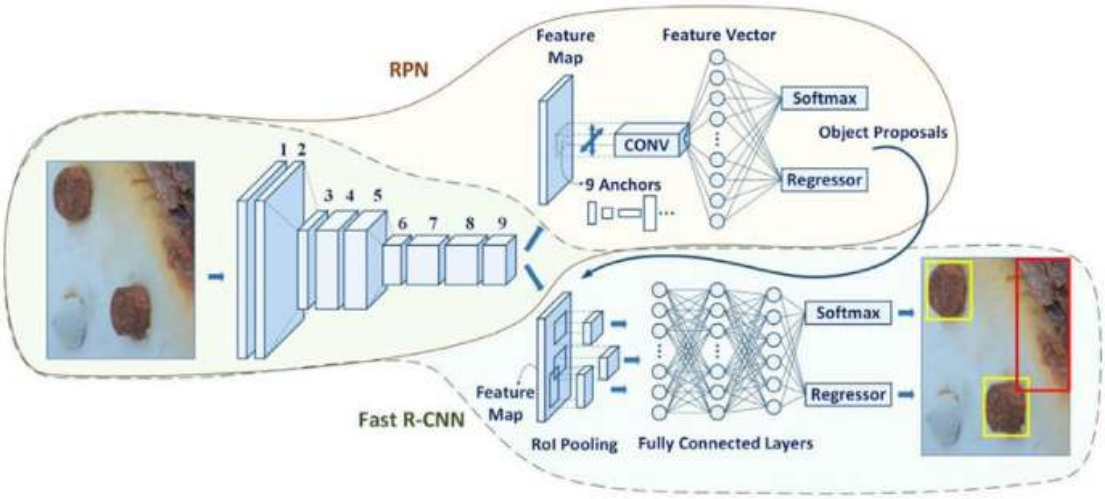
2.4.1 Opis modelu

Faster Region Based Convolutional Neural Network zbudowany jest z połączenia algorytmu RPN oraz architektury Fast R-CNN:

RPN (Region proposal Network) czyli konwolucyjna sieć odpowiadająca za wygenerowanie propozycji położeń obiektów. RPN składa się z klasyfikatora, bloku regresji położenia oraz koncepcji „kotwic", które są prostokątami o stałej wielkości i są dostarczane do Regionalnej sieci propozycji. Zakotwiczenie są zdefiniowane na ostatniej splotowej mapie obiektów. Dla każdej kotwicy RPN przewiduje ogólne prawdopodobieństwo umieszczenia obiektu i czterech współrzędnych korekcyjnych w celu przesunięcia i zmiany rozmiaru kotwicy do właściwej pozycji. Algorytm RPN musi zostać wyszkolony, a więc posiada własną funkcję błędu. Blok regresji zwraca koordynaty okna zawierającego potencjalny obiekt, a klasyfikator określa prawdopodobieństwo, że w danej ramce znajduje się pożądany obiekt.

„U podstaw Faster-RCNN leży zrozumienie, że moc reprezentacji ekstraktora cech jest wystarczająco duża, więc zewnętrzny generator RoI nie jest wymagany. Cały obraz jest dostarczany raz do ekstraktora cech, a jego dane wyjściowe rozchodzą się do szefa klasyfikacji i sieci propozycji regionów (RPN), która generuje propozycje i oceny obiektywności. Tym razem głowica regresji pudełkowej otrzymuje wynik FE dla całego obrazu, a nie wycinków odpowiadających RoI. Kod nie-neuronowy przechodzi w pętlę nad propozycjami - dla każdej propozycji wyjście FE odpowiadające tej poprawce jest wysyłane do klasyfikatora.”[1]

