

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Instytut Informatyki

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Sztuczna Inteligencja

Implementacja i zbadanie nowej architektury sieci
neuronowych wykorzystującej atencję w zadaniu
detekcji obiektów na obrazie

Łukasz Staniszewski

Numer albumu 304098

promotor
dr hab. inż. Paweł Wawrzyński

WARSZAWA 2023

Implementacja i zbadanie nowej architektury sieci neuronowych wykorzystującej atencję w zadaniu detekcji obiektów na obrazie

Streszczenie. Niniejsza praca prezentuje nowe rozwiązanie problemu detekcji obiektów na obrazie przy użyciu sieci neuronowych, architektury Focus-CNN, wykorzystującej mechanizm atencji do wspomagania swojego działania, poprzez stosowanie operacji przesuwania, przybliżania i obracania obrazu, by łatwiej móc identyfikować obiekty na nim się znajdujące. Koncepcja modelu zainspirowana została istniejącym już w tym obszarze rozwiązaniem Faster R-CNN i, tak samo jak ono, również stanowi sieć dwustopniową, której każdy z komponentów jest nauczalny. Prezentowane rozwiązanie składa się na trzy moduły: sieć skupiającą (pełniącą funkcję propozycji regionów z obiektem poprzez podawanie parametrów przekształcenia obrazu), moduł transformujący (przekształcający obraz na podstawie parametrów tak, aby otrzymać jego wycinek najlepiej wskazujący obiekt) i sieć klasyfikującą (podejmującą decyzję o kategorii obiektu bazując na wyciętym i przekręconym fragmencie obrazu). Celem pracy była implementacja podstawowej wersji modelu, a jej wynikiem kod zapewniający, że każda z części architektury spełnia swoje zadanie, aby łatwiej móc być rozszerzaną w przyszłości. Główny wniosek z przeprowadzonych badań na obrazach ze zbiorów PASCAL VOC 2012 oraz COCO-2017 i porównania wyników sieci z wynikami modelu Faster-RCNN, jest taki, że nowo proponowane rozwiązania osiąga lepszą skuteczność od swojego konkurenta w tym badaniu i ma predyspozycje, aby stać się jednym z najlepszych detektorów. Dodatkowo udało się zauważyć, że uwzględnienie rotacji w problemie, choć nie gwarantuje lepszej skuteczności w lokalizacji obiektów na obrazie, potrafi wzmacnić ich klasyfikację, co wskazuje potencjał tego parametru w rozwiązywanym zadaniu.

Słowa kluczowe: detekcja obiektów, atencja, wizja maszynowa, uczenie maszynowe

Implementation and evaluation of a new neural network architecture using attention in the task of object detection

Abstract. The thesis presents a new solution to the problem of detecting objects in an image using neural networks, the Focus-CNN architecture that uses the attention mechanism to support its processing by applying translation, scaling and rotation operations to identify objects in the image with greater ease. The concept of the model was inspired by Faster R-CNN - the existing solution in this area and, likewise, is also a two-stage network, with each component learnable. The presented solution consists of three modules: the focus network (which plays the role of a proposal of regions with objects by providing parameters for image transformation), a transforming module (which transforms the input image based on parameters to obtain the image fragment that best indicates the object) and a classifying network (which decides the category of the object based on the clipped and twisted image part). The thesis goal was to implement a basic version of the model, and the result was code ensuring that each part of the architecture fulfills its purpose so that it can be developed in the future. The main conclusion of the experiments carried out on both images from the PASCAL VOC 2012 and COCO-2017 datasets, with the comparison of the network results with those of the Faster-RCNN model, is that the newly proposed solution achieves better performance than its competitor in this study and has the capacity to become one of the best detectors. In addition, it has been observed that the inclusion of rotation in the problem, although it does not guarantee better performance in locating objects in the image, can enhance architecture classification ability, which indicates the potential of this parameter in the solved task.

Keywords: object detection, attention, computer vision, machine learning



Politechnika Warszawska

załącznik nr 3 do zarządzenia
nr 28 /2016 Rektora PW

Warszawa, 06.09.2023.

miejscowość i data

Fukasz Staniśiawski

imię i nazwisko studenta
304098

numer albumu
Informatyka

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 module APD systemu USOS są identyczne.

Stanisław Fukasz
czytelny podpis studenta

Spis treści

1. Wstęp	9
1.1. Cel i zakres pracy	11
1.2. Wykorzystane narzędzia	12
1.3. Słownik pojęć	12
2. Detekcja obiektów a sieci neuronowe	14
2.1. Kwestie techniczne związane z detekcją obiektów	14
2.1.1. Obwiednie obiektów	14
2.1.2. Strata w detekcji	15
2.1.3. Indeks Jaccarda (IoU)	16
2.1.4. Tłumienie niemaksymalne (NMS)	16
2.1.5. Ewaluacja modeli do detekcji obiektów	17
2.2. Istniejące rozwiązania	20
2.2.1. Podział architektur sieci neuronowych do detekcji	20
2.2.2. Architektura R-CNN	21
2.2.3. Architektura Fast R-CNN	22
2.2.4. Architektura Faster R-CNN	23
2.2.5. Architektura Mask R-CNN	25
2.3. Sieć rezydualna	25
2.4. Mechanizm atencji	26
2.4.1. Atencja jako mechanizm pochodzący z neuronauki	26
2.4.2. Wykorzystanie atencji przez sieci neuronowe	26
2.4.3. Matematyczna teoria atencji	28
3. Proponowane rozwiązanie	30
3.1. Ogólna architektura sieci	30
3.2. Sieć skupiająca - Focus Network	31
3.2.1. Ogólny opis	31
3.2.2. Działanie sieci	32
3.2.3. Architektura sieci	32
3.2.4. Funkcja straty	33
3.3. Moduł Transformujący - Transform Module	34
3.4. Sieć klasyfikująca - Classifier	36
3.4.1. Architektura sieci	37
3.5. Trening architektury	37
3.5.1. Trening wstępny sieci Focus	37
3.5.2. Preprocessing danych - sieć Focus	37
3.5.3. Trening wstępny sieci Classifier	41
3.5.4. Dostrojenie całej architektury	42

3.6. Wyznaczanie parametrów obwiedni na podstawie skalarów transformacji	43
3.7. Mechanizm skupiający	43
3.8. Potencjalne korzyści i ograniczenia rozwiązania	44
3.9. Proponowane rozszerzenia architektury	45
4. Implementacja rozwiązania	47
4.1. Różniczkowalne przekształcenie geometryczne	47
4.1.1. Inspiracja dla rozwiązania	47
4.1.2. Rozwiązanie przy użyciu PyTorch	48
4.1.3. Przetestowanie rozwiązania	49
4.2. Struktura kodu	51
5. Przeprowadzone badania	52
5.1. Zbiory danych	53
5.1.1. Zbiór PASCAL VOC 2012	53
5.1.2. Zbiór COCO-2017	53
5.1.3. Przygotowanie zbiorów danych	53
5.2. Zbadanie skuteczności działania architektury	54
5.2.1. Wyniki architektury Faster R-CNN	54
5.2.2. Przetrenowanie sieci skupiających	55
5.2.3. Przetrenowanie sieci klasyfikującej	59
5.2.4. Dostrajanie architektury	61
5.3. Zbadanie wpływu rotacji na wynik sieci	62
5.4. Zbadanie wpływu pełnego przepływu gradientu na wynik sieci	64
5.5. Przykłady działania	65
6. Podsumowanie	67
Bibliografia	69
Spis rysunków	74
Spis tabel	75

1. Wstęp

Rozpoznawanie obrazów, inaczej wizja maszynowa (ang. computer vision), jest obecnie jednym z najważniejszych obszarów informatyki, którego zastosowania występują w bardzo wielu miejscach życia codziennego. Obszar ten swoją skuteczność znaczco zauważa sztucznej inteligencji, a szczególnie uczeniu maszynowym, gdzie komputery uczone są jak należy interpretować świat wizualny po to, by potem reagować w odpowiedni sposób na to, co przetwarzają. Aby rozpoznawać obiekty, nie muszą one posiadać informacji jak zbudowany jest cel (przykładowo, że twarz posiada parę oczu, niżej nos i usta), tylko wymagają dostarczenia im odpowiedniej liczby przykładów, na podstawie których same stworzą dla siebie taką reprezentację.

