Politechnika Warszawska

WYDZIAŁ ELEKTRYCZNY

Instytut Sterowania i Elektroniki Przemysłowej
Zakład Sterowania

Praca dyplomowa inżynierska

na kierunku Automatyka i Robotyka Stosowana

Porównanie wybranych metod segmentacji instancji w obrazach cyfrowych

Magda Anna Koprowska
nr albumu 284991

promotor
dr inż. Witold Czajewski

WARSZAWA 2020

Porównanie metod segmentacji instancji w obrazach cyfrowych

Streszczenie

Celem niniejszej pracy dyplomowej było porównanie dwóch metod segmentacji instancji - Mask R-CNN oraz Yolact. Porównanie polegało na wytrenowaniu modeli wymienionych wyżej metod bazujących na sieciach konwolucyjnych oraz przeprowadzenie ewaluacji wyników z uwzględnieniem parametrów takich jak: czas uczenia, współczynnik mAP oraz szybkość przetwarzania. Przeprowadzono również porównanie modeli wytrenowanych przez autora z modelami dostarczonymi przez twórców rozwiązań. Praca pozwoliła na uzyskanie informacji na temat zasobów sprzętowych potrzebnych do przeprowadzenia badań z wykorzystaniem wybranych metod oraz wymaganej do tego ilości czasu. Wynikiem przeprowadzonych badań jest określenie optymalnego rozwiązania problemu rozpoznawania obiektów sceny pod względem dokładności, szybkości działania oraz zasobów potrzebnych do implementacji danej metody.

Słowa kluczowe: segmentacja instancji, machine learning, deep learning, sieci konwolucyjne, Mask R-CNN, Yolact

Comparison of methods of instance segmentation in digital images

Abstract

The purpose of this BSc thesis was to compare two methods of instance segmentation - Mask R-CNN and Yolact. The comparison consisted of training models of the aforementioned methods based on convolution networks and conducting evaluation of results, taking into account parameters such as: learning time, mAP coefficient and fps parameter. Additionally a comparison of the models trained by the author with the models provided by the developers of the solutions was made. The work allowed to obtain information on hardware resources needed to conduct tests using selected methods and required amount of time. The result of the conducted research determined the optimal solution to the problem of recognizing scene objects in terms of accuracy, speed and resources needed to implement a given method.

Keywords: instance segmentation, machine learning, deep learning, convolutional neural networks, Mask R-CNN, Yolact

Politechnika Warszawska

Warszawa, 31 stycznia 2020 r.

Magda Anna Koprowska

imię i nazwisko studenta

284991

numer albumu

Automatyka i Robotyka Stosowana

kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.
Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w systemie iSOD są identyczne.

czytelny podpis studenta

Spis treści

Rozdział 1

Wstęp

1.1 Wprowadzenie

Wizja komputerowa (ang. Computer Vision) istnieje już od ponad 50 lat, ale ostatnio można zauważyć odrodzenie zainteresowania tematem, jak „widzą" maszyny i jak można tę wiedzę wykorzystać [16].

Pierwsze prace nad wizją komputerową rozpoczęły się na uniwersytetach zajmujących się sztuczną inteligencją w późnych latach 60-tych poprzedniego wieku. Prace te miały na celu stworzenie algorytmu wzorującego się na ludzkim układzie wzrokowym, aby uczynić roboty „inteligentnymi". W latach 70. XX wieku powstało wiele technik widzenia komputerowego, które są obecnie wykorzystywane – rozpoznawanie krawędzi, podążanie za linią czy estymacja ruchu. W następnej dekadzie skupiono się bardziej na matematycznej analizie problemu wizji komputerowej. Prace obejmowały koncepcję skali, określanie kształtu obiektu na podstawie różnych przesłanek, takich jak cieniowanie, faktura czy modele konturów. Badania konstrukcji 3D doprowadziły do lepszego zrozumienia kalibracji kamery, a tym samym umożliwiło to stworzenie rekonstrukcji trójwymiarowych obiektów na podstawie kilku obrazów. W podobnym okresie stworzono również techniki będące podwaliną obecnej segmentacji obrazu.

W ostatnich latach nastąpiło odrodzenie metod opartych na ekstrakcji cech, stosowanych przy użyciu uczenia maszynowego. Przyczyn tego zjawiska można dopatrywać się w zauważalnym wzroście wydajności procesorów graficznych, który umożliwił bardziej efektywne stosowanie metod wykorzystujących techniki głębokiego uczenia. Techniki te zdominowały poprzednie metody służące do zadań obejmujących klasyfikację i segmentację obrazów [18].

Obecnie aplikacje wykorzystujące głębokie uczenie w procesie rozpoznawania obiektów są powszechnie wykorzystywane i to na ich opiera się wiele nowoczesnych technologii. Smartfony rozpoznają twarz użytkownika przy odblokowywaniu ekranu czy umożliwiają wyszukiwanie przedmiotów w galerii na podstawie tylko jego nazwy. Coraz szerzej rozpowszechnione są również autonomiczne samochody, które wytyczają trasę oraz reagują na sytuacje na drodze wykorzystując między innymi informacje z zastosowanych w nich kamer. A wszystko to dzieje się za pomocą technik rozpoznawania obiektów opartych na głębokich sieciach neuronowych i segmentacji instancji.

1.2 Cel pracy

Głównym celem niniejszej pracy inżynierskiej jest porównanie wybranych metod segmentacji instancji w obrazach cyfrowych z wykorzystaniem sieci głębokiego uczenia. Analiza ta powinna pozwolić na wybranie optymalnego rozwiązania problemu rozpoznawania obiektów sceny pod względem dokładności, szybkości działania oraz zasobów potrzebnych do implementacji danej metody. Zakres prac obejmuje także zgłębienie tematyki głębokiego uczenia oraz detekcji obiektów na obrazach.

1.3 Układ pracy

Praca składa się z siedmiu rozdziałów: wstępu, stanu wiedzy, przeglądu istniejących rozwiązań, opisu wybranej koncepcji rozwiązania problemu, opisu wybranych narzędzi i bibliotek, opisu przeprowadzonych prac oraz podsumowania wraz z wnioskami, których zawartość przedstawia się następująco:

- we wstępie wprowadzono czytelnika w rys historyczny problemu pracy, przedstawiono poruszane zagadnienia i obecnie wykorzystywane rozwiązania,
- rozdział drugi obejmuje teoretyczny opis tematów i zagadnień, które są poruszane w niniejszej pracy,
- w rozdziale trzecim przybliżone zostały czytelnikowi obecnie wykorzystywane rozwiązania w omawianej dziedzinie,
- rozdział czwarty traktuje o wyborze koncepcji postanowionego w tytule problemu badawczego,
- rozdział piąty zawiera zarówno szerszy opis narzędzi (w tym sprzętu), jak i bibliotek wykorzystane do zbadania problemu z rozdziału czwartego,
- w rozdziale szóstym następuje opis przeprowadzonych prac nad tematem, tj. opis konfiguracji wykorzystanych narzędzi, opis przeprowadzonych badań oraz analiza uzyskanych wyników,
- ostatni rozdział, rozdział siódmy, jest podsumowaniem pracy, przedstawieniem uzyskanych wyników oraz zawiera propozycję dalszego rozwoju w omawianej dziedzinie.

Rozdział 2

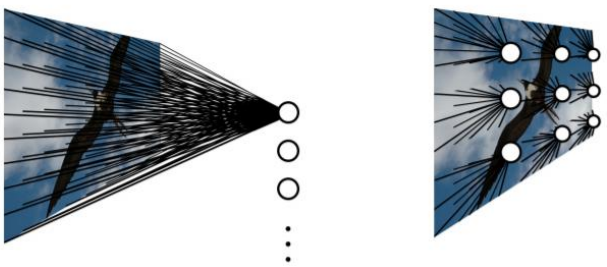
Stan wiedzy

2.1 Splotowe sieci neuronowe

Sztuczne sieci neuronowe istnieją już od dziesięcioleci i były z powodzeniem wykorzystywane w wielu zadaniach w obszarze uczenia maszynowego, z którym nie radziły sobie mniej zaawansowane algorytmy. Jednakże nie są one najlepszym rozwiązaniem w kontekście zadań związanych z analizą obrazu. Jeżeli wejściem dla sieci neuronowej są obrazy, tym bardziej, jeśli są to obrazy kolorowe, liczba połączeń sieci, a tym samym jej parametrów – współczynników wag i progów neuronów, drastycznie wrasta. Skutkuje to ogromną liczbą operacji, które trzeba przeprowadzić, aby wykorzystać daną sieć.

Ze względu na ten fakt, aby stosować sztuczne sieci neuronowe w obszarze analizy obrazu, konieczne stało się zmniejszenie wymiarów obrazów w zbiorze uczącym bądź zredukowanie liczby połączeń w każdej warstwie sieci. Rozwiązaniem tego problemu stały się splotowe (konwolucyjne) sieci neuronowe. Zostały one tak zaprojektowane, aby ich neurony łączyły się tylko z podzbiorem pikseli obrazu. Mechanizm ten uzyskuje się za pomocą warstw konwolucyjnych, w których obraz jest splatany z szeregiem filtrów.

W warstwach konwolucyjnych wszystkie neurony mają te same wagi i są połączone tylko z małą częścią obrazu. Ponadto każdy neuron jest odpowiedzialny za część obrazu zlokalizowaną tuż obok części, za którą odpowiedzialny jest poprzedni neuron, co tworzy przestrzenną siatkę na całym obrazie. Sposób działania warstw konwolucyjnych został przedstawiony na rys. 2.1.



Rysunek 2.1: Po lewej: warstwy całkowicie połączone - używane w zwykłych sieciach neuronowych. Po prawej: warstwy splotowe - używane w splotowych sieciach neuronowych. Źródło:[6]

Wspomniana wcześniej siatka neuronów zwraca splot wybranych wag z obrazem wejściowym. Wybrane wagi są nazywane filtrem, który spleciony z obrazem wejściowym generuje obraz wynikowy. Na każdej warstwie splotowej znajduje się kilka filtrów o tym samym rozmiarze, a wynikiem splotu obrazu z tymi filtrami jest kilka obrazów wyjściowych. Każdy obraz wynikowy jest nazywany mapą cech. Następnie stworzone wcześniej mapy cech są układane w stos danych 3D o wymiarach W x H x N, gdzie W i H oznaczają wymiary obrazów, a N jest liczbą filtrów używanych w danej warstwie. Ten stos danych można spleść z nowym zestawem filtrów, generując nowe wyniki.

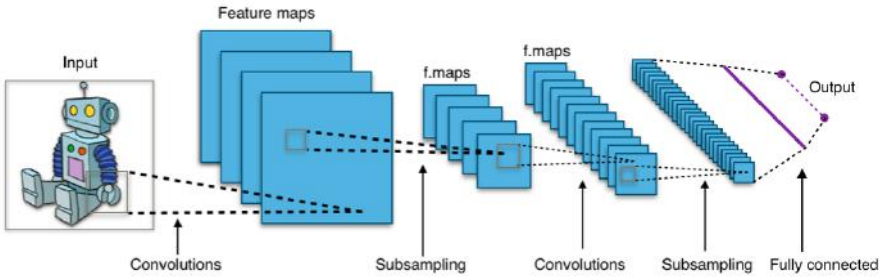
Aby określić, jakich rozmiarów powinna być warstwa konwolucyjna stosuje się formułę W x H x D x N, gdzie W i H określają rozmiar zastosowanego filtra (obszar, do którego zostanie podłączony każdy neuron danej warstwy), D określa, z iloma mapami cech zostaną połączone neurony, a N określa liczbę filtrów, które posiada ta warstwa konwolucyjna. Sieci splotowe, oprócz warstw splotowych, składają się często, choć nie zawsze, również z dwóch innych typów warstw – warstwy ReLU i warstwy skalującej. Warstwy ReLU (ang. Rectified Linear Unit Layers) obliczają wyjście dla danego pixela korzystając z równania:

$$ReLU(x) = \begin{cases} \mathrm{x}, & \mathrm{x} \geqslant 0 \\ 0, & \mathrm{x} < 0 \end{cases}$$

gdzie x jest wartością pixela wejściowego.

Rektyfikowane jednostki liniowe ReLU są stosowane w celu wyeliminowania problemu zanikającego gradientu, polegającego na coraz mniejszych zmianach wag coraz dalej położonych od wyjścia sieci neuronów ze względu na przemnażanie pochodnych będących z przedziału [0,1].

Warstwy skalujące (ang. pooling layers) zmniejszają każdą mapę cech do rozmiarów, w których zostaje zachowana jedynie jedna (najczęściej maksymalna, rzadziej średnia) wartość znaleziona na małym obszarze obrazu. Pozostałe wartości są odrzucane.



Rysunek 2.2: Typowa architektura sieci splotowej. Źródło: [19]

Częstą praktyką jest naprzemienne stosowanie warstw splotowych z warstwami ReLu i warstwami całkowicie połączonymi, aby mieć gwarancję, że zastosowane sploty są nieliniowe i stopniowo zmniejszają rozmiar danych, tym samym zmniejszając również wymagania obliczeniowe dla danej sieci.

W momencie, kiedy dany obraz wejściowy jest splatany z pierwszą serią filtrów, następnie wynik tego działania jest splatany z kolejną serią filtrów, jest duże prawdo-

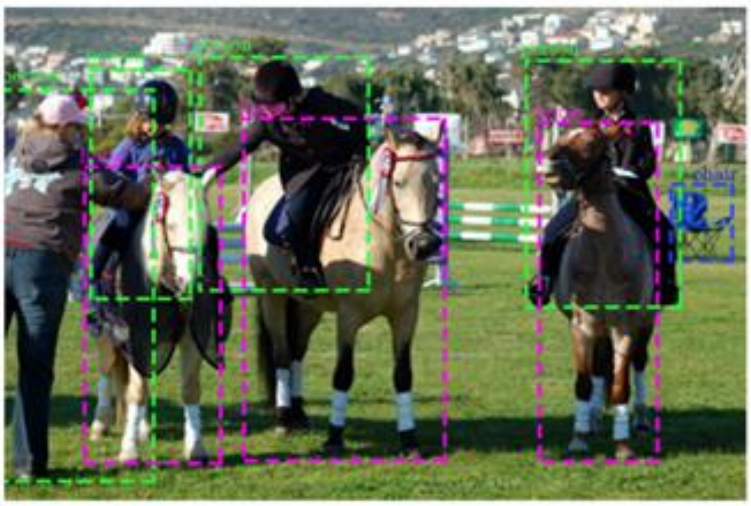
podobieństwo, że sieć będzie w stanie odkryć, które filtry są najlepsze – wyodrębniają z obrazu najbardziej istotne cechy.

W związku z tym, jednym ze sposobów interpretacji wyników na wyjściu zestawu warstw splotowych jest ekstraktor cech, który tworzy mały zbiór danych, będących jednocześnie najbardziej istotnymi dla danego zadania. Po serii warstw splotowych dodaje się często jedną (lub więcej) warstwę całkowicie połączoną i warstwę softmax, która próbuje dokonać regresji funkcji nieliniowej, aby naprowadzić wyodrębnione cechy na pożądany wynik.

Następnie, sieci splotowe są nauczane przy pomocy algorytmu propagacji wstecznej, który jest stosowany również w zwykłych sieciach neuronowych. Główną różnicą jest fakt, iż liczba wag jest dzielona przez liczbę neuronów, więc mechanizm propagacji wstecznej jest bardziej wydajny.

2.2 Wykrywanie obiektów

Jednym z głównych wyzwań w procesie analizy obrazu jest problem wykrywania obiektów na danym obrazie. Polega on na przewidywaniu lokalizacji danego obiektu na obrazie przy jednoczesnym przypisaniu obiektu do danej klasy. Lokalizacja ta jest określana poprzez stworzenie najmniejszego prostokąta, który jednocześnie zawiera cały obiekt, nazywanego ramką graniczną lub prostokątem otaczającym (ang. bounding box). Przykład ramki granicznej został przedstawiony na rys. 2.3.



Rysunek 2.3: Przykładowy wynik operacji wykrywania obiektów na obrazie. Źródło:[13]

Przez lata powstało wiele różnych koncepcji na poradzenie sobie z zadaniem wykrywania obiektów na obrazie. Obecnie najpowszechniej używanym rozwiązaniem jest zastosowanie specjalnego rodzaju konwolucyjnych sieci neuronowych – bazujących na regionie obrazu. Sieci te proponują ogromną liczbę obszarów (kandydatów na ramkę graniczną) wykorzystując tzw. algorytm wyszukiwania selektywnego [8]. Następnie propozycje są dopasowywane do wejściowych wymiarów ekstraktora cech sieci splotowej, który generuje wyjściowy wektor cech dla każdego obszaru.

Wektor ten jest w następnym kroku podawany na wejście warstw całkowicie połączonych, co ma na celu zaproponowanie klas dla obiektów, które znajdują się w

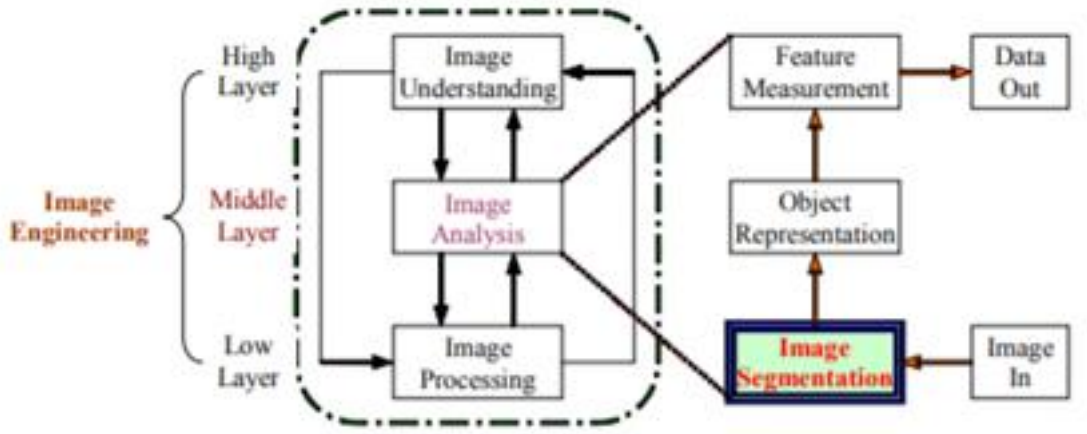
danym obszarze. Jednocześnie krok ten ma za zadanie odrzucenie klas, które są z małym prawdopodobieństwem obecne na danym obszarze.

