

Karta modelu sdv4 YOLOv8s

KlikaczeWKomputer

Projekt traffic signs detection z użyciem ML YOLOv8

TrafficEye

Zespół:

Maciej Kała - lider zespołu, frontend

Alina Upyrowa - UI designer, dokumentacja, stylizacja

Ruslan Zhukotynskyi - prawa ręka lidera, ML model trener, łączenie komponentów, labelowanie

Mateusz Kruszewski - backendowca, tester

Eryk Żabiński - backendowca, baza danych, labelowanie

Prowadzący: mgr inż. Krzysztof Ropiak

Firma: Visimind

Uniwersytet: UWM w Olsztynie

Spis treści

| | |
|--|-----------|
| 1. Informacje ogólne | 3 |
| 1.2 Krótki opis | 3 |
| 2. Narzędzia i technologie | 3 |
| 2.1 Framework i biblioteki | 3 |
| 2.2 Środowisko treningowe | 4 |
| 2.3 Śledzenie eksperymentów | 4 |
| 2.4 Ekspорт modelu | 4 |
| 3. Dane treningowe | 5 |
| 3.1 Źródło i charakterystyka | 5 |
| 3.2 Przygotowanie danych | 5 |
| 3.3 Augmentacja danych | 6 |
| 4. Architektura modelu | 7 |
| 4.1 Typ modelu | 7 |
| 4.2 Szczegóły architektury z pliku args.yaml | 8 |
| 4.3 Hiperparametry treningu modelu sdv4 | 8 |
| 5. Wyniki treningu | 8 |
| 5.1 Metryki walidacyjne | 8 |
| Straty walidacyjne | 9 |
| Artefakty w MLFlow | 9 |
| 5.2 Analiza wyników | 9 |
| 5.2.1 Wstęp | 9 |
| 5.2.2 Analiza Wyników | 10 |
| 6. Zastosowanie | 15 |
| Aplikacja mobilna | 15 |
| 7. Ograniczenia | 16 |
| 7.1 Wyzwania datasetu | 16 |
| 7.2 Ograniczenia modelu | 16 |
| 7.3 Potencjalne problemy | 16 |
| 8. Wnioski | 17 |
| 8.1 Osiągnięcia | 17 |
| 8.2 Lekcje wyciągnięte | 17 |

1. Informacje ogólne

Nazwa modelu: sdv4

Cel: Detekcja znaków drogowych na zdjęciach, filmach oraz w czasie rzeczywistym, z wizualizacją bounding boxów i opisami znaków, w celu wsparcia kierowców w rozpoznawaniu znaków w różnych warunkach oświetleniowych i pogodowych.

Kontekst projektu: Model jest częścią aplikacji mobilnej na platformę Android, opracowanej dla firmy Visimind, która ma na celu zwiększenie bezpieczeństwa na drogach poprzez automatyczne rozpoznawanie znaków drogowych.

Wersja: v3 (trenowana w Maju 2025) oraz v4 (trenowana w Czerwcu 2025)

Data treningu: 19.05.2025 , 01.06.2025

Autor modelu: Ruslan Zhukotynskyi

Eksperyment MLFlow: yolov8s_sign_detection_v4, yolov8s_sign_detection_v3, run train_baseline_with_aug

1.2 Krótki opis

Model sdv4 został zaprojektowany do identyfikacji i klasyfikacji znaków drogowych w obrazach, sekwencjach wideo oraz strumieniach wideo w czasie rzeczywistym.

Aplikacja mobilna, w której model jest zintegrowany, wyświetla bounding boxy wokół wykrytych znaków oraz opisy, co pomaga kierowcom w szybkim rozpoznawaniu znaków.

2. Narzędzia i technologie

2.1 Framework i biblioteki

- **Ultralytics 8.3.138:** Framework do treningu i inferencji YOLOv8s, wybrany za wysoką wydajność, wsparcie dla niestandardowych datasetów i łatwość integracji z PyTorch.
- **PyTorch 2.8.0.dev20250518+cu128:** Biblioteka do obliczeń tensorowych, z obsługą CUDA dla akceleracji GPU.
- **Albumentations 2.0.8:** Biblioteka do augmentacji danych, umożliwiająca dynamiczne przetwarzanie obrazów i bounding boxów w formacie YOLO.
- **MLFlow 2.22.0:** Platforma do śledzenia eksperymentów, logowania metryk, parametrów i artefaktów (np. wykresów, modeli).
- **ONNX Runtime:** Użyty do eksportu modelu do formatu ONNX i dalszej konwersji do TFLite dla aplikacji mobilnej.
- **NumPy 1.26.4:** Do operacji na danych i przetwarzania wyników.

- **tensorflow 2.14.0**: Framework do inferencji modelu w formacie TensorFlow Lite, używany w skrypcie visualize_tflite_detections.py do testowania detekcji na zdjęciach.
- **opencv-python**: Przetwarzanie obrazów, rysowanie bounding boxów i zielonych etykiet na wynikach detekcji
- **Pillow 11.2.1**: Wczytywanie i manipulacja obrazami w formacie RGB, alternatywa dla OpenCV w skrypcie inferencji.
- **PyYAML 6.0.2**: Wczytywanie nazw klas znaków (np. A-1, B-33) z pliku data.yaml dla poprawnego oznaczania detekcji.