Narzędzia uczenia maszynowego, które najlepiej radzą sobie z zadaniami związanymi z wizją maszynową to przede wszystkim sieci neuronowe, co pokazały badania przeprowadzone w ostatnich latach. Działanie sieci neuronowych podczas rozpoznawania obrazów może przypominać układanie puzzli - na początku rozróżniają one fragmenty obrazów poprzez wykrywanie krawędzi, te krawędzie łączą się w elementy, a te natomiast - w obiekty na obrazie, na podstawie których maszyny podejmują decyzję. Najpopularniejsze zadania sieci neuronowych w wizji maszynowej to m.in. klasyfikacja obrazów (polegająca na nadaniu kategorii danemu obrazowi - przykładowo zdjęcie człowieka), rozpoznawanie twarzy na obrazie (gdzie poza decyzją czy twarz znajduje się na obrazie, istotne jest również ustalenie tożsamości człowieka), segmentację (gdzie obraz dzielony jest na regiony jednakowe pod względem pewnych cech - przykładowo rozdzielenie na zdjęciu pikseli trawy od pikseli reprezentujących niebo) czy detekcję obiektów na obrazie, na której opisywana praca inżynierska się koncentruje.

Detekcja obiektów na obrazie polega na jednoczesnym wykrywaniu i rozpoznawaniu elementów na nich się znajdujących na podstawie ich cech wizualnych. Zadanie to wymaga wskazywania miejsca obiektów, czyli ich lokalizowania, co jest realizowane przy użyciu tzw. obwiedni (ang. bounding-boxes), które jest najłatwiej wyjaśnić jako najmniejsze możliwe prostokąty równoległe do ramek obrazu, w pełni otaczające obiekty konkretnych kategorii.

W ostatnich latach detekcja stała się jednym z popularniejszych w zastosowaniu zagadnieniem w rozpoznawaniu obrazów i głębokim uczeniu. Najczęstszym przypadkiem, w jakim można spotkać działanie algorytmów realizujących detekcję obiektów, jest działanie najnowszych samochodów autonomicznych, których kamery wykrywają istotne obiekty na drodze, takie jak pojazdy, znaki czy sygnalizację świetlną, aby umożliwić sterowanie samochodem bez ingerencji kierowcy.



Rysunek 1.1. Przykładowy widok z kamery samochodu autonomicznego. Obiekty są lokalizowane, a także kategoryzowane przy użyciu koloru prostokątów. Grafika pochodzi z [1].

Innym zastosowaniem, gdzie można dostrzec algorytmy detekcji, jest rozpoznawanie twarzy (ang. facial recognition). Zadanie to często dzieli się na dwa etapy, gdzie najpierw realizowana jest detekcja twarzy na obrazie - to tutaj wykorzystywany jest algorytm detekcji, który wskazuje miejsce twarzy na większym zdjęciu, po to by ją "wydobyć" z obrazu. Następnie konieczne jest wykorzystanie innego algorytmu (często realizowanego również przez sieć neuronową), który ma za zadanie porównać taki wycinek z innymi twarzami w bazie danych w celu ostatecznego ustalenia tożsamości osoby. Przykładem takiego zastosowania jest porównywanie twarzy osoby ze zdjęciem w paszporcie na lotniskach.

Innym rodzajem aplikacji detekcji jest śledzenie obiektów (ang. object tracking). Gdy algorytmowi podawana jest pojedyncza grafika, otrzymywane jest miejsce obiektów na niej się znajdujących, natomiast podając takiej sieci film (zestaw klatek), otrzymywana jest zmieniająca się w czasie lokalizacja interesujących bytów. Takie rozwiązanie zastosować można m.in. do nagrań meczów piłki nożnej, w celu analizy zachowania zawodników na boisku i ustalania taktyki na mecz, z czego korzystają największe kluby piłkarskie w Europie, czy do celów militarnych (np. wspomagania działania dronów śledczych).

Detekcja obiektów wykorzystywana jest również przy analizie zdjęć satelitarnych. Z powodu, że tego typu zdjęcia charakteryzują się ogromnymi rozmiarami, najczęściej dzieli się je na mniejsze grafiki, które podaje się sieciom neuronowym. Dzięki tego typu rozwiązaniom możliwe jest przykładowo zliczanie ludzi na wydarzeniach o charakterze masowym, czy analizowanie liczby wolnych miejsc na parkingach o odkrytym dachu.



Rysunek 1.2. Przykłady zastosowań dla zadań detekcji obiektów na obrazie. Kolejno: rozpoznawanie twarzy, analiza zdjęcia satelitarnego, śledzenie obiektów. Źródła obrazów: [2], [3], [4].

1.1. Cel i zakres pracy

Ideą niniejszej pracy jest implementacja nowej architektury sieci neuronowych realizującej zadanie detekcji obiektów na obrazie, pod nazwą **Focus-CNN**, przy użyciu popularnej biblioteki do programowania modeli głębokiego uczenia oraz wstępne zbadanie jej działania na powszechnie dostępnych zbiorach danych i porównanie z istniejącym już w tym obszarze podejściem - **Faster R-CNN**. Koncepcja ta zadana została konkretnymi wytycznymi - ma być podzielona na dwie części, gdzie pierwsza z nich, część skupiająca, swoim działaniem ma przypominać znany w uczeniu maszynowym mechanizm atencji i ułatwić działanie drugiej z nich, klasyfikatorowi, poprzez proponowanie jej takich fragmentów obrazu, na podstawie których podjęcie decyzji o kategorii obiektu na obrazie będzie łatwiejsze. Praca ma przede wszystkim charakter badawczy, koncentruje się wokół realizacji podstawowej wersji modelu oraz ma na celu sprawdzić, czy proponowane wprowadzenia mogą wspomóc sieci neuronowe w problemie detekcji. Stąd też praca nie stanowi ostatecznej wersji koncepcji i może zostać rozwinięta w przyszłości.

W ramach tej pracy istotnym elementem jest odpowiednie zaprojektowanie architektury, poparte analizą dotychczasowych rozwiązań w dziedzinie detekcji przy użyciu dostępnej literatury i artykułów naukowych. Kolejnym krokiem jest przejrzenie, zbadanie i wybranie zbiorów danych do badań, a następnie implementacja rozwiązania przy użyciu biblioteki programistycznej do uczenia Sieci Neuronowych, PyTorch, oraz stworzenie skryptów przygotowujących dane benchmarkowe do pracy ze stworzonym modelem.

Po wykonaniu wszelkich działań dotyczących implementacji architektury, w kolejnym kroku wytrenowane zostaną, na początku, poszczególne pojedyncze sieci, a następnie dostrojona zostanie całe rozwiązanie i nastąpi jego ewaluacja na 2 zbiorach danych. W tym celu konieczny będzie przegląd powszechnie obowiązujących metryk w zadaniu detekcji obiektów na obrazie i wybranie spośród nich tych, które najbardziej pasują do zadania.

Ostatnim etapem pracy będzie wytrenowanie i zmierzenie skuteczności działania architektury Faster R-CNN na wybranych wcześniej zbiorach oraz zestawienie wyników tych badań z wynikami osiąganymi przez proponowane rozwiązanie w celu sprawdzenia, czy implementacja działa oraz, czy model osiąga satysfakcjonującą skuteczność.

Implementacja wykonana w ramach tej pracy ograniczona została do wykrywania tylko jednego obiektu danej klasy na obrazie, natomiast niniejszy tekst przedstawia jedno z proponowanych podejść do rozszerzenia rozwiązania na wiele obiektów, co stanowi miejsce na potencjalny przyszły rozwój architektury.