FE – Feature Extractor



Rys. 1 Graficzne przedstawienie metody Faster-RCNN

2.4.2 Moja implementacja modelu

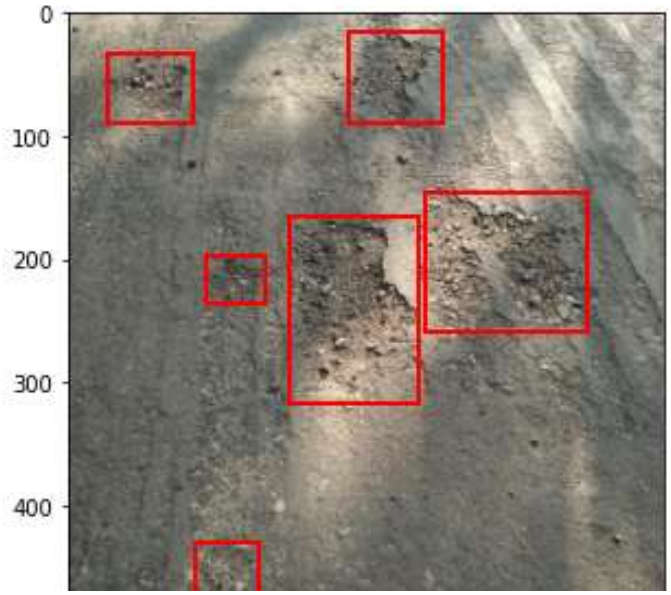
Jako moją pierwszą próbę rozwiązania postawionego zadania postanowiłem, wykorzystując metodę transfer learningu, przeszkolić już wytrenowaną sieć neuronową pobraną ze strony PyTorch. Dzięki temu mimo małego zbioru danych, powinienem uzyskać lepsze rezultaty. Po wyszkoleniu sieci na własnym zbiorze danych, przez 15 epok, uzyskałem następujące wyniki:

Średnia Precyzja dla IoU=0.5:0.95 wyniosła 42.2%
Średnia Precyzja dla IoU=0.5 wyniosła 71.5%
Średnia Precyzja dla IoU=0.75 wyniosła 43.8%

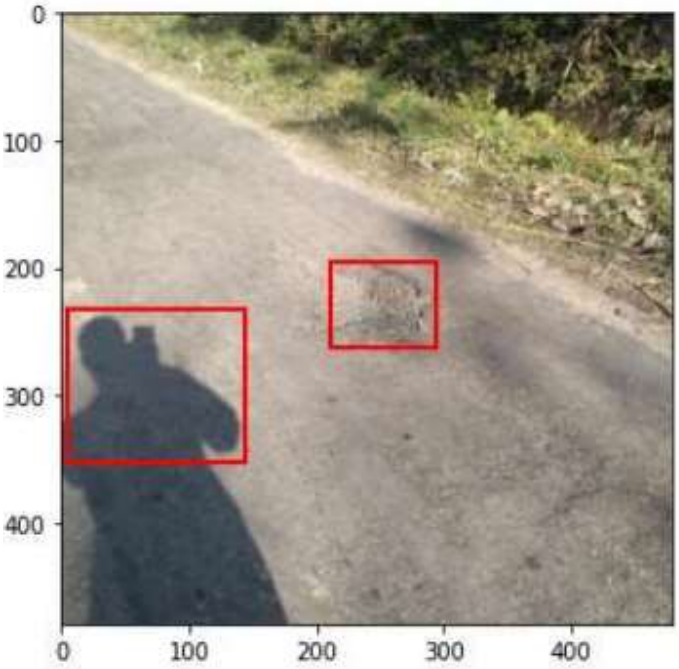
Wyników nie można było uznać za satysfakcjonujące, program radził sobie lepiej z dużymi obiektami dla których średnia precyzja dla IoU z przedziału 0.50:0.95 wzrosła do 60.9%, jednak w przypadku małych obiektów zmalała, aż do 12.5 %.

2.4.3 Efekty działania programu

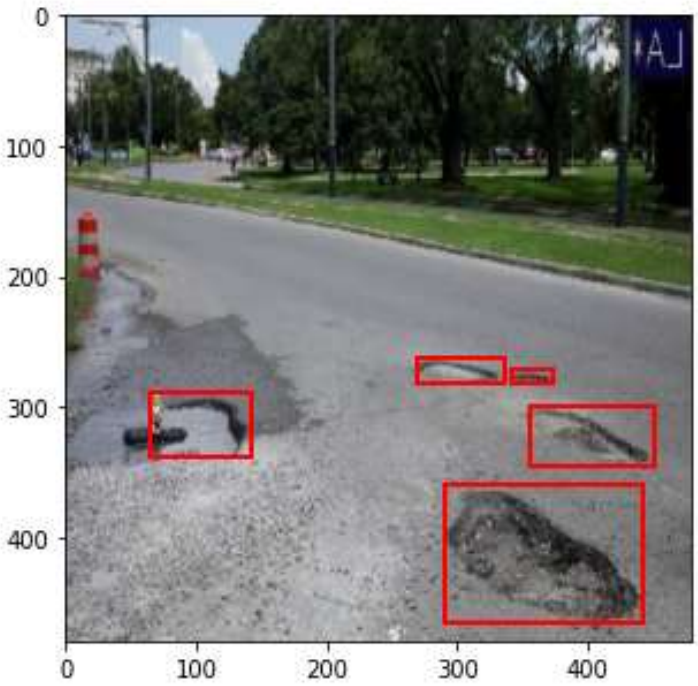
Po wyszkoleniu modelu Faster-RCNN poddajemy go próbie na zdjęciach zawartych w zbiorze testowym w celu wykrycia niedokładności i innych błędów sieci.Poniższe obrazy prezentują działanie programu na testowym zbiorze danych