Oprócz tego wektor cech jest również podawany na wejście funkcji regresji, którego celem działania jest poprawa dopasowania ramki granicznej.

Ze względu na dużą wyższą wydajność w stosunku do innych metod, sieci splotowe opierające się na regionie obrazu są teraz powszechnym sposobem na poradzenie sobie z detekcją obiektów na obrazie. W tej pracy została wykorzystana sieć Mask R-CNN, która jest rozszerzeniem techniki Fast R-CNN – sieci splotowej opartej na regionie, stworzonej w 2015 roku.[3] Pomimo upływu kilku lat, sieć ta jest w stanie osiągać konkurencyjne wyniki w porównaniu z innymi technikami detekcji obiektów.

2.3 Segmentacja obrazu

Obrazy są dla człowieka głównym źródłem informacji na temat świata zewnętrznego. Aby wydobyć z nich jeszcze więcej informacji opracowano szereg różnych technik i aplikacji. Wszystkie te techniki można zawrzeć w jednym pojęciu – inżynierii obrazu, która składa się z trzech głównych warstw: przetwarzania obrazu, analizy obrazu i rozumienia obrazu.



Rysunek 2.4: Schemat podziału inżynierii obrazu. Źródło: [7]

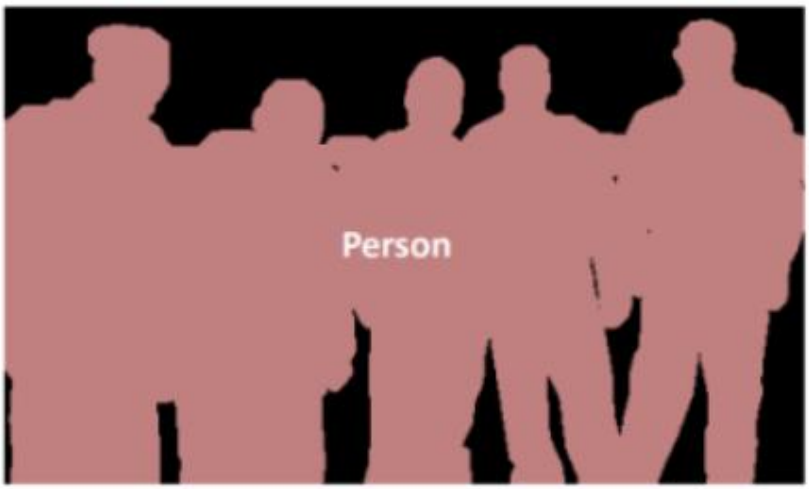
Segmentacja obrazu jest jednym z najważniejszych etapów analizy obrazu. Polega ona na uproszczeniu obrazu (często poprzez wydzielenie mniejszych jego obszarów) do postaci, która jest łatwiejsza do interpretacji i zawiera więcej znaczących informacji. Segmentacja obrazu jest zarówno pierwszym, jak również jednym z ważniejszych, o ile nie najważniejszym, zadaniem wspomnianego powyżej procesu. Jej wyniki mają znaczący wpływ na dalsze etapy samej analizy obrazu oraz zadań wyższego poziomu, takich jak zrozumienie przedstawionej sceny.

Problem segmentacji obrazu jest zagadnieniem sięgającym ponad 40 lat wstecz. Pierwszą zaproponowaną techniką był detektor krawędzi Robertsa wynaleziony w 1963 roku[14]. Był on pierwszym krokiem w kierunku dekompozycji obrazu, co stało się podwaliną pojęcia jego segmentacji.

Od tego czasu powstało wiele technik i algorytmów bazujących bądź będących wariacjami na ten temat. Zasadniczo można podzielić techniki segmentacji obrazu na dwa typy – segmentację semantyczną i segmentację instancji.

2.3.1 Segmentacja semantyczna

Segmentacja semantyczna polega na zakwalifikowaniu każdego piksela danego obrazu do konkretnej klasy (zdefiniowanej wcześniej w zestawie klas). W przeciwieństwie do zadania wykrywania obiektów, segmentacja semantyczna nie skupia się na wyodrębnieniu osobnych obiektów (ich instancji), co pokazano na rys. 2.5.



Rysunek 2.5: Przykład segmentacji semantycznej. Źródło: [15]

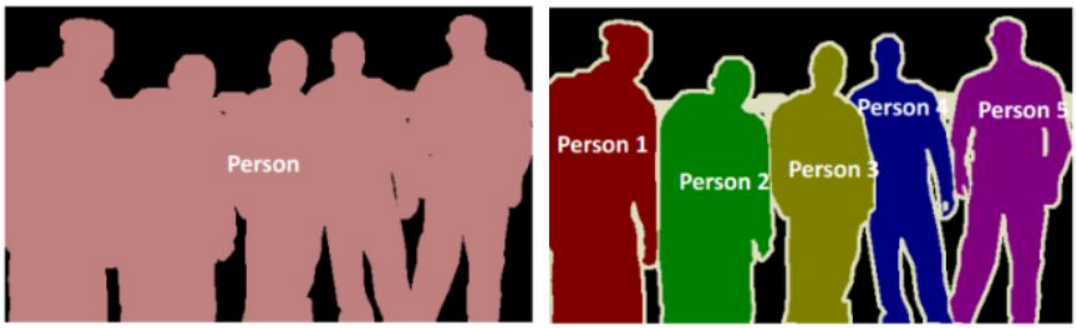
Jak można zauważyć wyodrębniono tutaj dwie klasy – klasę *person* oraz tło. Nie niesie to jednak żadnej informacji na temat konkretnych osób znajdujących się na obrazie.

Przełomem w dziedzinie segmentacji semantycznej był artykuł zatytułowany Fully Convolutional Networks for Semantic Segmentation. Artykuł ten przedstawia, w jaki sposób zreinterpretowanie warstw w standardowych sieciach konwolucyjnych i dodanie drobnych modyfikacji pozwoliło na stworzenie nowego typu sieci konwolucyjnych – FCN, które okazały się idealne dla złożonych zadań predykcyjnych, takich jak właśnie segmentacja semantyczna.

2.3.2 Segmentacja instancji

Segmentację instancji można rozumieć jako jednoczesne wykrywanie obiektów oraz segmentację semantyczną. Polega ona na zlokalizowaniu każdego obiektu na obrazie, przypisaniu mu klasy spośród zestawu zdefiniowanych wcześniej klas oraz stworzeniu maski zawierającej każdy z pikseli należący do tego konkretnego obiektu.

Zasada działania segmentacji instancji (w porównaniu do segmentacji semantycznej) została przedstawiona na rys. 2.6.



Rysunek 2.6: Porównanie segmentacji semantycznej z segmentacją instancji. Źródło: [15]

Algorytmy, które są wykorzystywane do zadania segmentacji instancji są dużo bardziej efektywne niż w przypadku poprzednich zadań analizy obrazu tj. wykrywanie obiektów czy segmentacja semantyczna, ponieważ konieczne jest pełne zrozumienie sceny przedstawionej na obrazie.

2.4 Metryki ewaluacyjne

Klasyfikacja jest to proces, w którym obiekty występujące na obrazie przypisuje się do danej klasy. Narzędziem wykorzystywanym w zadaniu klasyfikacji jest klasyfikator, który, aby poprawnie wykonywać swoje zadanie, musi zostać wcześniej do niego przyuczonym. Proces uczenia klasyfikatora polega na pokazaniu mu zbioru obiektów z przypisanymi już klasami. Ten sposób uczenia klasyfikatora nazywa się uczeniem nadzorowanym (z nauczycielem). Następnie klasyfikatorowi przedstawia się zbiór testowy, zawierający obiekty z już przypisanymi klasami i sprawdza się czy klasa predykowana przez klasyfikator zgadza się z klasą przypisaną do obiektu.

Aby móc ocenić wydajność klasyfikacji, stosuję się tablicę pomyłek - tabelę złożoną z dwóch wierszy i dwóch kolumn. Wiersze przedstawiają klasy predykowane a kolumny klasy rzeczywiste.

		klasa rzeczywista	
		pozytywna	**negatywna**
klasa predykowana	**pozytywna**	prawdziwie pozytywna (TP)	fałszywie pozytywna (FP)
	negatywna	fałszywie negatywna (FN)	prawdziwie negatywna (TN)

Rysunek 2.7: Tablica pomyłek. Źródło: [20]

Na podstawie prawdziwych i nieprawdziwych klasyfikacji można wyróżnić szereg miar oceniających wydajność danej klasyfikacji:

Precyzja (ang. Precision) jest to stosunek wyników prawdziwie pozytywnych do sumy wyników prawdziwie pozytywnych oraz fałszywie pozytywnych.

$$Precyzja = \frac{TP}{TP + FP}$$

Informuje ona o tym, jak zgodne są jego wyniki w stosunku do innych pomiarów tej wielkości.

Czułość (ang. Recall) jest to stosunek wyników prawdziwie pozytywnych do sumy wyników prawdziwie pozytywnych i fałszywie pozytywnych.

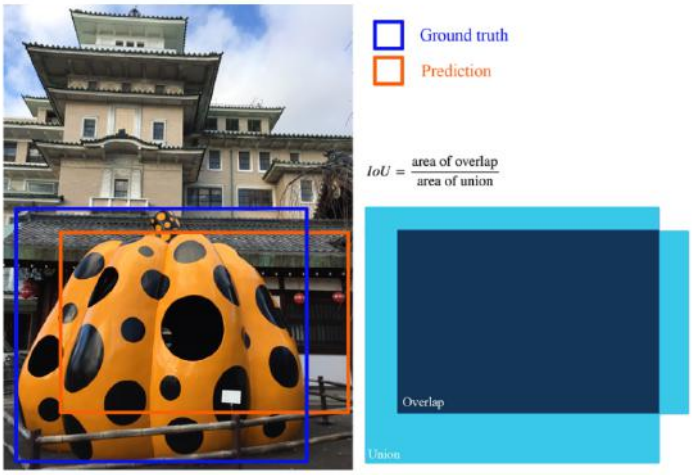
$$Czulosc = \frac{TP}{TP + FN}$$

Informuje ona o tym, jak dobra jest jego metoda w znajdowaniu wyników pozytywnych.

Oprócz wyżej wymienionych miar, oceny dokonuje się również na podstawie takich metryk jak:

IoU (ang. IoU – Intersection Over Union), współczynnik Jaccarda

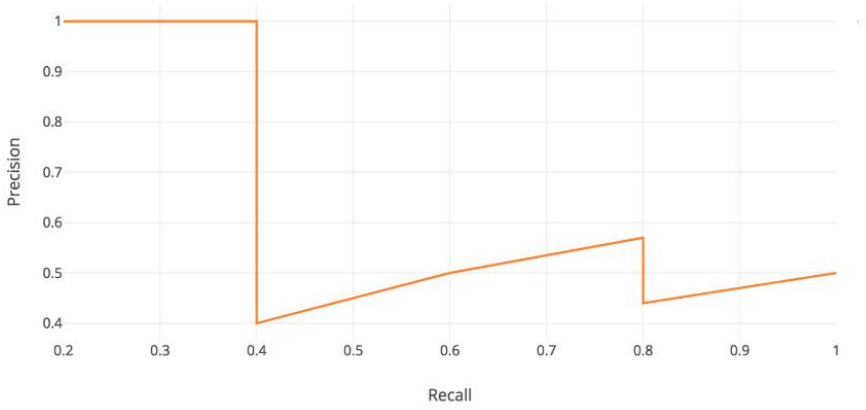
Wskaźnik IoU (zwany czasem współczynnikiem Jaccarda) został stworzony do oceny precyzji wykrywania obiektów i informowania, jak bardzo obszar zaproponowanej przez detektor ramki granicznej zgadza się z rzeczywistym obrysem obiektu. To ważny wskaźnik również w kontekście wspomnianej wcześniej precyzji pomiaru. Jego wartość stanowi o zakwalifikowaniu próbki do określonej kategorii. Dla przykładu – najczęściej przyjmuje się, że próbka jest prawdziwie pozytywna, jeżeli wartość IoU jest większa lub równa 0,5.



Rysunek 2.8: Ilustracja miary IoU Źródło: [11]

Średnia precyzja (ang. AP – Average Precision)

Często wykorzystywanym wskaźnikiem w ocenie detektorów obiektów, takich jak Fast R-CNN czy SSD, jest wskaźnik AP. Zawiera się on w zakresie od 0 do 1 i określa pole powierzchni pod wykresem precyzji w funkcji czułości.

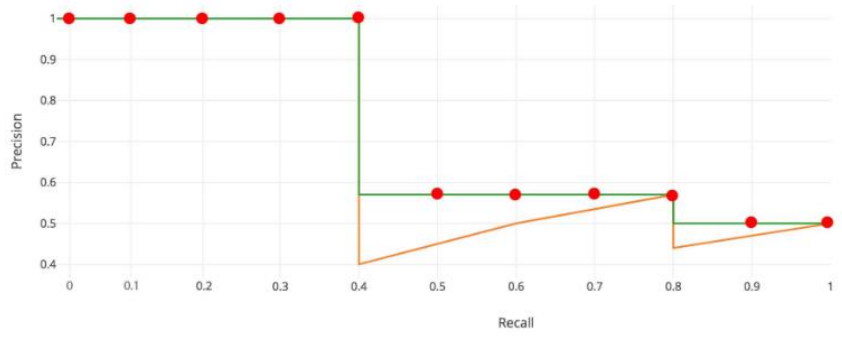


Rysunek 2.9: Przykład wykresu średniej precyzji (Average Precision)) Źródło: [11]

$$AP = \int_0^1 p(r)dx$$

Interpolowana średnia precyzja (ang. Interpolated Average Precision)

Innym wskaźnikiem, głównie wykorzystywanym do oceny modeli uczonych na popularnym dla detekcji obiektów zbiorze Pascal VOC, jest interpolowana średnia precyzja (ang. Interpolated Average Precision). Bazowano na niej do 2008 roku, wykorzystując 11 punktów wykresu precyzji w funkcji czułości.



Rysunek 2.10: Przykład wykresu interpolowanej średniej precyzji (ang. Interpolated Average Precision). Źródło: [11]

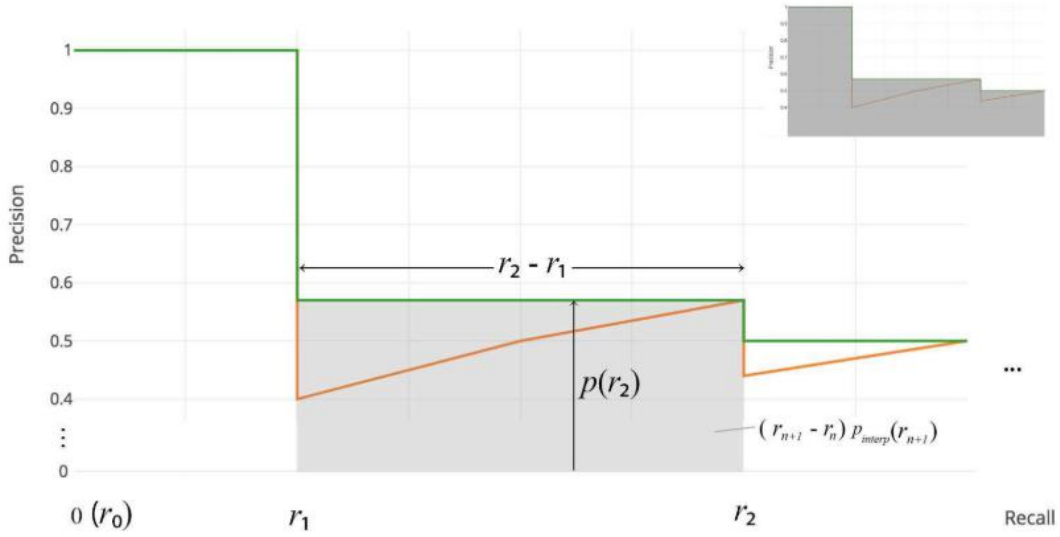
Aby wyliczyć średnią precyzję zastosowano wzór:

$$AP = \frac{1}{11} \times (AP_r(0) + AP_r(0.1) + ... + AP_r(1.0))$$

Metoda ma dwie zasadnicze wady. Po pierwsze, jest mało precyzyjna. Po drugie, nie jest aplikowalna dla przykładów o małych współczynnikach AP. Wady te przyczyniły się do zmiany wskaźnika dla Pascal VOC.

AUC (ang. Area Under Curve)

Wskaźnik wykorzystywany w zawodach Pascal VOC po 2008 roku. Polega na próbkowaniu krzywej precyzji w funkcji czułości w unikalnych punktach (r1, r2), w których wartość precyzji drastycznie spada. Dzięki tej zmianie, obszar pod interpolowaną krzywą jest dokładnie zmierzony.



Rysunek 2.11: Przykład wykresu AUC (ang. Area Under Curve). Źródło: [11]

W tej metodzie nie potrzebna jest aproksymacja czy interpolacja. Zamiast próbkowania 11 punktów krzywej, próbkuje się wartości precyzji, jak wspomniano wyżej, w unikalnych punktach (r1, r2), w których jej wartość drastycznie spada.

$$AP = \sum (r_{n+1} - r_n) \times p_{interp}(r_{n+1})$$

$$p_{interp}(r_n + 1) = max[p(r)]$$

Różni się ona od interpolowanej średniej metody uwzględnieniem wartości w unikalnych punktach, których poprzednia metoda nie uwzględniała.

