Laboratorium 7: Detekcja obiektów za pomocą sieci neuronowych

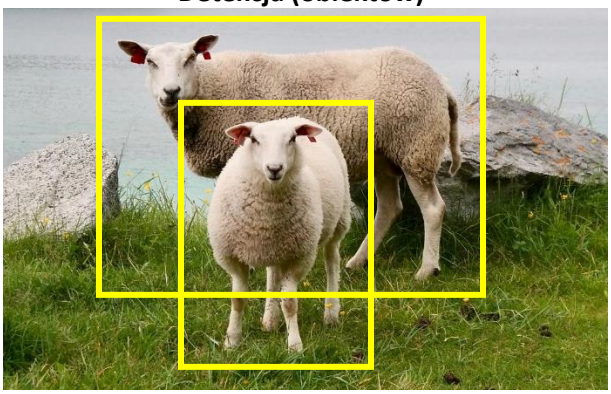
Zarys zadania

W literaturze znaleźć można wiele różnych typów zadań analizy obrazów. Cztery z najczęściej spotykanych przedstawiono na rysunku poniżej. Ta klasyfikacja przyjęła się wraz z upowszechnieniem zastosowania technik uczenia głębokiego w wizji komputerowej, dlatego będziemy również ją stosować.

Klasyfikacja

„ten obraz przedstawia owce”

Detekcja (obiektów)



„tutaj na obrazie znajdują się owce”

Segmentacja semantyczna

„te piksele należą do klasy *owca*”

Segmentacja instancyjna

„te piksele należą do *jednej* owcy, a te do *drugiej*”

W niniejszej instrukcji zajmiemy się tematem *detekcji obiektów*, ponieważ zrozumienie algorytmów detekcji jest pomocne w zrozumieniu innych, bardziej złożonych zadań; ponadto jest to problem występujący bardzo często w praktyce. Nieraz zastosowanie detekcji bywa wystarczające od strony aplikacyjnej (tj. nie potrzeba rozszerzać systemu o bardziej złożone komponenty wykonujące np. segmentację), a do tego systemy realizujące tylko detekcję są z zasady dużo szybsze oraz łatwiejsze do nauczenia (dlaczego?), niż porównywalne podejścia wykonujące bardziej złożone predykcje.

Ujmując rzecz bardziej formalnie, detekcja jest zadaniem przewidzenia pewnej (nieznanej z góry) liczby prostokątów otaczających rejony obrazu zawierające interesujące obiekty. Z reguły uznaje się zadanie

zaklasyfikowania tych rejonów za integralną część problemu, stąd nowoczesne algorytmy do wykrytych koordynat dodają także wektor prawdopodobieństw przynależności do znanych klas.

Historycznie detektory obiektów oparte były o różnego rodzaju ekstraktory cech (np. detektor Viola-Jones wykorzystujący falki Haara, nb. stosowany do dziś w warunkach bardzo ograniczonej mocy obliczeniowej) oraz tzw. okno przesuwne (*sliding window*). Najprostszym podejściem jest bowiem sekwencyjne sprawdzanie wszystkich możliwych lokalizacji na obrazie za pomocą algorytmu klasyfikującego binarnie: tło-vs-obiekt. Widzisz już pewnie, że działanie takich metod mogło pozostawiać dużo do życzenia, zwłaszcza w kwestii szybkości.

Dzisiejsze podejścia również w pewien sposób opierają się o koncepcję okna przesuwnego, wykorzystując jednak sieci neuronowe do wykonania predykcji dla całego obszaru obrazu naraz – przypomnij sobie koncepcję FCN (*Fully Convolutional Network*). Nowoczesne metody detekcji można w ogólności podzielić na dwie klasy:

- detektory dwuetapowe (*two-stage detectors*), które w pierwszym etapie przygotowują pewien zbiór „propozycji" regionów zainteresowań, a w drugim oceniają i korygują zaproponowane rejony; do tej klasy przypadają metody typu R-CNN (*Region-based CNN*), m.in. Faster R-CNN czy Mask R-CNN;
- detektory jednoetapowe (*one-stage detectors*), wykonujące całą predykcję w jednym przejściu; do tej grupy przypada SSD, większość rodziny YOLO, RetinaNet, FCOS…

