Wydział Fizyki i Informatyki Stosowanej AGH

Uczenie maszynowe
projekt grupowy

Zgłoszenie tematu

Określenie niepewności przewidywań modelu YOLO na ogólnie dostępnych benchmarkach.

Autorzy:
Jedrzej Szostak, **Piotr Libucha Mariusz Biegański**

19 czerwca 2023

1 Cel i zakres projektu

Celem projektu jest określenie niepewności przewidywań modelu YOLO w zakresie liczby i poprawności wykrytych obiektów.

2 Oprogramowanie

YOLO może działać w oparciu o bibliotekę Darknet albo o Tensorflow
https://pjreddie.com/darknet/yolo/

3 Dane wejściowe

Należy zgromadzić jak największy zbiót otagowanych danych. Jako pukt wyjścia zostaną wzięte standardowe benchmarki używane dla YOLO.

4 Kamienie milowe projektu

4.1 1-sze 2 tygodnie

Implementacja i testy eksploracyjne bazowej wersji modelu. Rozpoczęliśmy od próby wykorzystania YOLO3 na platformie darknet. Zajęło nam to sporo czasu. Ostatecznie się udało, lecz biorąc pod uwagę problemy, które pojawiły się po drodze (problem z dołączaniem bibliotek i wiele innych). Doszliśmy do wniosku, że musimy zmienić platformę. Wybór padł na YOLO5 dostarczone przez Ultralytics.

https://github.com/ultralytics/yolov5

Był to kompromis między dostępnością różnego rodzaju pomocy/instrukcji oraz dokładnością modelu. (Jedrzej Szostak 33% Piotr Libucha 33% Mariusz Bieganski 33%)

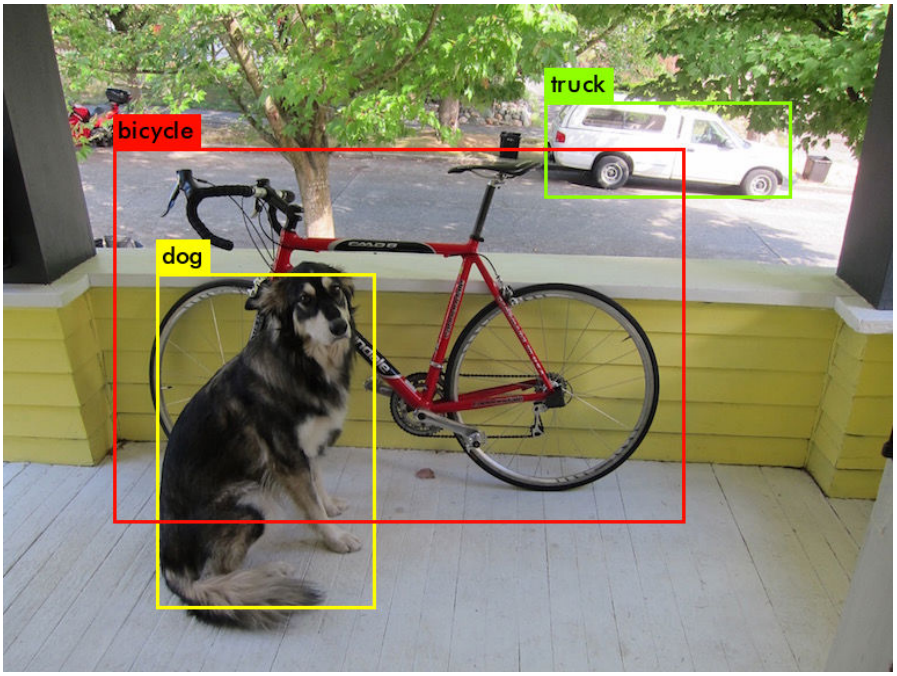


Figure 4.1: Przykładowy wynik testów na platformie darknet.

4.2 2-gie 2 tygodnie

Reprodukcja dotychczasowych benchmarków jako test architektury. Model YOLO5 udostępnia kilka specjalistycznych modeli, o różnej wielkości i dokładności. Po paru testach empirycznych doszliśmy do wniosku, że z naszymi możliwościami sprzętowymi możemy sobie pozwolić maksymalnie na yolov5s.