COCO mAP

Ostatnie badania bazują głównie na zbiorze COCO Dataset. Do ewaluacji rezultatów trenowania na zbiorze COCO została stworzona indywidualna metryka zwana COCO mAP. Używa ona 101-punktowej definicji AP. W przypadku COCO, progiem IoU nie jest już tylko wartość 0,5. Ocenia się wyniki dla różnych wartości IoU z przedziału od 0,5 do 0,95 z krokiem 0,05. Podczas konkursów COCO oceniana jest AP z na 10 poziomach IoU i dla 80 kategorii obiektów. Wykorzystywanych jest również kilka innych metryk.

```
Average Precision (AP):
   AP                  % AP at IoU=.50:.05:.95 (primary challenge metric)
   AP^IoU=.50          % AP at IoU=.50 (PASCAL VOC metric)
   AP^IoU=.75          % AP at IoU=.75 (strict metric)
AP Across Scales:
   AP^small            % AP for small objects: area < 32^2
   AP^medium           % AP for medium objects: 32^2 < area < 96^2
   AP^large            % AP for large objects: area > 96^2
Average Recall (AR):
   AR^max=1            % AR given 1 detection per image
   AR^max=10           % AR given 10 detections per image
   AR^max=100          % AR given 100 detections per image
AR Across Scales:
   AR^small            % AR for small objects: area < 32^2
   AR^medium           % AR for medium objects: 32^2 < area < 96^2
   AR^large            % AR for large objects: area > 96^2
```

Rysunek 2.12: Poszczególne metryki uwzględnione w COCO mAP. Źródło: [11]

Rozdział 3

Przegląd istniejących rozwiązań

Istnieje szereg rozwiązań zajmujących się tematem segmentacji instancji. Jest ich znacznie mniej niż w przypadku segmentacji semantycznej, mimo to nie sposób przybliżyć je wszystkie. Z tego właśnie powodu, poniżej zostało opisanych tylko kilka z nich - szczególnie kluczowych dla rozwoju tego pola nauki czy też prezentujących inne podejście niż wcześniej zaprezentowane rozwiązania.

3.1 Mask R-CNN

Maska R-CNN jest konceptualnie prostą oraz bardzo efektywną strukturą stworzoną do zadania segmentacji instancji. Wykrywa ona obiekty na obrazie, tworząc jednocześnie wysokiej jakości maskę segmentacji dla każdej instancji obiektu. Przykładowe wyniki zastosowania Mask R-CNN przedstawiono na rys. 3.1

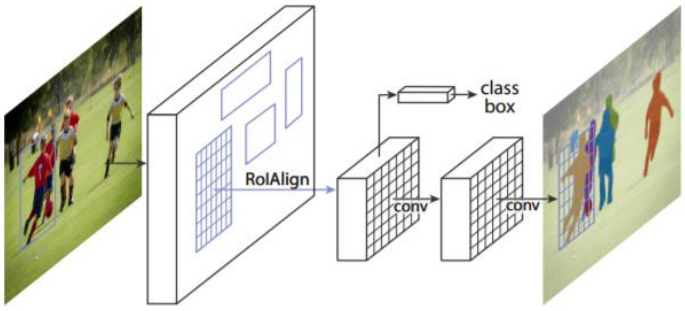
Rysunek 3.1: Przykładowe wyniki zastosowania Mask R-CNN na zbiorze COCO Dataset. Źródło: [4]

Maska R-CNN jest rozszerzeniem koncepcji Faster R-CNN poprzez dodanie maski dla każdej instancji obiektu do już istniejącego rozwiązania wykrywania obiektów na obrazie. Jest ona łatwa do implementacji oraz osiąga topowe rezultaty. W dniu publikacji przewyższała ona wszystkie istniejące rozwiązania w dziedzinach - segmentacji instancji, wykrywania obiektów z użyciem ramki granincznej oraz rozpoznawania pozycji ludzkiego ciała, włączając w to zwycięzców COCO 2016 Challenge

- metodę FCIS (Fully Convolution Instance-aware Semantic Segmentation) przybliżoną na dalszych stronach tej pracy. [4]

W ostatnich latach techniki przeznaczone do wykrywania obiektów oraz przeprowadzania segmentacji semantycznej zrobiły ogromny postęp. W dużej mierze stało się to za sprawą wynalezionych w tamtym czasie potężnych rozwiązań: Fast/Faster RCNN i Fully Convolutional Network (FCN) – struktur stworzonych do odpowiednio wykrywania obiektów oraz segmentacji semantycznej. Metody te są proste, intuicyjne i elastyczne oraz oferują akceptowalny czas trenowania. Maska R-CNN jest próbą stworzenia tak dobrej metody jak wspomniane wcześniej – Faster R-CNN oraz FCN, w tym przypadku dla zadania segmentacji instancji.

Segmentacja instancji jest zadaniem trudnym – wymaga jednoczesnego wykrywania wszystkich obiektów na obrazie i segmentacji każdej z instancji obiektu. Dlatego łączy ona dwa mechanizmy – wykrywania obiektów oraz lokalizacji każdego z nich przy pomocy ramki granicznej oraz segmentacji semantycznej, która ma za zadanie zakwalifikowanie każdego piksela obrazu do konkretnej klasy. Biorąc to pod uwagę, można by oczekiwać, że maska R-CNN będzie bardzo złożonym algorytmem. Jednakże jest ona bardzo prosta i prześciga wiele dotychczasowych metod segmentacji instancji.

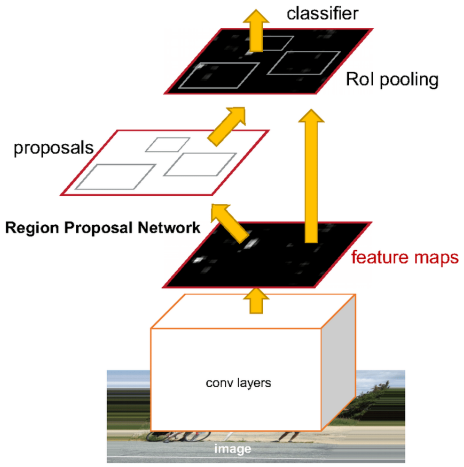


Rysunek 3.2: Architektura Mask R-CNN źródło: [4]

Rozszerzenie działania Faster R-CNN w metodzie Mask R-CNN polega na dodaniu predykcji masek segmentacji na każdym regionie zainteresowania (ang. RoI – Region of Interest) przy równoległym istniejącym mechanizmie klasyfikacji i predykcji ramki granicznej.

Mechanizm masek składa się z małej sieci FCN, zaaplikowanej na każdym obszarze zainteresowania, która przewiduje maskę segmentacji na poziomie pojedynczego piksela. Mask R-CNN, ze względu na swoją niezbyt skomplikowaną strukturę, jest łatwa do implementacji oraz nie niesie za sobą dużych wymagań obliczeniowych.

Faster R-CNN składa się z dwóch głównych etapów. Pierwszy etap wykorzystuje sieć RPN (Region Proposal Network) do zaproponowania ramek granicznych wokół obiektów. Drugi etap, który jest w istocie metodą Fast R-CNN (poprzednią wersją wymienionego wyżej mechanizmu), wyodrębnia cechy z każdego obszaru zawierającego się w ramce granicznej i na ich podstawie dokonuje klasyfikacji obiektu oraz lepszego dopasowania ramki granicznej. Architektura Faster R-CNN została przedstawiona na rys. 3.3



Rysunek 3.3: Architektura Faster R-CNN Źródło: [17]

W mechanizmie Mask R-CNN zaadoptowano tę dwuetapową procedurę techniki Faster R-CNN – pierwszy etap jest ten sam, natomiast przy drugim etapie, oprócz predykcji ramki obiektu i jego klasyfikacji, zostaje stworzona maska obiektu. Jest to całkowicie inne podejście w porównaniu do poprzednich technik, które dokonywały klasyfikacji obiektu na podstawie istniejącej już maski.

3.2 Yolact

W ciągu ostatnich kilku lat zostały poczynione duże postępy w kontekście rozwoju dziedziny, jaką jest segmentacja instancji. Działo się tak głównie dlatego, że również algorytmy wykrywania obiektów - Faster R-CNN czy R-FCN, stały się coraz bardziej efektywne, co skutkowało rozwojem bazujących na nich algorytmów segmentacji instancji - Mask R-CNN czy FCIS. Jednak metody te skupiały się głównie na wydajności, a nie prędkości, przez co segmentacja instancji w czasie rzeczywistym pozostawała obszarem nieosiągalnym, chociaż detektory obiektów w czasie rzeczywistym, takie jak YOLO czy SSD, były już szeroko wykorzystywane.

Pomimo dobrych wyników wspomnianych wcześniej detektorów zastosowanie tego samego podejścia – usunięcia jednego stopnia z dwustopniowych detektorów i nadrabiania utraconej wydajności na inne sposoby, nie było możliwe w zadaniu segmentacji instancji. Nie było takiej możliwości, ponieważ najnowocześniejsze dwustopniowe metody tej techniki bazowały na lokalizacji ramek granicznych, aby później wyprodukować na ich podstawie maski.

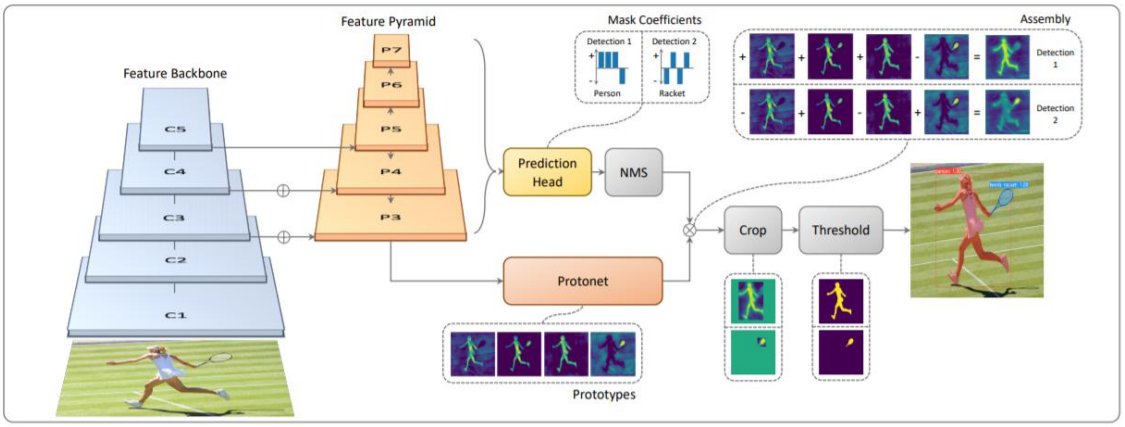
Oznacza to, że metody te ponownie ekstrahowały cechy z obszarów ramek granicznych, które następnie trafiały do predyktora maski. Podejście to – z natury wymagające określonej sekwencji działań – było z tego powodu trudne do przyspieszenia. Metody, które wykonują te zadania równocześnie, na przykład metoda FCIS, wymagała znacznych zasobów już po lokalizacji ramki granicznej, przez co była daleka od wykonywania w czasie rzeczywistym.

Z tego właśnie powodu został stworzony algorytm Yolact umożliwiający przeprowadzenie procesu segmentacji instancji w czasie rzeczywistym.

Yolact (ang. You Only Look At Coefficients) jest prostym, w pełni splotowym modelem przeznaczonym do segmentacji instancji w czasie rzeczywistym, który

osiąga najlepsze wyniki spośród wszystkich istniejących obecnie algorytmów służących do wspomnianego wcześniej zagadnienia [1].

Wynik ten jest uzyskiwany dzięki podzieleniu zadania segmentacji instancji na dwa równoległe podzadania: generowanie zestawu masek prototypowych na całym obrazie oraz predykcję współczynników maski dla danej instancji obiektu. Następnie przeprowadzana jest segmentacja instancji dla całego obrazu poprzez liniowe połączenie masek prototypowych wykorzystując odpowiadające przewidywane współczynniki i wycinane z obrazu za pomocą przewidywanej ramki granicznej. Architektura Yolact została przedstawiona na rys. 3.4



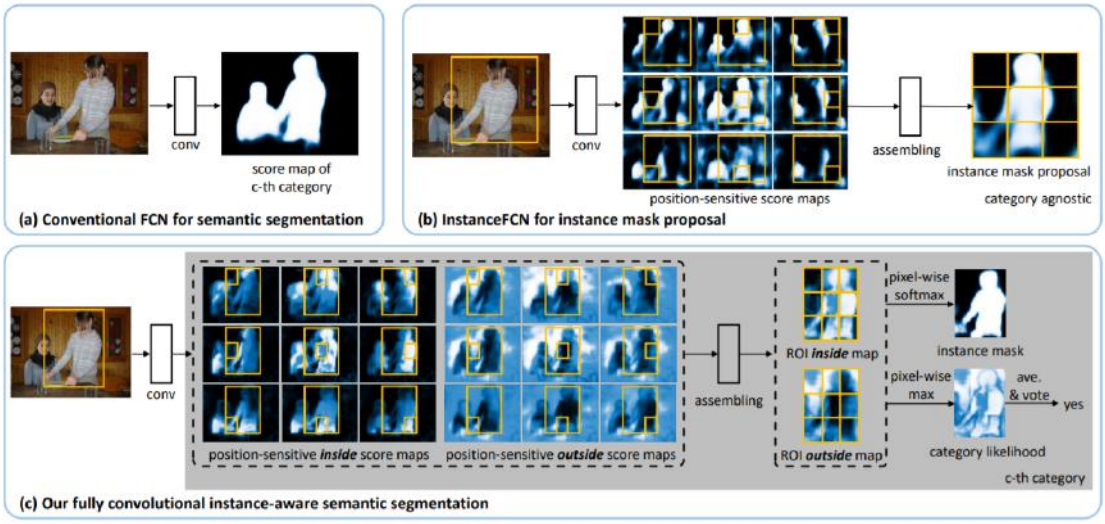
Rysunek 3.4: Architektura Yolact. Źródło: [1]

Liczba masek prototypowych jest niezależna od liczby zadeklarowanych klas, co prowadzi do sytuacji, w której liczba prototypów może być mniejsza niż liczba klas. Yolact uczy się rozproszonej reprezentacji, w której każda instancja obiektu jest segmentowana za pomocą kombinacji prototypów, które są wspólne dla różnych klas. Ta rozproszona reprezentacja prowadzi do interesujących wyników, w których prototypy dokonują różnych operacji – dzielą obraz na części, lokalizują instancje obiektów, niektóre lokalizują tylko kontury danego obiektu, a wynikiem działania większości z nich jest po prostu kombinacja wyżej wymienionych operacji.

3.3 Fully Convolutional Instance-aware Semantic Segmentation

Fully Convolutional Instance-aware Semantic Segmentation (FCIS) jest rozwiązaniem wykorzystującym Fully Convolutional Network (FCN) do zadania segmentacji semantycznej przy jednoczesnym wyodrębnianiu instancji obiektów. FCIS osiąga wysokie wyniki pod względem dokładności i wydajności, dzięki czemu zostało zwycięzcą COCO 2016 Segmentation Competition dystansując konkurencję [5].

Zasadnicze działanie sieci FCN jest podzielone na dwa podzadania oraz między wszystkie obszary zainteresowań (ang. Region of Interest - RoI). Sieci FCN pobierają obraz wejściowy o dowolnym rozmiarze i wykorzystując kilka warstw splotowych generują mapy wyników z dokładnością do piksela, jak pokazano na rys. 3.5.



Rysunek 3.5: Ilustracja idei Fully Convolutional Instance-aware Semantic Segmentation. Źródło: [5]

Dzięki swojej prostocie i wydajności zapewniają dokładne, szybkie i kompleksowe rozwiązanie dla zadania segmentacji semantycznej. Jednak klasyczne sieci FCN nie mają zastosowania w przypadku segmentacji semantycznej z uwzględnieniem instancji obiektu, które wymaga jednocześnie wykrywania i segmentacji poszczególnych instancji obiektu.

Jednoczesne wykrywanie i segmentacja jest trudna, ponieważ dany piksel jest klasyfikowany bez analizy szerszego kontekstu danej sceny. Segmentacja semantyczna przy jednoczesnej segmentacji instancji musi działać na poziomie mniejszego obszaru, aby jeden piksel mógł być rozumiany w kontekście konkretnego obszaru obrazu, a nie jego całości. Ten mechanizm nie może być przeprowadzony przez pojedynczą sieć FCN na poziomie całego obrazu. Problem różniących się od siebie klasyfikacji konkretnego piksela na podstawie różnych RoI został przedstawiony na rys. 3.6.