2.2 Środowisko treningowe

- **System operacyjny**: Windows 11
- **GPU**: NVIDIA GeForce RTX 4060, 8 GB VRAM
- **CUDA version**: 12.8
- **Pamięć RAM**: 16 GB

2.3 Śledzenie eksperymentów

- **Platforma**: MLFlow, skonfigurowana lokalnie z backend store i artifact store w file:runs/mlflow.
- **Serwer**: Lokalny serwer MLFlow uruchomiony polecienniem:
`mlflow server --backend-store-uri file:runs/mlflow --default-artifact-root
file:runs/mlflow --host 0.0.0.0 --port 5000`
- **Interfejs użytkownika**: Dostępny pod adresem <http://127.0.0.1:5000>.
- **Eksperyment**: yolov8s_sign_detection_v4, yolov8s_sign_detection_v3, run train_baseline_with_aug.
- **Artefakty**:
 - Wykresy: confusion_matrix.png, PR_curve.png, F1_curve.png .
 - Plik konfiguracyjny: data.yaml.
 - Model: Zapisany w MLFlow jako yolov8s_model oraz jako best.onnx.

2.4 Eksport modelu

Model został wyeksportowany do formatu **ONNX** (opset=13) dla uniwersalnej kompatybilności oraz skonwertowany do **TFLite** (float32, float16) dla inferencji na urządzeniach mobilnych.

3. Dane treningowe



3.1 Źródło i charakterystyka

- **Dataset:** Własny dataset przygotowany przez firmę Visimind, zawierający obrazy znaków drogowych w różnych warunkach oświetleniowych i kątach widzenia. Oraz dodane nowe zdjęcia, znaczy utworzony dodatkowy własny dataset z 700 zdjęć wraz z augmentacją co wyszło na 2500 zdjęć dodatkowych. Ale za pomocą automatycznego pobierania i labelowania zdjęć, co skrypt stworzył kolega z innego zespołu, udało się rozszerzyć dataset jeszcze o 1000 zdjęć. (Link do GitHub tego projektu do automatycznego pobierania:
<https://github.com/ArturSierakowski/traffic-sign-detection-workflow>)
- **Struktura:**
 - **Treningowy:** 5046 obrazów (project_dataset/images/train)
 - **Walidacyjny:** 1009 obrazów (project_dataset/images/val)
 - **Testowy:** 674 obrazów (project_dataset/images/test)
- **Liczba klas:** 100.
- **Format etykiet:** YOLO (pliki .txt z identyfikatorami klas i współrzędnymi bounding boxów).
- **Źródła obrazów:** Zebrane z rzeczywistych nagrań drogowych, z uwzględnieniem różnicowanych warunków środowiskowych, aby zapewnić robustość modelu. Zdjęcia do dodatkowego datasetu zostały pobrane z serwisu Mapillary.

3.2 Przygotowanie danych

- Obrazy zostały przeskalowane do rozdzielczości 640x640 pikseli, zgodnie z parametrem imgsz=640.

- Etykiety bounding boxów zweryfikowano pod kątem poprawności formatu YOLO i minimalnej widoczności obiektów (min_visibility=0.1).
- Dataset podzielono na zbiory treningowy, walidacyjny i testowy w proporcjach ~75:15:10, zapewniając równomierne rozłożenie klas.

3.3 Augmentacje danych

Aby zwiększyć generalizację modelu, zastosowano niestandardowe augmentacje za pomocą biblioteki Albumentations:

- **Rotacja:** ± 15 stopni, prawdopodobieństwo 50%, symuluje różne kąty widzenia znaków.
- **Losowy crop:** 640x640 pikseli, prawdopodobieństwo 40%, zwiększa różnorodność perspektyw.
- **CLAHE (Contrast Limited Adaptive Histogram Equalization):** Clip limit=4.0, tile grid size=8x8, prawdopodobieństwo 30%, poprawia kontrast w trudnych warunkach oświetleniowych.
- **Losowa jasność i kontrast:** $\pm 20\%$, prawdopodobieństwo 30%, symuluje zmiany oświetlenia (np. zmierzch, refleksy).
- **ToTensorV2:** Konwersja obrazów do tensorów PyTorch, wymagana dla modelu YOLOv8.

Dodatkowo, wbudowane augmentacje YOLOv8:

- **Mosaic:** Prawdopodobieństwo 80%, łączy 4 obrazy w jeden, zwiększając różnorodność scen.
- **HSV:** Modyfikacja odcienia (hsv_h=0.015), nasycenia (hsv_s=0.7), jasności (hsv_v=0.4), p=100%, symuluje różne warunki kolorystyczne.
- **Przesunięcie (Translate):** $\pm 10\%$ obrazu, p=100%, zmienia pozycję znaków.
- **Skalowanie (Scale):** $\pm 50\%$, p=100%, symuluje znaki w różnej odległości.
- **Ścinanie (Shear):** $\pm 2^\circ$, p=100%, odzwierciedla zniekształcenia perspektywy.
- **Random Erasing:** p=40%, usuwa losowe fragmenty obrazu, zwiększając odporność na częściowe zasłonięcie znaków.

Przykład augmentacji:

Oryginalny Obraz



Augmentowany Obraz



Oryginalny Obraz



Augmentowany Obraz



Oryginalny Obraz



Augmentowany Obraz



4. Architektura modelu

4.1 Typ modelu

- **YOLOv8s:** Wersja "small" rodziny YOLOv8, wybrana jako kompromis pomiędzy dokładnością a wydajnością na urządzeniach mobilnych.
- **Pretrenowany model:** yolov8s.pt, wstępnie wytrenowany na zbiorze danych połączonych od firmy i od własnego.