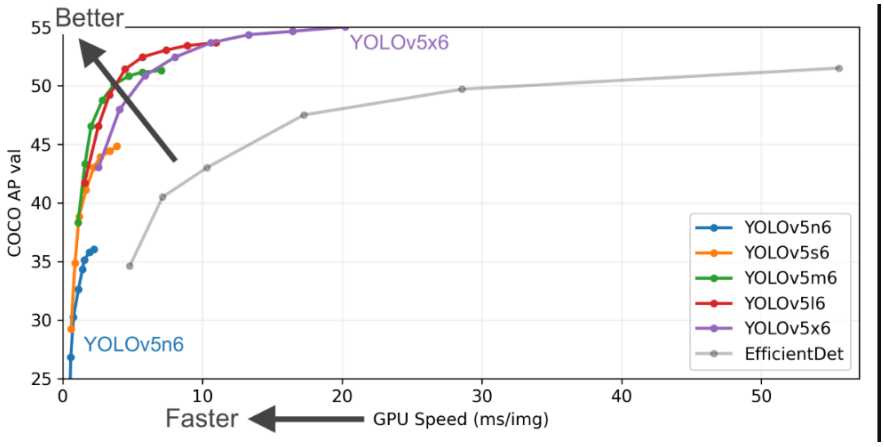


Figure 4.2: Porównanie modeli yolov5.

Przeszliśmy więc do próby odtworzenia benchmarków. Żeby to zrobić najpierw potrzebowaliśmy mieć jakiś punkt odniesienia. Najłatwiej dostępny jest benchmark dla datasetu COCO.

Model	size (pixels)	mAP^{val} 50-95	mAP^{val} 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6	1280	55.0	72.7	3136	26.2	19.4	140.7	209.8
+ TTA	1536	**55.8**	**72.7**	-	-	-	-	-

Figure 4.3: Benchmarki dla różnych modeli yolov5, nas interesuje yolov5s.

Proces sprawdzanie zaczęliśmy od sklonowania repozytorium i zainstalowania wymaganych bibliotek (wszystkie opisywane kroki można znaleźć tutaj - https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/).
Co istotne w zasadzie powinno się rozpocząć od zainstalowania pythona, następnie cudy, a na końcu pytorcha z kompatybilną wersją dla oprogramowania Nvidii. Po tych krokach

możemy przejść do tworzenia datasetu. Ultralytics dostarcza skrypty, dla niektórych datasetów do automatycznego pobierania danych i ich przygotowania. Warto zwrócić uwagę, że o ile na linuxie działa to bez zarzutu to na systemie Windows, czasem trzeba się troszke nagimnastykować. Można to też zrobić samemu (co uczyniłem i jest na githubie, ale nie polecam jak nie trzeba). Struktura plików powinna wyglądać w następujący sposób:

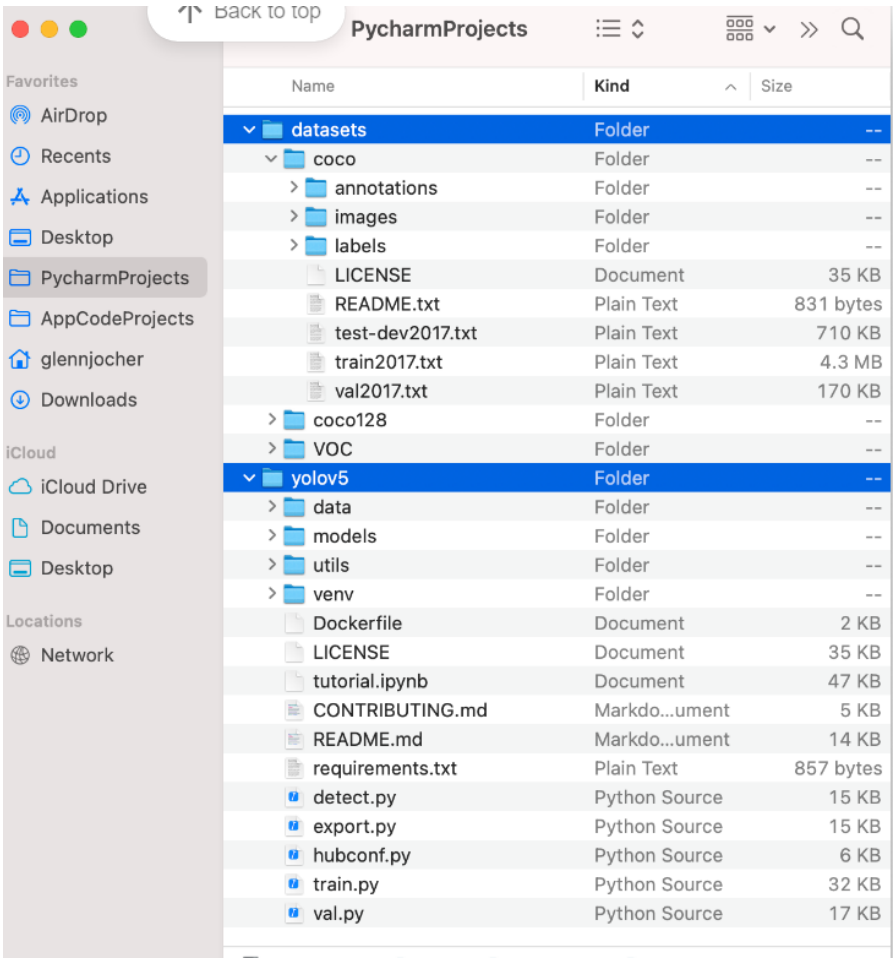


Figure 4.4: Wymagana struktura plików dla projektu.

Z tak przygotowanym projektem i wybranym modelem możemy przejść do trenowania.

```
python train.py --img 640 --epochs 3 --data coco128.yaml --weights yolov5s.pt
```

Figure 4.5: Komenda do wytrenowania modelu.

Jak widać jest ono całkiem proste i przejrzyste. Przekazujemy dostarczony przez ultrlytics skrypt do trenowania, ustawiamy rozmiar obrazu, liczbę epok, nasz plik kon-

figuracyjny yaml, zawierający ścieżki do części trenującej, walidującej oraz wyszczególnione labelsy z przypisanymi ID. W tym przykładzie przekazany jest od razu plik z wagami. Niestety łatwo dostępny jest tylko takowy dla datasetu COCO. W innym przypadku musimy zdefiniować model dla, którego trenujemy i dla niego wynikiem naszego trenowania będą właśnie pliki z wagami. Zachowując analogiczne podejście udało nam się otrzymać całkiem dobre wyniki.

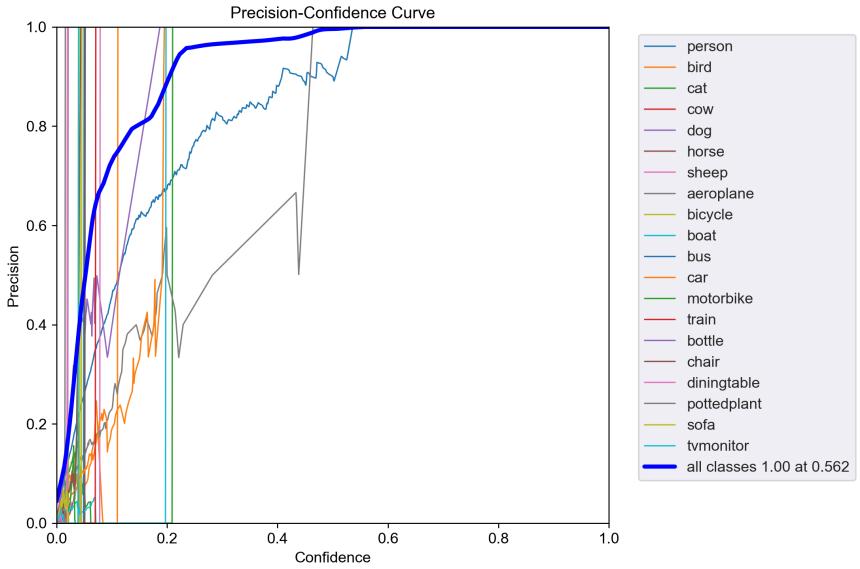
Class	Images	Instances	P	R	mAP50	mAP50-95:
all	5000	36335	0.672	0.519	0.566	0.371

Figure 4.6: mAP dla datasetu COCO. Pokrywa się z danymi dostarczonymi przez twórców.