Detektory jednoetapowe są z reguły znacznie szybsze, więc stosuje się je wszędzie tam gdzie liczy się detekcja w czasie rzeczywistym. Modele dwuetapowe bazujące R-CNN dają większą swobodę konfiguracji, a także łatwiej je rozszerzać o dodatkowe komponenty, np. wykonujące predykcję maski, uzyskując system do segmentacji semantycznej (Mask R-CNN).

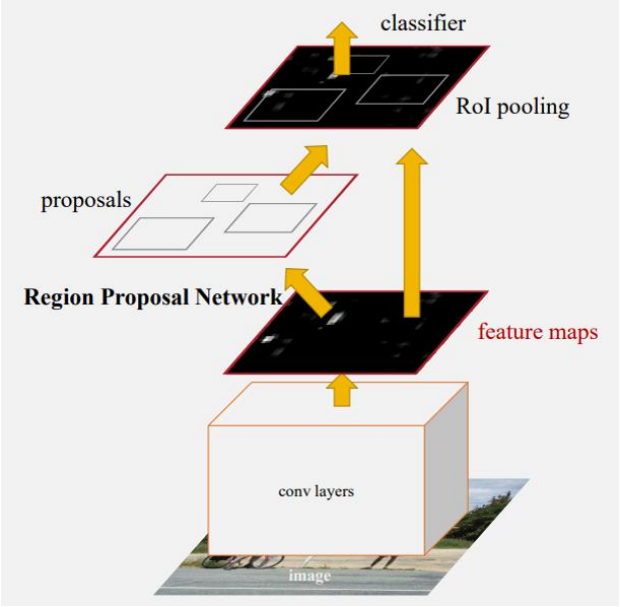
W tej instrukcji podejmiemy model z grupy dwuetapowych, konkretnie Faster R-CNN [1].

Faster R-CNN

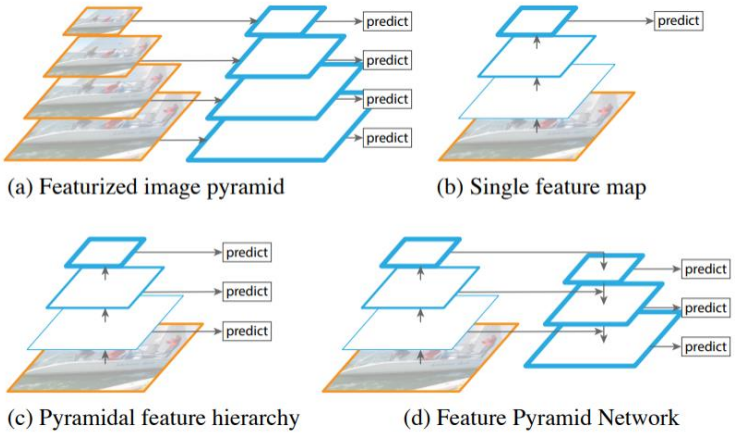
Architektura Faster R-CNN składa się z trzech głównych komponentów:

- *backbone* – sieć wykonująca ekstrakcję cech z obrazu źródłowego,
- *Region Proposal Network* – bardzo lekka sieć zwracająca propozycje rejonów dla całej powierzchni obrazu (w stylu *sliding window*),
- *RoIHead* – głowica wykonująca dokładne predykcje dla zaproponowanych rejonów.