1.2. Wykorzystane narzędzia

1. **Python** - wysokopoziomowy, obiektowy, interpretowany język programowania, który dzięki m.in. dynamicznemu typowaniu i dobrze rozwiniętym implementacjom podstawowych struktur danych stał się jednym z najlepszych narzędzi do błyskawicznego tworzenia aplikacji (ang. Rapid Application Development). Język ten bardzo dobrze radzi sobie z tworzeniem skryptów i łączeniem części kodu w całość. Łatwość używania Pythona oraz wysoki poziom zaangażowania społeczności w jego rozwój zapewniły, że stał się on podstawowym narzędziem programistycznym do uczenia maszynowego (ang. machine learning) i analizy danych (ang. data science), ponieważ gwarantuje on jak najmniejszy wkład programisty w tworzenie kodu (dzięki licznym dostępnym modułom zapewniającym gotowe rozwiązania typowych problemów), a pozwala mu się skupić na wysokopoziomowej realizacji algorytmów. Python stanowi główny język programowania części implementacyjnej tej pracy dyplomowej. Strona domowa języka Python: [5].
2. **PyTorch** - narzędzie (framework) uczenia maszynowego (a przede wszystkim uczenia głębokiego), używane w aplikacjach takich jak przetwarzanie języka naturalnego czy wizja maszynowa. Oprogramowanie te stworzone zostało w celu implementacji sztucznych sieci neuronowych i napisanie przy użyciu języków Python, C++ oraz architektury kart graficznych CUDA. PyTorch jest otwarto-źródłową, darmową biblioteką, na której zbudowane zostały popularne na całym świecie rozwiązania takie jak Tesla Autopilot, Hugging Face Transformers czy Pyro (autorstwa Ubera). Dwie ma głównymi funkcjami, jakie zapewnia PyTorch, jest system automatycznego różniczkowania grafu obliczeń sieci neuronowych, a także realizacja obliczeń przy użyciu Tensorów, na których operacje mogą być wykonywane przy użyciu kart graficznych. W ramach tej pracy PyTorch wykorzystany został do implementacji proponowanej architektury. Strona domowa: [6].

1.3. Słownik pojęć

W ramach tej pracy zostały wprowadzone pojęcia i definicje, których dodatkowe wyjaśnienia mogą wspomóc czytelnika w rozumieniu dalszej części tekstu:

1. **Atencja** - popularny w dziedzinie uczenia głębokiego (ang. deep learning) mechanizm, który ma za zadanie wspomagać sieć neuronową w jej działaniu poprzez proponowanie jej jedynie takich fragmentów z danych wejściowych, które z jej punktu widzenia są istotne (w przypadku tej pracy - konkretne obiekty na obrazie), a odrzucenie tych, które nie powinny wpływać na jej decyzję (przykładowo tło zdjęcia). Mechanizm atencji, jego pochodzenie oraz wykorzystanie przez sieci neuronowe zostało szczegółowo opisane w podrozdziale 2.4.

2. **Detekcja** - zadanie z dziedziny wizji komputerowej (ang. computer vision), polegającym na jednoczesnym wskazywaniu kategorii i miejsca znajdowania się obiektu na obrazie (klasyfikacji i lokalizacji).
3. **Ewaluacja** - etap pracy z modelami uczenia maszynowego, w trakcie którego badana jest skuteczność rozwiązań z tego obszaru, poprzez liczenie metryk (liczbowych statystyk określających jak dobry jest model), jakie są przez nie osiągane na zbiorze walidacyjnym (próbce danych, do których model nie ma dostępu w trakcie uczenia). Ewaluacja pozwala ocenić jak rozwiązanie zadziałałoby, gdyby zostało wykorzystane w rzeczywistym zadaniu, a także pomaga stwierdzić, czy jedno rozwiązanie jest lepsze od drugiego.
4. **Obwiednia (prostokąt otaczający)** - sposób adnotacji lokalizacji obiektów na obrazie we wszelkich zbiorach benchmarkowych, na których wykonywane są badania dotyczące zadania detekcji. Wizualnie przedstawiane są jako najmniejsze prostokąty otaczające w pełni obiekty konkretnych kategorii na obrazie. Oryginalna, angielska nazwa tego pojęcia to bounding-box, a dokładniejszy jego opis zamieszczony został w podrozdziale 2.1.1
5. **Preprocessing danych** - etap pracy z zadaniami uczenia maszynowego, polegający na takim odpowiednim przetworzeniu zbioru danych, aby można było go wykorzystać jako wejście do modelu.
6. **Zbiór benchmarkowy** - zestaw danych wraz z prawidłowymi oznaczeniami (ang. labels) skonstruowany dla specyficznego problemu w celu porównania opracowanych algorytmów sztucznej inteligencji rozwiązujących dany problem i określeniu najskuteczniejszego z nich. Najczęściej, zbiór ten zawiera w sobie wydzielony podzbiór treningowy, na którym modele są optymalizowane oraz podzbiór walidacyjny, na którym modele są ewaluowane. Architektury, które osiągają najlepsze metryki na najpopularniejszych zbiorach dla danego problemu, nazywane są mianem State Of The Art (SOTA).

2. Detekcja obiektów a sieci neuronowe

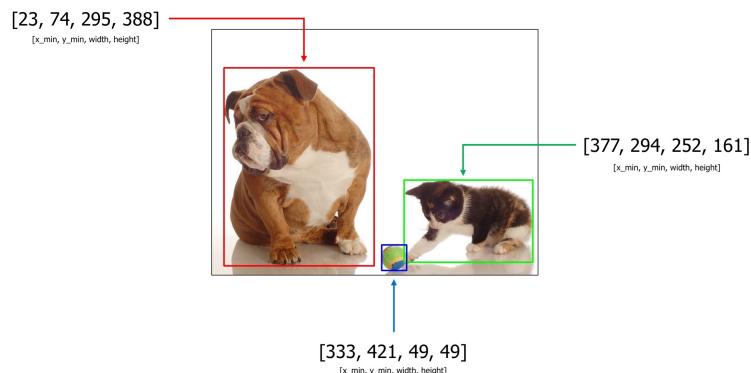
Detekcja obiektów może być realizowana zarówno przy użyciu metod opartych na regułach, jak i metod uczenia maszynowego. Obecnie największe sukcesy w tej dziedzinie osiąga się dzięki użyciu sieci neuronowych, które potrafią nauczyć się samodzielnie wykrywać obiekty na obrazie dzięki wcześniejszemu treningowi na dużych zbiorach danych. Niniejszy rozdział opisuje kwestie techniczne związane z zagadnieniem detekcji obiektów na obrazie w kontekście uczenia maszynowego, jakie mogą się przydać czytelnikowi w celu zrozumienia działania proponowanej architektury. Dodatkowo opisane zostały tutaj najpopularniejsze istniejące już architektury sieci neuronowych w tym zadaniu, a także przedstawiony został mechanizm atencji i jego wykorzystanie w sztucznej inteligencji.

2.1. Kwestie techniczne związane z detekcją obiektów

Zadanie to jest nieco trudniejsze od typowej klasyfikacji obrazów - wymaga ono dodatkowo, poza kategoryzowaniem obiektów, wskazywania ich miejsca na obrazie.

2.1.1. Obwiednie obiektów

Lokalizacja obiektów realizowana jest technicznie przy użyciu obwiedni (najmniejszych prostokątów otaczających obiekty). Są to figury równoległe do konturów obrazu, które oznaczają najmniejszy prostokąt w pełni zawierający obiekt. Przykład takich prostokątów został przedstawiony na rysunku 2.1.



Rysunek 2.1. Przykład obrazu z trzema obiektami i ich obwiedniami. Źródło: [7].