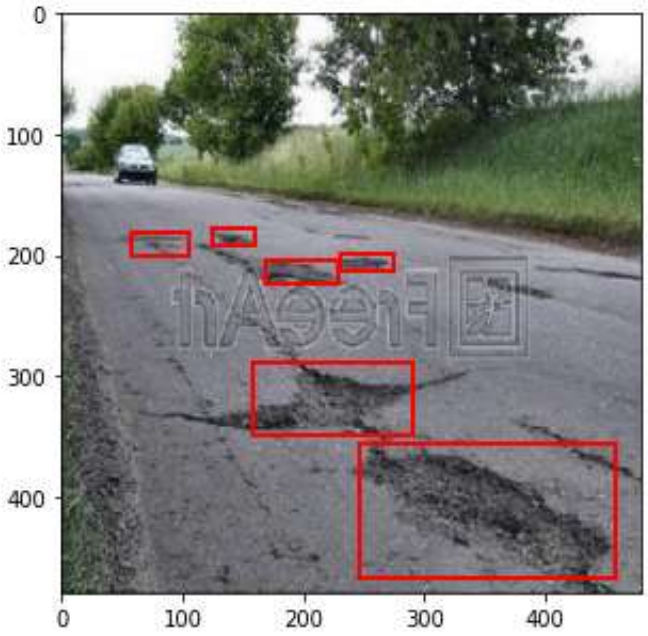
Rys. 2 Program radzi sobie dobrze z dużymi dziurami.



Rys. 3 Program mylił cienie z dziurami, może być to spowodowane zbyt małym zbiorem danych treningowych.



Rys. 4 Z kałużami program radził sobie znacznie lepiej



Rys. 5 Program również radził sobie dobrze z dziurami na tle samochodów.

2.5 Yolo v5

2.5.1 Opis modelu

Algorytmy YOLO (You only look once) dzielą obraz w siatkę jednolitych kwadratów, a następnie generuje N predykcji ramek zawierających obiekt. Każda z ramek jest ograniczona do posiadania wyłącznie jednej klasy w swoim obszarze. Ramki charakteryzujące się wystarczająco dobrą szansą na posiadanie wewnątrz siebie obiektu wybranej klasy lokalizują położenie obiektu na obrazie. Sieć Yolo łączy w sobie zadanie wyznaczenia jak najlepszej ramki wraz z zadaniem klasyfikacji obiektu, dzięki czemu jest w stanie działać przy wysokiej liczbie klatek na sekundę, co przyczynia się do bardzo dobrych wyników podczas detekcji obiektów na filmach. Algorytmy YOLO opierają się na Darknecie, czyli wolno dostępnej struktury sieci neuronowej, YOLO v5 wyróżnia się zastosowaniem warstwy „Focus", która zastępuje 3 pierwsze warstwy YOLO v3. Zaletami jej zastosowania jest zredukowanie potrzebnej pamięci CUDA (Compute Unified Device Architecture) oraz zredukowanie liczby warstw. Ponadto Yolov5 zajmuje mniej miejsca i jest bardziej dokładne od Yolo v4.

