Dokumentacja

Kacper Sokół

Projekt ma na celu stworzenie systemu wykrywania i identyfikacji zwierząt w polskich lasach przy użyciu frameworku Yolo z biblioteką ultralytics. W zamyśle program ma współpracować z danymi z kamer fotopułapek i dronów, system ma umożliwić automatyczną analizę obrazów i filmów w celu rozpoznawania różnych gatunków zwierząt w ich naturalnym leśnym środowisku. Rozwiązanie to ma znaleźć zastosowanie w licznych obszarach, pomoże w monitorowaniu populacji dzikich zwierząt, śledzeniu ich zachowań oraz identyfikowaniu obszarów o szczególnym znaczeniu ekologicznym. System wykrywania zwierząt będzie mógł analizować różne warunki środowiskowe od dziennych ujęć, poprzez nocne nagrania w podczerwieni, aż po trudne warunki atmosferyczne co sprawi, że stanie się narzędziem uniwersalnym i niezawodnym. Dzięki integracji technologii z naturalnym środowiskiem możliwe będzie efektywniejsze zarządzanie zasobami leśnymi, śledzenie migracji zwierząt, ich siedlisk oraz identyfikacji obszarów szczególnie cennych przyrodniczo. Rozwiązanie to ma zapewnić przeciwdziałanie kłusownictwu oraz ochronie zagrożonych gatunków.

Korzyści wynikające z realizacji projektu:

- **Efektywne monitorowanie populacji zwierząt:** Dzięki automatycznemu wykrywaniu i identyfikacji możliwe będzie szybkie pozyskiwanie wiarygodnych danych o stanie fauny leśnej.
- **Reakcja na zmiany środowiskowe:** System pozwoli na wczesne wykrywanie zmian w ekosystemach, które mogą wymagać interwencji, takich jak spadek liczebności gatunków czy zmiany w ich zachowaniu.
- **Oszczędność czasu i zasobów:** Automatyzacja procesu monitorowania przyrody zminimalizuje potrzebę czasochłonnego przeglądania materiałów wideo przez ludzi.
- **Współpraca międzysektorowa:** Projekt umożliwi wymianę danych między instytucjami naukowymi, organizacjami ekologicznymi oraz instytucjami zarządzającymi lasami.

Cele tego projektu są bardzo zróżnicowane zaczynając od:

Zautomatyzowane rozpoznawanie zwierząt
Opracowanie algorytmów zdolnych do identyfikacji różnych gatunków zwierząt na podstawie analizy zdjęć i nagrań

Ochrona gatunków zagrożonych
Umożliwienie skutecznego monitorowania populacji rzadkich i chronionych gatunków,.

Zwalczanie kłusownictwa i nielegalnych działań w lasach
Wdrożenie mechanizmów umożliwiających szybkie wykrywanie nietypowej aktywności w lasach, takiej jak nielegalne polowania, co przyczyni się do ograniczenia kłusownictwa i poprawy bezpieczeństwa zwierząt.
Wsparcie dla leśników i ekologów

System dostarczy dane niezbędne do zarządzania zasobami leśnymi, w tym planowania działań w zakresie gospodarki leśnej i ochrony przyrody.

Rozwój badań naukowych
Projekt stworzy szeroką bazę danych, która będzie wykorzystywana w badaniach nad zachowaniem zwierząt.

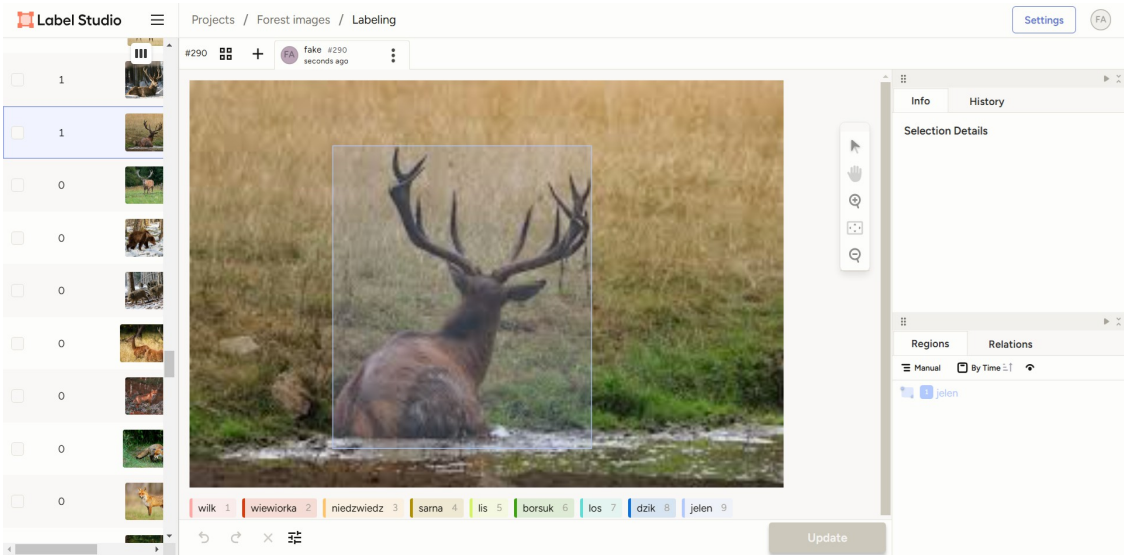
Zaimplementowano następujące gatunki zwierząt:
-Wilk
-Wiewiórka
-Sarna
-Jeleń
-Łoś
-Borsuk
-Dzik
-Lis
-Niedźwiedź

Przebieg projektu

1. Zbieranie zdjęć potrzebnych do trenowania modelu. Wszystkie są zebrane z grafik Google na wolnych licencjach. Przygotowanych jest około 20 zdjęć na każdy gatunek zwierzęcia oraz kilka zdjęć łączonych aby zapewnić dobre warunki do trenowania modelu.
2. Zaznaczanie na zdjęciach odpowiednich klas w programie Label Studio
3. Przygotowanie środowiska oraz potrzebnych skrytpów do trenowania modelu
4. Trenowanie modelu
5. Eksport wytrenowanego modelu na maszynę do głębszych testów

Zaznaczanie klas

Zaznaczanie poszczególnych klas zrealizowałem w środowisku ..Label studio".
Uruchomiłem je lokalnie na mojej maszynie. Przebieg tego wyglądał następująco:



Każde zdjęcie dostało swoje poprawne klasy(około 180 zdjęć). Uruchomienie strony lokalnie przebiegało przy pomocy anaconda promt, gdzie został utworzony cały proejkt

Następnie weksportowałem w formacie .zip, który zawiera obrobione zdjęcia oraz nazwy klas(images, labels, classes).