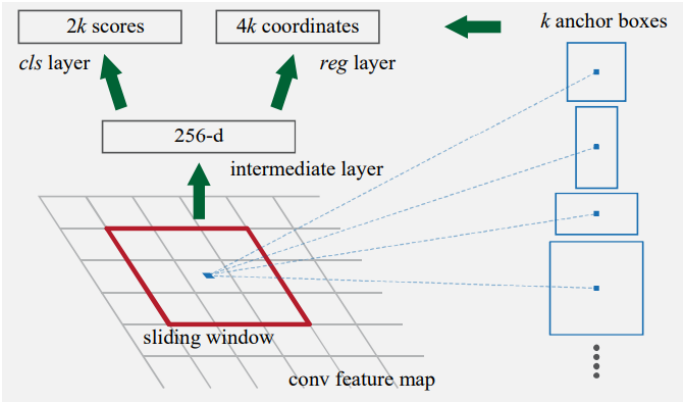
Poglądowy schemat architektury znajduje się na rysunku obok (źródło [1]).



Backbone może być dowolnym modelem neuronowym, najczęściej stosuje się technikę *transfer learning* czyli dobiera sieć wstępnie nauczoną na zadaniu klasyfikacji (ze względu na łatwość pozyskania dużej liczby zaetykietowanych danych w tym zadaniu). Aby umożliwić wykonywanie predykcji przy różnych skalach obserwowanych obiektów, stosuje się różne meta-architektury łączenia cech z poszczególnych warstw sieci, najczęściej FPN (*Feature Pyramid Network* – patrz obraz poniżej; źródło [2]). Idea FPN wynikła z problemów klasycznych detektorów (działających najczęściej w układzie *featurized image pyramid*, co było bardzo wolne, a w dobie sieci konwolucyjnych i transformerów zupełnie nieakceptowalne) i polega na takim połączeniu cech wyekstrahowanych przez pewną liczbę ostatnich warstw sieci backbone, by przy jak najmniejszym koszcie obliczeniowym umożliwić jak najlepszą zdolność predykcyjną przy wielu skalach. Jeśli ciekawią Cię szczegóły, znajdziesz je w pracy [2].



Tak wyekstrahowane i zagregowane cechy trafiają w modelu Faster R-CNN do komponentu RPN (*Region Proposal Network*). Jest to strukturalnie prosta sieć neuronowa, która na zasadzie okna przesuwnego (realizowanego oczywiście za pomocą konwolucji) estymuje koordynaty *bounding boksów* w każdym punkcie obrazu, a także pewną wartość, tzw. *objectness score*, interpretowaną jako prawdopodobieństwo, że w danym rejonie znajduje się obiekt należący do jednej ze znanych klas.



Istotnym szczegółem działania RPN jest zastosowanie tzw. *anchors* (i to nie jest rzecz unikalna dla R-CNN – YOLOv2 również opiera się o *anchors*). Naiwne podejście do RPN, polegające na proponowaniu 5 wartości (4 koordynaty + *objectness*) per piksel byłoby ograniczone do przewidywania tylko jednego obiektu w danym rejonie; gdyby obraz zawierał dwa nachodzące na siebie obiekty (np. osoba stojąca na tle swojego samochodu) tylko jeden z nich byłby możliwy do wykrycia. *Anchors* pozwalają na obejście tego ograniczenia. Koncepcja polega na przyjęciu pewnej liczby (k) prostokątów o założonych wymiarach i proporcjach; oryginalnie Faster R-CNN przyjmuje 9 prostokątów (wszystkie kombinacje wymiarów 128, 256, 512). RPN wykonuje zatem predykcje dla 9 predefiniowanych prostokątów ($2 \cdot 9 = 18$ liczb dla *objectness*, $4 \cdot 9 = 36$ liczb dla koordynat *bounding boksów*: X, Y, H, W). Tak wyznaczone koordynaty należy interpretować jako różnice względem odpowiadających im koordynat

anchor boksów. To pozwala na wykrywanie obiektów o różnych kształtach, a zbliżonym położeniu w przestrzeni.