4.2 Szczegóły architektury z pliku args.yaml

- **Rozmiar wejściowy:** Obrazy z datasetu (project_dataset/images/train, val, test) są skalowane do 640x640 podczas treningu i inferencji, z zachowaniem proporcji (padding, jeśli potrzebne).
- **Format:** Tensory PyTorch o wymiarach [batch, 3, 640, 640] (batch=16)
- **Funkcja straty:**
 - Box loss (CIOU): Minimalizacja różnicy między przewidywanymi a rzeczywistymi bounding boxami. Waga: box: 7.5.
 - Classification loss: Dla predykcji klas. Waga: cls: 0.5.
 - DFL (Distribution Focal Loss): Dla poprawy precyzji w trudnych przypadkach, zwiększenie precyzji lokalizacji małych obiektów. Waga: dfl: 1.5.

4.3 Hiperparametry treningu modelu svd4

Na podstawie kodu treningowego:

- **Epoki:** 70
- **Rozmiar obrazu:** 640x640 pikseli
- **Batch size:** 16 (zoptymalizowany dla GPU z 8 GB VRAM)
- **Optimizer:** AdamW
- **Learning rate:** Początkowy (lr0): 0.001
- **Patience:** 10 epok (wczesne zatrzymanie, jeśli walidacyjne metryki nie poprawiają się)
- **Liczba klas:** 100

Dodatkowe ustawienia:

- **Workers:** 0 (wyłączone multiprocessing dla kompatybilności z Windows).
- **Augmentacje wbudowane:** Mosaic, HSV, translate (szczegóły w sekcji "Augmentacje danych").

5. Wyniki treningu

5.1 Metryki walidacyjne

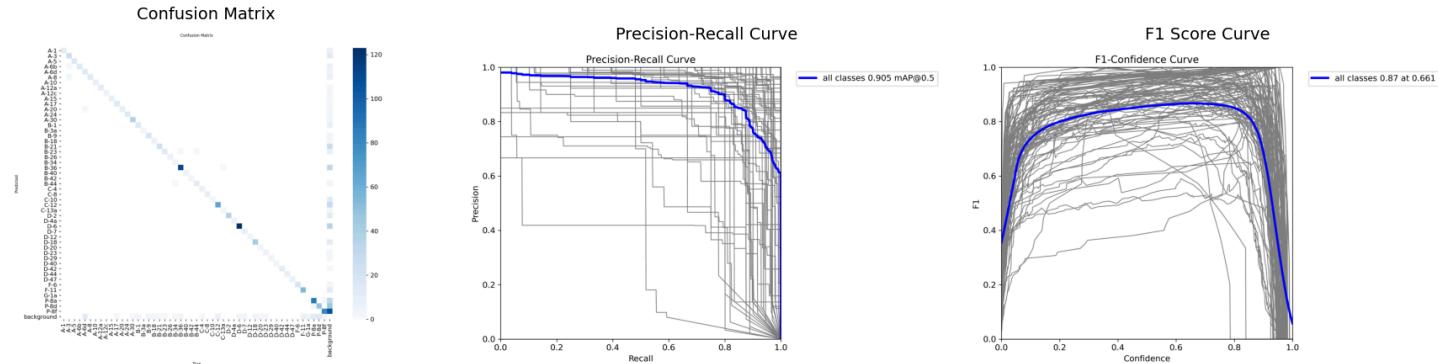
Metryki uzyskane na zbiorze walidacyjnym (1009 obrazów).

- **mAP@50:** 0.93589
- **mAP@50:95:** 0.76237
- **Precision:** 0.9224
- **Recall:** 0.8985

Straty walidacyjne

- **Box loss:** 0.72337
- **Classification loss:** 0.39293
- **DFL loss:** 0.85337

Artefakty w MLFlow



- **Wykresy:**
 - confusion_matrix.png: Macierz pomyłek, pokazująca dokładność klasyfikacji dla każdej klasy.
 - PR_curve.png: Krzywa Precision-Recall, ilustrująca trade-off między precyzją a recall.
 - F1_curve.png: Krzywa F1, pokazująca balans między precyzją a recall.
- **Plik konfiguracyjny:** data.yaml, zawierający strukturę datasetu i nazwy klas.
- **Model:** Zapisany jako w postaci .pt oraz w postaci .onnx

5.2 Analiza wyników

5.2.1 Wstęp

Nasz zespół przeprowadził szereg eksperymentów treningowych w celu optymalizacji modelu YOLOv8 do detekcji znaków drogowych w ramach projektu TrafficEye. Proces obejmował testowanie różnych konfiguracji hiperparametrów, augmentacji danych oraz stopniowe rozszerzanie datasetu. Początkowo wykorzystano dataset Visimind (2600+ obrazów), który został później powiększony o dodatkowe 2500 obrazów a potem jeszcze o 1046 (łącznie 5046 treningowych, 1009 walidacyjnych, 674 testowych), w tym dane augmentowane. Treningi realizowano na GPU NVIDIA GeForce RTX 4060, 8 GB VRAM, z różną liczbą epok (50-70) i batchami (16), stosując modele yolov8n.pt , yolov8s.pt oraz yolov8m.pt. Celem było poprawienie metryk takich jak mAP@50, mAP@50:95, precyzja i straty (Box Loss, Cls Loss, DFL Loss), aby osiągnąć wysoką dokładność detekcji 100 klas znaków drogowych. Wyniki tych eksperymentów zostały zebrane i przeanalizowane, co pozwoliło na wybór finalnego modelu.