Prostokąty te opisywane są przy użyciu 4-elementowego wektora $[x_A, y_A, w, h]$, na który składa się:

- (x_A, y_A) - współrzędne (x, y) piksela obrazu, w którym znajduje się górny lewy róg prostokąta (lub centrum prostokąta),
- w - szerokość prostokąta,
- h - wysokość prostokąta.

Alternatywnie, w literaturze można spotkać opis $[x_A, y_A, X_D, y_D]$, gdzie punkt D oznacza prawy dolny róg obwiedni, ponieważ z założenia figura otaczająca obiekt jest prostokątem, którego boki są równoległe do obu osi układu współrzędnych. Ostatecznie jednak, w ramach tej pracy, stosowany jest zapis zawierający szerokość i wysokość prostokąta.

2.1.2. Strata w detekcji

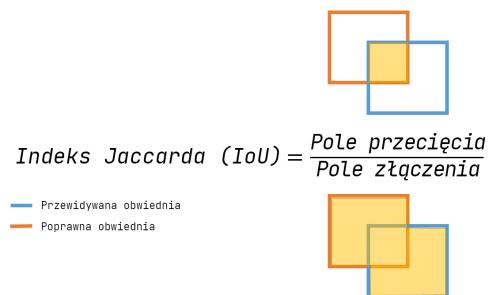
Modele sieci neuronowych trenowane są poprzez optymalizowanie straty jaką ponoszą, gdy są uczone. W takiej sytuacji zadaniem optymalizatora jest minimalizowanie matematycznej funkcji różnicowej między tym, co powinny zwracać, a tym, co zwracają.

Dwoma najpopularniejszym rodzajami funkcji straty są funkcje straty klasyfikacyjnej (ang. classification loss) oraz funkcje straty regresyjnej (ang. regression loss). Pierwsze z nich stosowane są w zadaniach, gdzie istotne jest, aby sieć wybierała ze zbioru ściśle ustalonych klas kategorię, do jakiej należy to, co otrzymuje na wejściu. Ze względu na to, że oczekuje się od niej, aby jak najgorszą ocenę przypisywała klasom, których nie ma na obrazie, a jak najlepszą kategorią, jaką przykład się oznacza, jej wyjście modelowane jest jako wektor liczb - prawdopodobieństw należenia wejścia do konkretnych klas. Przykładem takich funkcji jest strata entropii krzyżowej (ang. cross-entropy loss) czy strata ogniskowa (ang. focal loss). Drugie z nich, funkcje straty regresyjnej, mają nieco odmienne zadanie niż straty klasyfikacyjne - dominują one w zadaniach, gdzie zamiast wybrania kategorii ze ściśle ustalonego zbioru klas, sieć ma za zadanie wyznaczać wartość zmiennej wyjaśnianej. W takiej sytuacji funkcja straty ma oceniać, jak blisko predykcje modelu znajdują się względem oczekiwanych liczb poprzez określanie liczbowe dystansu między nimi. Liczba ta wyliczana jest tutaj jako pewna funkcja odległości między obiema wartościami, gdzie przykładowo dla błędu kwadratowego (ang. square error) jest to odległość podniesiona do kwadratu, a dla błędu bezwzględnego (ang. absolute error) - wartość bezwzględna odległości.

W zadaniu detekcji obiektów problem wybrania odpowiedniej funkcji straty nieco się komplikuje - z jednej strony mamy do czynienia z zadaniem klasyfikacji (ponieważ decydujemy o kategorii obiektu wykrywanego), a z drugiej strony podejmujemy się również zadania regresji (ponieważ próbujemy wyznaczyć koordynaty i wielkości obwiedni). Stąd też, w tego typu zadaniach wykorzystywana jest suma ważona tych dwóch strat, tzn. osobno wyliczany jest błąd klasyfikacyjny na decyzji modelu o prawdopodobieństwie konkretnej klasy, a osobno liczony jest błąd regresji zarówno na współrzędnych (x, y) środka obwiedni, jak i jej wysokości i szerokości, po czym jest uśredniany na tych czterech liczbach. Po otrzymaniu takich dwóch błędów każdy z nich mnożony jest przez odpowiednie wagi, a liczby te są sumowane, aby ostatecznie otrzymać liczbę definiującą ostateczny błąd, do którego każda ze strat kontrybuje w konkretnym stopniu tak, by sieć uczyła się obu zadań jednocześnie.

2.1.3. Indeks Jaccarda (IoU)

Indeks Jaccarda (inaczej IoU - z ang. Intersection over Union, przedstawiony w [8]) jest metryką stosowaną w detekcji do mierzenia pokrycia pomiędzy dwiema różnymi obwiedniami. W większości przypadków służy ona do sprawdzenia, jak prostokąt propo- nowany przez model pokrywa się z prawdziwym prostokątem, uznanym za prawidłowy w zbiorze danych. Liczba ta, jak jej angielska nazwa wskazuje, stanowi iloraz pól przecięcia (intersekcji) i złączenia (unii) dwóch obwiedni. Schematycznie tę operację przedstawia rysunek 2.2.

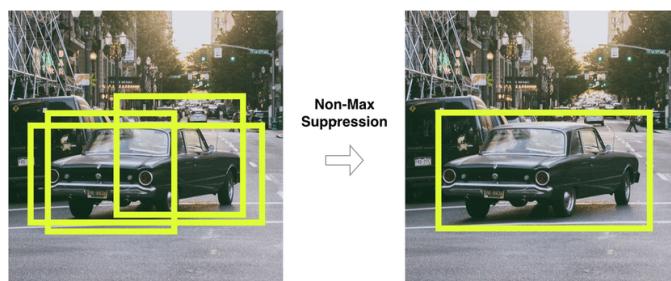


Rysunek 2.2. Schemat liczenia miary IoU dla dwóch dowolnych obwiedni.

Indeks Jaccarda dla dwóch prostokątów może przyjmować wartości z przedziału $[0, 1]$, gdzie liczba 0 oznacza kompletny brak ich części wspólnej, natomiast liczba 1 - ich idealne pokrycie. Zazwyczaj, w miejscach gdzie wykorzystywana jest wartość IoU, ustalany jest dodatkowy parametr zwany progiem (ang. threshold). Jest to liczba, której przekroczenie oznacza stwierdzenie, że prostokąty wskazują to samo miejsce na obrazie.

2.1.4. Tłumienie niemaksymalne (NMS)

Jedną z istotnych technik stosowanych głównie w detekcji obiektów jest tłumienie niemaksymalne (ang. non-maximum suppression). Metoda ta polega na wybraniu ze zbioru kilku obwiedni nakładających się na siebie najlepszej, w celu przefiltrowania wszystkich proponowanych przez sieć regionów i doprowadzeniu do sytuacji, gdy obiekty na obrazie są wskazywane przez maksymalnie jedną obwiednię. Przykład zastosowania techniki NMS został przedstawiony na rysunku 2.3.



Rysunek 2.3. Przykład zastosowania techniki tłumienia niemaksymalnego (NMS). Źródło: [9].

Metodę tłumienia niemaksymalnego przeprowadza się osobno dla każdej z rozpatrywanych klas. W pierwszym kroku algorytmu porządkuje się zbiór wszystkich predykcji modelu, malejąco względem pewności klasyfikatora co do klasy obiektu. Następnie algorytm realizuje pętlę, w której to wybiera pierwszą predykcję z listy (tą z największą pewnością) i eliminuje ze zbioru obwiedni te, które posiadają wysokie z nią pokrycie (indeks Jaccarda wyliczony między nimi jest większy niż ustalony próg). Na końcu obwiednia ta przenoszona jest do zbioru ostatecznych prostokątów otaczających i kroki te są powtarzane do momentu, aż zbiór wszystkich obwiedni przewidzianych przez model będzie pusty.

Metoda ta wykorzystywana jest m.in. w architekturach takich jak te z rodziny YOLO (wprowadzone w [10]) i R-CNN (poruszone po raz pierwszy w [11]), gdzie eliminuje pokrywające się ze sobą obwiednie produkowane przez modele. W proponowanej architekturze metoda tłumienia niemaksymalnego nie została wykorzystana ze względu na fakt, że model proponuje zawsze co najwyżej jeden obiekt danej klasy na obrazie, natomiast uznano, że opisanie tej techniki może ułatwić czytelnikowi zrozumienie architektur opisywanych w dalszej części tego rozdziału.