Katalog z gotowymi zdjęciami mieści się w wyższym katalogu „yolo" gdzie trzymane będą wszystkie potrzebne informacje i skrypty.

Trenowanie modelu

Trenowanie modelu odbędzie w następujących krokach:

1. Upload danych do środowiska google collab.
2. Rozpakowanie wcześniej przygotowanych danych
    1. unzip -q /content/data.zip -d /content/custom_data
3. Przygotowanie odpowiedniej struktury katalogowej, której wymaga ultralytics
    1. Train: To są rzeczywiste obrazy używane do trenowania modelu. W jednej epoce uczenia każdy obraz w składzie jest przekazywany do sieci neuronowej. Algorytm uczący dostosowuje wagi sieci do danych na obrazach.
    2. Validation: obrazy te służa do sprawdzania wydajności modelu na koniec każdej epoki szkoleniowej.
    3. W pliku „katalogi.py" znajduje się kod, który tworzy potrzebne katalogi oraz losowo rozdziela 90% danych do katalogu treningowego oraz 10% do walidacyjnego
4. Instalacja biblioteki do trenowania modelu ultralytics
5. Przygotowanie środowiska treningowego ultralytics
    1. Przygotowanie pliku konfiguracyjnego plik YAML. Plik ten będzie określał lokalizację potrzebnych plików(train, validation) i zdefiniuje klasy modelu. Kod ten znajduje się w pliku „konfig.py”
6. Trenowanie modelu
    1. Parametry konfiguracyjne trenowania:
        1. YOLO model → Głównie wykorzystywałem model yolo11 oraz jego różne wersje zaczynając do modelu s kończąc na l
        2. Liczba epok → W uczeniu maszynowym jedna „epoka” to pojedyncze przejście przez pełny zestaw danych szkoleniowych. Ustawienie liczby epok decyduje o tym, jak długo model będzie się uczył. Dla mojego modelu zacząłem od 60 epok a zakończyłem na 120.
        3. Rozmiary obrazów → Dla każdej próby rozmiar obrazów został ustawiony na 640x640.

Pierwsza próba trenowania modelu

!yolo detect train data=/content/data.yaml model=yolo11s.pt epochs=60 imgsz=640

Nie mogę powiedzieć, że pierwszy test był porażka w 100%, jednak nie prezentował on zamierzanych efektów.
Przedstawienie wyników pierwszego trenowania:

```
               all         18         25      0.662      0.828      0.747      0.623

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    115/120        11G      0.255     0.2691     0.8616         17        640: 100% 10/10 [00:06<00:00,  1.45it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  4.19it/s]
                   all         18         25      0.636      0.829      0.745      0.621

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    116/120        11G     0.2514     0.2313     0.8599         15        640: 100% 10/10 [00:07<00:00,  1.38it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  3.15it/s]
                   all         18         25      0.532      0.829      0.688      0.574

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    117/120        11G     0.2395     0.2112     0.8501         16        640: 100% 10/10 [00:06<00:00,  1.47it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  3.53it/s]
                   all         18         25      0.585      0.799      0.691      0.562

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    118/120        11G     0.2268     0.2242     0.8416         15        640: 100% 10/10 [00:07<00:00,  1.41it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  3.96it/s]
                   all         18         25      0.536      0.765      0.665      0.541

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    119/120        11G        0.245      0.211     0.8468         17        640: 100% 10/10 [00:07<00:00,  1.42it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  3.28it/s]
                   all         18         25      0.533      0.816      0.667      0.553

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    120/120        11G     0.2211     0.1957     0.8009         15        640: 100% 10/10 [00:06<00:00,  1.45it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  4.16it/s]
                   all         18         25      0.537      0.811      0.667      0.559
```

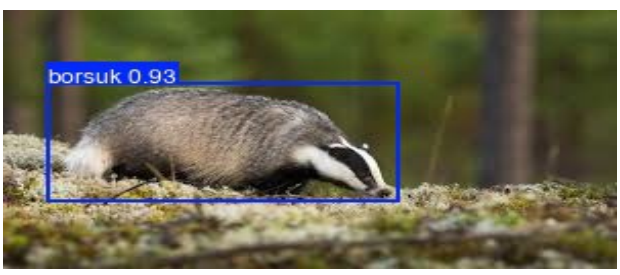
Do zoobrazowania kilku zdjęć wytrenowanych przez model użyłem następujących komend:

!yolo detect predict model=runs/detect/train/weights/best.pt source=data/validation/images save=True

Kod do wyświetlania:

```
import glob
from IPython.display import Image, display
for image_path in glob.glob(f'/content/runs/detect/predict/*.jpg')[:10]:
  display(Image(filename=image_path, height=400))
  print('\n')
```

Początek trenowanego modelu był dobry:



Jednak dalej model nie radził sobie aż tak dobrze:

A czasami nie dawał żadnych znaków:
Na tym zdjęciu model nic nie wykrył

Wnioski:
Potencjalnych problemów dla tego modelu może być dużo, jednym z nich może być liczba zdjęć, dokładniej, że jest ich za mało(w tej próbie trenowania liczba zdjęć to była 10 na zwierzę + kilka łączonych). Problemem mógł być sam model, który był za mały lub niewystarczająca liczba epok, a nawet nieodpowiednie dobranie klas.

Druga próba trenowania modelu

Przed podejściem do drugiej próby trenowania mojego modelu dokonałem kilku poprawek, przede wszystkim sprawdziłem czy klasy były poprawnie zaznaczone na zdjęciach, oraz spróbowałem zwiększenie liczby epok do 100.

Niestety i tym razem rezultaty nie były zadowalające, efekty modelu był praktycznie taki sam jak w przypadku poprzedniej próby trenowania. Zdjęcia przedstawione w pierwszej próbie trenowania modelu zachowywały się tak samo przy drugiej próbie.