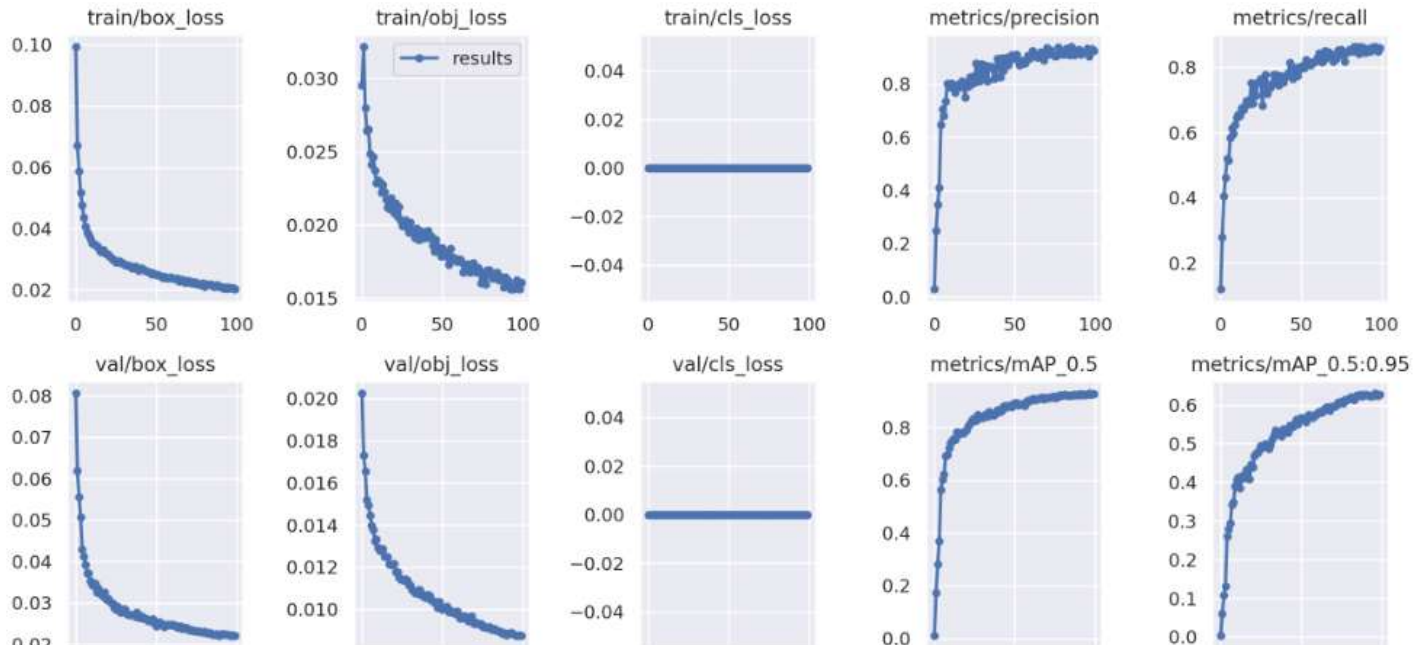
2.5.2 Moja implementacja modelu

Podobnie jak w przypadku Faster RCNN zastosowałem wcześniej wytrenowany model na zbiorze COCO i metodą transfer learningu przekształciłem go, aby rozwiązywał pożądane zadanie.

W celu uzyskania lepszych wyników zastosowałem metodę Fine-Tuning (dostrajanie), która polega na zamrożeniu wytrenowanego modelu, a następnie wyszkoleniu go na nowo na tym samym zbiorze danych. Efektem takiego działania jest przyspieszenie procesu trenowania sieci oraz zniwelowanie efektów zastosowania małego zbioru danych.

Sieć trenowałem przez 100 epok przy wielkości dawki równej 8.

Poniższa grafika przedstawia efekty po pierwszym szkoleniu. Wykresy błędów maleją, a wykresy precyzji rosną, więc efekty wyglądają prawidłowo.



Rys. 6 Wykresy przedstawiające efekty uczenia się sieci

Najważniejsze miary: mAP dla IoU równego 0.5 wyniosła 0.929, a dla Iou z zakresu 0.5 do 0.95 wyniosła 0.63.