2.1.5. Ewaluacja modeli do detekcji obiektów

Posiadając kilka modeli sieci neuronowych przeznaczonych do konkretnego zadania, zwykle istnieje potrzeba porównania ich ze sobą. Potrzebne jest wtedy zdefiniowanie pewnej miary, której większa wartość osiągana przez dany model pozwala nam stwierdzić, że radzi sobie on lepiej w jakimś zadaniu. Liczba ta nazywana jest metryką.

Ze względu na fakt, że jedną z funkcji jaką pełnią detektory jest przypisywanie kategorii do obiektów, w zadaniu detekcji wykorzystywane są m.in. metryki znane z zadania klasyfikacji. W przypadku klasyfikacji binarnej wykrywana jest tylko jedna kategoria, do jakiej mogą należeć obiekty, tzn. jeśli do niej należą, otrzymują oznaczenie 1, natomiast gdy są jakiekolwiek innej kategorii, otrzymują oznaczenie 0. Teraz uwzględniając fakt, że model może wyznaczać predykcje o wartościach 0 lub 1, a także obiekty powinny zostać oznaczone jako 0 lub 1, możliwe są 4 przypadki:

- Model nie pomylił się, oznaczając obiekt jako przypadek pozytywny (1) - TP (ang. True Positive).
- Model nie pomylił się, oznaczając obiekt jako przypadek negatywny (0) - TN (ang. True Negative).
- Model pomylił się - oznaczył obiekt jako przypadek pozytywny (1), a powinien jako negatywny (0) - FP (ang. False Positive).
- Model pomylił się - oznaczył obiekt jako przypadek negatywny (0), a powinien jako pozytywny (1) - FN (ang. False Negative).

Następnie, posiadając liczby przykładów należących do każdej z tych czterech klas, zdefiniować można następujące metryki:

- Dokładność (ang. accuracy) - definiująca jaki procent przykładów jest przypisywany do poprawnej kategorii: $\frac{|TP|+|TN|}{|TP|+|TN|+|FP|+|FN|}$
- Precyza (ang. precision) - definiująca jaki procent przykładów wyznaczanych przez model jako pozytywne jest poprawne: $\frac{|TP|}{|TP|+|FP|}$
- Czułość (ang. recall) - definiująca jaki procent przykładów, które powinny być pozytywne, są poprawnie wyznaczane przez model: $\frac{|TP|}{|TP|+|FN|}$
- Miara $F1$ (ang. F1-score) - średnia harmoniczna precyzji i czułości: $\frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}}$

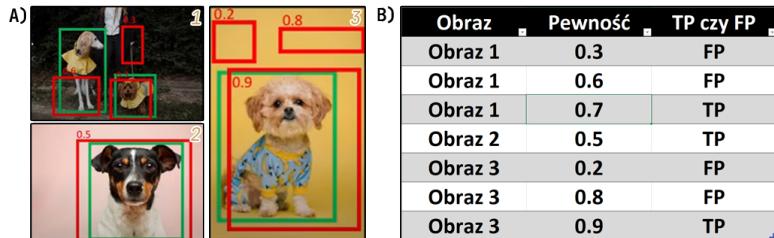
Aby uogólnić powyższe metryki na zadanie z wieloma klasami, dla każdej z rozpatrywanych kategorii niezależnie wyliczane są wartości $|TP|$, $|FP|$, $|TN|$ i $|FN|$ poprzez sprowadzenie problemu do N zadań binarnych "1 kontra reszta", gdzie N oznacza liczbę kategorii. Następnie ostaczne wartości metryk wyliczane są w zależności od podejścia:

- Makro-uśrednianie - metryki takie jak dokładność, precyza, czułość czy miara $F1$ są wyliczane osobno dla każdej z klas, a następnie uśredniane. Podejście to zapewnia, że każda z klas ma taki sam wpływ na ostateczną metrykę niezależnie od liczby przykładów danej kategorii w zbiorze.
- Mikro-uśrednianie - uzyskane liczby $|TP|$, $|FP|$, $|TN|$ i $|FN|$ dla każdego z zadań binarnych są ze sobą sumowane, a następnie na podstawie tych sum wyznaczane są powyższe wskaźniki jakości. Wtedy do ostatecznego wyniku metryki konkretnych kategorii kontrybuują zgodnie z rozkładem klas w zbiorze danych.

Należy jednak pamiętać, że w detekcji obiektów nie jest wystarczające sprawdzanie, jak dobrze wyznaczane są klasy przez model - dodatkowo ma on dobrze lokalizować obiekty na obrazie poprzez poprawne wyznaczanie prostokątów otaczających obiekty. Metryka, która łączy mierzenie poprawności lokalizowania i kategoryzowania obiektów oraz która uznawana jest za najlepszą do decydowania o jakości detektorów to miara mAP (z ang. Mean Avarage Precision).

Metryka ta wykorzystuje dwa wyjścia sieci neuronowej - przewidywane obwiednie obiektów oraz liczby oznaczające pewność modelu co do najbardziej prawdopodobnej klasy. Lokalizacja obwiedni proponowana przez model jest uznawana za poprawną (TP), gdy indeks Jaccarda liczony między tą obwiednią a którąś prawdziwą obwiednią na tym obrazie dotyczącej tej samej klasy przekracza ustalony próg, w innym przypadku lokalizacja uznawana jest za niepoprawną (FP). Aby ułatwić czytelnikowi zrozumienie obliczeń, wszystko wytłumaczone zostało na przykładach.

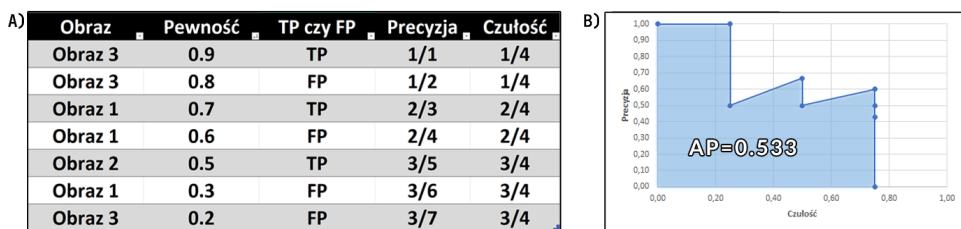
Aby policzyć mAP , najpierw konstruowana jest tabela, w której wierszach umieszczane są, dla każdego ze zdjęć, informacje z wszystkich predykcji pozytywnych modelu (czyli takie, gdzie model oznajmuje, że obiekt występuje), a dokładnie pewność modelu co do występowania obiektu na obrazie oraz oznaczenie poprawności lokalizacji (TP lub FP).



Rysunek 2.4. Przykład konstruowania tabeli z predykcjami modelu w metryce mAP liczonej dla obiektów psów. Na grafice (A) przedstawiono 3 obrazy uwzględnione w przykładzie, gdzie na zielono zaznaczono prostokąty opisujące prawdziwe obiekty, a na czerwono - predykcje modelu (obwiednie i pewność co do klasy obiektów); natomiast na grafice (B) przedstawiono tabelę skonstruowaną na podstawie tych predykcji.

Następnie, wiersze tabeli są sortowane w kolejności malejącej co do pewności modelu o klasie obiektu. Teraz, do każdego wiersza tabeli dodawane są dwie nowe wartości - precyza i czułość, gdzie dla danego wiersza precyzję wyznaczamy jako iloraz liczby TP do tej pory (licząc od góry tabeli) i liczby $TP + FP$ do tej pory (również licząc od góry tabeli), natomiast czułość dla danego wiersza wyznaczamy jako iloraz liczby TP do tej pory i liczby wszystkich prawidłowych obwiedni, jakie istnieją w zbiorze danych.