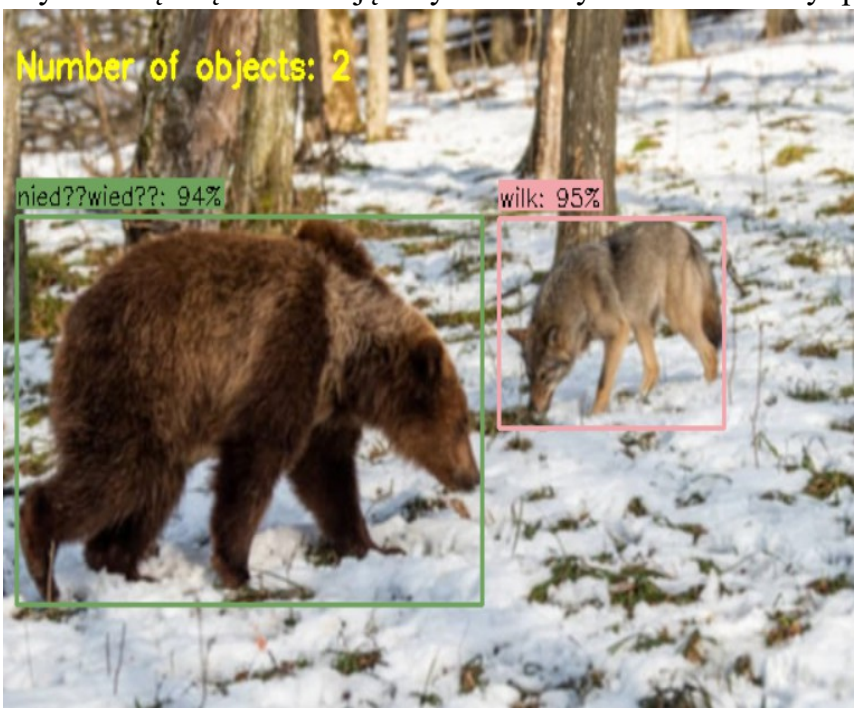
Wnioski:

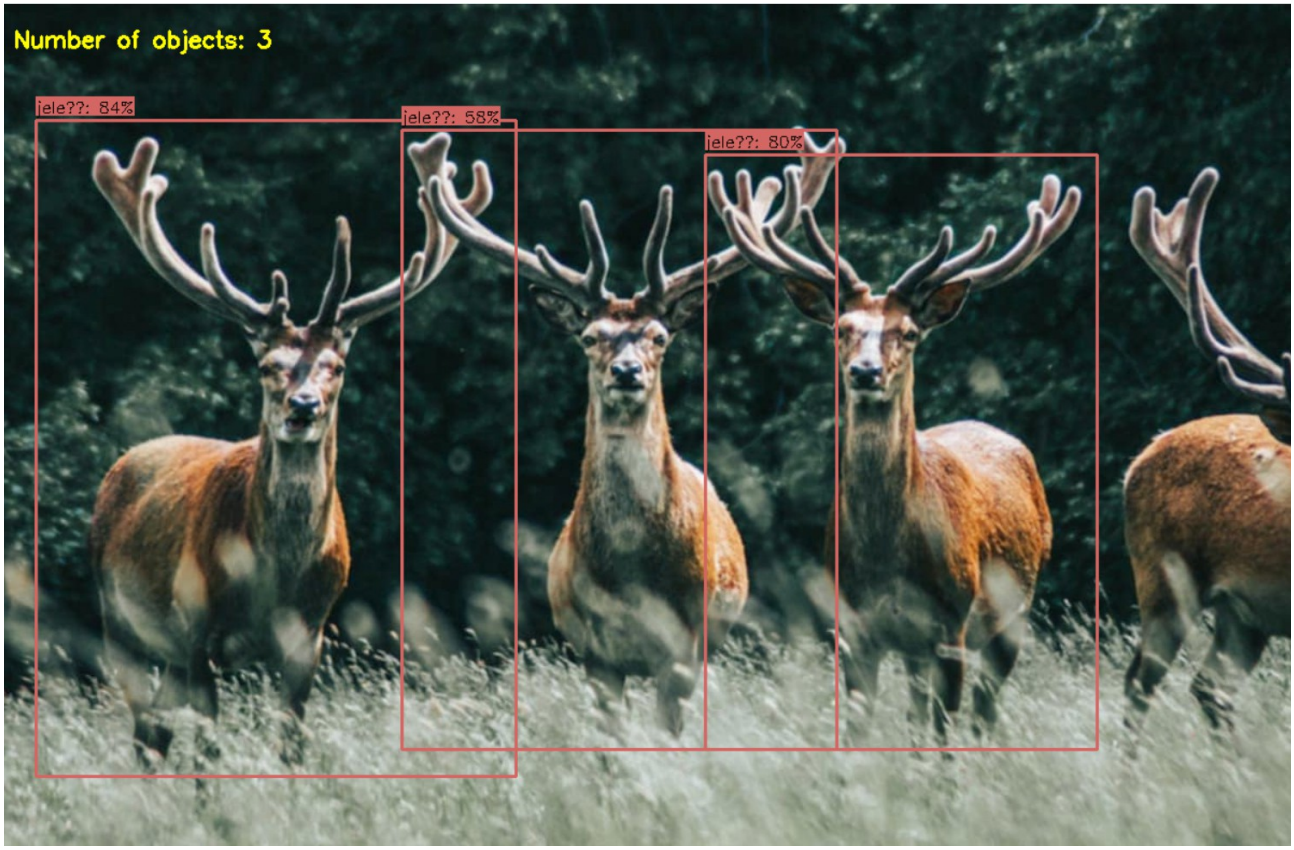
Po dokładnym sprawdzeniu zdjęć ich zaznaczone klasy są poprawne więc problem może być w ich ilości lub ponownie w wielkości modelu i/lub liczbie epok.

Trzecia próba trenowania modelu

W tej próbie postawiłem na większy model treningowy oraz na zwiększoną liczbę epok. Model w tej próbie to yolo11l oraz 120 użytych epok. Liczba zdjęć niezmienna.

Ten model okazał się o wiele większym postępem projektu niż poprzednia próba. Zdecydowaną większość zdjęć wytrenowany model zaznaczył poprawnie:





Jak widać na załączonych obrazkach model radził sobie już przyzwoicie, jednak zdarzały się dalej pewne błędy jak tutaj:

Gdzie model zaznaczał dwie klasy na jednym zwierzęciu,

Wnioski:

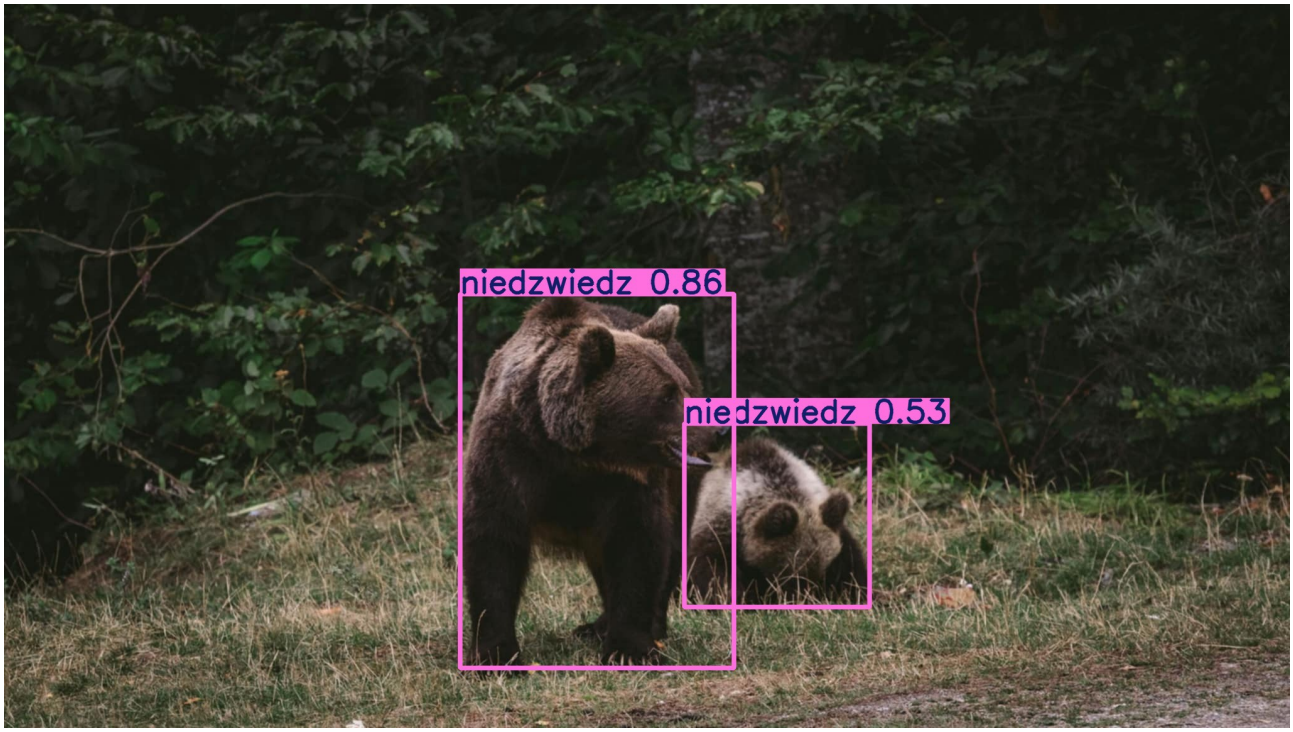
Po tej próbie wydaję mi się, że model oraz liczba epok jest odpowiednia dla tych wymagań, klasy na zdjęciach są również poprawnie zaznaczone, więc dla kolejnej próby będzie potrzeba zwiększenia liczby zdjęć na każde zwierzę.

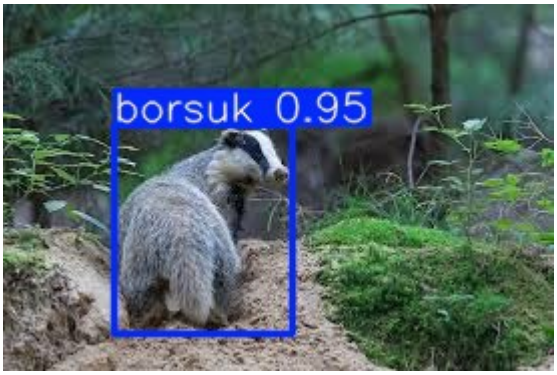
Czwarta próba trenowania modelu

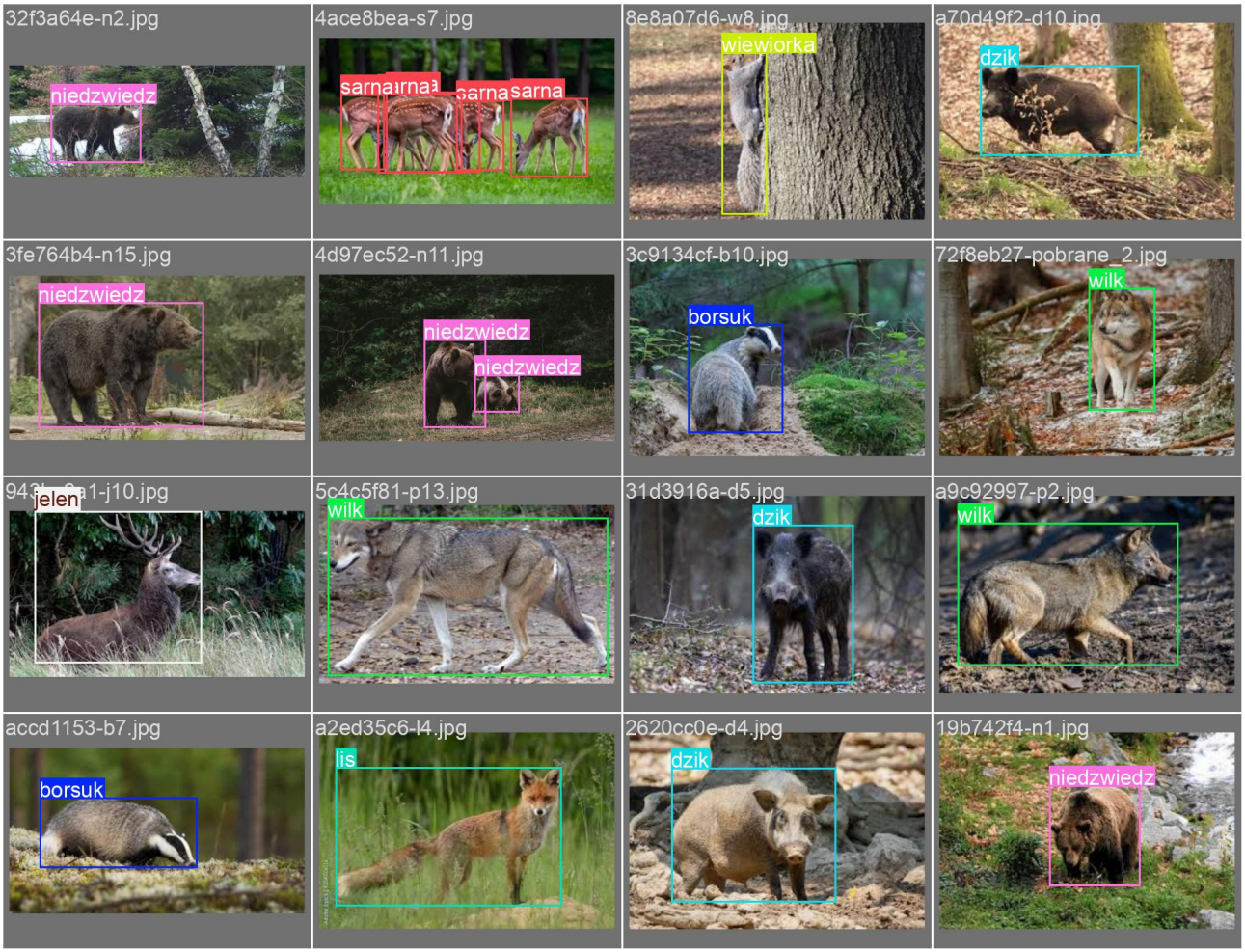
W tym podejściu do trenowania modelu, jego wersja jak i liczba epok zostaje bez zmian, jednak teraz liczba zdjęć na każde zwierzę wynosi 20 oraz dodatkowo kilka połączonych zdjęć gdzie jest kilka na jednym obrazku, łącznie daje to około 200 zdjęć.