5.2.2 Analiza Wyników

Poniżej przedstawiono analizę siedmiu wytrenowanych modeli YOLOv8 (od FIRST MODEL do YOLOv8s Final), opartą na metrykach uzyskanych podczas treningu z różnymi konfiguracjami i datasetami. Dane pochodzą z porównania wyników na zbiorze testowym.

| Model | Dataset Extension | Epochs | Precision ↑ | Recall ↑ | mAP50 ↑ | mAP50 -95 ↑ | Box Loss ↓ | Cls Loss ↓ | DFL Loss ↓ |
|-------------------------------|---------------------------------|--------|-------------|-------------|-------------|-------------|------------|------------|------------|
| YOLOv8s.pt (FIRST MODEL) | Visimind dataset 2600+- | 50 | 0.86613 | 0.800 68 | 0.8638 1 | 0.667 27 | 0.69533 | 0.39464 | 0.84124 |
| YOLOv8n.pt | +2500 images | 70 | 0.926 | 0.869 | 0.928 | 0.709 | 0.8557 | 0.54 | 0.897 |
| YOLOv8m.pt | +2500 images | 50 | 0.948 | 0.903 | 0.947 | 0.757 | 0.6974 | 0.3717 | 0.8561 |
| YOLOv8s.pt (default train) | +2500 images | 70 | 0.93658 | 0.923 52 | 0.9531 5 | 0.778 51 | 0.62419 | 0.33248 | 0.83531 |
| YOLOv8s (aug+params) | +2500 images (augmente d) | 70 | 0.94127 | 0.910 69 | 0.9509 8 | 0.766 88 | 0.75699 | 0.3895 | 0.86131 |
| YOLOv8m Last | +2500 + 1046 images (aug) | 70 | 0.92624 | 0.911 76 | 0.9389 2 | 0.766 47 | 0.47121 | 0.27451 | 0.7281 |
| YOLOv8s Final | +2500 + 1046 images (aug) | 70 | 0.9224 | 0.898 5 | 0.9358 9 | 0.762 37 | 0.72337 | 0.39293 | 0.85337 |

Tabela 1. Porównanie metryk wydajności modeli YOLOv8

Precyzja: Procent poprawnych detekcji ($TP / (TP + FP)$), wskazujący minimalizację fałszywych alarmów.

Recall: Procent wykrytych rzeczywistych obiektów ($TP / (TP + FN)$), mierzący kompletność detekcji.

mAP@50: Średnia precyzja przy $\text{IoU}=0.5$ dla 100 klas, odzwierciedlająca ogólną skuteczność.

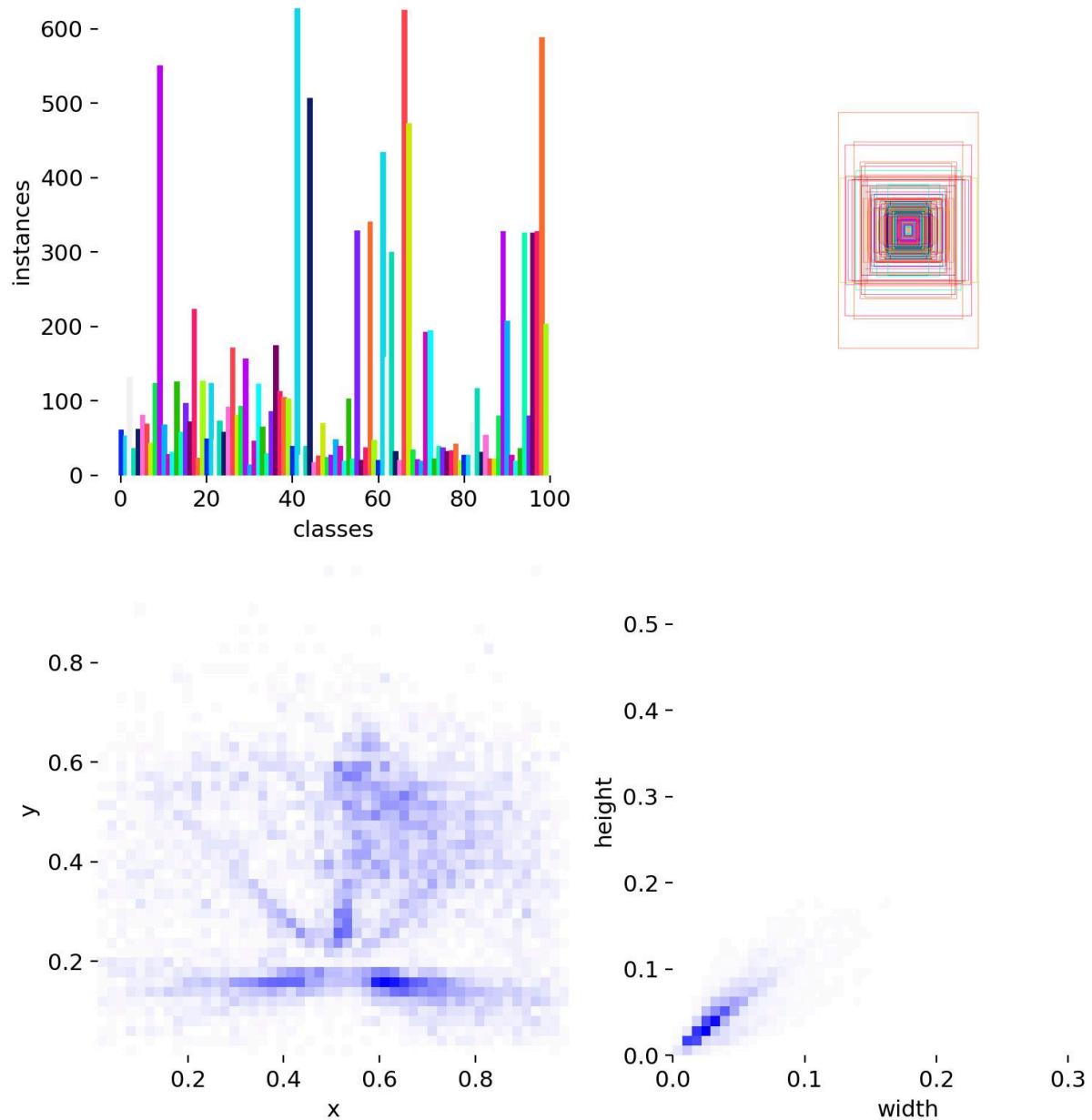
mAP@50:95: Średnia precyzja dla IoU od 0.5 do 0.95, oceniąca dokładność lokalizacji.