Na końcu rysowany jest wykres poprzez zaznaczenie punktów z tabeli jako współrzędnych (czułość, precyza) i połączenie ze sobą w krzywą. Liczba AP (ang. Average Precision) wyznaczana jest jako pole pod otrzymaną figurą, natomiast mAP ustalane jest jako średnia AP po wszystkich klasach uwzględnianych w zadaniu. Dodatkowo może być tak, że w ramach liczenia metryki potrzebne jest uwzględnienie kilku progów dla indeksu Jaccarda - wtedy mAP wyliczane jest tak jak do tej pory, dla każdego z progów osobno, a następnie uśredniane. Przykładowo, dla progu 0.5 miara ta opisywana jest jako $mAP@0.5$, a dla progów 0.5, 0.55, 0.6, ..., 0.95 jako $mAP@0.5:0.05:0.95$.

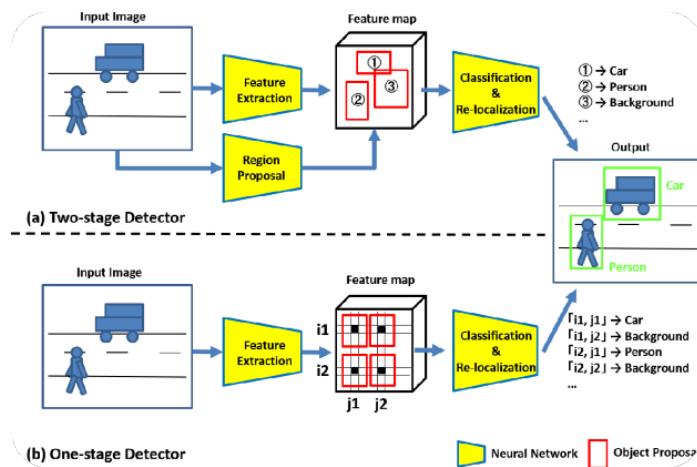


Rysunek 2.5. Kontynuacja przykładowu liczenia mowy AP dla pojedynczej klasy. Najpierw (grafika A) predykcje sortowane są względem wartości pewności malejąco i wyliczana jest precyza i czułość dla każdego z progów pewności. Na końcu (grafika B) rysowany jest wykres zależności precyzyi od czułości i wyznaczane jest pole pod tym wykresem.

2.2. Istniejące rozwiązania

2.2.1. Podział architektur sieci neuronowych do detekcji

W przypadku zadania detekcji obiektów na obrazie, projektanci architektur sieci neuronowych stają przed poważnym dilemmakiem związanym z przeciwnością dwóch głównych celów, jakie przyświecają algorytmom mającym realizować te zadanie - z jednej strony oczekiwane jest, jak w przypadku wszystkich sieci neuronowych, aby rozwiązanie było jak najbardziej skuteczne i przebiło wszystkie dotychczasowe rozwiązania pod względem metryk - co sugeruje używanie potężnych architektur zawierających duże liczby neuronów - natomiast, z drugiej strony, oczekiwane jest od tych sieci, żeby ich działanie było jak najszybsze, co wynika z charakteru zastosowań takich algorytmów (zadania czasu rzeczywistego), a co jednocześnie sugeruje korzystanie z jak najprostszych i najmniejzych architektur neuronowych (wykonujących jak najmniej operacji). Te dwa podejścia spowodowały podział architektur sieci neuronowych do detekcji obiektów na obrazie na sieci jednostopniowe (ang. one-stage networks) oraz sieci dwustopniowe (ang. two-stage networks).



Rysunek 2.6. Porównanie działania architektur jednostopniowych i dwustopniowych do detekcji obiektów na obrazie. W podpunkcie (b) przedstawione zostało działanie sieci jednostopniowych, gdzie jednocześnie produkowane są kategorie i lokalizacje obiektów na obrazie. W podpunkcie (a) zostało przedstawione działanie sieci dwustopniowych, zakładających pracę w dwóch etapach - propozycję regionów, w których mogą być obiekty, a następnie pracę na tych regionach w celu skategoryzowania obiektów. Źródło rysunku: [12].

Sieci jednostopniowe są sieciami o znacznie szybszym działaniu, ponieważ produkują zarówno kategorie, jak i miejsca występowania obiektów w jednym kroku (zdjęcie wejściowe przechodzi przez pojedynczą głęboką sieć neuronową), a klasyfikacja i lokalizacja nie wymaga w tym przypadku pracy na wcześniej ustalonych propozycjach regionów.

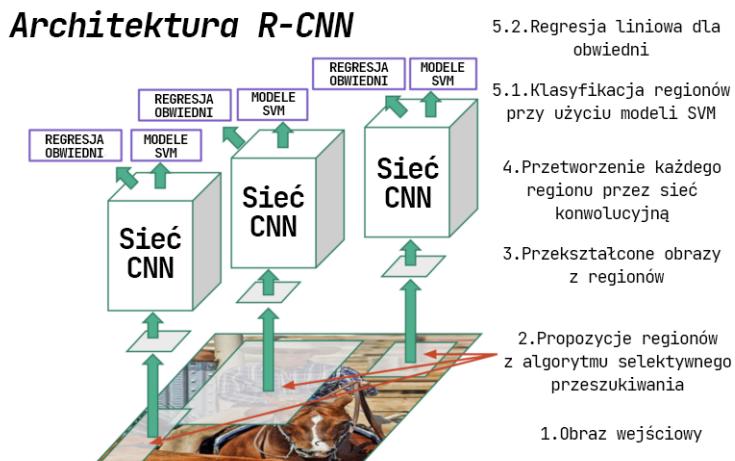
Najpopularniejszymi architekturami tego typu są architektury z rodziny YOLO (ang. You Only Look Once, przedstawiona w [10]), zakładające podział obrazu na siatkę prostokątów (przykładowo 10×10), gdzie każdy element takiej siatki realizuje zadanie opisywania obiektu, którego centrum znajduje się właśnie w tym obiekcie. Opis pojedynczego obiektu realizuje przy użyciu $C + 2 \cdot 4$ skalarów, wyznaczając C prawdopodobieństw należenia obiektu do każdej z kategorii oraz dwie proponowane obwiednie (opisane przy użyciu 4 liczb) definiujące lokalizację tego obiektu (gdzie tylko jeden z nich - ten o lepszym indeksie jaccarda - wykorzystywany jest jako ostateczna lokalizacja w trakcie ewaluacji). Modele YOLO osiągają rekordową szybkość działania na poziomie nawet 90 klatek na sekundę przy jednocześnie niezłą skuteczności działania, ale gorszej od sieci dwustopniowych.

Drugim rodzajem architektur neuronowych w tym zadaniu są sieci dwustopniowe, które, jak nazwa wskazuje, dokonują swoich decyzji w dwóch etapach. Pierwszy etap z nich - etap propozycji regionów zainteresowania - ma przedstawić zbiór fragmentów obrazów, w których mogą znajdować się obiekty. Następnie, w drugim etapie, sieć pracuje na uprzednio wskazanych regionach i na ich podstawie dokonuje swoich decyzji. Zastosowanie takiego podejścia pozwoliło zapewnić dużą skuteczność takich sieci kosztem wolniejszego ich działania względem jednostopniowych detektorów. Najpopularniejszym metodami z tej kategorii są Konwolucyjne Sieci Neuronowe Bazujące na Regionach - RCNN (ang. Region Based Convolutional Neural Networks). Ze względu na fakt, że proponowana architektura również plasuje się w kategorii sieci dwustopniowych, rozwiązania RCNN zostały szerzej opisane w kolejnych podrozdziałach.