Po dodaniu klas na nowe zdjęcia przeszedłem do trenowania nowego modelu.

Efekty wytrenowanego modelu były w końcu zadowalające:







Wnioski:
Jak widać, model już poprawnie rozpoznaje zwierzęta w większości z wysokim prawdopodobieństwem. Co potwierdza wcześniej przedstawione wnioski, że problemem mogła być niewystarczająco duża liczba zdjęć oraz dobranie odpowiedniego modelu.

Eksport wytrenowanego modelu na maszynę lokalną

Skrypt do spakowania modelu:

```
!mkdir /content/my_model
!cp /content/runs/detect/train/weights/best.pt /content/my_model/my_model.pt
!cp -r /content/runs/detect/train /content/my_model

%cd my_model
!zip /content/my_model.zip my_model.pt
!zip -r /content/my_model.zip train
%cd /content
```

Testowanie modelu na lokalnej maszynie

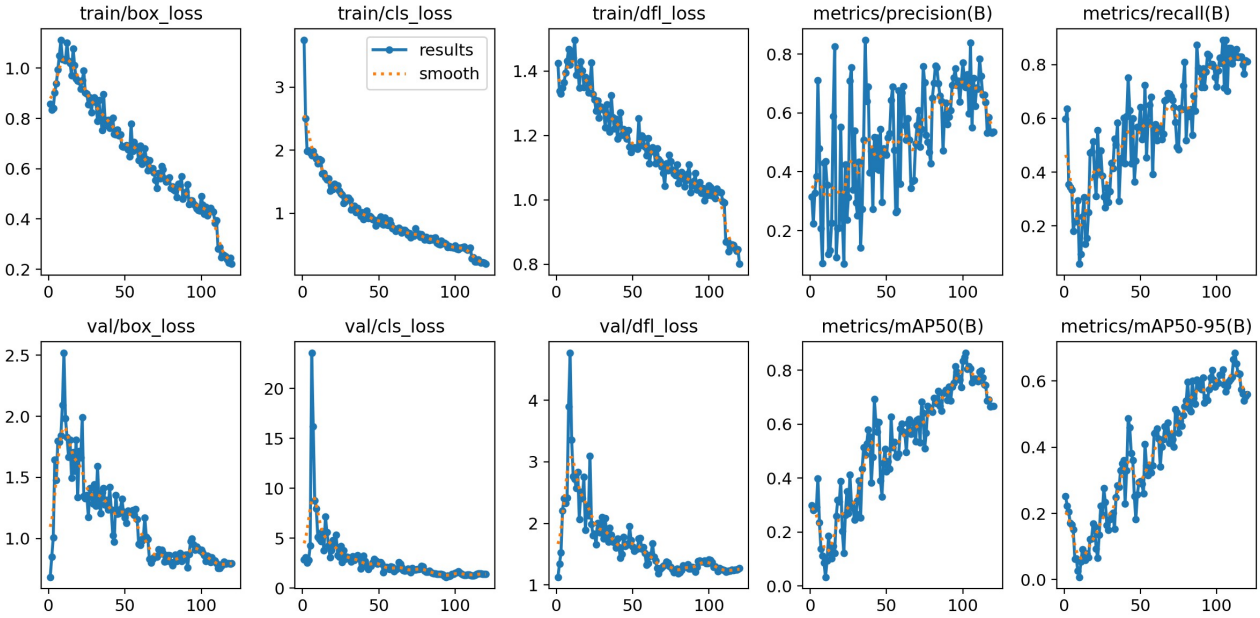
Po pobraniu i rozpakowaniu modelu przy pomocy anaconda promt, w stworzonym środowisku, przechodzę do lokalizacji pobranego modelu i uruchamiam skrypt python „yolo_detect.py":

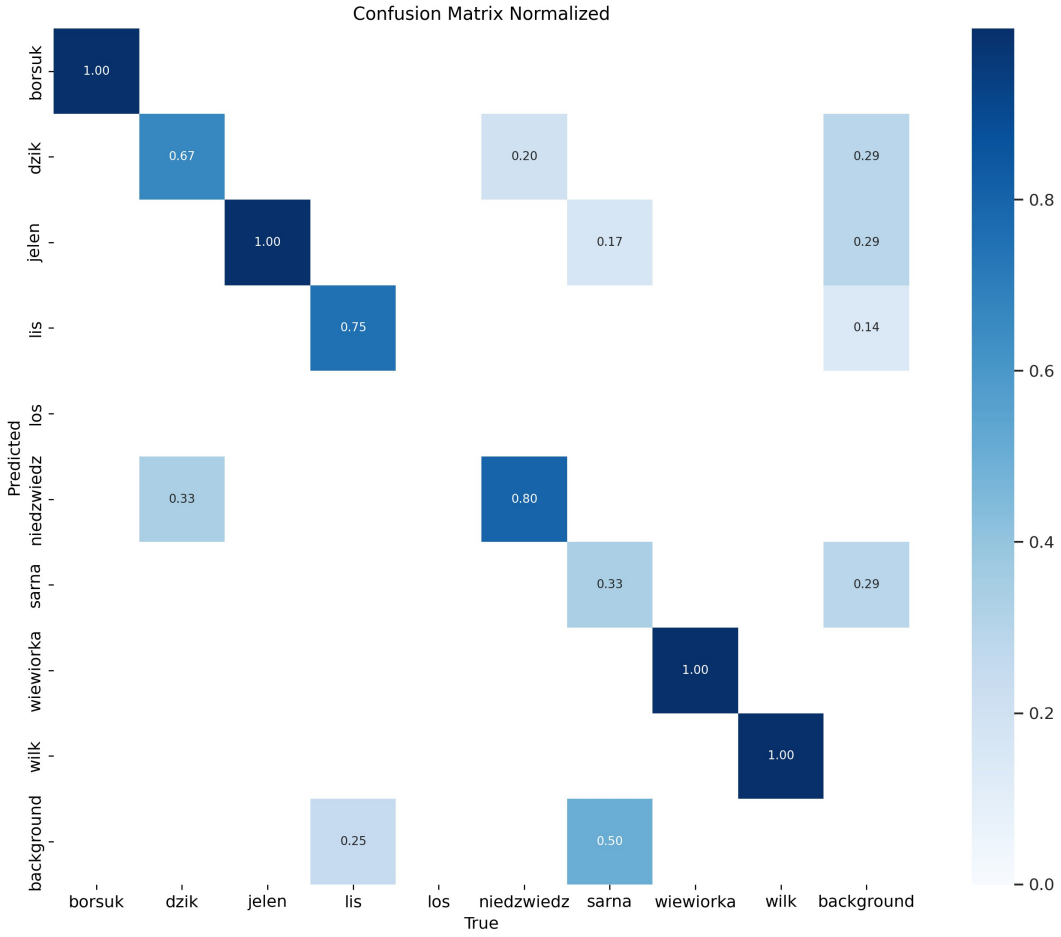
Python yolo_detect.py – –model my_model.pl – –source (nazwa wraz z rozszerzeniem do zdjęcia, filmu, czy kamerki)