Analiza:

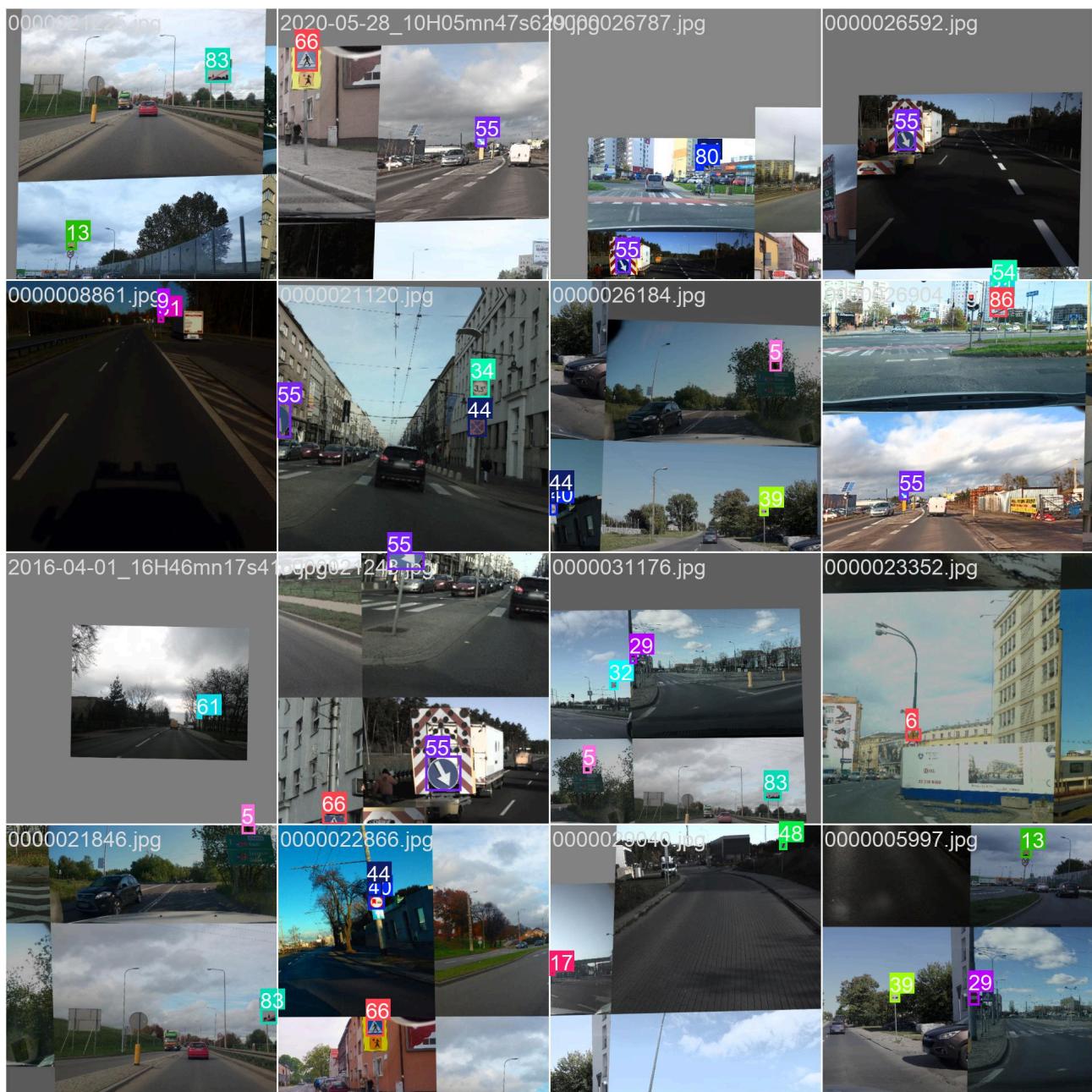
- **YOLOv8s.pt (FIRST MODEL):** Bazowy model na datasetcie Visimind (2600+ obrazów) osiągnął mAP@50:95=0.66727 po 50 epokach. Wysoka precyza (0.86613) i recall (0.80068) wskazują na dobrą detekcję, ale straty (Box Loss=0.69533) sugerują niedostateczną lokalizację boxów.
- **YOLOv8n.pt:** Po dodaniu 2500 obrazów i 70 epokach, mAP@50:95 wzrósł do 0.709, ale straty (Box Loss=0.8557) są wyższe, co wskazuje na trudności w optymalizacji mniejszego modelu (YOLOv8n).
- **YOLOv8m.pt:** Po 50 epokach mAP@50:95=0.757, ale brak dalszej poprawy przy 70 epokach sugeruje, że większy model (YOLOv8m) nie wykorzystał w pełni datasetu.
- **YOLOv8s.pt (default train):** Domyślna konfiguracja z 70 epokami dała najlepszy wynik mAP@50:95=0.77851, z niskimi stratami (Box Loss=0.62419), co pokazuje skuteczność domyślnych augmentacji.
- **YOLOv8s (aug-params):** Augmentacje (rotacja, CLAHE, itp.) poprawiły mAP@50:95 do 0.76688, ale wyższe straty (Box Loss=0.75699) wskazują na potrzebę dalszej optymalizacji.
- **YOLOv8m Last:** Z dodatkowymi 1046 obrazami augmentowanymi mAP@50:95=0.76647, ale niższe straty (Box Loss=0.47121) sugerują lepszą stabilność większego modelu.
- **YOLOv8s Final:** Finalny model osiągnął mAP@50:95=0.76237, balansując precyza (0.9224) i recall (0.8985). Straty (Box Loss=0.72337) są umiarkowane, co potwierdza wybór tego modelu.