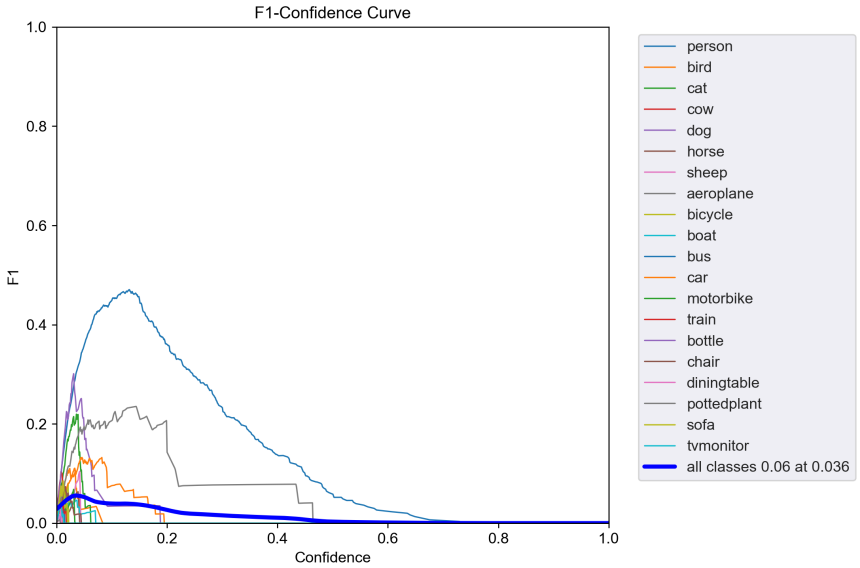
(Jedrzej Szostak 33% Piotr Libucha 33% Mariusz Bieganski 33%)

4.3 3-cie 2 tygodnie

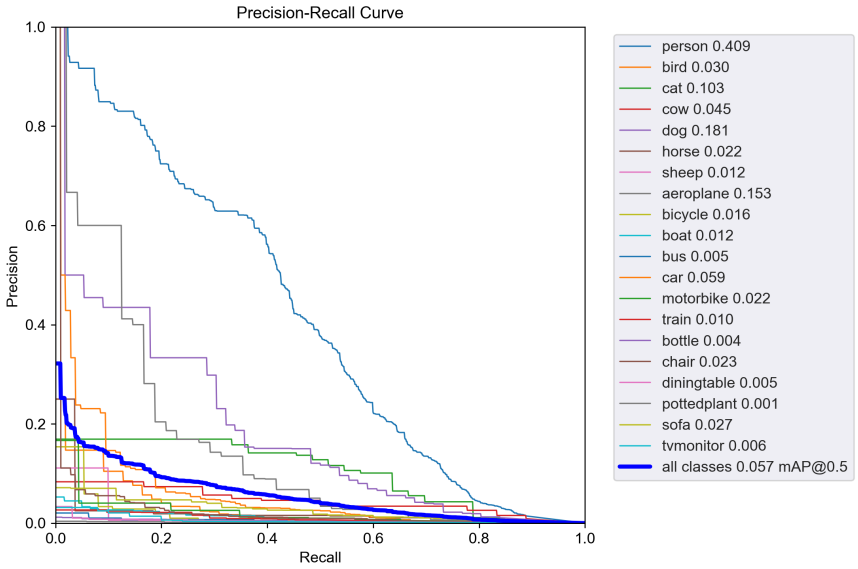
Przygotowanie docelowego środowiska roboczego, tj. modelu i danych oraz wybór metody określania niepewności (jackknife, bootstrap, dropout). Po przetestowaniu utwierdziliśmy się w przekonaniu, że wybraliśmy odpowiedni model. Ze względu na ograniczenia czasowe, sprzętowe (przy większych datasetach waliło mi za małym ramem :((() wybraliśmy stosunkowo nieduże datasety COCO, PASCAL VOC i VIS DRONE. Niestety nawet dla nich potrzebna jest większa liczba epok, aby otrzymać sensowne wyniki.