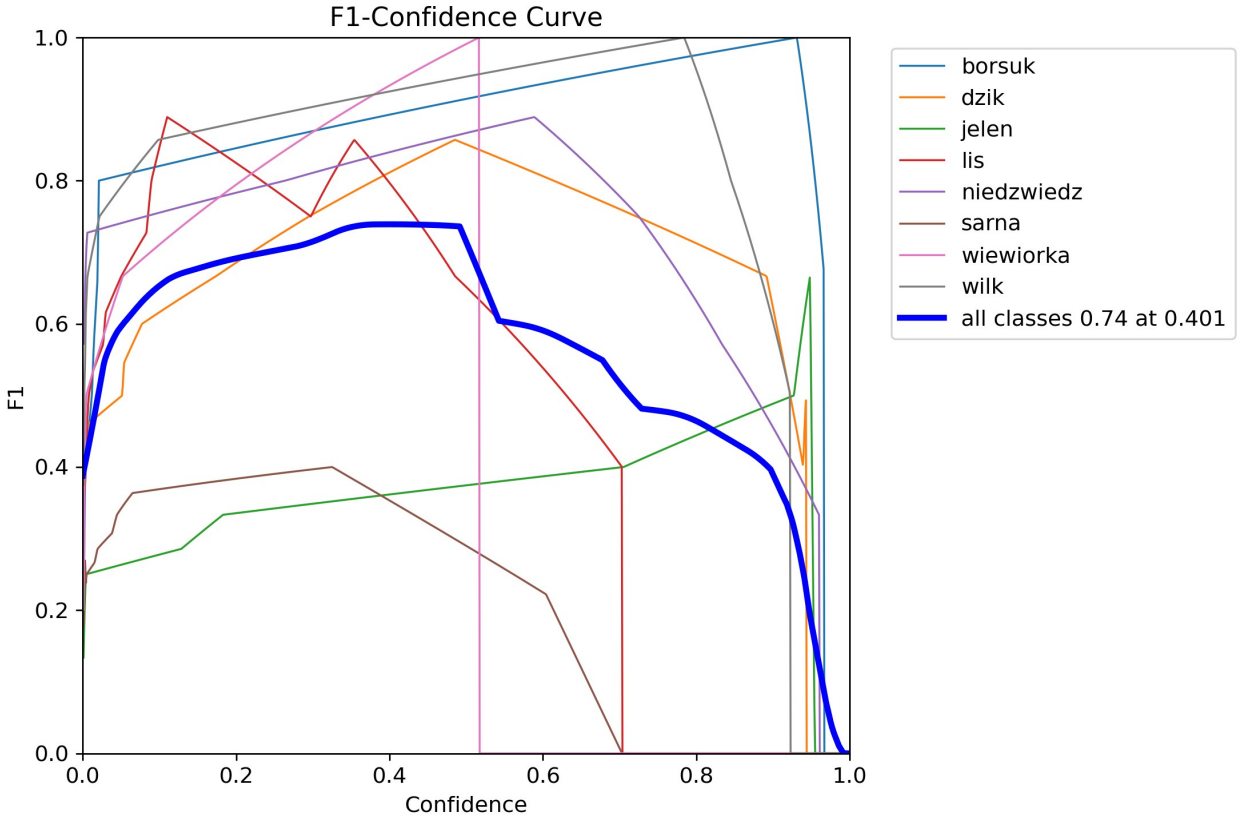
Niektóre z testów wykonanych na lokalnej maszynie:

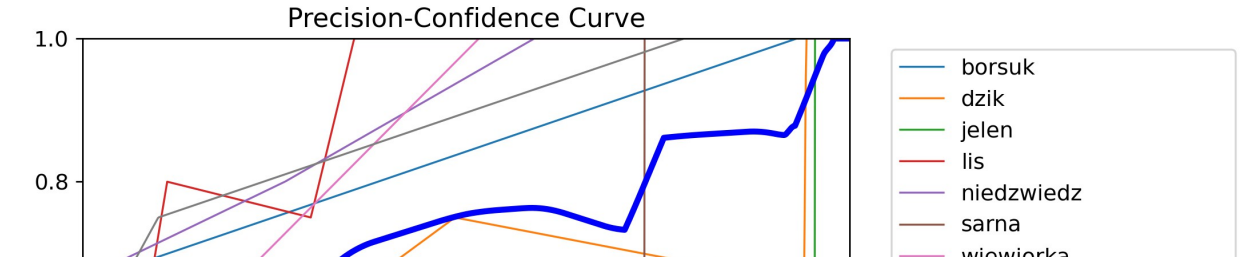
Model działał również idealnie podczas przesłania filmiku, jednak ciężko pokazać to, ponieważ miałem 2-3 FPS podczas niego.