Rysunek 3.6: Klasyfikacja danego piksela na podstawie szerszej sceny. Źródło [5]

Aby rozwiązać ten problem, segmentację semantyczną z uwzględnieniem instancji obiektu osiąga się poprzez zastosowanie różnych typów podsieci w trzech etapach:

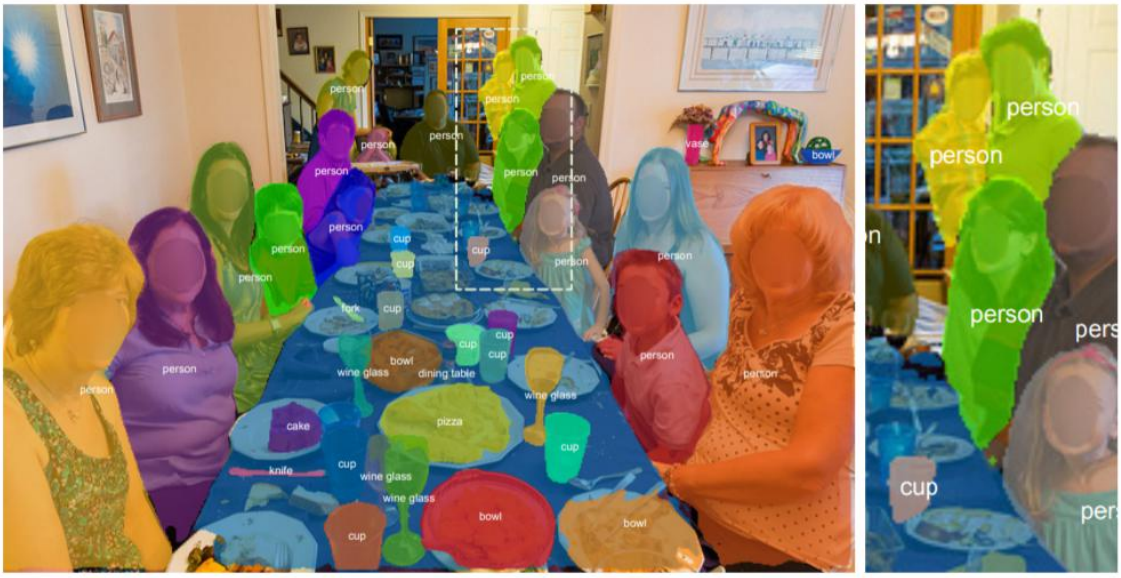
1. Sieć FCN jest stosowana na całym obrazie w celu wygenerowania map pośrednich oraz wspólnych cech.
2. Z każdej mapy cech wspólnych warstwa łącząca wypacza każdy obszar zainteresowania w mapę cech stałego rozmiaru dla każdego obszaru zainteresowania.
3. Jedna lub więcej warstw całkowicie połączonych w ostatniej sieci zamienia mapy cech dla każdego regionu zainteresowania w maski dla każdego regionu zainteresowania.

Jednak takie podejście ma również wiele wad. Między innymi:

1. Krok, w którym warstwa łącząca działa na obszarze zainteresowania prowadzi do utraty szczegółów przestrzennych z powodu wypaczenia cech i zmiany rozmiarów obszarów zainteresowania. Zmiana ich rozmiarów jest konieczna, aby uzyskać stały rozmiar wejściowy dla kolejnych warstw sieci. Jest to szczególny problem w przypadku dużych obiektów, w przypadku których zmniejszenie rozmiaru zmniejsza znacząco dokładność segmentacji.
2. Warstwy całkowicie połączone nadmiernie parametryzują zadanie, nie wykorzystując regularności lokalnego podziału wag.

3.4 TensorMask

Detektory obiektów oparte na przesuwającym się oknie, generujące ramki graniczne wokół przewidywanych obiektów na gęstej, regularnej siatce szybko się rozwinęły i stały się popularne. Natomiast w przypadku metod segmentacji instancji nadal dominują metody oparte na Faster R-CNN i Mask R-CNN, które najpierw wykrywają ramki ograniczające, a następnie lokalizują obiekty w ich obrębie. Próbą zaimplementowania paradygmatu przesuwającego się okna w zadaniu segmentacji instancji jest metoda TensorMask.

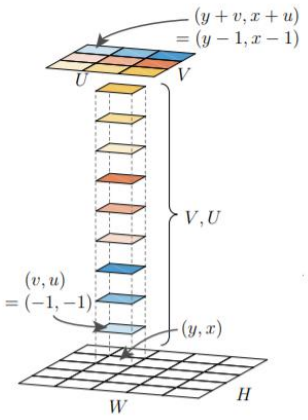


Rysunek 3.7: Przykładowe rezultaty wykorzystania algorytmu Tensormask. Źródło [2]

W przypadku segmentacji instancji zadanie to zasadniczo różni się od zadań wykorzystujących paradygmat przesuwającego się okna w zadaniu segmentacji semantycznej czy wykrywaniu ramek ograniczających, ponieważ dane wyjściowe w każdym miejscu przestrzennym są same w sobie geometryczną konstrukcją o własnych wymiarach przestrzennych.

W przeciwieństwie do ramek ograniczających, które są stałego wymiaru, maski segmentacji mogą bazować na bogatszych, bardziej uporządkowanych reprezentacjach. Koncepcja TensorMask zakłada reprezentowanie masek za pomocą tensorów wysokowymiarowych, który pozwala na eksplorowanie nowatorskich architektur sieci stworzonych do gęstej predykcji maski [2].

Główną ideą TensorMask jest użycie strukturalnych tensorów 4D (V,U,H,W) do przedstawienia masek obiektów w przestrzeni domen, w których zarówno para H i W, która reprezentuje pozycję obiektu, jak i para V i U reprezentująca względne położenie maski są subtensorami geometrycznymi, to znaczy mają osie o znaczeniu geometrycznym w odniesieniu do obrazu. Pozwala to na umożliwienie działania sieci bezpośrednio na subtensorze (V, U), przedstawionym na rys. 3.8, w sensie geometrycznym, włączając w to transformację współrzędnych, skalowanie czy zastosowanie piramid skali.



Rysunek 3.8: Przykład subtensora używanego w metodzie TensorMask. Źródło [2]

Zasada działania polega na opracowaniu piramidy z indeksowaną według skali listą tensorów 4D, którą nazywamy bipiramidą tensorową. Analogicznie do piramidy cech, która jest listą map cech w różnych skalach, bipiramida tensorowa zawiera listę tensorów 4D.

Rozdział 4

Opis wybranej koncepcji rozwiązania problemu

Tematem niniejszej pracy inżynierskiej jest porównanie wybranych metod segmentacji instancji w obrazach cyfrowych. Biorąc pod uwagę dostępny sprzęt, czas oraz stopień trudności implementacji istniejących rozwiązań przedstawionych w rozdziale trzecim, do porównania zostały wybrane dwie opisane wcześniej metody - Mask R-CNN oraz Yolact.

Porównanie uwzględnia również dostępność wyżej wymienionych modeli w konfiguracjach z wykorzystaniem różnych rdzeni (ang. backbone), w przypadku modelu Yolact są to Darknet53, Resnet101 oraz Resnet50, natomiast dla Mask R-CNN jest to Resnet101. Dla metody Yolact i rdzenia Resnet101, zostaną przedstawione również dwie konfiguracje zróżnicowane pod względem rozmiarów obrazów zbioru uczącego.

Badania mają na celu wytrenowanie modeli na zbiorze COCO Dataset przy wykorzystaniu różnych konfiguracji oraz przeprowadzenie ich ewaluacji polegającej na:

- wyliczeniu współczynnika mAP przy różnych wartościach parametru IoU dla kolejnych poziomów wytrenowania modeli, aby móc ocenić precyzję poszczególnych konfiguracji danej metody,
- wyznaczeniu szybkości przetwarzania (w kl./s) dla konkretnej konfiguracji, pozwalającej ocenić zdolność danej konfiguracji do wykrywania oraz segmentacji obiektów w czasie rzeczywistym lub zbliżonym do rzeczywistego,
- stworzeniu zestawienia czasów uczenia w zależności od konkretnej konfiguracji, aby uzyskać informację na temat potrzebnych zasobów czasowych oraz sprzętowych do zastosowania danej metody,
- porównaniu wyników wytrenowanych modeli z modelami udostępnionymi przez twórców rozwiązań.

Kolejnym krokiem jest analiza zebranych wyników badań oraz sformułowanie wniosków, pozwalających na wybranie odpowiedniego modelu i konfiguracji na potrzeby konkretnej implementacji.

Rozdział 5

Opis wybranych narzędzi i bibliotek

W celu realizacji wybranej koncepcji rozwiązania problemu niniejszej pracy, można skorzystać z wielu narzędzi oraz bibliotek. Po wnikliwej analizie ich dostępności, łatwości implementacji oraz kompatybilności z wykorzystywanym sprzętem i oprogramowaniem, została wyselekcjonowana grupa narzędzi oraz bibliotek do realizacji tego zadania. Zostały one pokrótce scharakteryzowane w kolejnych sekcjach pracy.

5.1 Wykorzystane narzędzia

PyCharm jest to środowisko programistyczne (IDE – Integrated Development Environment) stworzone dla języka programowania Python przez firmę JetBrains. Umożliwia ono edycję kodu, jego debugowanie oraz integrację z systemem kontroli wersji. Pracuję na wszystkich powszechnie stosowanych platformach systemowych tj. Microsoft Windows, Linux i macOS. W niniejszej pracy inżynierskiej zostało ono wykorzystane na platformie Microsoft Windows 10.

Anaconda jest środowiskiem wspierającym tworzenie kilku projektów wykorzystujących różne biblioteki. Za jej pomocą można zainstalować biblioteki dla danego projektu, bez obawy o konflikt z bibliotekami zainstalowanymi w przypadku projektu, który wymaga, na przykład, poprzedniej wersji danej biblioteki.

Jupyter Notebook (wcześniej IPython Notebook) jest to internetowe interaktywne środowisko obliczeniowe zaprojektowane do swoistych „notatników". Termin „notatnik" (Notebook) może być rozumiany na kilka sposobów m.in. jako aplikacja internetowa Jupyter, serwera www Jupyter lub formatu dokumentu typu Jupyter. Dokument Jupyter to dokument JSON zawierający uporządkowaną listę komórek wejściowych i wyjściowych, które mogą zawierać kod w języku Python, tekst (również dowolnie sformatowany), obliczenia, wykresy czy multimedia. Rozszerzenie pliku typu Jupyter Notebook to .ipynb.

Może być on przekonwertowany na wiele standardowych formatów tj. HTML, LaTeX, PDF czy Python poprzez użycie „pobierz jako" bezpośrednio w internetowym środowisku, z wykorzystaniem biblioteki nbconvert lub poprzez komendę „jupyter nbconvert" w wierszu poleceń.

Pip jest to system zarządzania pakietami oprogramowania napisanymi w języku Python. Większość wersji Pythona - Python 2.7.9 i wyższe oraz Python 3.4 i wyższe - jest dystrybuowanych z preinstalowanym pipem.

Git jest to rozproszony system kontroli wersji przeznaczony do śledzenia zmian w kodzie źródłowym podczas tworzenia oprogramowania. Głównie służy do koordynowania pracy zespołu programistów, ale korzysta się z niego również do śledzenia zmian w dowolnym zestawie plików.

Aby wykorzystać możliwości karty graficznej w procesie uczenia sieci zaimplementowane zostało CUDA i CuDNN.

CUDA (ang. Compute Unified Device Architecture) jest to zarówno platforma obliczeniowa, jak również model programowania opracowany przez firmę Nvidia w celu przeprowadzania obliczeń na graficznych jednostkach przetwarzających GPU.

CuDNN (ang. CUDA Deep Neural Network) – biblioteka wspierana przez GPU komputera zapewniająca podstawowe funkcje dla głębokich sieci neuronowych. Zapewnia ona wysokiej jakości implementacje standardowych procedur takich jak: konwolucje do przodu i do tyłu (ang. forward and backward convolution), oraz warstw łączących (ang. pooling layers), warstw normalizujących (ang. normalization layers) i warstw aktywacji (ang. activation layers).

Badacze głębokich sieci neuronowych szeroko wykorzystują CuDNN w celu uzyskania wysokiej wydajności na GPU. CuDNN wspiera szeroko stosowane platformy uczenia maszynowego takie jak Caffe, Caffe2, Chainer, Keras, TensorFlow czy PyTorch.

5.1.1 Wykorzystany sprzęt

Pierwsze próby zostały przeprowadzone na sprzęcie charakteryzującym się poniższymi parametrami:

- **Procesor:** Intel Core i5 9gen (9300H)
  2,4 GHz max. 4,1 GHz /8GB RAM/4-rdzeniowy/8-wątkowy
- **Karta graficzna:** Nvidia GeForce 1660Ti with Max-Q Design
  o pamięci 6 GB + Intel UHD Graphics 630
- **System operacyjny:** Microsoft Windows 10

Ze względu na problemy z ilością dostępnej pamięci i wydajnością używanej karty graficznej, skorzystano ze sprzętu należącego do zasobów uczelni i charakteryzującego się poniższymi parametrami:

- **Procesor:** AMD Ryzen Threadripper 1920X 12-Core Procesor
- **Karta graficzna:** Nvidia GeForce 2080Ti o pamięci 11 GB
  + Nvidia GeForce 980ti o pamięci 6 GB
- **System operacyjny:** Ubuntu 18.04

5.2 Wykorzystane biblioteki

TensorFlow - biblioteka stworzona do szybkich numerycznych obliczeń, używana do tworzenia modeli głębokiego uczenia.

PyTorch – biblioteka open-source stworzona do uczenia maszynowego. Została opracowana przez Facebook Artifical Inteligence Research Group.

Matplotlib – biblioteka kreślarska dla języka Python oraz jego roszerzenia numerycznego NumPy. Zapewnia obiektowy interfejs API służący do osadzania grafów, wykresów etc. w aplikacjach.

NumPy – biblioteka wspierająca duże, wielowymiarowe tablice i macierze oferując ogromną ilość złożonych matematycznych funkcji do działania na tych tablicach.

SciPy – biblioteka wyposażona w moduły do różnych, często używanych zadań w programowaniu naukowym, w tym algebry liniowej, rozwiązywania równań różniczkowych i przetwarzania sygnałów.

Pillow – biblioteka będąca rozszerzeniem PIL (Python Image Library). Została ona oparta na kodzie PIL, jednak ewoluowała do lepszej nowszej i bardziej przyjaznej wersji. Umożliwia ona obsługę otwierania, manipulowania i zapisywania wielu różnych formatów plików graficznych.

Cython – optymalizujący kompilator statyczny zarówno dla języka programowania python, jak i dla rozszerzonej języka programowania cython (opartego na Pyrex). Umożliwia pisanie rozszerzeń w języku C, tak samo prosto jak po prostu w języku Python.

SciKit-image – biblioteka typu open-source dla języka programowania Python. Zawiera algorytmy do segmentacji, transformacji geometrycznych, manipulacji przestrzenią kolorów, analizy, filtorwania, morfologii, wykrywania cech i wiele innych. Jest przeznaczona do współpracy z bibliotekami numerycznymi i naukowymi NumPy i SciPy.

opencv-python – biblioteka zaprojektowana do rozwiązywania problemów związanych z wizją komputerową. Korzysta ona z biblioteki NumPy – wszystkie tablice są konwertowane do i z tablic NumPy.

h5py – interfejs Pythona do formatu danych binarnych hdf5. Pozwala na przechowywanie ogromnych ilości danych liczbowych i łatwe manipulowanie danymi pochodzącymi z NumPy. Umożliwia podzielenie wielobajtowe zestawy danych na mniejsze tablice obsługiwane przez NumPy.

imgaug – biblioteka służąca do powiększania obrazu w zadaniach uczenia maszynowego. Obsługuje szeroki zakres technik augmentacji, pozwala łatwo łączyć je ze sobą, wykonywać w losowej kolejności lub na wielu rdzeniach procesora. Ma prosty, ale potężny interfejs stochastyczny i może nie tylko powiększać obrazy, ale także punkty kluczowe/punkty orientacyjne, ramki graniczne, mapy cieplne i mapy segmentacji.

IPython – interpreter języka Python zapewniający bogaty zestaw narzędzi pozwalający na pracę z kodem w Jupyther Notebooksach i innych interaktywnych nakładkach.

5.3 Problemy napotkane podczas instalacji i konfiguracji narzędzi oraz bibliotek

Realizując cel niniejszej pracy dyplomowej napotkano na kilka problemów podczas instalacji i konfiguracji potrzebnych narzędzi. Problemy, wraz z propozycjami ich rozwiązania, zostały pokrótce opisane poniżej.

Kompatybilność wersji Pythona z bibliotekami
Należy zwrócić uwagę, aby wykorzystywane biblioteki wspierały wersję Pythona, która ma zostać wykorzystana. Początkowo popełniono błąd instalując najnowszą wersję Pythona - 3.8 i chcąc później zainstalować bibliotekę matplotlib, która tej najnowszej wersji nie wspierała. Rozwiązaniem tego problemu było odinstalowanie wersji 3.8 i zainstalowanie wersji 3.7.5, która była już wersją sprawdzoną.

Problem z instalacją biblioteki imgaug
Instalacja biblioteki imgaug może stwarzać problem ze względu na swoje zależności z inną biblioteką - Shapely. Rozwiązaniem tego problemu jest zainstalowanie tylko biblioteki imgaug, bez jej zależności względem innych bibliotek wykorzystując polecenie: pip install git+https://github.com/aleju/imgaug –no-deps.

Problem z instalacją biblioteki pycocotools
Często podstawowa komenda: pip install pycocotools nie wystarcza do zainstalowania tej biblioteki i generuje błędy. Na początek trzeba pamiętać, żeby przed instalacją tej biblioteki zainstalować inną bibliotekę - Cython. Jeśli jednak to nie pomoże, należy instalować pycocotools ze źródła czyli githuba, używając komendy: pip install git+https://github.com/philferriere/cocoapi.gitegg=
pycocotoolssubdirectory=PythonAPI

Problem z biblioteką Tensorflow
- 'module' object has no attribute 'placeholder'.
Problem ten pojawia się przy wersji Tensorflow 2.x. Można wykorzystać wersję 1.x, jednak niektóre rozwiązania, a tym bardziej nowopowstałe i przyszłe z nich, wymagają wersji co najmniej 2.x. Rozwiązaniem tego problemu jest zmiana formuły importowania biblioteki tensorflow na:
import tensorflow.compat.v1 as tf tf.disable_v2_behavior()

Problem: 'Coco has no attribute CocoConfig'
Jeżeli plik coco.py został pobrany z lokalizacji: https://github.com/waleedka/coco, nie zawiera on CocoConfig. Aby rozwiązać ten problem, należy pobrać coco.py z lokalizacji: https://github.com/karolmajek/Mask_RCNN/blob/master/coco.py.