Wyjściem „surowego" RPN są dwa obrazy o stałych rozmiarach (w powyższym przypadku 18-kanałowy i 36-kanałowy) zawierające mapę prawdopodobieństw oraz mapę *offsetów*. Naturalnym krokiem jest progowanie po prawdopodobieństwie – otrzymujemy wtedy (dowolnej długości!) listę rejonów potencjalnie zawierających obiekty. Często zdarza się jednak tak, że w okolicach rejonu zawierającego wyraźny obiekt RPN zwróci więcej niż jedną propozycję. Standardowo w metodach bazujących na RPN stosuje się tzw. *Non-Maximum Suppresion* (NMS), czyli dodatkową filtrację propozycji, które za bardzo na siebie nachodzą (zwykle wg metryki IoU).

Ostatnim krokiem w sieci Faster R-CNN (i ogólnie metodach dwuetapowych z dedykowanymi głowicami) jest rozpatrzenie poszczególnych propozycji, tzn. *czy* faktycznie zawierają jakiś obiekt, *jaki* to obiekt, oraz *gdzie* dokładnie w obrębie rejonu się on znajduje. Aby to osiągnąć, należy pozyskać cechy z rejonu zaproponowanego przez RPN i na ich podstawie dokonać odpowiednich predykcji. Głowica (*RoIHead*) w sieci Faster R-CNN realizuje to zadanie poprzez:

1. obliczenie koordynat zaproponowanego rejonu w oparciu o jego *anchor*,
2. wycięcie tego rejonu z mapy cech wytworzonej przez backbone,
3. przeskalowanie tak uzyskanego tensora do stałego wymiaru – tzw. *RoIPooling* (stosuje się różne alternatywne metody i tricki techniczne, patrz np. *RoIAlign* [3]),
4. odpytanie niewielkiej sieci neuronowej (lub wielu) dla tego obrazu w celu uzyskania oczekiwanych odpowiedzi (klasyfikacja, regresja dokładnych offsetów, …).

Kluczowe w tym procesie jest ponowne odwołanie się do już raz przygotowanej mapy cech (ewaluacja backbone'a to bardzo ciężka operacja, chcemy wykonać ją tylko raz) i odpytywanie tylko lekkich głowic na tej podstawie.

W tym układzie bardzo łatwo jest dobudować kolejne głowice, np. do predykcji punktów zainteresowań (np. szacowanie pozy) czy maski segmentacyjnej (segmentacja instancyjna, vide Mask R-CNN [3]). Wadą podejścia jest natomiast wydajność: liczba propozycji nie jest znana do momentu wykonania RPN, a następnie każdą z nich rozpatruje się osobno (RoIPooling/RoIAlign).

Procedurę uczenia sieci klasy R-CNN w niniejszej instrukcji pominiemy. Należy wiedzieć, że zadanie detekcji obiektów jest połączeniem typowej klasyfikacji (*objectness*, przynależność klasowa) z *regresją* (koordynaty prostokątów otaczających obiekty). Warto też zapoznać się z funkcjami kosztu stosowanymi w modelu Faster R-CNN, oraz których komponentów one dotyczą.

Implementację modelu znajdziesz w źródłach PyTorch/torchvision – należy zapoznać się przynajmniej z klasą `FasterRCNN` i `GeneralizedRCNN`:

https://github.com/pytorch/vision/blob/main/torchvision/models/detection/faster_rcnn.py

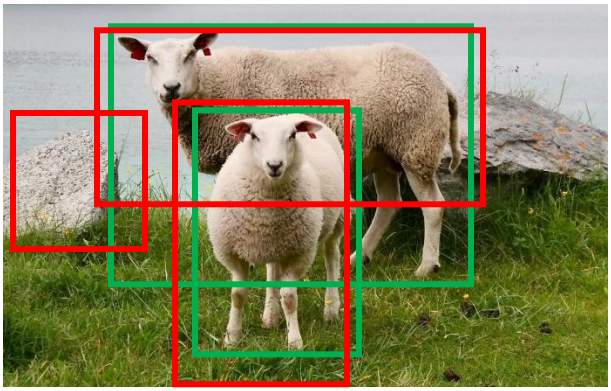
Zarówno implementacja modelu jak i procedura uczenia będą podane w notebooku, natomiast w razie potrzeby lub ciekawości zachęcam do spojrzenia w standardowe skrypty uczenia w module `references` pakietu torchvision:

https://github.com/pytorch/vision/blob/main/references/detection/train.py

Ocena jakości

Jak w każdym zadaniu, w detekcji obiektów rozpatruje się szereg różnych metryk służących ocenie jakości predykcji uzyskanych przez badany algorytm. Najczęściej podawanym w kontekście detekcji wynikiem jest *AP* (od *average precision*), nieraz w formie „AP75", „AP@50", „AP@[.5:.05:.95]", czy *mAP* (*mean average precision*). Oznaczenia potrafią być mylące, dlatego w tej części instrukcji prześledzimy proces oceny jakości detekcji.

Pierwszym krokiem – nie tylko przy ocenie, ale oczywiście także podczas uczenia – jest dopasowanie predykcji modelu do etykiet. To znaczy, znalezienie prawdziwych bounding boksów odpowiadających poszczególnym predykcjom. Celem jest zakwalifikowanie każdej predykcji jako poprawnej lub nie: TP albo FP (zauważ, że nie ma możliwości powstania *wprost* predykcji negatywnej – TN).



Wracając do przykładu z owcami: kolorem zielonym zaznaczono etykiety, czerwone prostokąty stanowią predykcje modelu.

Aby stwierdzić, czy predykcja odpowiada etykiecie, oblicza się miarę IoU pomiędzy nachodzącymi na siebie prostokątami. Następnie, w oparciu o przyjętą uprzednio wartość progu, dokonuje się binarnej klasyfikacji:

jeśli IoU(predykcja, etykieta) > próg → predykcja TP

W przypadku wielokrotnych predykcji odpowiadających tej samej etykiecie, przyjmujemy najlepszą z nich a pozostałe traktujemy jako FP.

Dla powyższego przykładu: jeśli przyjmiemy próg IoU=0.5, otrzymamy 2 predykcje poprawne i 1 fałszywą; jednak jeśli zwiększyć próg do poziomu 0.9, to predykcja dla owcy widzianej od boku zostanie potraktowana jako fałszywa.

Taki proces powtarzamy dla całego zbioru danych (tj. dla wszystkich predykcji po wszystkich obrazach), uzyskując dla zadanej wartości progu listę predykcji z ocenami TP/FP. Następnym krokiem jest obliczenie wartości precyzji i czułości (*precision and recall*):

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{wszystkie\ predykcje}$$

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{wszystkie\ etykiety}$$

Zauważ, że w pewnych warunkach detekcje nadmiarowe mogą spowodować, że uzyskana zostanie czułość bliska 1; z drugiej strony zbyt ostrożny algorytm uzyska wysoką precyzję kosztem licznych fałszywych negatywów. Aby uwzględnić błędy obu typów, w ocenie jakości algorytmów detekcji stosuje się przybliżenie powierzchni pod krzywą precyzji i czułości, nazywane AP (*average precision*), które dodatkowo uwzględnia *pewność* algorytmu – fałszywy pozytyw, który został oceniony jako obiekt z prawdopodobieństwem 90% jest uznawany za błąd poważniejszy, niż gdyby prawdopodobieństwo wynosiło np. 70%.

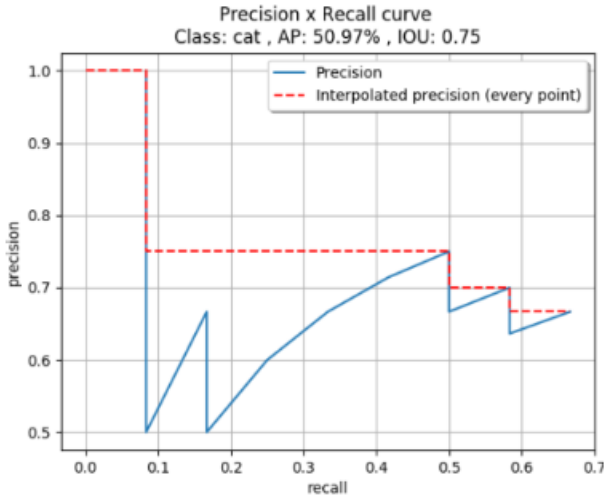
Procedura obliczenia AP jest następująca:

1. Uszereguj wszystkie predykcje w tablicę, sortując malejąco po pewności predykcji
2. Oznacz każdą predykcję jako TP lub FP na podstawie IoU z etykietą
3. Przechodząc od góry tabeli, zliczaj skumulowaną precyzję *na danym poziomie pewności* (czyli: dla każdej **i**-tej próbki licz precyzję osobno, tak jakby istniało tylko pierwsze **i** próbek)
4. Przechodząc od góry tabeli, zliczaj skumulowaną czułość (analogiczna procedura, tylko mianownik nie zmienia się – patrz wzór)

bounding box	confidence (τ)	IOU	IOU > 0.75?	$\sum$TP(τ)	$\sum$FP(τ)	$Pr(\tau)$	$Rc(\tau)$
D	99%	0.91	Yes	1	0	1.0000	0.0833
K	98%	0.70	No	1	1	0.5000	0.0833
C	95%	0.86	Yes	2	1	0.6667	0.1667
H	95%	0.72	No	2	2	0.5000	0.1667
L	94%	0.91	Yes	3	2	0.6000	0.2500
I	92%	0.86	Yes	4	2	0.6667	0.3333
A	89%	0.92	Yes	5	2	0.7143	0.4167
F	86%	0.87	Yes	6	2	0.7500	0.5000
J	85%	-	No	6	3	0.6667	0.5000
B	82%	0.84	Yes	7	3	0.7000	0.5833
E	81%	0.74	No	7	4	0.6364	0.5833
G	76%	0.76	Yes	8	4	0.6667	0.6667

Stan tabeli po wykonaniu kroków 1-4 (Źródło [4], GitHub)

5. Tak uzyskane pary precyzja-czułość umieść na wykresie
6. Oblicz pole pod tym wykresem według wybranej metody interpolacji (*11-point*/*every-point*)



Krzywa precyzja-czułość z naniesioną interpolacją (Źródło: tamże)

Uzyskana wartość stanowi *średnią precyzję* – AP. Z reguły podaje się próg IoU, przy jakim wartość została wyznaczona, stąd właśnie oznaczenia „AP75" (AP przy progu IoU równym 0.75). Jeśli natomiast chodzi o *mAP* (*mean average precision*), jest to miara uwzględniająca rozbicie AP na poszczególne klasy:

$$mAP = \frac{1}{n}\sum AP_k$$

(gdzie AP_k stanowi AP wyliczone dla klasy k).

W praktyce nieraz spotkasz się z różnymi dodatkowymi miarami, jak np. APS/APM/APL – AP dla obiektów *Small*, *Medium* oraz *Large* (według przyjętych kryteriów wielkości) albo mAP uśredniane dla różnych wartości progów – tak robi się np. w wyzwaniu COCO [5]. Dobrze jest, czytając artykuły czy analizując wyniki, zwracać uwagę na przyjmowane przez różnych autorów definicje, gdyż – jak wiele pojęć w tak dynamicznie rozwijających się dziedzinach nauki – potrafią się niekiedy od siebie różnić.

Bibliografia

[1] Ren et al. – *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* (2015) https://arxiv.org/abs/1506.01497

[2] Lin et al. – *Feature Pyramid Networks for Object Detection* (2016) https://arxiv.org/abs/1612.03144

[3] He et al. – Mask R-CNN (2017) https://arxiv.org/abs/1703.06870

[4] Padilla et al. - *A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit* (2021) https://doi.org/10.3390/electronics10030279
https://github.com/rafaelpadilla/Object-Detection-Metrics
https://github.com/rafaelpadilla/review_object_detection_metrics

[5] https://cocodataset.org/#detection-eval