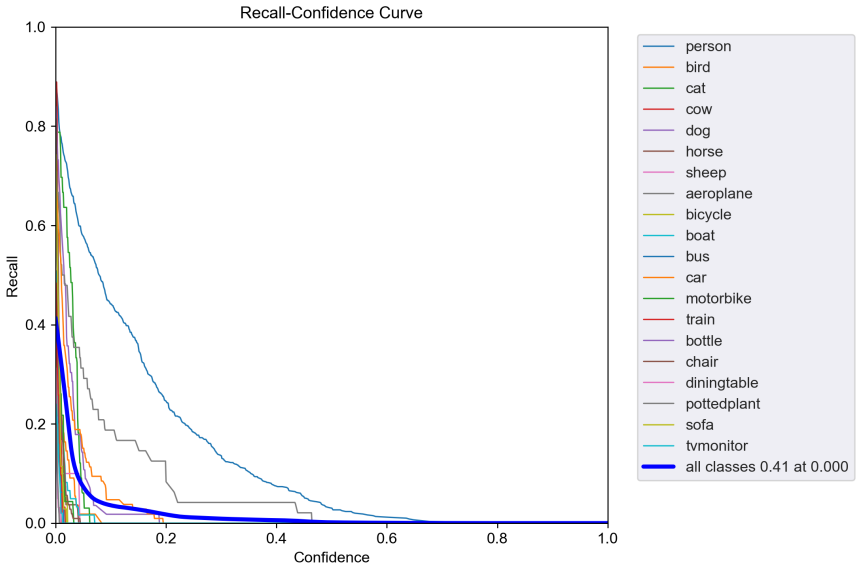
(a) Caption for Image 1



(b) Caption for Image 2



(a) Caption for Image 3



(b) Caption for Image 4

Figure 4.8: Overall caption for the combined figure

Wszystkie te datasety zawierają też stosunkowo podobny zakres klas co ułatwia ich złączenie. (Jedrzej Szostak 33% Piotr Libucha 33% Mariusz Bieganski 33%)

4.4 4-te 2 tygodnie

Eksperymenty numeryczne - do połączenie datasetów konieczna jest konwersja labeli. Do tego celu został napisany skrypt przechodzący przez wszystkie labele i jako, że id są na początku to konwertowane były na kodowanie odpowiadające naszemu docelowaemu modelowi. Zdażało się też, że klasy były do siebie bardzo podobne lub wręcz identyczne pomiędzy setami więc wtedy były wrzucane do jednego worka. Przez to samo dopasowywanie i porównywanie labeli musiało się odbywać ręcznie (która klasa przechodzi w którą). Za podstawe wzięliśmy dataset COCO jako, że ma najwięcej labeli. Dla VOC i Drone wszystko zostało odpowiednio zamienione w docelowym pliku YAML oraz poszczególnych plikach tesktowych opisujących zdjęcia.

```yaml
path: ../datasets/coco  # dataset root dir
train:
  - images/train2014
  - ../pascal_voc/images/train
  - ../VisDrone/VisDrone2019-DET-train/images

val:
  - images/val2017
  - ../pascal_voc/images/val
  - ../VisDrone/VisDrone2019-DET-val/images


# Classes
names:
  0: person
  1: bicycle
  2: car
  3: motorcycle
  4: airplane
  5: bus
  6: train
  7: truck
  8: boat
  9: traffic light
  10: fire hydrant
  11: stop sign
  12: parking meter
  13: bench
  14: bird
  15: cat
  16: dog
  17: horse
  18: sheep
  19: cow
  20: elephant
  21: bear
  22: zebra
  23: giraffe
  24: backpack
  25: umbrella
  26: handbag
  27: tie
  28: suitcase
  29: frisbee
  30: skis
  31: snowboard
  32: sports ball
  33: kite
```

Figure 4.9: Plik konfiguracyjny yaml.

Jak widać powyżej łączenie datasetów jest stosunkowo proste. Wystarczy dodać wszystkie adtasety do odpowiednich scieżek. I zdefiniować wszystkie lebele - czyli de facto zmergować labele z klasa składowych. (Jedrzej Szostak 33% Piotr Libucha 33% Mariusz

Bieganski 33%)

4.5 5-te 2 tygodnie

Opracowanie wyników. Dużą zaletą wybranego przez nas rozwiązania jest to, że wiele rzeczy i wykresów robi się automatycznie w trakcie wykonywania programu i nie trzeba pisać dodatkowego kodu. Wszystkie hiperparametry pozostawiliśmy domyślne (batch size np. nie dało się zmienić na wyższe, żeby komputer to wytrzymał), przy pobieżnej próbie manipulacji innymi parametrami przeważnie coś się wywalało. (Jedrzej Szostak 33% Piotr Libucha 33% Mariusz Bieganski 33%)

5 Analiza parametryczna

Szczegółowe omówienie poszczególnych etapów i operacji składowych zaplanowanych na potrzeby poszukiwania optymalnych parametrów klasyfikatora. W tej części można zamieszczać fragmenty kodu źródłowego, jeśli zachodzi taka potrzeba oraz schematy blokowe.