Problem: -scipy.mics has no attribute 'imresize'
Na początku należy sprawdzić, jaka wersja biblioteki sciPy została użyta. Jeżeli jest to wersja 1.3.0 i wyższa, nie zawiera ona atrybutu „imresize". Należy wtedy odinstalować obecną wersję i zainstalować wcześniejszą wersję tejże biblioteki. W niniejszej pracy została wykorzystana wersja 1.2.2. Polecenie: pip install scipy==1.2.2

Problem z biblioteką Tensorflow:
- Could not load dynamic library 'cudart64_100.dll
Biblioteka tensorflow-gpu, którą należy zainstalować, aby móc korzystać z wydajności procesora graficznego w procesie trenowania sieci, wymaga zainstalowania CUDA Toolkit. Należy jednak mieć na uwadze, jaka wersja CUDA Toolkit została zainstalowana. Wersja CUDA Toolkit musi zgadzać się z wersją biblioteki Tensorflow. Wersja z września 2018 zawiera biblioteki xx100.dll, starsze wersje będą zawierały kolejno xx101.dll, xx102.dll. Rozwiązaniem tego problemu jest pobranie odpowiedniej wersji CUDA Toolkit kierując się wersją zainstalowanej biblioteki Tensorflow.

Konfiguracja narzędzi oraz bibliotek do zrealizowania celu tej pracy jest zadaniem co najmniej skomplikowanym. Należy jednak podkreślić, że cała wiedza potrzebna do przeprowadzenia tego procesu jest dostępna w Internecie. Istnieje wiele forów internetowych, gdzie użytkownicy dzielą się wymaganą wiedzą. Jako główne źródła informacji poleca się stackoverflow.com oraz github.com.

Rozdział 6

Opis przeprowadzonych prac

Prace przeprowadzone nad realizacją celu niniejszej pracy dyplomowej zostały podzielone na dwie części: konfigurację potrzebnych narzędzi oraz przeprowadzenie badań i analizę uzyskanych wyników.

6.1 Opis konfiguracji narzędzi

W celu przeprowadzenia badań niezbędnych do realizacji celu niniejszej pracy, należało w pierwszej kolejności skonfigurować potrzebne narzędzia. Proces ich konfiguracji został przedstawiony w kolejnych podsekcjach.

6.1.1 Python

Pierwszym podjętym krokiem było zainstalowanie środowiska Python na wykorzystywanym komputerze. W tym celu odwiedzono stronę python.org i postępując zgodnie z instrukcjami pobrano wersję Python 3.8.

Następnie zainstalowano biblioteki potrzebne do uruchomienia dostępne rozwiązania. Wersje Pythona od 3.4 są dostarczane z już preinstalowanym, wspomnianym wcześniej w opisie narzędzi, systemem zarządzania pakietami pip, więc nie wymagał on instalacji. Instalowanie bibliotek polegało na uruchomieniu terminala oraz wykorzystywaniu poleceń pip install 'nazwa biblioteki' dla poszczególnych bibliotek.

Podczas tego procesu wystąpiło kilka komplikacji. Zostały one opisane w podrozdziale 5.3 wraz ze wskazówkami ich rozwiązań. Problemy z instalacją dotyczyły, między innymi, biblioteki matplotlib, imgaug, pycocotools, scipy czy tensorflow.

W trakcie rozwiązywania błędu przy próbie instalacji biblioteki matplotlib zorientowano się, że nie wspiera ona Pythona w wersji 3.8, więc wersja 3.8 została odinstalowana i zainstalowano sprawdzoną już wersję 3.7.5. Dla tej wersji przeprowadzono instalację bibliotek w taki sam sposób, jak dla poprzedniej wersji.

6.1.2 Mask R-CNN

W celu konfiguracji Mask R-CNN, pierwszym podjętym działaniem była próba sklonowania repozytorium, które zawierało całą architekturę Mask R-CNN. Do tego celu wymagana była, w pierwszej kolejności, instalacja kolejnego narzędzia – systemu git. Skorzystano z wersji dostępnej na stronie https://git-scm.com/downloads.

Po instalacji systemu git sklonowano repozytorium zawierające Mask R-CNN za pomocą terminala i komendy, git clone https://github.com/matterport/Mask_RCNN.

Następnie zostały pobrane częściowo wytrenowane (na zbiorze COCO Dataset) wagi dla Mask R-CNN dostępne pod adresem:
https://github.com/matterport/Mask_RCNN/releases jako mask_rcnn_coco.h5,
aby móc przetestować dostępne rozwiązania, zanim zostaną wytrenowane własne modele.

Repozytorium zawierające Mask R-CNN okazało się posiadać również pliki .ipynb – Jupyther Notebooks. Aby móc je otworzyć i z nich skorzystać, zainstalowany został program Anaconda. Anaconda pozwala na korzystanie z Jupyter Notebooks i za jej pomocą można również skonfigurować biblioteki, z których notebooki korzystają. Naturalnym kolejnym zadaniem było skonfigurowanie Anacondy – zainstalowanie bibliotek przy użyciu przystosowanego interfejsu oraz, w przypadku braku dostępnej wersji lub innych problemów z instalacją przez interfejs, przy pomocy terminala anaconda prompt i poleceń: conda install „nazwa biblioteki".

Następnie podjęto próbę wytrenowania własnego modelu. Pierwsza próba polegała na skorzystaniu z przygotowanego przez twórców repozytorium Notebooka do stworzenia modelu uczonego na dostępnym zbiorze kształtów. Wyniki tej próby zostały przedstawione poniżej.

Kolejnym krokiem była próba wytrenowania własnego modelu na zbiorze COCO Dataset. W tym celu został on pobrany ze strony
http://cocodataset.org/download. Aby móc wytrenować, testować oraz ewaluować wyniki należało pobrać paczki train, test i val oraz ich adnotacje (ang. annotations).

Następnie przy pomocy polecenia przeznaczonego do trenowania własnego modelu, znajdującego się w pliku readme wykorzystywanego rozwiązania, rozpoczęto trenowanie własnego modelu bazując na częściowo wytrenowanych już wagach.

Trenowanie modelu przebiegało prawidłowo, jednak trwało ono znacznie dłużej niż przewidywano na podstawie informacji o czasie wymaganym do wytrenowania własnych modeli wynikających z dostępnych artykułów. Po konsultacji z promotorem odnaleziono przyczynę długiego czasu trenowania modeli - procesor graficzny nie uczestniczył w trenowaniu modelu, ponieważ nie zostały zainstalowane biblioteki CUDA i CuDNN.

Kolejnym krokiem było zainstalowanie CUDA i CuDNN. CUDA zostało zainstalowane zgodnie z krokami opisanymi na oficjalnej stronie dystrybutora wykorzystywanej karty graficznej: https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html. Natomiast CuDNN zgodnie z tym linkiem:
https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html. Istotnym jest sprawdzenie, jaka wersja powyższych bibliotek będzie kompatybilna z dostępną kartą graficzną i używaną biblioteką tensorflow (problem z biblioteką tensorflow i CUDA Toolkit został opisany w rozdziale 5.3).

Kolejna próba wytrenowania modelu zakończyła się częściowym sukcesem. Procesor graficzny wziął udział w uczeniu, poszczególne biblioteki dynamiczne należące do CUDA Toolkit zostały załadowane, jednak pamięć procesora graficznego (6GB) okazała się niewystarczająca, aby móc pomieścić cały zbiór uczący COCO Dataset.

Po konsultacji z promotorem zdecydowano się na zmianę parametru, mającego kluczowe znaczenie w kontekście ilości pamięci zajmowanej przez zbiór uczący - zmniejszenie rozmiaru partii (ang. batch size).

6.1.3 Yolact

W celu konfiguracji Yolact, pierwszym działaniem, jakie zostało podjęte, było również sklonowanie repozytorium, które zawierało całą architekturę Yolact, przy pomocy terminala i komendy: git clone https://github.com/dbolya/yolact.git.

Następnie zostały pobrane częściowo wytrenowane (na zbiorze ImageNet) modele dla Yolact: Resnet50, Darknet53 oraz Resnet101 ze strony https://github.com/dbolya/yolact.

Ze względu na problemy z niewystarczającą ilością pamięci oraz długim czasem uczenia z wykorzystaniem dostępnego sprzętu, promotor zaproponował wykorzystanie uczelnianych zasobów sprzętowych o dużo większej mocy i wydajności. Opis sprzętu znajduje się w sekcji 5.1.1. Została wykorzystana wydziałowa sieć VPN oraz klient VNC Viewer do zdalnego połączenia się z komputerem i rozpoczęcia na nim pracy. Konfiguracja sprzętu należącego do uczelni była zadaniem dużo łatwiejszym niż w przypadku wcześniej wykorzystywanego sprzętu, ponieważ wiele narzędzi zostało już zainstalowanych, m.in. Python, CUDA Toolkit oraz większość bibliotek. Dalsze prace były wykonywane na sprzęcie należącym do uczelni.

6.2 Opis przeprowadzonych badań i analiza uzyskanych wyników

Przeprowadzone badania polegały na wytrenowaniu modeli wybranych dwóch metod segmentacji instancji - Mask R-CNN oraz Yolact na zbiorze COCO Dataset dla różnych konfiguracji, zróżnicowanych pod względem wykorzystanych rdzeni i rozmiarów obrazu zbioru uczącego oraz przeprowadzeniu ewaluacji wytrenowanych modeli biorąc pod uwagę:

- współczynnik mAP przy różnych wartościach parametru IoU dla kolejnych poziomów wytrenowania modeli,
- szybkość przetwarzania,
- czas uczenia

dla konkretnej konfiguracji oraz porównanie uzyskanych wyników z wynikami wytrenowanych modeli dostarczonych przez twórców rozwiązań.

Konfiguracje wykorzystane w poniższym zestawieniu:

- yolact_base_config - metoda Yolact, backbone: Resnet101, rozmiar obrazów zbioru uczącego: 550x550px,
- yolact_resnet50_config - metoda Yolact, backbone: Resnet50, rozmiar obrazów zbioru uczącego: 300x300px,
- yolact_im700_config - metoda Yolact, backbone: Resnet101, rozmiar obrazów zbioru uczącego: 700x700px,
- mask_rcnn - metoda Mask R-CNN, backbone: Resnet101, rozmiar obrazów zbioru uczącego: 1024x800px.

Początkowo do zestawienia miała zostać włączona również konfiguracja yolact_darknet53_config, jednak próba przeprowadzenia uczenia nie powiodła się.

Trenowanie modeli zostało przeprowadzone do 16 epoki, ze względu na możliwości sprzętowe oraz czas uczenia danych modeli, jednak zestawienie uwzględnia również modele dostarczone przez twórców rozwiązań wytrenowane do 54 epoki.

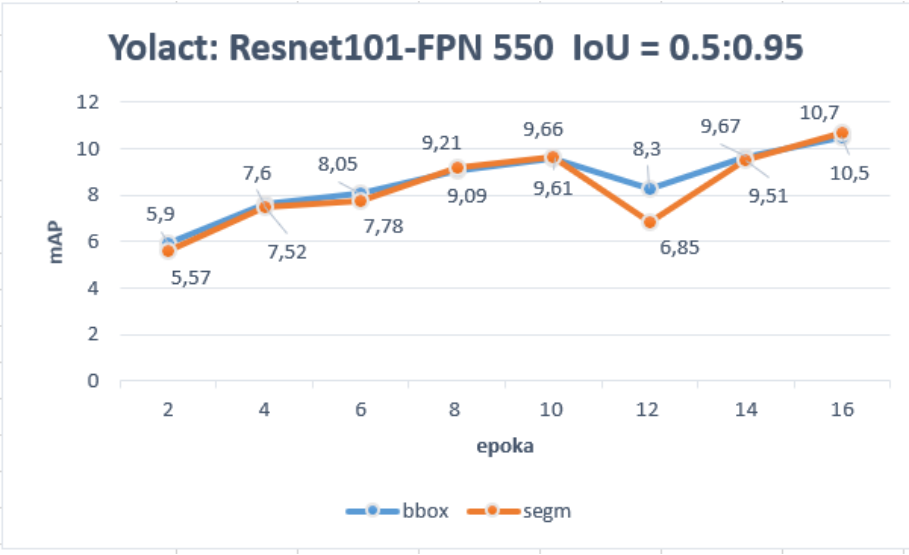
Ewaluacja modeli została przeprowadzona na zbiorze 1000 obrazów losowo wybranych ze zbioru walidacyjnego.

6.2.1 Współczynnik mAP

Pierwszym czynnikiem branym pod uwagę w procesie ewaluacji wytrenowanych modeli był współczynnik mAP przy różnych wartościach parametru IoU dla kolejnych poziomów wytrenowania modeli. Czynnik ten został przeanalizowany, aby móc ocenić precyzję poszczególnych konfiguracji danej metody.

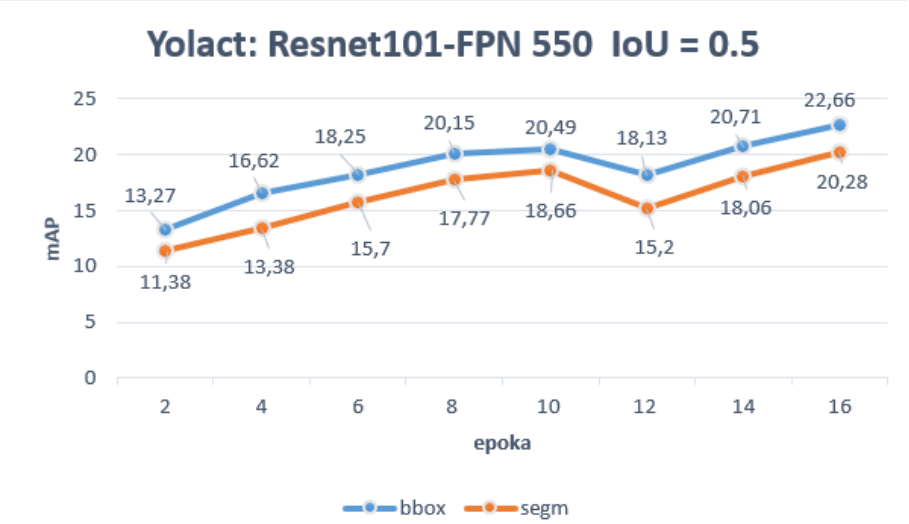
Yolact: Resnet101-FPN 550

Pierwszą uwzględnioną w badaniach konfiguracją był model Yolact ze rdzeniem Resnet101. Poniżej przedstawiono wyniki ewaluacji współczynnika mAP dla różnych wartości IoU.



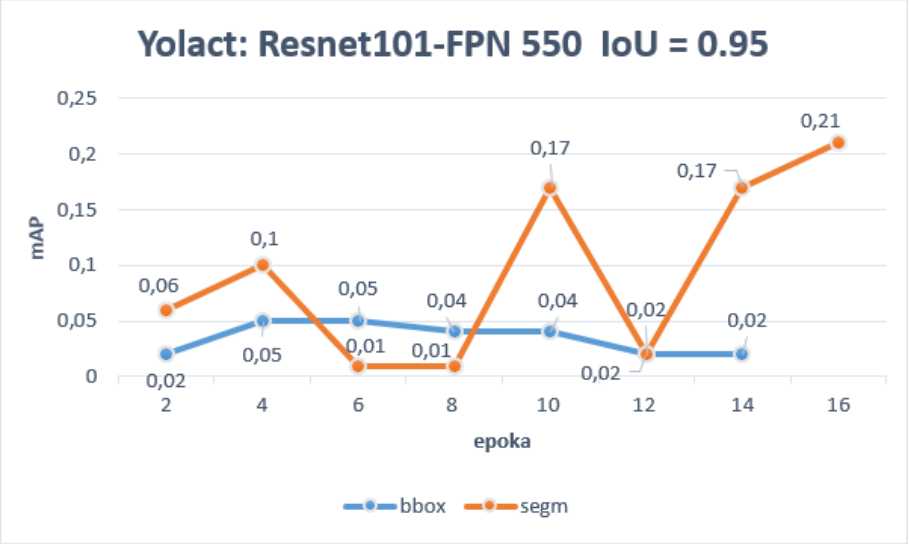
Rysunek 6.1: Wyniki ewaluacji modelu Yolact: Resnet101-FPN 550 dla IoU = 0,5:0,95. Źródło: opracowanie własne

Parametr mAP przy IoU = 0,5:0,95 rośnie wraz numerem epoki, do której został wytrenowany model. Dzieje się tak zarówno dla ramki ograniczającej, jak i segmentacji instancji. Występuje lekki spadek mAP dla epoki dwunastej, jednak trend utrzymuje się dla epoki czternastej oraz szesnastej.



Rysunek 6.2: Wyniki ewaluacji modelu Yolact: Resnet101-FPN 550 dla IoU = 0,5. Źródło: opracowanie własne

Przy IoU = 0,5 również rośnie wraz numerem epoki, do której został wytrenowany model, zarówno dla ramki ograniczającej, jak i segmentacji instancji, jednak różnica jest większa niż w przypadku IoU = 0,5:0,95.



Rysunek 6.3: Wyniki ewaluacji modelu Yolact: Resnet101-FPN 550 dla IoU = 0,95. Źródło: opracowanie własne

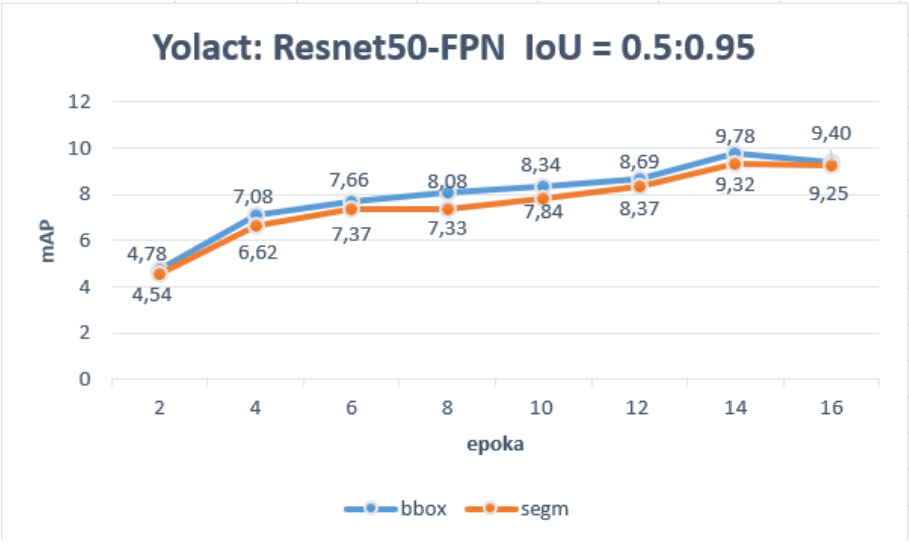
Przy IoU równym 0,95 zależność współczynnika mAP od epoki, do której został wytrenowany model, przestaje być jednoznaczna.