Wnioski:

- Rozszerzanie datasetu (2600 + 2500 + 1046) i augmentacje zwiększyły mAP@50:95 z 0.66727 do 0.76237.
- Początkowo model YOLOv8s z domyślnymi ustawieniami i augmentacjami okazał się optymalnym wyborem pod względem stosunku wydajności do rozmiaru, osiągając mAP@50:95=0.77851. Jednak finalnie wybrano model YOLOv8s Final, trenowany na rozszerzonym datasetcie (łącznie 6729 obrazów), aby lepiej wykorzystać większą różnorodność danych. Chociaż mAP@50:95 spadł do 0.76237, model zachowuje wysoką wydajność (precyza 0.9224, recall 0.8985) przy umiarkowanych stratach (Box Loss=0.72337). Wybór ten wynika z faktu, że dalsze optymalizacje najlepszego modelu (np. YOLOv8s z mAP@50:95=0.77851) byłyby czasochłonne, a YOLOv8s Final oferuje wystarczającą dokładność przy zachowaniu kompaktowego rozmiaru, co jest kluczowe dla wdrożenia na urządzeniach mobilnych (np. Android).

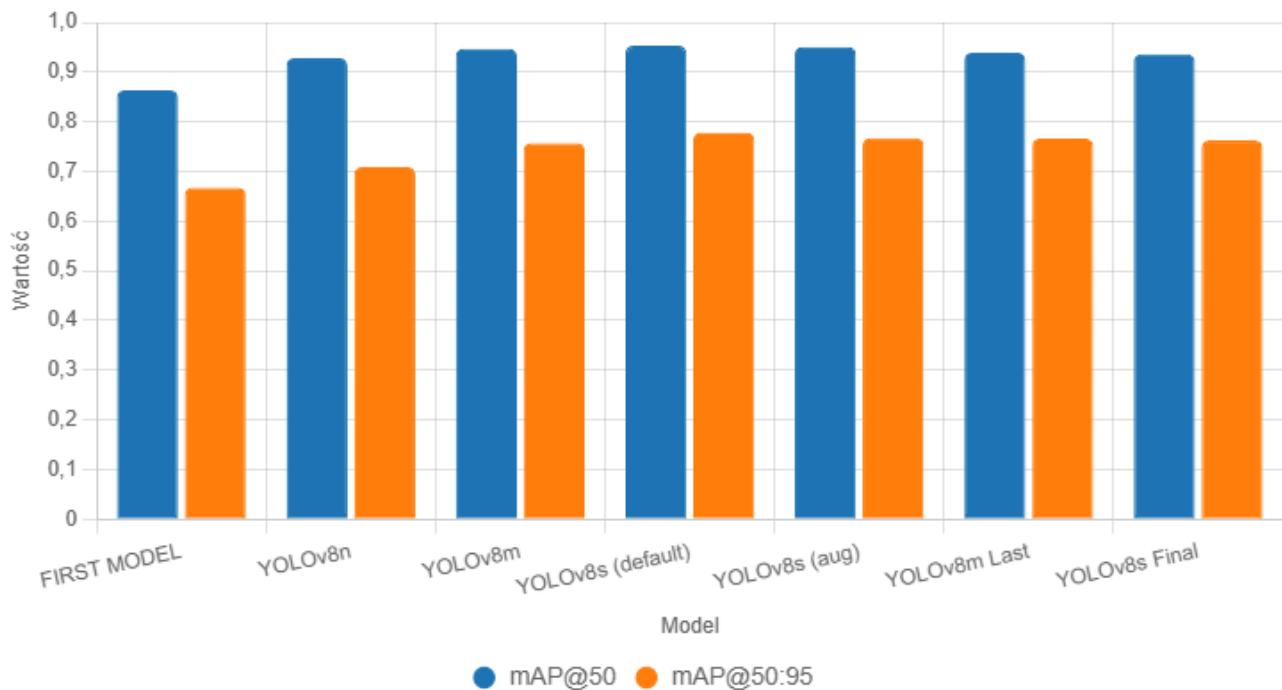


Wykres 1. Szczegółowa analiza zestawu danych treningowych z adnotacjami



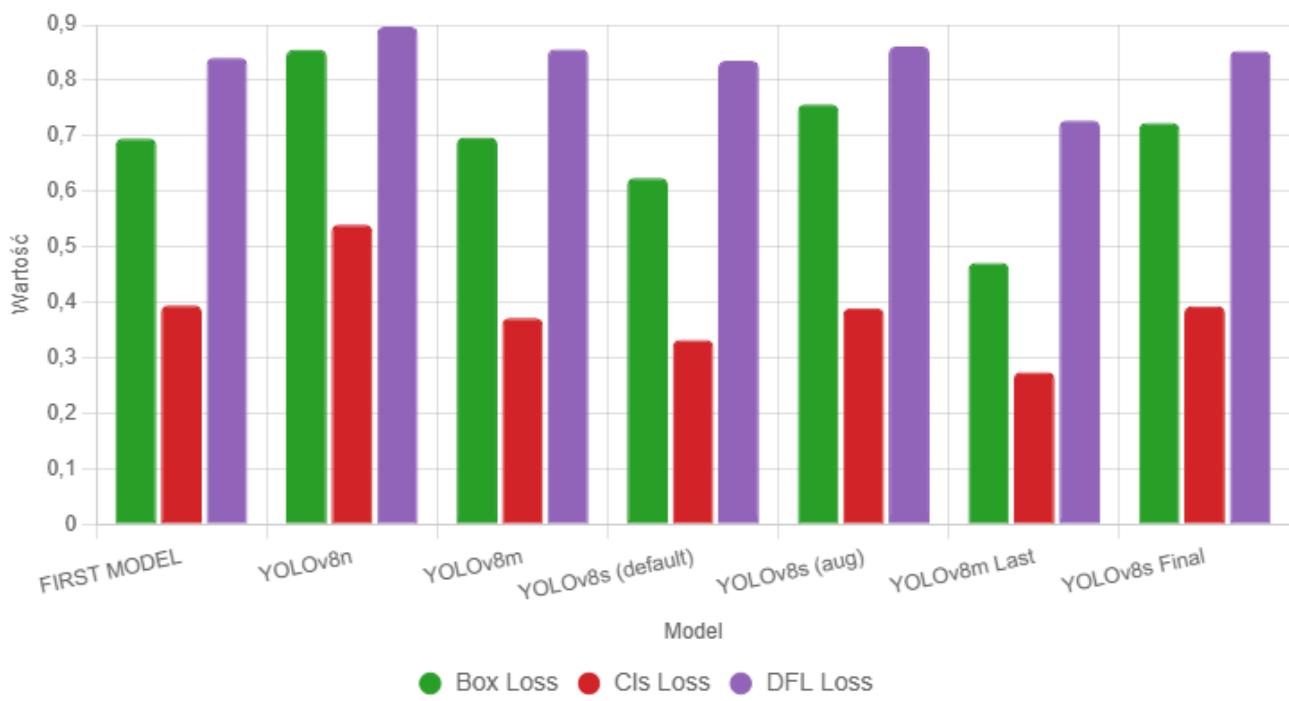
Wykres 2. Przykładowy train batch

Porównanie mAP@50 i mAP@50:95



Wykres 3. Porównanie mAP@50 i mAP@50:95

Porównanie Strat

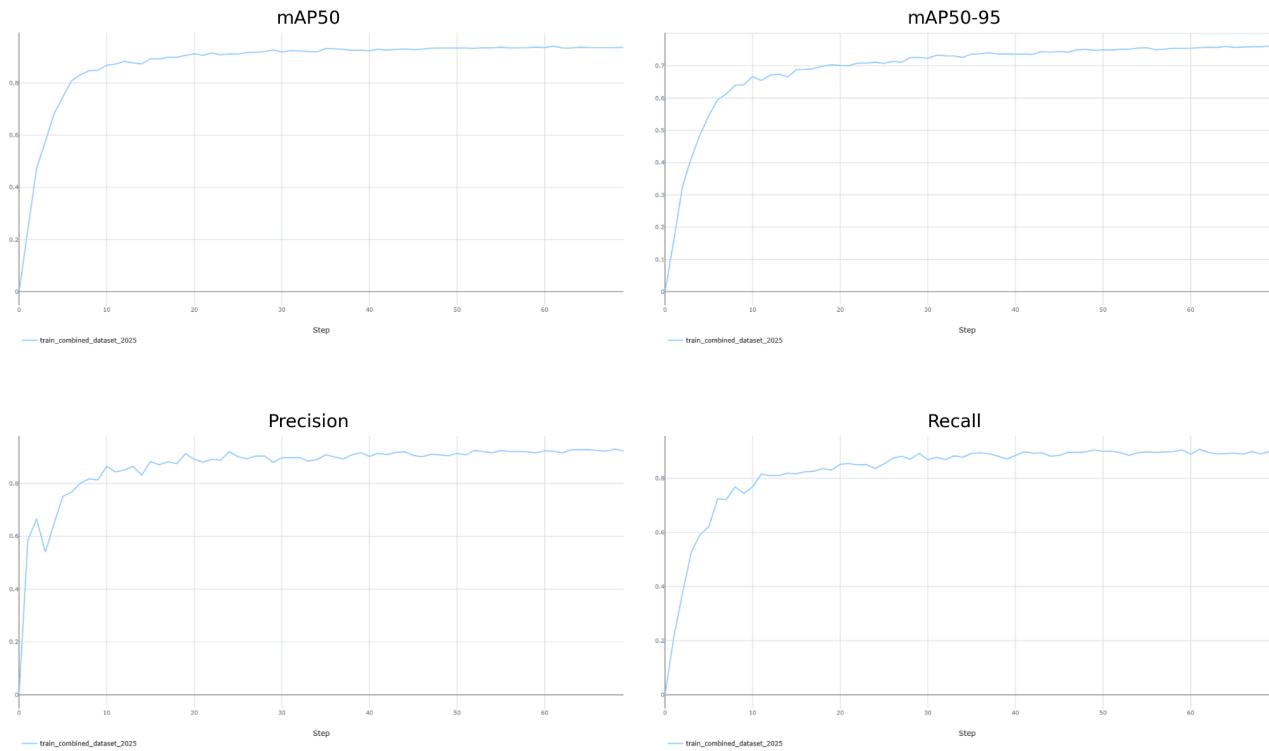


Wykres 4. Porównanie Strat (Box Loss, Cls Loss, DFL Loss)