6 Wyniki eksperymentalne

Rezultaty oceny poprawności działania programu w formie tabel/wykresów.

Class	Images	Instances	P	R	mAP50	mAP50-95:
all	6063	76552	0.00243	0.0362	0.00167	0.00049

Figure 6.1: mAP dla naszego modelu uczonego wygląda mizernie.

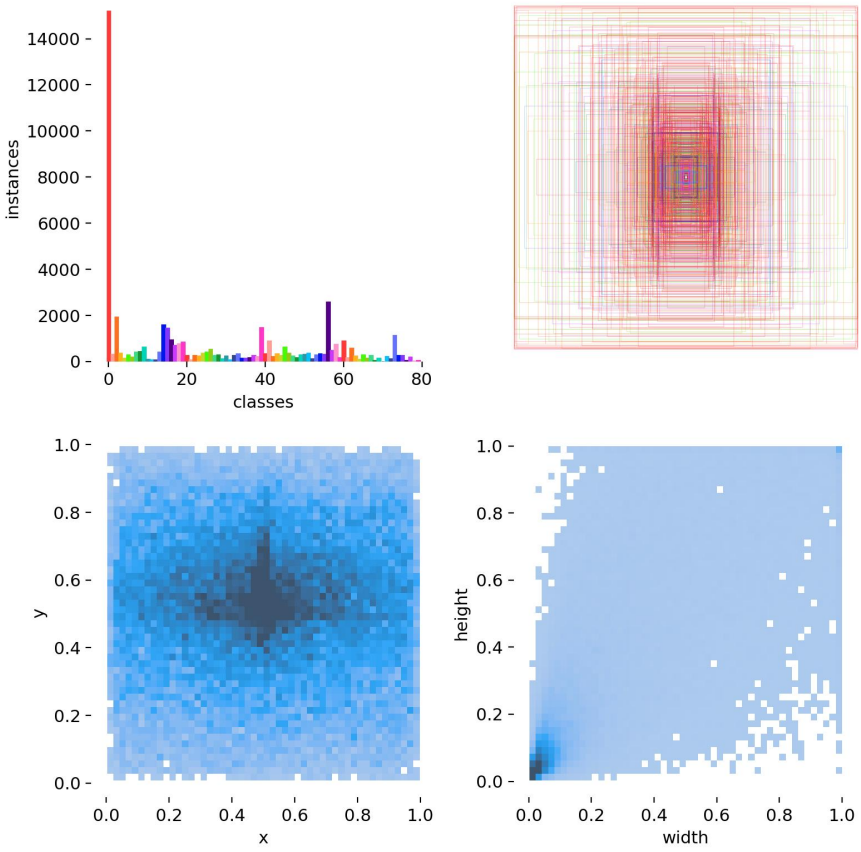


Figure 6.2: mKolejnym problemem mogło być niewłaściwe rozdzielenie klas, widać że ludzi jest zdecydowanie najwięcej.

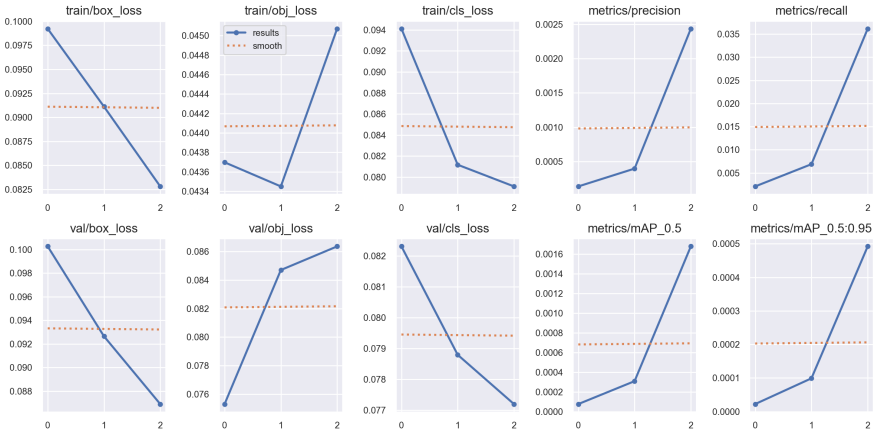


Figure 6.3: Wyniki dla naszego modelu. Widzimy, że przydałoby się trochę epok bo np mAP gwałtownie wzrasta.