Poniżej, na rys. 6.4, pokazano porównanie modelu wytrenowanego do 16 epoki z modelem dostarczonym przez twórców rozwiązania wytrenowanym do 54 epoki.

Rysunek 6.4: Porównanie wyników ewaluacji modelu Yolact: Resnet101-FPN 550 na obrazie dla epoki 16 i 54. Źródło: opracowanie własne

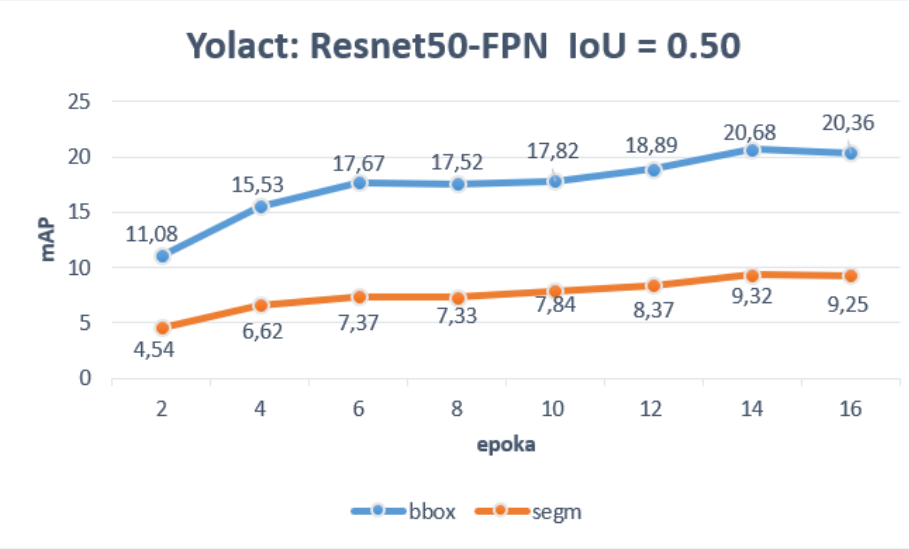
Yolact: Resnet50-FPN

Drugą uwzględnioną w badaniach konfiguracją był model Yolact ze rdzeniem Resnet50. Poniżej przedstawiono wyniki ewaluacji współczynnika mAP dla różnych wartości IoU.



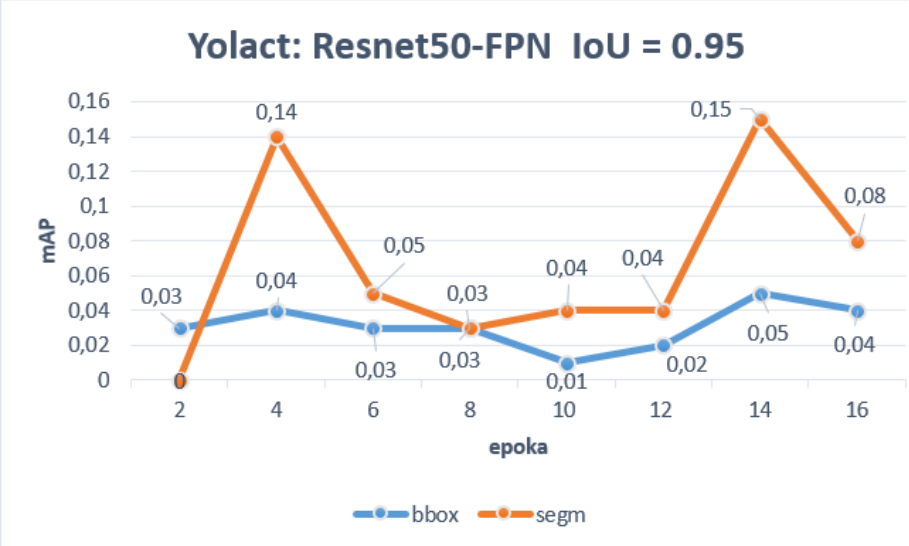
Rysunek 6.5: Wyniki ewaluacji modelu Yolact: Resnet50-FPN dla IoU = 0,5:0,95. Źródło: opracowanie własne

Zależności między mAP a numerem epoki są podobne, jak w przypadku konfiguracji z wykorzystaniem rdzenia Resnet101 opisanej poprzednio. Jednak różnica między mAP ramki ograniczającej a segmentacji jest minimalna.



Rysunek 6.6: Wyniki ewaluacji modelu Yolact: Resnet50-FPN dla IoU = 0,5. Źródło: opracowanie własne

Przy IoU = 0,5 różnica między mAP ramki ograniczającej a segmentacji drastycznie wzrasta.



Rysunek 6.7: Wyniki ewaluacji modelu Yolact: Resnet50-FPN dla IoU = 0,95. Źródło: opracowanie własne

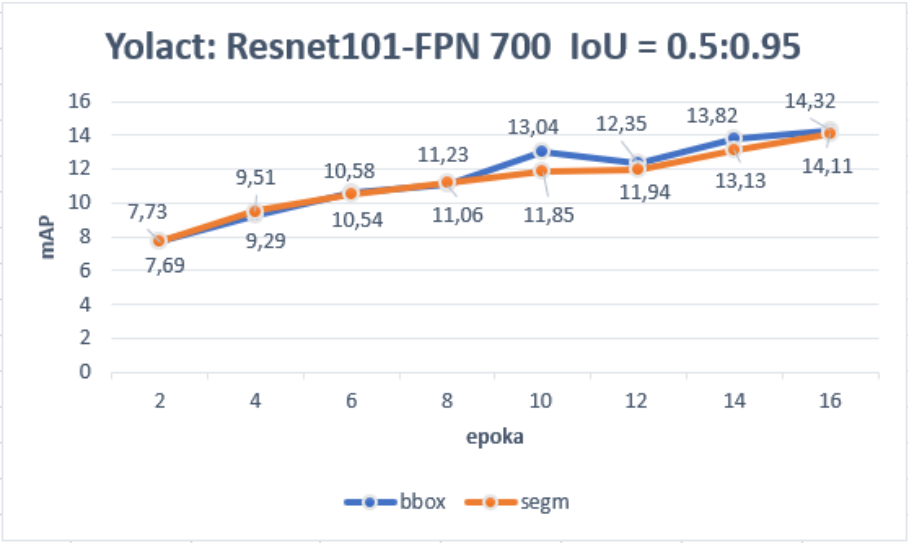
Przy IoU równym 0,95 zależność współczynnik mAP od epoki, do której został wytrenowany model przestaje być jednoznaczna również dla konfiguracji z wykorzystaniem rdzenia Resnet50.

Poniżej, na rys. 6.8, pokazano porównanie modelu wytrenowanego do 16 epoki z modelem dostarczonym przez twórców rozwiązania wytrenowanym do 54 epoki.

Rysunek 6.8: Porównanie wyników ewaluacji modelu Yolact: Resnet50-FPN na obrazie dla epoki 16 i 54. Źródło: opracowanie własne

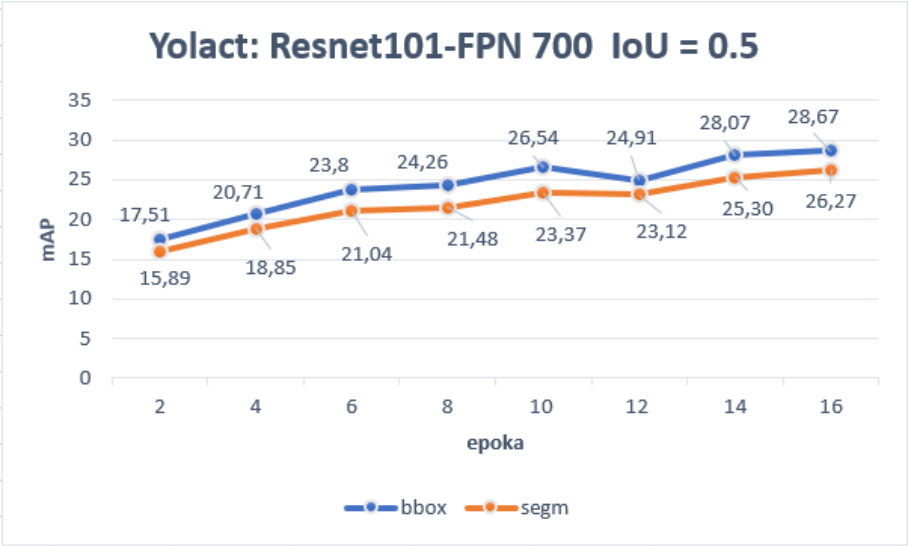
Yolact: Resnet101-FPN 700

Trzecią uwzględnioną w badaniach konfiguracją był model Yolact ze rdzeniem Resnet101-FPN. Poniżej przedstawiono wyniki ewaluacji współczynnika mAP dla różnych wartości IoU.



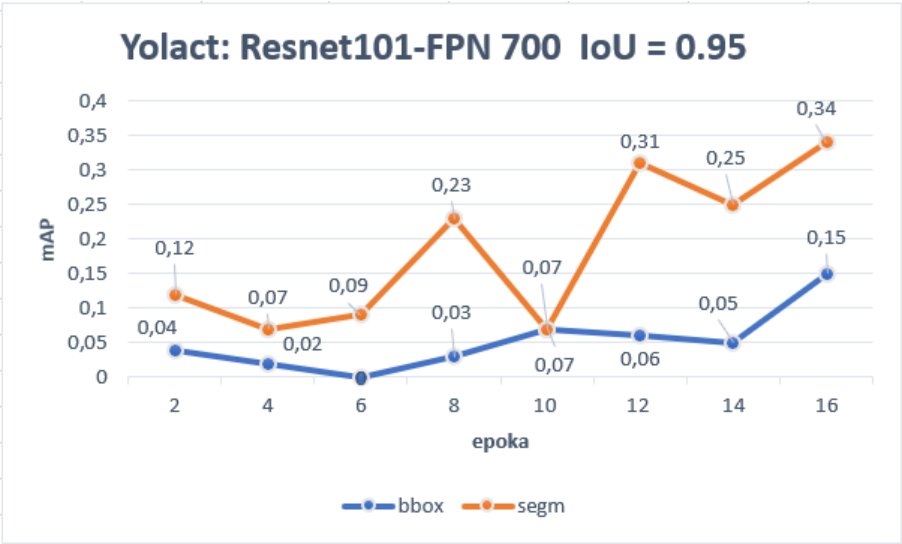
Rysunek 6.9: Wyniki ewaluacji modelu Yolact: Resnet101-FPN 700 dla IoU = 0,5:0,95. Źródło: opracowanie własne

Zależności między mAP a numerem epoki dla tej konfiguracji nie różnią się od wyników konfiguracji dla rdzenia Resnet101-FPN 550 oraz Resnet50-FPN. Różnica między mAP ramki ograniczającej a segmentacji jest minimalna.



Rysunek 6.10: Wyniki ewaluacji modelu Yolact: Resnet101-FPN 700 dla IoU = 0,5. Źródło: opracowanie własne

Przy IoU = 0,5 różnica między mAP ramki ograniczającej a segmentacji jest minimalna.



Rysunek 6.11: Wyniki ewaluacji modelu Yolact: Resnet101-FPN 700 dla IoU = 0,95. Źródło: opracowanie własne

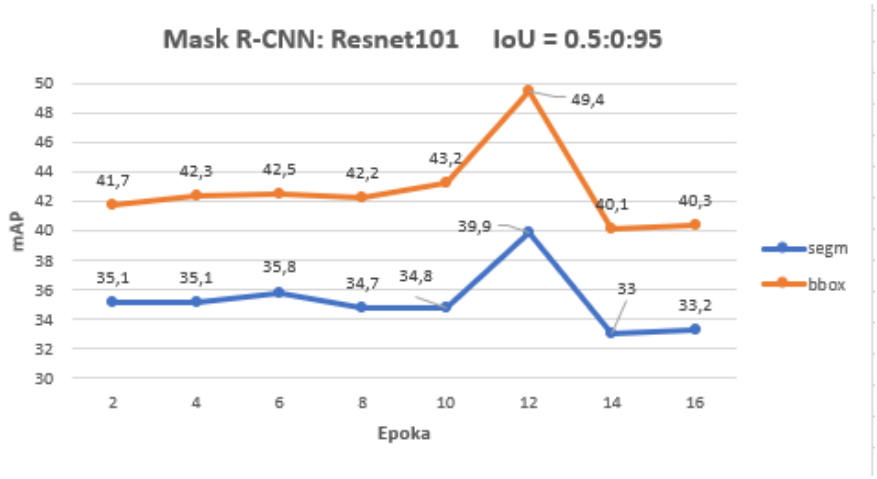
Przy IoU równym 0,95 wartości współczynnika mAP są minimalne, a jego zależność od numeru epoki, do której został wytrenowany model przestaje być jednoznaczna również dla tej konfiguracji.

Poniżej, na rys. 6.12, pokazano porównanie modelu wytrenowanego do 16 epoki z modelem dostarczonym przez twórców rozwiązania wytrenowanym do 54 epoki.

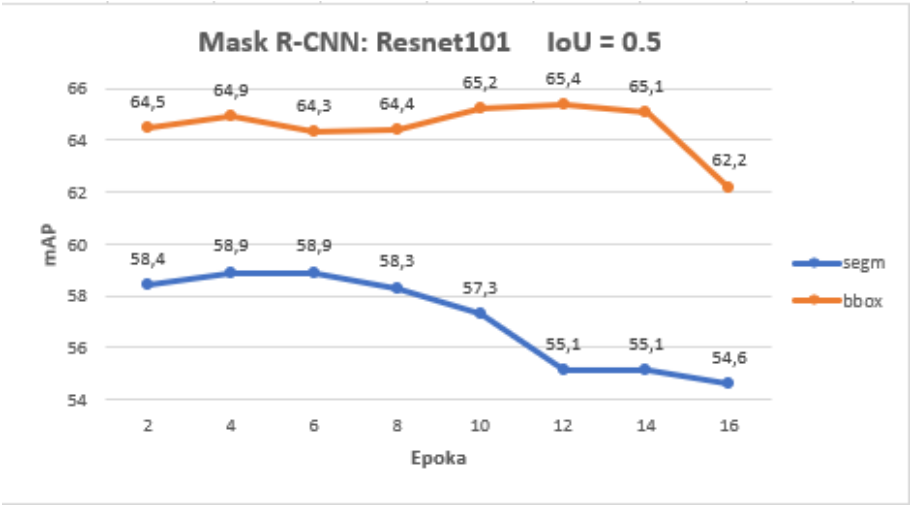
Rysunek 6.12: Porównanie wyników ewaluacji modelu Yolact: Resnet101-FPN 700 na obrazie dla epoki 16 i 54. Źródło: opracowanie własne

Mask R-CNN: Resnet101-FPN

Ostatnią uwzględnioną w badaniach konfiguracją był model Mask R-CNN ze rdzeniem Resnet101-FPN. Poniżej przedstawiono wyniki ewaluacji współczynnika mAP dla różnych wartości IoU.



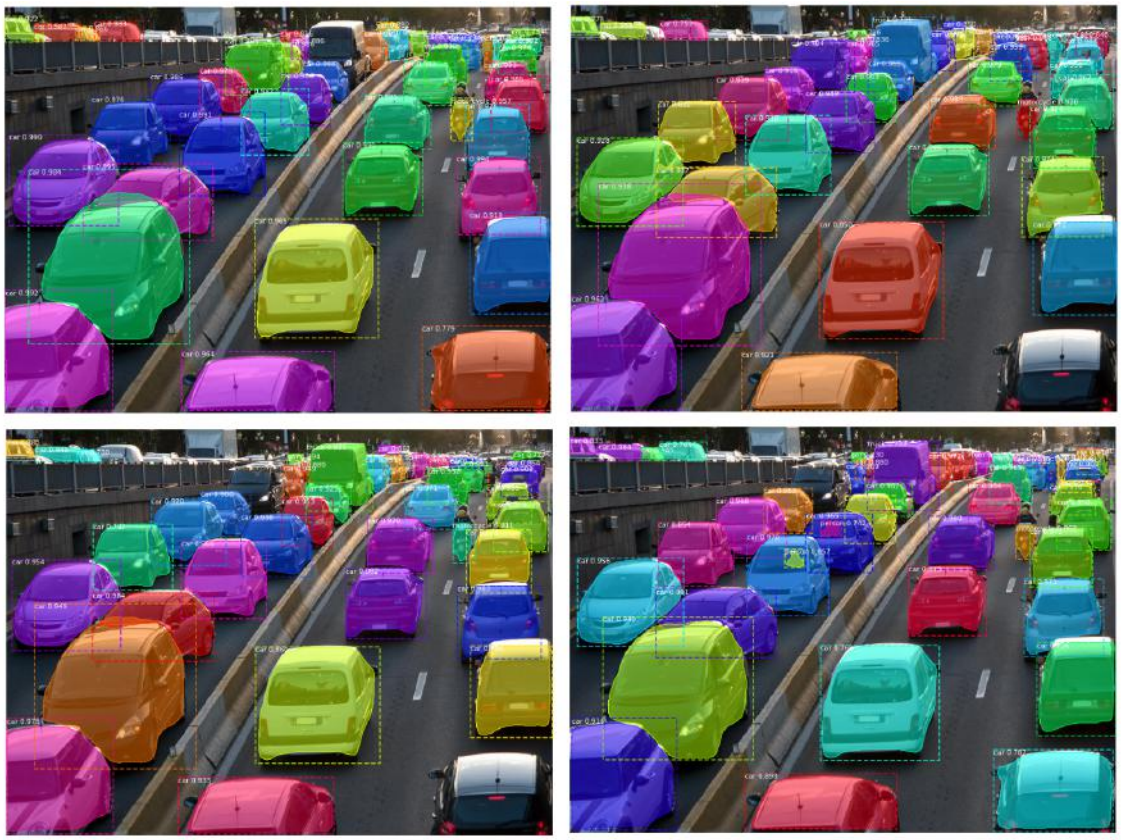
Rysunek 6.13: Wyniki ewaluacji modelu Mask R-CNN: Resnet101-FPN dla IoU = 0,5. Źródło: opracowanie własne



Rysunek 6.14: Wyniki ewaluacji modelu Mask R-CNN: Resnet101-FPN dla IoU = 0,5. Źródło: opracowanie własne

Przeprowadzona ewaluacja wytrenowanego modelu Mask R-CNN pozwoliła na uzyskanie wyników parametru mAP dla IoU = 0,5:0,95 oraz IoU = 0,5. W przypadku modelu Mask R-CNN wartość mAP nie zmienia się znacząco w zależności od numeru epoki, do której został wytrenowany model, a nawet częściowo spada przy ostatnich wytrenowanych epokach, zarówno dla ramki ograniczającej, jak i segmentacji.