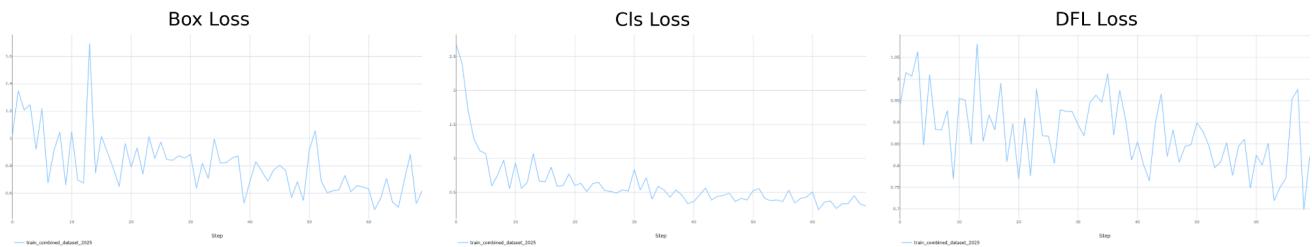
Model osiągnął wysoką dokładność detekcji ($\text{mAP}@50:95 = 0.76237$, zakładany cel projektu), co potwierdza skuteczność augmentacji i przetrenowanego modelu yolov8s.pt.

Wysoka wartość mAP@50 wskazuje na niezawodne wykrywanie znaków w większości przypadków, podczas gdy mAP@50:95 sugeruje dobrą jakość lokalizacji bounding boxów. Precision i recall wskazują na zrównoważoną zdolność modelu do minimalizowania fałszywych pozytywów i negatywów.

Końcowe wykresy modelu sdv4:



Wykres 5. Metryki na zbiorze walidacyjnym (model sdv4)



Wykres 6. Straty podczas treningu (model sdv4)

6. Zastosowanie

Aplikacja mobilna

- **Platforma:** Android.
- **Funkcjonalność:**
 - **Detekcja na zdjęciach:** Użytkownik wczytuje obraz, a model generuje bounding boxy i opisy znaków.

- **Detekcja w czasie rzeczywistym:** Model przetwarza strumień wideo z kamery, wyświetlając bounding boxy i opisy w czasie rzeczywistym.
- **Format modelu:** TFLite (float32), skonwertowany z ONNX dla efektywnej inferencji na urządzeniach mobilnych.
- **Integracja:** Model TFLite zintegrowany z aplikacją za pomocą TensorFlow Lite Interpreter, z użyciem Kotlin w Android Studio.

7. Ograniczenia

7.1 Wyzwania datasetu

- **Niebalansowanie klas:** Niektóre rzadkie znaki mogą mieć mniej przykładów w danych treningowych, co obniża precyzję dla tych klas.
- **Ekstremalne warunki:** Model może mieć trudności z wykrywaniem znaków w bardzo złych warunkach pogodowych (np. gęsta mgła, silny śnieg) z powodu ograniczonej liczby takich obrazów w danych.

7.2 Ograniczenia modelu

- **Rozmiar modelu:** YOLOv8s, choć zoptymalizowany, może być zbyt zasobocherenny dla starszych urządzeń Android z ograniczoną mocą obliczeniową.
- **Czas inferencji:** Na urządzeniach bez GPU inferencja może spaść poniżej 20 FPS, co wpływa na płynność w trybie real-time.

7.3 Potencjalne problemy

- **Falszywe pozytywy:** Model może błędnie klasyfikować obiekty podobne do znaków jako znaki drogowe.
- **Generalizacja:** Model trenowany głównie na polskich znakach, może wymagać dodatkowego dostrojenia dla innych regionów z różnymi stylami znaków.

8. Wnioski

8.1 Osiągnięcia

- Model YOLOv8s sdv4 osiągnął wysoką dokładność detekcji ($mAP@50:95 > 0.75$), spełniając założenia projektu.
- Zastosowanie augmentacji (Albumentations, mosaic) poprawiło generalizację modelu w zróżnicowanych warunkach.
- Lokalny serwer MLFlow umożliwił efektywne śledzenie eksperymentów, z trwałym przechowywaniem danych w runs/mlflow.

8.2 Lekcje wyciągnięte

- **Zarządzanie danymi:** Wczesne zabezpieczenie folderu runs/mlflow (np. kopie zapasowe) zapobiega utracie wyników.
- **Augmentacje:** CLAHE i RandomBrightnessContrast znaczaco poprawiły wydajność w trudnych warunkach oświetleniowych.
- **Optymalizacja:** Batch size=16 był kluczowy dla treningu na GPU z 8 GB VRAM, zapewniając stabilność.
- **MLFlow:** Lokalna konfiguracja (file:runs/mlflow) jest wystarczająca dla małych projektów.