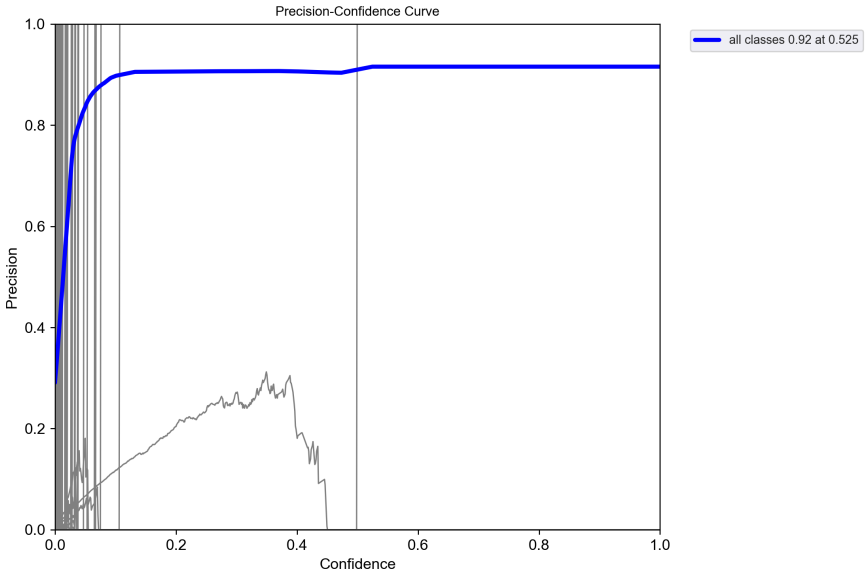


Figure 6.4: P_curve.

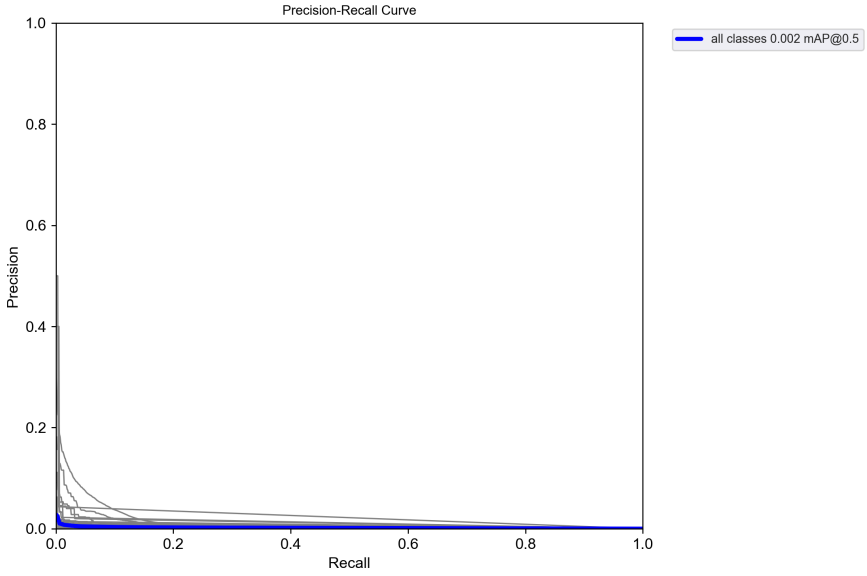


Figure 6.5: PR_curve.