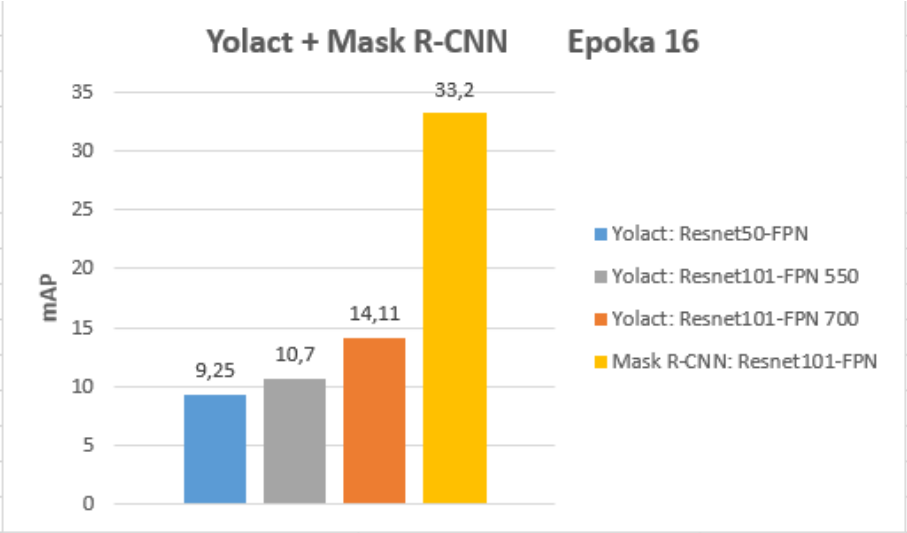
Poniżej, na rys. 6.15, pokazano porównanie wyników ewaluacji na obrazie konfiguracji Mask R-CNN: Resnet101-FPN dla modeli wytrenowanych do epoki drugiej, szóstej, dziesiątej i czternastej.



Rysunek 6.15: Wyniki ewaluacji modelu Mask R-CNN: Resnet101-FPN na obrazie dla epoki drugiej, szóstej, dziesiątej i czternastej. Źródło: opracowanie własne

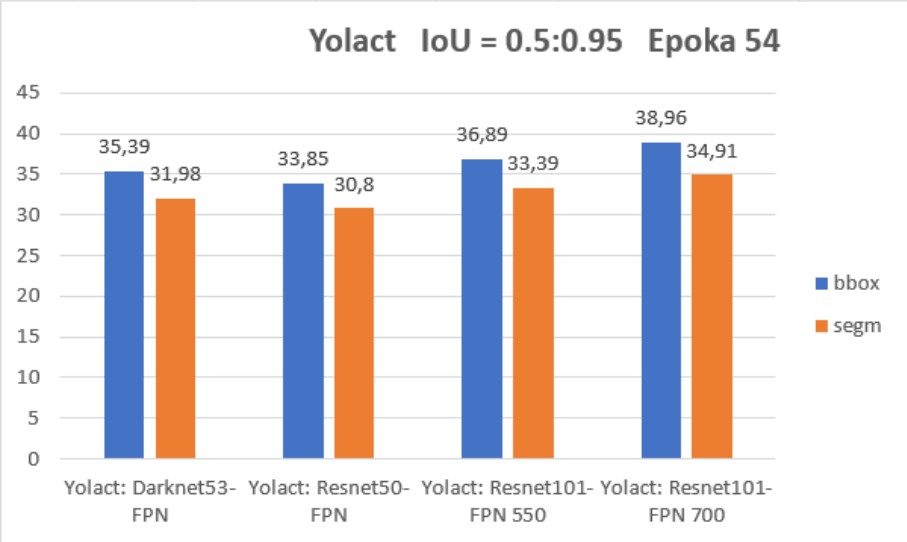
Wyniki ewaluacji modelu Mask R-CNN na obrazie wskazują, iż już od drugiej epoki model działa z dużą precyzją będąc w stanie wykryć większość obiektów i dokładnie dopasować maski.

Współczynnik mAP - podsumowanie



Rysunek 6.16: Wyniki ewaluacji modeli Mask R-CNN i Yolact: Resnet101-FPN 550, Resnet101-FPN 700 i Resnet50 dla IoU = 0,5:0,95 dla 16 epoki. Źródło: opracowanie własne

W zestawieniu wytrenowanych modeli Yolact oraz Mask R-CNN niekwestionowanym zwycięzcą pod względem precyzji segmentacji jest model Mask R-CNN: Resnet101, który osiągnął wartość mAP ponad dwa razy wyższą od najlepszego wyniku modelu Yolact - Resnet101-FPN 700.

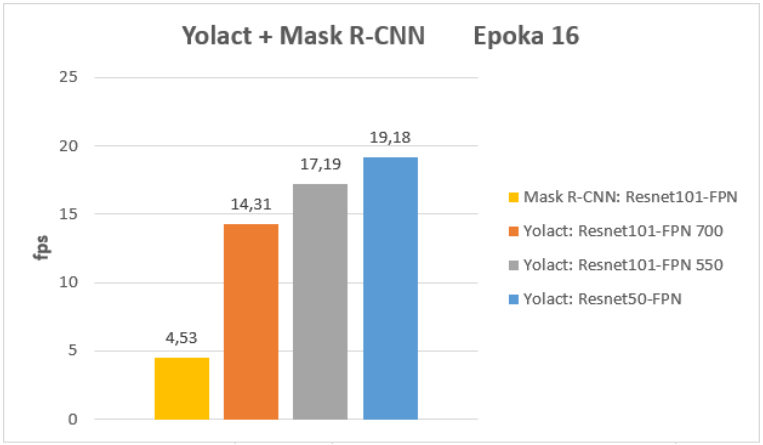


Rysunek 6.17: Wyniki ewaluacji parametru mAP dla IoU=[0,5:0,95] modeli Yolact wytrenowanych do 54 epoki. Źródło: opracowanie własne

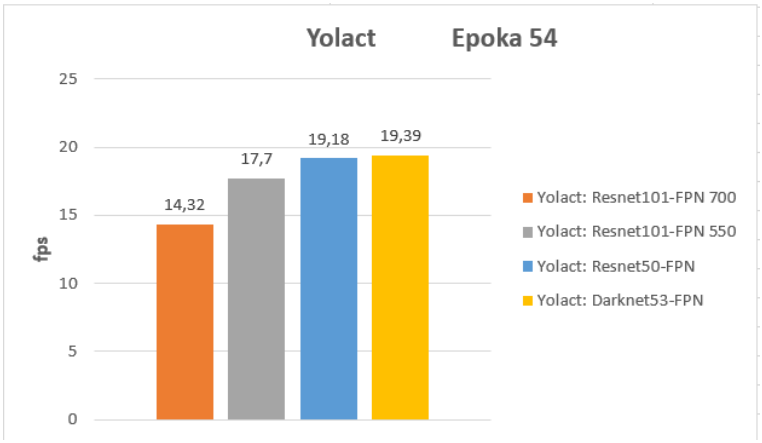
Modele Yolact dla poszczególnych konfiguracji wytrenowane do 54 epoki osiągają wartość współczynnika mAP porównywalną do wytrenowanego do 16 epoki modelu Mask R-CNN.

6.2.2 Szybkość działania sieci

Poniżej zostały przedstawione wyniki badań szybkości działania sieci (w kl./s) dla poszczególnych konfiguracji. Badania zostały przeprowadzone na zbiorze tysiąca obrazów ze zbioru walidacyjnego. Do ich przeprowadzenia wykorzystano kartę graficzną Nvidia GeForce 2080Ti.



Rysunek 6.18: Wyniki ewaluacji szybkości działania sieci Mask R-CNN: Resnet101-FPN i Yolact: Resnet101-FPN 550, Resnet50-FPN, Resnet101-FPN 700 wytrenowanych do 16 epoki. Źródło: opracowanie własne



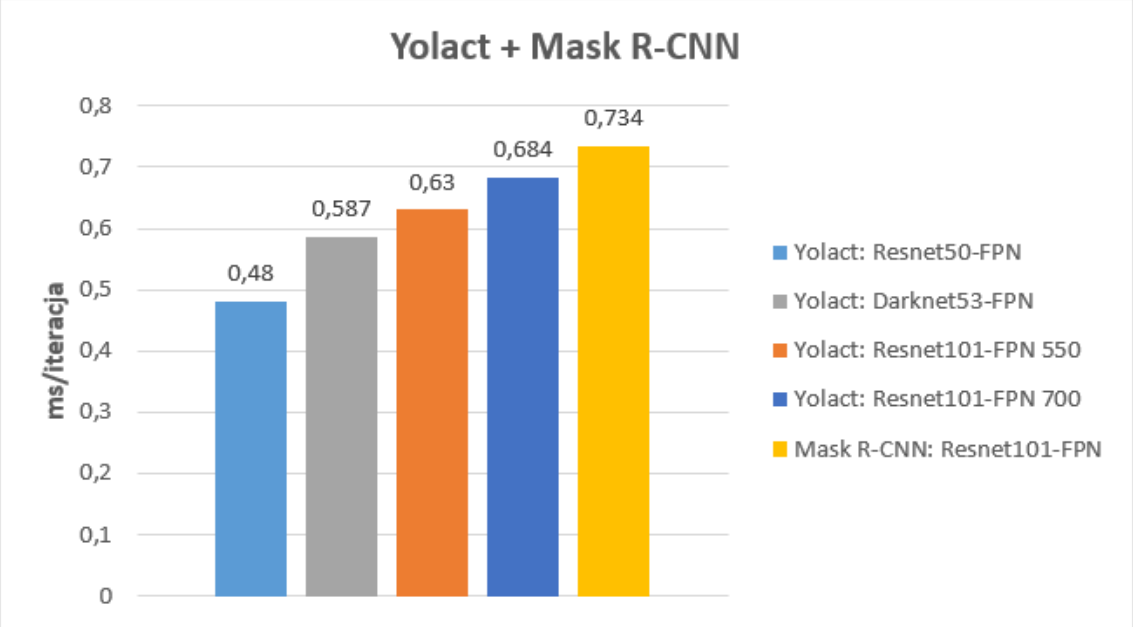
Rysunek 6.19: Wyniki ewaluacji szybkości działania sieci modelu Yolact: Resnet101-FPN 700, Resnet101-FPN 550, Resnet50-FPN, Darknet53-FPN wytrenowanych do 54 epoki. Źródło: opracowanie własne

Powyższe zestawienie wskazuje, że najlepszym wyborem pod względem szybkości działania sieci jest model Yolact wykorzystujący rdzeń Resnet50. Wyniki ewaluacji modeli wytrenowanych przez autora zgadzają się z wynikami ewaluacji modeli dostarczonych przez twórców rozwiązań. W zestawieniu wyników tych modeli została uwzględniona konfiguracja darknet, jednak nie udało się samodzielnie wytrenować modelu dla tej konfiguracji.

Żadna konfiguracja nie osiągnęła wyniku pozwalającego na użycie jej w zastosowaniach wymagających działania w czasie rzeczywistym - powyżej 30 kl./s.

6.2.3 Czas uczenia

Poniżej zostały przedstawione wyniki badań czasu uczenia dla poszczególnych konfiguracji Yolact oraz Mask R-CNN.



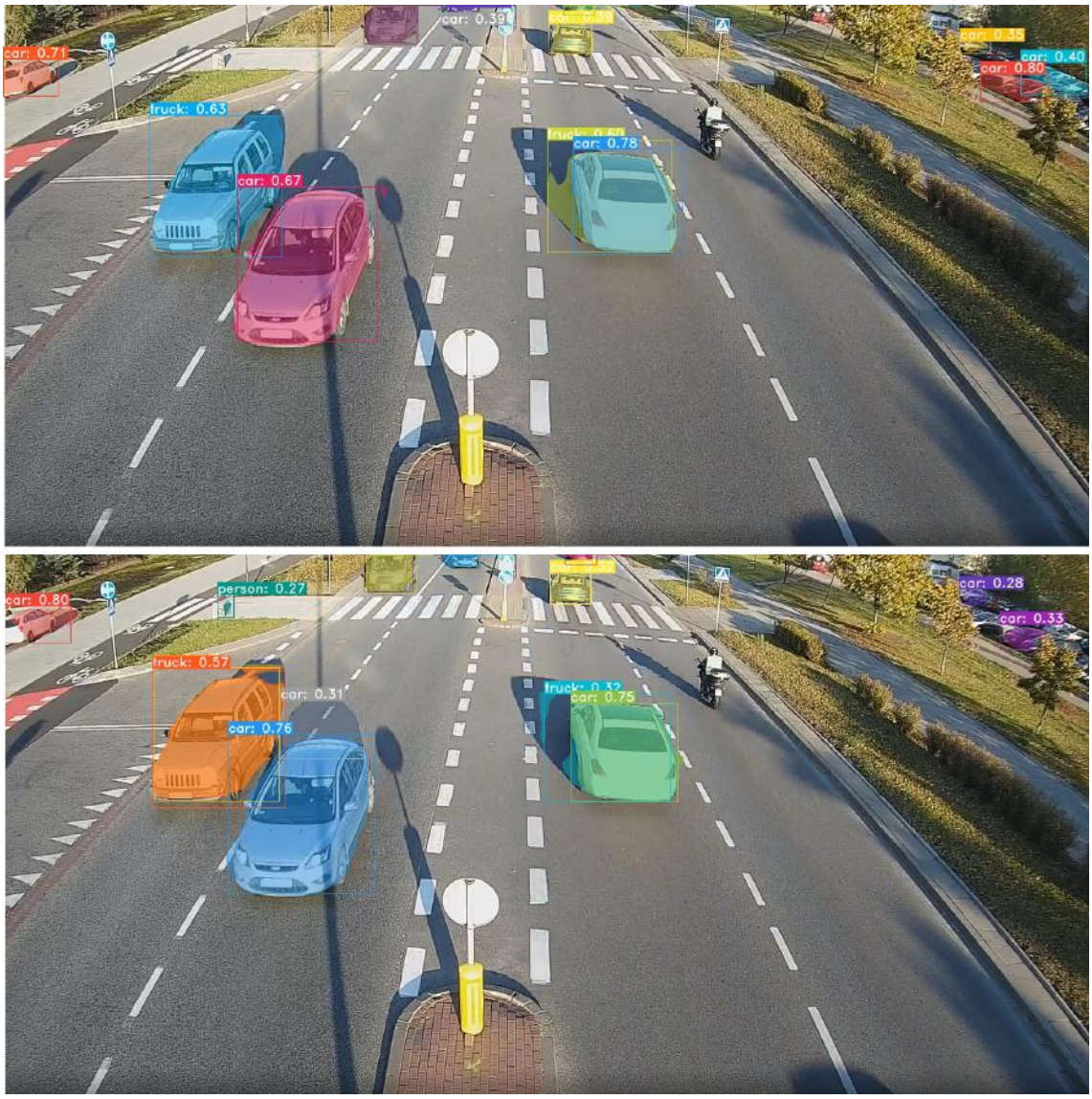
Rysunek 6.20: Wyniki ewaluacji czasu uczenia dla różnych konfiguracji modelu Yolact i Mask R-CNN. Źródło: opracowanie własne

Czas uczenia jest najwyższy dla Mask R-CNN. Przyczyną tego rezultatu jest fakt, iż rozmiar obrazów zbioru uczącego Mask R-CNN jest dużo większy niż dla wszystkich konfiguracji Yolact, co zostało przedstawione w opisie konfiguracji wykorzystanych w zestawieniu. Dla różnych konfiguracji Yolact, najwyższy czas uczenia osiąga Resnet101-FPN 700, natomiast najniższy Resnet50-FPN. Resnet101 jest architekturą bardziej rozbudowaną niż Resnet50, dlatego czas uczenia modelu opartego na tym rdzeniu okazał się wyższy.

6.2.4 Porównanie wizualne efektów działania wybranych sieci

Efekty działania sieci są zróżnicowane również pod względem perspektywy, jakości zdjęcia, czy występowania cieni obiektów na obrazie. W tej sekcji zostaną przedstawione wyniki działania wybranych sieci (dla modeli wytrenowanych do 54 epoki) na obrazach prezentujących sceny uliczne zróżnicowane pod względem wyżej wymienionych czynników wraz z oceną ich wpływu na jakość działania sieci.