2.2.2. Architektura R-CNN

Model R-CNN pojawił się po raz pierwszy w artykule [11], jako jeden z pierwszych rozwiązań do detekcji obiektów przy użyciu konwolucyjnych sieci neuronowych. Architektura w pierwszym etapie miała za zadanie wydobyć z obrazu propozycje regionów (ang. region proposals) - miejsc, w których mogą znajdować się obiekty na obrazie. To realizowane było przy użyciu przeszukiwania selektywnego (ang. selective search) - algorytmu będącego następcą podejścia z przesuwnym oknem (ang. sliding window). Algorytm przeszukiwania selektywnego najpierw generował wstępnią "podsegmentację" obrazka wejściowego, a następnie używał zachłannego podejścia do grupowania mniejszych regionów w większe. W R-CNN wyjście tego algorytmu stanowiło ok. 2000 propozycji regionów z obiekta jako prostokątów należących do obrazu, które następnie były zniekształcone do określonych rozmiarów i podawane do sieci konwolucyjnej, która miała za zadanie wydobyć cechy z każdego z regionów w postaci wielowymiarowego wektora. Ten wektor natomiast stanowił wejście dla dwóch ostatnich elementów architektury - maszyn wektorów nośnych (SVM) dokonujących klasyfikacji cech obrazu (decydujących o występowaniu obiektów w regionach) oraz do sieci dokonującej regresji liniowej w celu otrzymania wymiarów obwiedni.

Z modelem tym związane były problemy wynikające z decyzji projektowych. Dużo czasu zajmowało generowanie 2000 propozycji regionów na obrazie, przez co model nie był gotowy do działania w czasie rzeczywistym - przetwarzał obraz w czasie ok. 47 sekund. Dodatkowo algorytm przeszukiwania selektywnego był niezmiennym algorytmem - nie był trenowany, więc mógł generować niepoprawne propozycje. W trakcie uczenia R-CNN proces przetwarzania był trzyetapowy - najpierw sieć konwolucyjna miała za zadanie produkować wektor cech regionów, następnie model SVM wykorzystywał ten wektor w celu klasyfikacji, a dopiero na końcu wykonywana była regresja parametrów obwiedni.



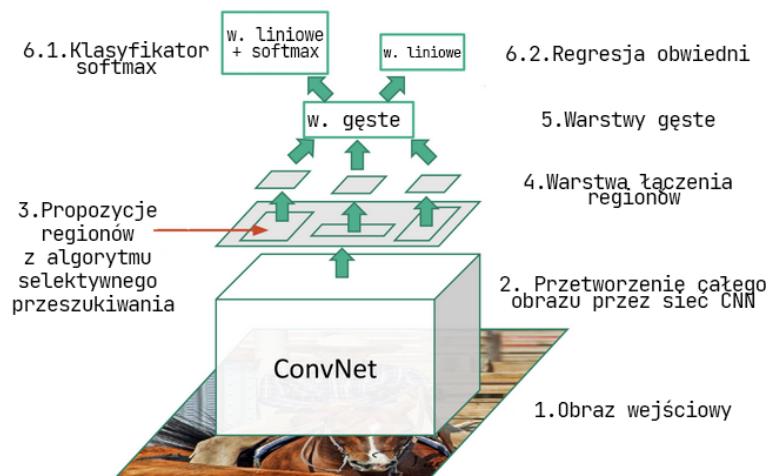
Rysunek 2.7. Schemat działania modelu R-CNN.

2.2.3. Architektura Fast R-CNN

Model Fast R-CNN stanowił bezpośrednią następcę modelu R-CNN i został wprowadzony w artykule [13], a główne jego innowacje związane były z polepszeniem szybkości treningu i inferencji architektury oraz zwiększeniem skuteczności jej detekcji. Rewolucyjnym rozwiązaniem okazał się pomysł zastosowania tylko raz sieci konwolucyjnej - do wydobycia cech z całego obrazu i podzielenia się tymi cechami dla każdego regionu proponowanego. Tak więc model otrzymywał na wejściu cały obrazek wraz z propozycjami regionów i przetwarzał go całego raz, przy użyciu zestawu warstw konwolucyjnych (CNN) oraz warstw łączenia maksymalnego (ang. max pooling) - w ten sposób wydobywane były cechy obrazu. Następnie wykorzystywana była warstwa łączenia regionów zainteresowania (RoI - z ang. Region of Interest), która produkowała wektor cech obrazu związanych z proponowanymi regionami na obrazie. Na końcu, każdy z tych wektorów stanowił wejście dla zestawu warstw gęstych, które rozgałęziały się na dwie części: część klasyfikującą (estymującą prawdopodobieństwa konkretnych kategorii lub ich braku dla obiektu w tym regionie) oraz część regresyjną (estymującą wartości parametrów tworzących obwiednię dla obiektu w tym regionie). Od tamtego momentu, część neuronową tej architektury nazwano siecią detekcji (ang. detection network).

Fast R-CNN okazał się lepszym modelem od swojego poprzednika - osiągał on lepsze wyniki pod względem metryki *mAP* na popularnych zbiorach benchmarkowych, trening odbywał się w jednym kroku i przy użyciu straty łączącej dwa zadania (klasyfikację obiektów i regresję ich obwiedni) naraz, a także trening mógł aktualizować wagi całej architektury (poza częścią proponującą regiony). Model R-CNN przebitý został przez tę architekturę również pod względem szybkości działania - czas przetwarzania obrazów przez sieć neuronową skrócił się do ok. 0,3 sekundy, natomiast pozostała wciąż konieczność korzystania z algorytmu przeszukiwania selektywnego do wydobywania propozycji regionów.

Architektura Fast R-CNN



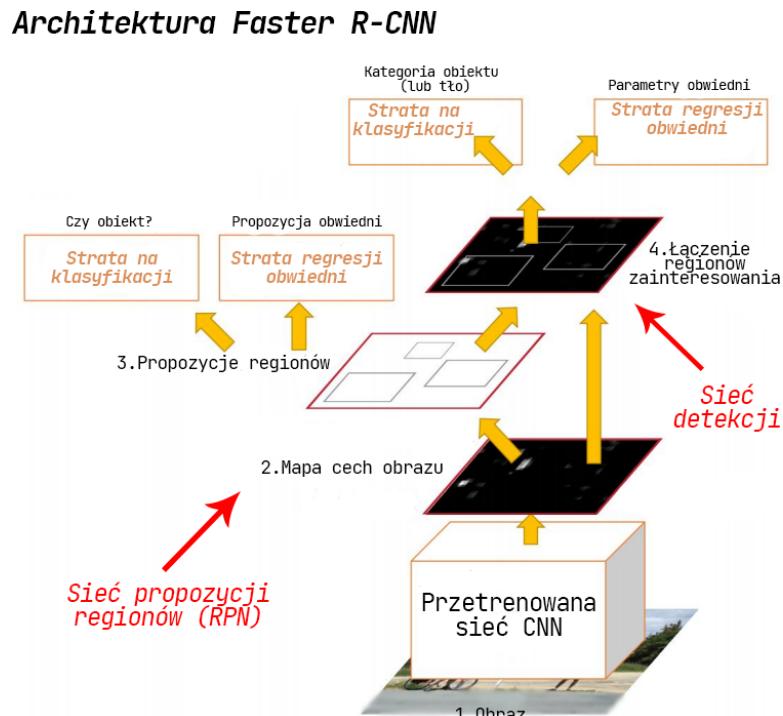
Rysunek 2.8. Schemat działania modelu Fast R-CNN.

2.2.4. Architektura Faster R-CNN

Przełomowym rozwiązaniem w detekcji obiektów okazał się model Faster-RCNN, wprowadzony w [14], któremu udało się rozwiązać większość problemów architektury Fast R-CNN. Wprowadził on przede wszystkim Sieć Propozycji Regionów (RPN - z ang. Region Proposal Network), która zastąpiła algorytm przeszukiwania selektywnego. Jej głównym atutem był fakt, że stanowiła również konwolucyjną sieć neuronową z warstwami gęstymi i zaprojektowana została do działania na całym obrazie, przez co mogła dzielić wszystkie konwolucyjne cechy obrazu z siecią detekcji, w ten sposób zapewniając niemal bezkosztowną obliczeniowo propozycję regionów. Sieć RPN, na podstawie cech obrazu, równocześnie miała za zadanie przewidywać wstępne obwiednie obiektów i informację o tym, czy obiekt jest w danym regionie (bez rozróżnienia na kategorie) i uczona była po to, aby jak najlepiej wskazywać propozycje regionów, które potem zostaną wykorzystane przez model detekcji - dokładnie tak samo, jak w modelu Fast R-CNN.