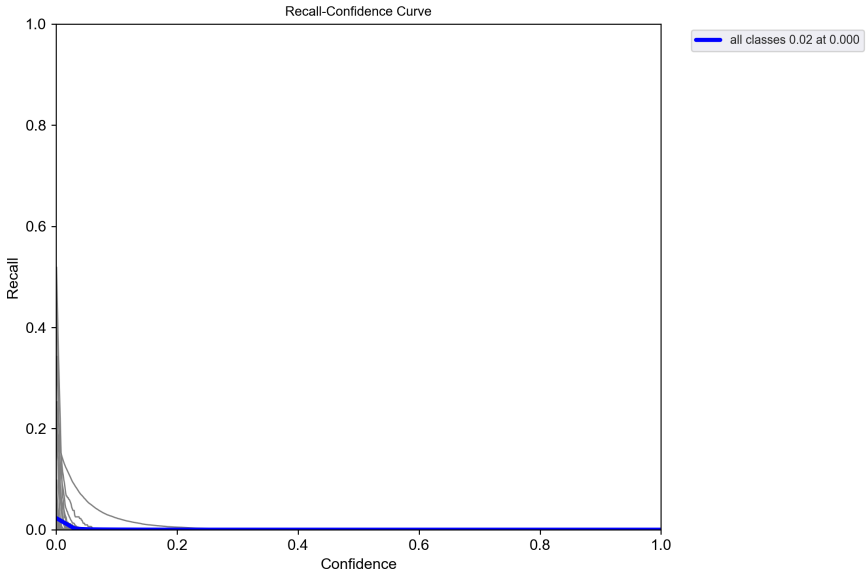


Figure 6.6: R_curve.

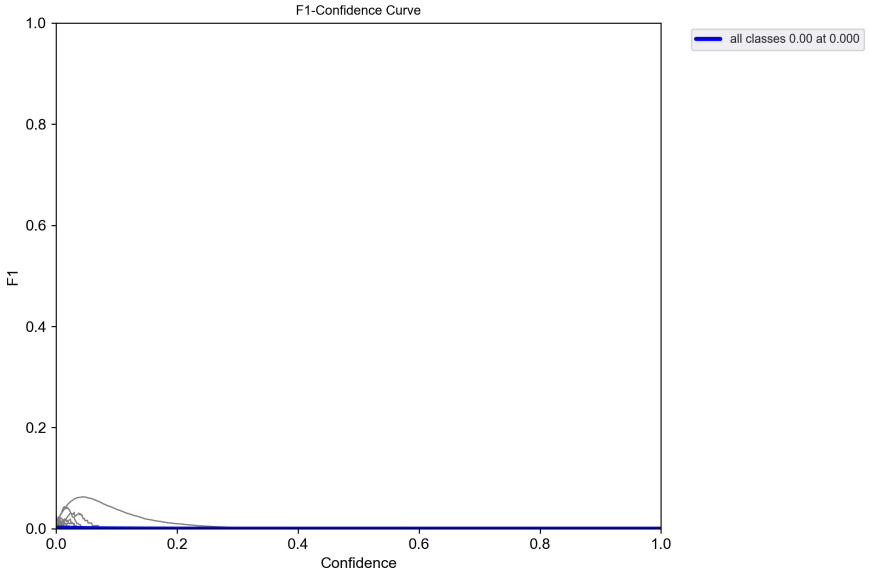


Figure 6.7: F1_curve.

Wykresy nie wyglądaja zbyt dobrze, ani nic nie mówią, ze względu na to, że model był trenowany tylko na 3 epokach (1.5h).

7 Wnioski

Analiza uzyskanych w części eksperymentalnej rezultatów, przedstawienie wniosków i obserwacji. Jak widać niestety nie udało nam się w pełni dobrze wytrenować sieci. Głównym problemem było za poźne zabranie się za projekt, w którym duża część czasu to czekanie na wyniki. Opracowywanie takich zagadnień wymaga dużych mocy obliczeniowych (4GB ramu na GPU to zdecydowanie za mało). Przed podjęciem się takiego tematu dobrze by było też sprawdzić wymagania sprzętowe, bo w moim wypadku nie było technicznie możliwe zrobienie na większym datasecie, a nawet na takim niewielkim wyszło średnio. Nauczyliśmy się jak mniej więcej działa yolo i jak się je uruchamia co pozwoli na wykorzystanie w późniejszych projektach. Liczba epok potrzebna do solidnego wytrenowania wynosi około 200/300. Instalacja odpowiednich sterowników graficznych jest szczególnie ważna, bo można sobie rozwalić system, a nawet trzy.

8 Github

https://github.com/codespaghettifier/yolo_model_metrics/tree/rest