Wpływ występowania cieni obiektów



Rysunek 6.21: Wynik ewaluacji modelu Yolact: Resnet50-FPN i Darknet53-FPN na obrazie zawierającym cienie obiektów. Źródło: opracowanie własne

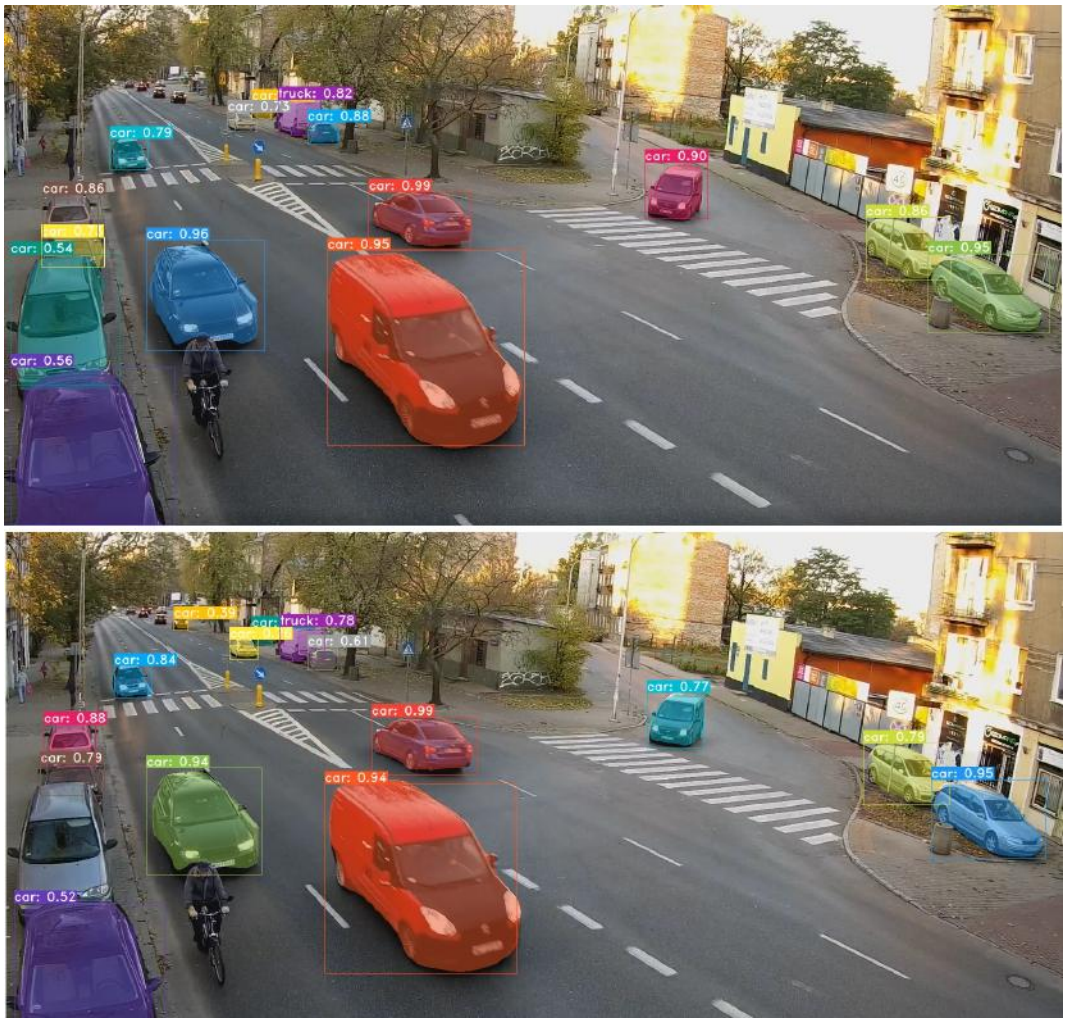
Jak można zauważyć na rys. 6.21, występowanie cieni obiektów ma istotny wpływ na działanie sieci. Kwalifikują one cienie jako część danego obiektu lub inny obiekt, często należący nawet do innej klasy.

Wpływ jakości obrazu sceny

Rysunek 6.22: Wynik ewaluacji modelu Yolact: Darknet53-FPN i Resnet50-FPN na obrazie słabej jakości. Źródło: opracowanie własne

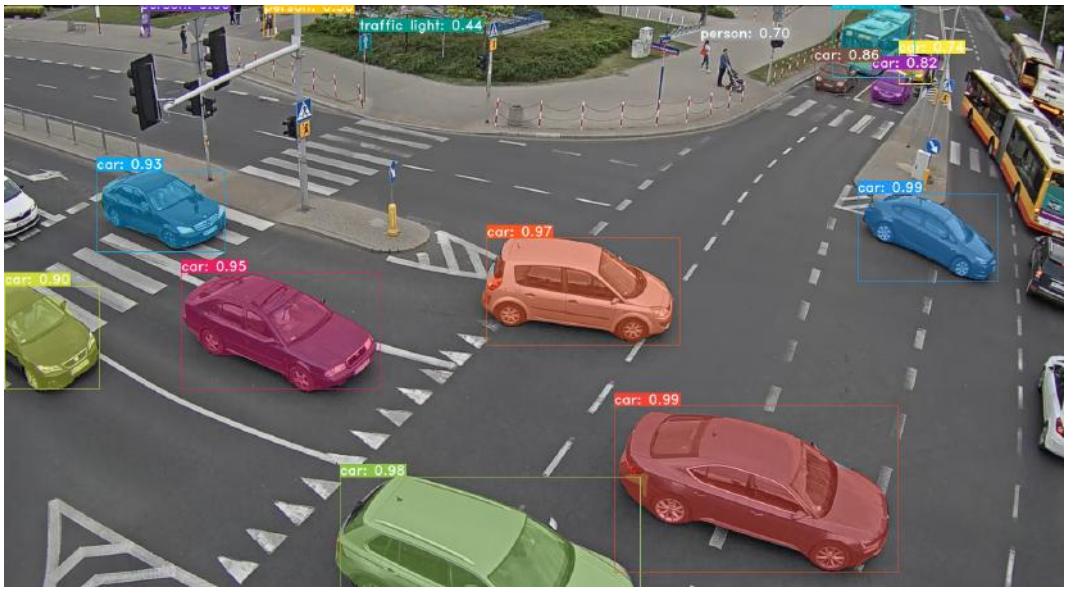
Jakość obrazu podawanego na wejście sieci ma również niebagatelny wpływ na jej efektywność. Jak zostało pokazane na rys. 6.22, przy słabej jakości obrazu sceny wagi przypisane obiektom występującym na obrazie są niskie, sieci nie rozpoznają wielu obiektów sceny oraz wykrywają nieistniejące obiekty.

Wpływ perspektywy sceny



Rysunek 6.23: Wynik ewaluacji modelu Yolact: Resnet101-FPN 700 na obrazie dla ukośnej perspektywy sceny. Źródło: opracowanie własne

Dla pokazanej na rys. 6.23 perspektywy rezultaty osiągane przez modele Yolact: Resnet101-FPN 700 oraz Yolact: Resnet101-FPN 550 są zadowalające. Większość obiektów występujących na obrazie jest sklasyfikowana poprawnie. Występujący na pierwszym planie rowerzysta nie został wykryty i sklasyfikowany pomimo istnienia dużej liczby obiektów tego typu w zbiorze uczącym COCO Dataset.



Rysunek 6.24: Wynik ewaluacji modelu Yolact: Resnet50-FPN na obrazie dla górno-ukośnej perspektywy sceny. Źródło: opracowanie własne

Rysunek 6.25: Wynik ewaluacji modelu Yolact: Resnet101-FPN 550 na obrazie dla górno-ukośnej perspektywy sceny. Źródło: opracowanie własne

Pomimo trudnej perspektywy, wynik ewaluacji sieci można ocenić jako bardzo dobry. Jak zostało pokazane na rys. 6.24 oraz 6.25, przypisane obiektom wagi są wysokie, większość z występujących na obrazie obiektów została wykryta i poprawnie rozpoznana. W przeciwieństwie do rowerzysty przedstawionego na 6.23, rowerzysta znajdujący się na pasach został poprawnie wykryty oraz sklasyfikowany, pomimo występowania większej odległości między obiektem a kamerą oraz mniejszej powierzchni zajmowanej przez obiekt na obrazie.

Poniżej przedstawiono wyniki ewaluacji modeli na obrazach zróżnicowanych pod względem prezentowanej sceny.

Rysunek 6.26: Wynik ewaluacji modelu Yolact: Resnet101-FPN 550 na obrazie. Źródło: opracowanie własne na podstawie [9]

Prezentowana na rys. 6.26 scena zawiera wiele obiektów, jednak są one zbyt małe, aby mogły być poprawnie wykryte oraz sklasyfikowane.

Rysunek 6.27: Wynik ewaluacji modelu Yolact: Resnet101-FPN 700 na obrazie. Źródło: opracowanie własne na podstawie [12]

Pomimo frontalnej perspektywy obrazu przedstawionego na rys. 6.27, wagi zostały przypisane tylko części znajdujących się na nim obiektów. Przyczyną tego rezultatu jest najprawdopodobniej duża liczba elementów zlokalizowanych na obrazie w bliskiej odległości.

Rysunek 6.28: Wynik ewaluacji modelu Yolact: Darknet53-FPN na obrazie. Źródło: opracowanie własne

Przedstawiony na rys. 6.28 przykład sceny dynamicznej uwypukla problem z wykrywalnością obiektów, w szczególności osób, i poprawnym ich sklasyfikowaniem, kiedy przyjmują one niecodzienne pozy.

Rysunek 6.29: Wynik ewaluacji modelu Yolact: Darknet53-FPN na obrazie. Źródło: opracowanie własne na podstawie [10]

Wagi przepisane obiektom sceny przedstawionej na rys. 6.29 są wysokie. Obiekty są wykryte i sklasyfikowane poprawnie. Stworzone ramki graniczne oraz maski dopasowują się do obszaru zajmowanego przez obiekty. Najprawdopodobniej wynika to z faktu, iż sieć była trenowana na zbiorze uczącym zawierającym wiele obrazów przedstawiających tę klasę.

Rozdział 7

Podsumowanie i wnioski

W niniejszej pracy zostało przeprowadzone porównanie dwóch metod segmentacji instancji - Mask R-CNN oraz Yolact, kierując się takimi parametrami jak współczynnik mAP, szybkość działania oraz czas wytrenowania modeli. Omówione zostały również zagadnienia sieci konwolucyjnych, detekcji obiektów, segmentacji semantycznej, metryk stworzonych do ewaluacji badań oraz inne wykorzystywane metody segmentacji instancji. Ponadto został przedstawiony proces konfiguracji porównywanych metod wraz z rozwiązaniami napotkanych problemów i wskazówkami przy rozwiązywaniu błędów możliwych do napotkania.

Głównym założeniem pracy było przeprowadzenie porównania wybranych metod segmentacji instancji w obrazach cyfrowych z wykorzystaniem sieci głębokiego uczenia, pozwalające na wybranie optymalnego rozwiązania problemu rozpoznawania obiektów sceny pod względem dokładności, szybkości działania oraz zasobów potrzebnych do implementacji danej metody. Cel ten został zrealizowany. Niemniej jednak, analiza ta mogłaby być zrealizowana szerzej - z uwzględnieniem większej liczby metod oraz czynników, pozwalających na lepsze zgłebienie tematu i dających szerszy ogląd na wybór metody segmentacji instancji do implementacji w konkretnym rozwiązaniu. Bardziej szczegółowy opis prac, które mogłyby zostać wykonane, aby uczynić pracę bardziej kompleksową, został zawarty w sekcji 7.1.

Główne problemy w realizacji niniejszej pracy dotyczyły rozmiaru pamięci i wydajności karty graficznej potrzebnej do realizacji celu niniejszej pracy. Problem został częściowo rozwiązany przy pomocy sprzętu zapewnionego przez uczelnię, który pozwolił na wytrenowanie modeli wykorzystanych do przeprowadzonego badania w racjonalnym czasie. Należy jednak zwrócić uwagę na bardzo istotny fakt, iż problem ten wymaga specjalistycznych i, tym samym, kosztownych zasobów sprzętowych, które mogą być często nieosiągalne dla przeciętnego użytkownika, chcącego wykorzystać te metody w swojej implementacji.

Wyniki przeprowadzonych badań wskazują na wyższość Mask R-CNN w kontekście precyzji segmentacji, jednak charakteryzuje się ona również najwyższym czasem uczenia. Biorąc pod uwagę szybkość działania sieci najlepszy wynik uzyskał model Yolact z rdzeniem Resnet50, jednak zajmuje on ostatnie miejsce w zestawieniu parametru mAP dla poszczególnych konfiguracji. Porównując wyniki wszystkich parametrów nie można wskazać najlepszej konfiguracji, ponieważ przy lepszych wynikach precyzji dla danej konfiguracji, posiada ona wyższy czas uczenia oraz gorsze wyniki szybkości przetwarzania sieci.

7.1 Dalsze prace

Niniejszą pracę dyplomową można byłoby wzbogacić o porównanie większej liczby metod segmentacji instancji - o omówioną w rozdziale trzecim metodę FCIS (Fully Convolutional Instance-aware Semantic Segmentation) czy opublikowaną w ostatnim czasie (03.12.2019) metodę Yolact++ rozszerzającą opisaną w niniejszej pracy metodę Yolact. Ponadto trenowanie modeli wykorzystanych do porównania mogłoby zostanie przeprowadzone przez większą liczbę epok, możnaby zróżnicować modele pod względem rozmiarów obrazów zbioru uczącego oraz ewaluacyjnego czy wykorzystać inne zbiory danych uczących (innych niż użyty COCO Dataset) oraz stworzyć własne zbiory do uczenia i ewaluacji. Oprócz tego można byłoby przeprowadzić porównanie dla sprzętu cechującego się różnymi parametrami, różnej pamięci kart graficznych i innej ich wydajności.

Bibliografia

[1] Bolya, D., et al. YOLACT: real-time instance segmentation. *CoRR abs/1904.02689* (2019).

[2] Chen, X., et al. Tensormask: A foundation for dense object segmentation. *CoRR abs/1903.12174* (2019).

[3] Girshick, R. B. Fast R-CNN. *CoRR abs/1504.08083* (2015).

[4] He, K., et al. Mask R-CNN. *CoRR abs/1703.06870* (2017).

[5] Li, Y., et al. Fully convolutional instance-aware semantic segmentation. *CoRR abs/1611.07709* (2016).

[6] Pinto, J. P. O. Masknet: An instance segmentation algorithm - leveraging object detection and semantic segmentation to tackle instance segmentation (2017).

[7] Zhang, Y.-J. *An Overview of Image and Video Segmentation in the Last 40 Years*. 2006.

[8] Chandel, V. S., Learnopencv: Selective Search for Object Detection. data dostępu: 02.01.2020. `https://www.learnopencv.com/selective-search-for-object-detection-cpp-python/`.

[9] Dreamstime: Shibuya Crossing. data dostępu: 30.01.2020. `https://pl.dreamstime.com/`.

[10] Granumfn: Cows on pasture. data dostępu: 30.01.2020. `https://www.granumfn.pl/`.

[11] Hui, J., mAP (mean Average Precision) for Object Detection. data dostępu: 10.01.2020. `https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173`.

[12] Learningenglish: Ellen Degeneres Oskar Photo. data dostępu: 30.01.2020. `https://learningenglish.voanews.com/a/ellen-degeneres-history-selfie-oscars-china/1859238.html`.

[13] Pinto, J., Masknet: An Instance Segmentation Algorithm. `http://publications.lib.chalmers.se/records/fulltext/250417/250417.pdf`.

[14] Robert's cross. data dostępu: 02.01.2020. `https://en.wikipedia.org/wiki/Roberts_cross`.

[15] Sharma, P. , Implementing Mask R-CNN for Image Segmentation. data dostępu: 08.01.2020. https://www.analyticsvidhya.com/blog/2019/07/computer-vision-implementing-mask-r-cnn-image-segmentation/.

[16] Singh, R., Towards Datascience: Computer Vision Introduction. data dostępu: 30.12.2019. https://towardsdatascience.com/computer-vision-an-introduction-bbc81743a2f7.

[17] Tran, G. S.,Faster R-CNN. data dostępu: 05.01.2020. https://www.researchgate.net/figure/Faster-R-CNN-Architecture-9_fig1_324549019.

[18] Wikipedia: Computer Vision History. data dostępu: 02.01.2020. https://en.wikipedia.org/wiki/Computer_vision.

[19] Wikipedia: Convolutional Neural Network. data dostępu: 02.01.2020. https://en.wikipedia.org/wiki/Convolutional_neural_network.

[20] Wikipedia: Tablica pomyłek. data dostępu: 28.01.2020. https://pl.wikipedia.org/wiki/Tablica_pomy%C5%82ek.

Wykaz symboli i skrótów

AP	Average Precision
AR	Average Recall
AUC	Area Under Curve
bbox	bounding box
CNN	Convolutional Neural Network
CPU	Central Processing Unit
$CUDA$	Compute Unified Device Architecture
$CuDNN$	CUDA Deep Neural Network
$FCIS$	Fully Convolutional Instance-aware Semantic Segmentation
FCN	Fully Convolutional Network
FPN	Feature Pyramid Network
fps	frame per second
GPU	Graphical Processing Unit
IoU	Intersection over Union
mAP	mean Average Precision
$R-CNN$	Region-based Convolutional Neural Network
$ReLU$	Rectified Linear Unit Layers
RoI	Region of Interest
RPN	Region Proposal Network
SSD	Single Shot Detector
VOC	Virtual Object Classes
VPN	Virtual Private Network
$YOLACT$	You Only Look At Coefficients
$YOLO$	You Only Look Once

Spis rysunków