Wykresy oznaczają funkcję błędu poszczególnych cech:

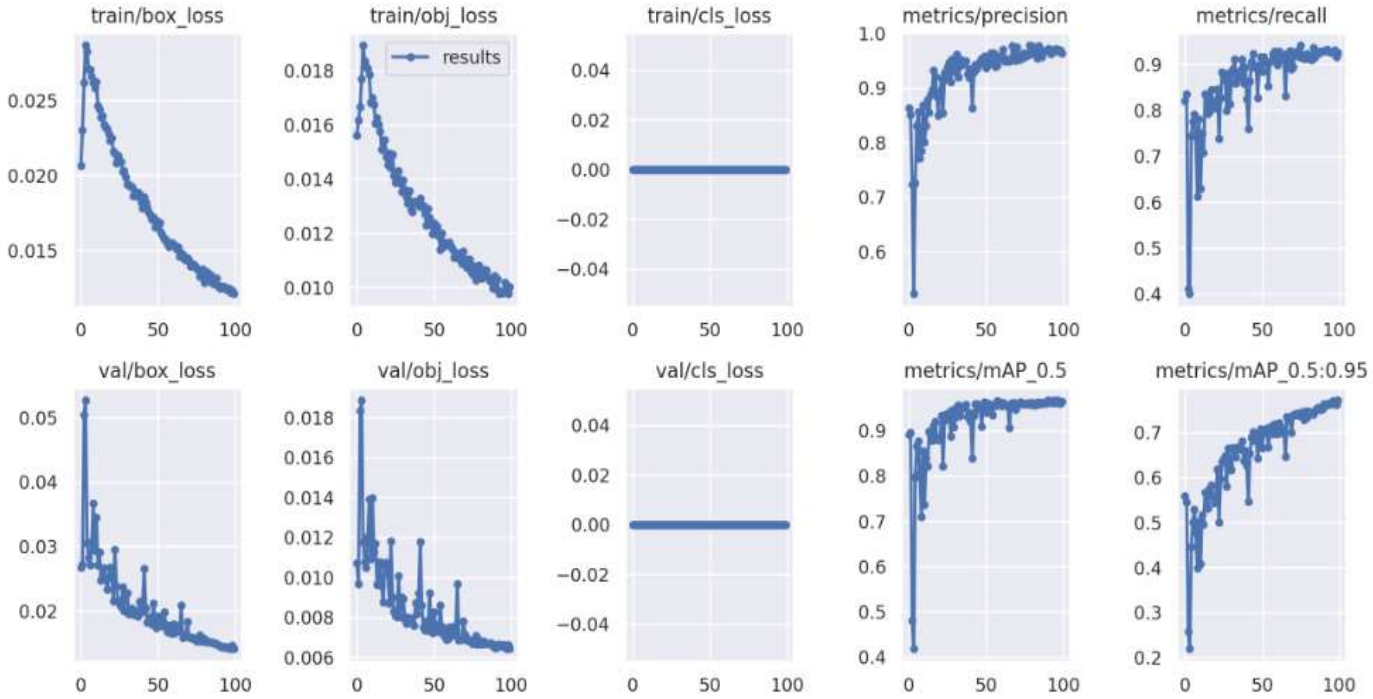
Box_loss – przedstawia wartości funkcji błędu położenia ramki, która jest określana jako różnica pomiędzy przewidzianym położeniem ramki, a prawdziwym.

Obj_loss – przedstawia wartości funkcji błędu czyli prawdopodobieństwo, że wewnątrz ramki nie występuje żaden obiekt.

Cls_loss – przedstawia jak dobrze algorytm potrafi zidentyfikować prawidłową klase, w naszym przypadku szukmay tylko jednej klasy, więc nie może być ona zidentyfikowana jako inna klasa, stąd błąd klasyfikacji wynosi zawsze zero.

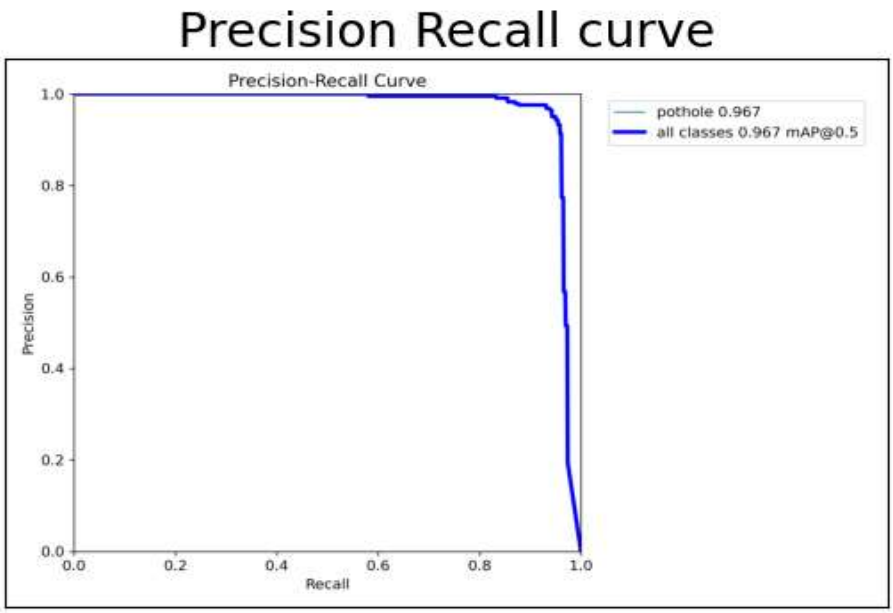
Funkcje błędu dla zbioru treningowego jak i walidacyjnego zbiegają do zera, czyli skuteczność modelu w wyznaczaniu ramek oraz wykrywaniu obiektów zwiększa się.