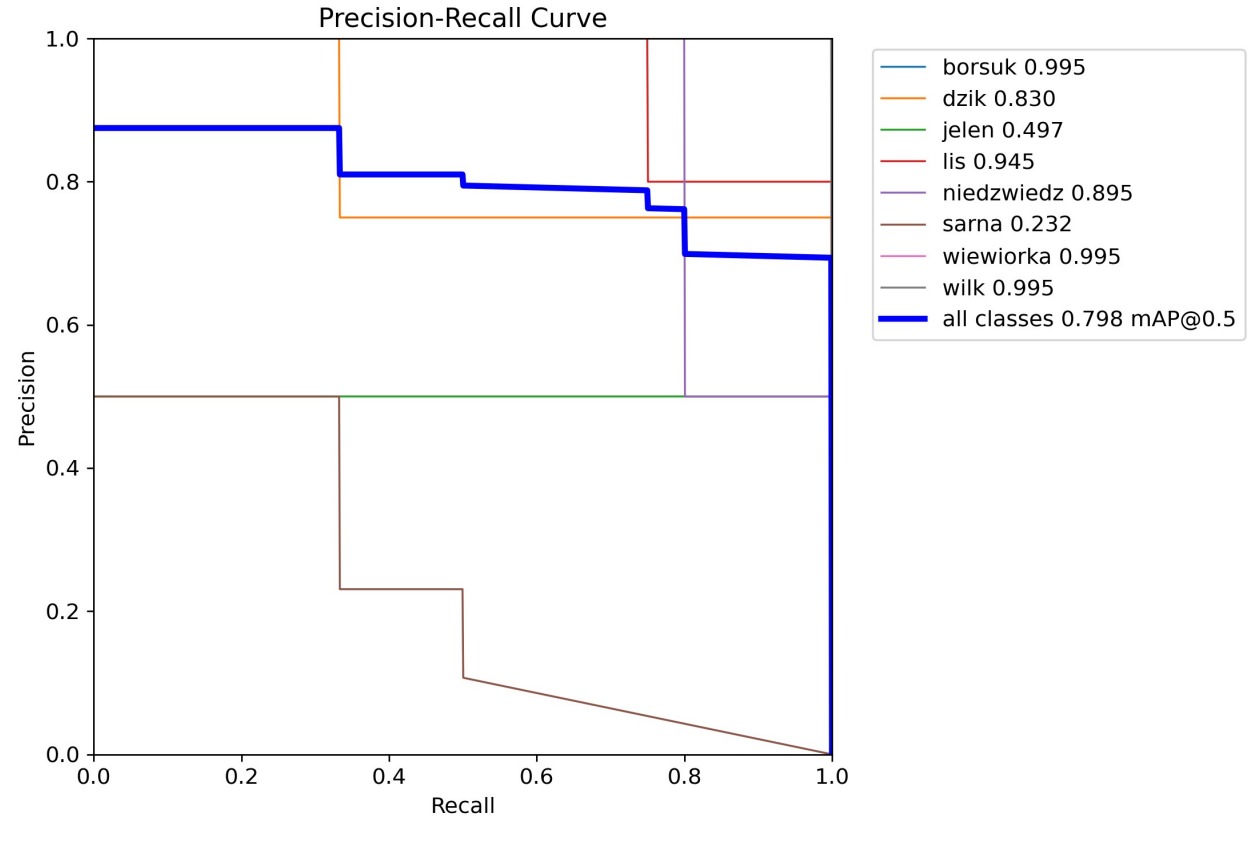
Wykresy wygenerowane podczas trenowania

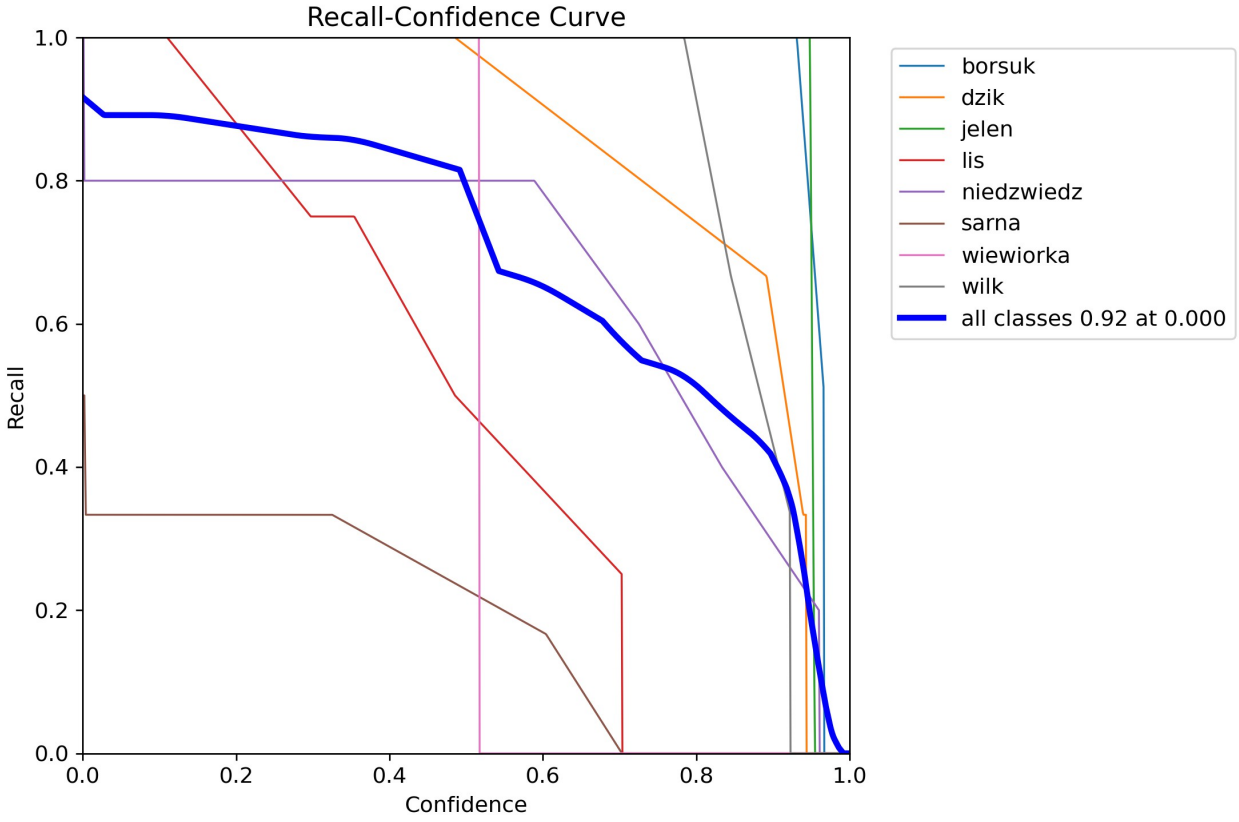


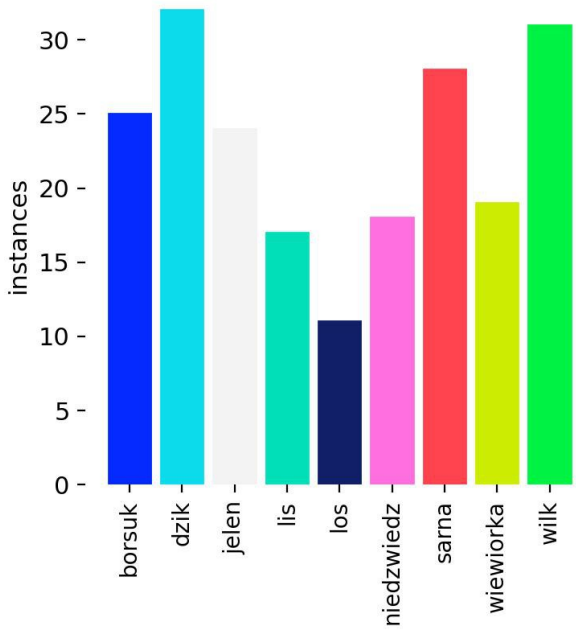

Confusion Matrix Normalized
borsuk
dzik
jelen
lis
los
niedzwiedz
sarna
wiewiorka
wilk
background
Predicted
True
1.00
0.67
0.20
0.29
1.00
0.17
0.29
0.75
0.14
0.33
0.80
0.33
0.29
1.00
1.00
0.25
0.50
0.0
0.2
0.4
0.6
0.8

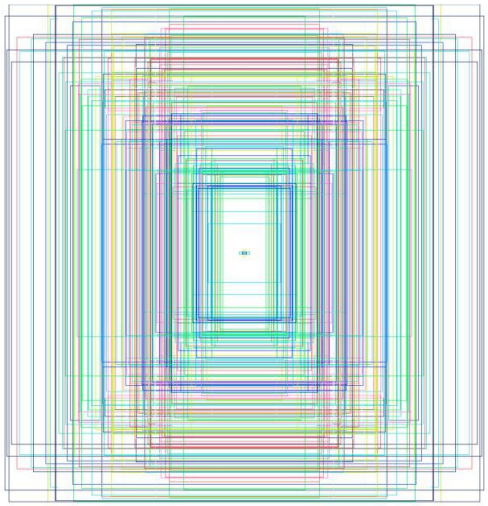

F1-Confidence Curve
F1
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
borsuk
dzik
jelen
lis
niedzwiedz
sarna
wiewiorka
wilk
all classes 0.74 at 0.401

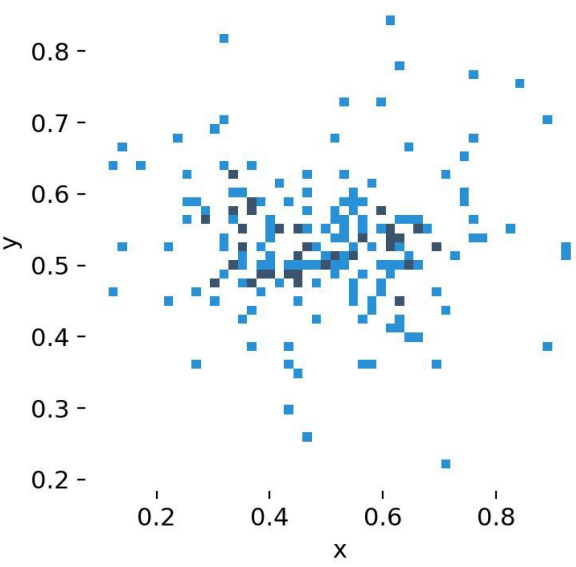

Precision-Confidence Curve
0.8
1.0
borsuk
dzik
jelen
lis
niedzwiedz
sarna

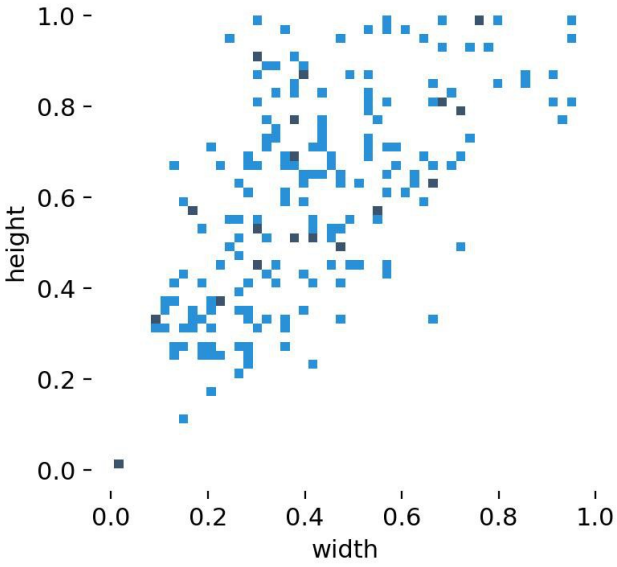

Precision-Recall Curve
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
borsuk 0.995
dzik 0.830
jelen 0.497
lis 0.945
niedzwiedz 0.895
sarna 0.232
wiewiorka 0.995
wilk 0.995
all classes 0.798 mAP@0.5


Recall-Confidence Curve
Recall
Confidence
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
borsuk
dzik
jelen
lis
niedzwiedz
sarna
wiewiorka
wilk
all classes 0.92 at 0.000


instances
30
25
20
15
10
5
0
borsuk
dzik
jelen
lis
los
niedzwiedz
sarna
wiewiorka
wilk




y
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.2