Poniższe wykresy przedstawiają wyniki modelu po zastosowaniu Fine-Tuningu, podobnie jak na poprzednich wykresach funkcja błędu maleje, a precyzja rośnie. Wykresy mają podobny kształt do efektów poprzedniego treningu.



Rys. 7 Wykresy przedstawiające efekty uczenia się sieci przy zastosowaniu Fine Tuningu

Ostatecznie po przetestowaniu działania sieci osiągnąłem satysfakcjonujące wyniki:



2.5.3 Rezultaty działania sieci w praktyce

Po wyszkoleniu modelu Faster-RCNN poddajemy go próbie na zdjęciach zawartych w zbiorze testowym w celu wykrycia niedokładności i innych błędów sieci.

Rys. 8 Program nie wykrywa cieni jako dziury drogowe

Rys. 9 Program nie wykrywa studzienek jako dziury drogowe

Rys. 10 Program radzi sobie dobrze w trudnych, wykrywa dziurę mimo znajdującej się w niej wody i zniszczonego asfaltu

Rys. 11 Program nie wykrywa elementów samochodów jako dziury drogowe

Dodatkowe sprawdzenie działania programu zostało przedstawione w załączonym filmie z kamery samochodowej dla auta jadącego z prędkością około 85km/h.

3. Podsumowanie

Efekty akie udało mi się osiągnąć uważam za bardzo dobre, pomimo małego zbioru danych program radzi sobie w różnych warunkach. Program poprawnie wykrywa dziury na filmie z kamery samochodowej oraz na zdjęciach.

Wszystkie obliczenia były prowadzone na prywatnym komputerze nie stworzonym do uczenia maszynowego, dlatego postanowiłem nie przekraczać 100 epok. W celu uzyskania większej mocy obliczeniowej skorzystałem z architektury CUDA i trening prowadziłem przy wykorzystaniu karty graficznej.

4. Wnioski

Zbiór danych nie zawierał obrazów dziur z dalekiej odległości, mogło by to pomóc wykrywać dziury z dalszej odległości i informować kierowcę szybciej o zbliżającym się uszkodzeniu. W celu polepszenia rezultatów na pewno należałoby zastosować bogatszy zbiór danych oraz szkolić sieć przez większą ilość epok, dodatkowo istnieje możliwość polepszenia wydajności przy zastosowaniu różnych technik takich

jak: umożliwienie asynchronicznego wczytywania danych, czy wyłączenie debugowania interfejsu programowania aplikacji.

5. Źródła:

https://towardsdatascience.com/faste r-rcnn-object-detection-f865e5ed7fc4
https://medium.com/deelvin-machine-learning/pytorch-performance-tuning-in-action-7c4d065d4278
https://medium.com/swlh/one-stop-for-object-detectors-2c99daa08c50
https://pytorch.org/vision/main/models/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn.html
https://towardsdatascience.com/the-practical-guide-for-object-detection-with-yolov5-algorithm-74c04aac4843
https://github.com/ultralytics/yolov5
https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173
https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
https://www.kaggle.com/datasets/sachinpatel21/pothole-image-dataset
https://public.roboflow.com/object-detection/pothole
https://pytorch.org/hub/ultralytics_yolov5
Wykrywanie obiektów z głębokim uczeniem: RCNN, kotwice, bez maksymalnego tłumienia (ichi.pro)[1]