0.4
0.6
0.8
x


height
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
width

Wnioski końcowe

Poprawne wytrenowanie modelu wymaga zwracania ogromnej uwagi na wszelkie szczegóły oraz na dogłębną analizę potrzeb. Dla mojego projektu największym problemem okazała się za mała liczba zdjęć dla jednego zwierzęcia, model nie miał wystarczającej bazy trenującej dla tylu klas i przez to popełniał błędy. Po zwiększeniu liczby zdjęć precyzja wykrywania była na bardzo wysokim poziomie wykrywając praktycznie wszystkie zwierzęta w przedziale (85%-90%+), co jest bardzo dobrym wynikiem, jak na bazę w okolicach 200 zdjęć. Gdyby zwiększyć liczbę zdjęć w bazie model świetnie radził by sobie w wykrywaniu nawet gdy obrazy były by zniekształcone przez deszcz czy śnieg, gdzie już teraz potrafił poprawnie wykryć zwierzę w śniegu. Kluczową rolę odegrała również liczba epok(120), która dała dobre efekty a zwiększenie modelu na yolo11L tylko poprawiło wyniki. Z pewnością ograniczenie się do mniejszej ilości klas o takiej samej liczbie zdjęć zwiększyło by poprawne wykrywanie modelu do jeszcze wyższych parametrów, jednak wytrenowanie modelu dla większej ilości klas dało szersze zastosowanie projektu w celach przedstawionych na początku.