Sieć propozycji regionów, po wydobyciu cech z obrazu przy użyciu warstw konwolucyjnych, stosowała zredukowaną wersję algorytmu przesuwnego okna, gdzie dla k ustalonych kotwic (ang. anchors) w mapie cech (punktów na niej), wybierała wstępne regiony (o różnych skalach i proporcjach), aby dla każdego takiego regionu wydobyć informację o tym, czy reprezentuje on obiekt i jak tak, to jakie współrzędne wstępne ma obwiednia w nim się zawierająca. Następnie, tak samo jak w Fast R-CNN, proponowane regiony nakładane były na mapę cech obrazu przy użyciu warstwy łączenia regionów oraz spłaszczane do wektorów reprezentujących cechy regionów. Wektor ten następnie stanowił wejście dla dwóch warstw gęstych, gdzie pierwsza z nich odpowiada za wyznaczenie klasy dla obiektu (lub stwierdzeniu, że obiekt tam nie występuje), a druga wyznaczała ostateczne parametry obwiedni dla obiektów. Jak można zauważyć, architektura w trakcie swojego treningu optymalizowała funkcję straty składającą się na 4 zadania - binarną klasyfikację i regresję dla części proponującej regiony oraz klasyfikację i regresję dla części działającej na cechach tych regionów. Istotnym elementem w projektowaniu architektury był również dobór jej szkieletu - części odpowiadającej za ekstrakcję cech z całego obrazu. Dwoma najpopularniejszymi podejściami było wykorzystanie części konwolucyjnych (czyli bez warstw gęstych) z architektur ResNet-101 lub VGG-16.

Architektura Faster R-CNN pokonała model Fast R-CNN pod względem miary mAP na najpopularniejszych zbiorach do detekcji obiektów, a dodatkowo okazała się znacznie szybszym rozwiązaniem - potrafiła przetwarzać obrazy z szybkością 200ms.



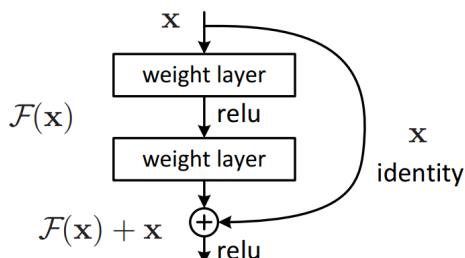
Rysunek 2.9. Schemat działania modelu Faster R-CNN.

2.2.5. Architektura Mask R-CNN

Model Mask R-CNN, zaproponowany w [15], zaprojektowany został jako następca Faster R-CNN, jednak realizuje on nieco innego zadanie niż detekcja obiektów, a mianowicie segmentację obrazów, polegającą na dzieleniu obrazu na segmenty - zbiory pikseli posiadających cechy wspólne. Zadanie to wydaje się nieco trudniejsze niż detekcja, ponieważ granice obiektów nie muszą być teraz prostokątami, a dowolnymi figurami. Mask R-CNN stanowi rozwinięcie modelu Faster R-CNN o dodatkową gałąź poza tymi do klasyfikacji i regresji - warstwy neuronowe przewidujące maski obiektów - zbiory pikseli z obrazu opisujące obiekty na nim się znajdujące.

2.3. Sieć rezydualna

Sieć rezydualna - ResNet (ang. Residual Neural Network, wprowadzona w [16]) - stanowi jedną z najpopularniejszych architektur sieci neuronowych w wizji maszynowej. Swój sukces zawdzięcza opieraniu się na tzw. blokach rezydualnych, w których do wyjścia produkowanego przez standardowe warstwy konwolucyjne dodawane jest wejście do całego bloku. Rozwiążanie to zapewnia, że możliwe jest łączenie bloków rezydualnych ze sobą w długie sekwencje przy jednoczesnym niepogarszaniu skuteczności takiego modelu, ponieważ jeśli blok nie ma czego się uczyć, realizuje on przekształcenie tożsamościowe i przeprowadza swoje wejście do kolejnego bloku. Schemat bloku rezydualnego przedstawiony został na rysunku 2.10.



Rysunek 2.10. Schemat bloku rezydualnego, gdzie do wyniku operacji warstw splotowych dodawane jest przekształcenie tożsamościowe wejścia do bloku. Źródło rysunku: [16].

Sieci rezydualne wykorzystywane są bardzo często jako szkielety dużych architektur neuronowych. Stosowany jest wtedy transfer uczenia (ang. transfer learning), który polega na skorzystaniu z wytrenowanej już architektury neuronowej na innym zadaniu i zbiorze danych, wycięciu z niej warstw końcowych (tzw. warstw gęstych) i wykorzystaniu jedynie tych, które są odpowiedzialne za wydobywanie cech z obrazu. Do warstw tych dołączane są następnie własne warstwy gęste w celu wytrenowania ostatecznego modelu na własnym zadaniu. Proces ten często zapewnia szybszy trening sieci i jest niezwykle popularnym działaniem w głębokim uczeniu.

2.4. Mechanizm atencji

Niezbędnym mechanizmem użytym w proponowanej architekturze, który może zapewnić jej skuteczność, jest znany i powszechnie wykorzystywany w sieciach neuronowych mechanizm atencji. Jak się okazuje, jego idea wywodzi się z neuronauki.

2.4.1. Atencja jako mechanizm pochodzący z neuronauki

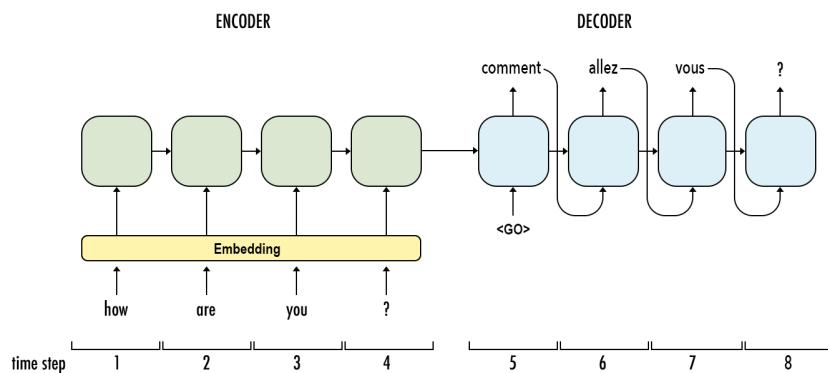
Zagłębiając się w tematyce sztucznych sieci neuronowych, można zauważać, że wiele pomysłów i definicji z tej dziedziny wywodzi się z tego, jak funkcjonuje i reaguje ludzki mózg. Człowiek, aby wykonać zadanie jak najlepiej, potrzebuje skupić się na celu, jaki ma do zrealizowania, co robi odrzucając nieinteresujące dla siebie bodźce zewnętrzne na rzecz tych, które mogą mu pomóc zrealizować to zadanie. Przykładowo, gdy chce przeczytać tekst i go jak najlepiej zrozumieć, musi odrzucić wszystkie czynniki, które go rozpraszały, takie jak słuchanie muzyki czy rozmawianie z inną osobą, a skoncentrować swój wzrok i mózg na przetwarzaniu informacji zawartej w tekście, czyli - inaczej mówiąc - skupić swoją uwagę na konkretnej części z czynników zewnętrznych, jakie do niego docierają. Dokładnie takie zachowanie ludzkiego mózgu można nazwać uwagą, natomiast słowo "atencja" jest kalką językową angielskiego tłumaczenia uwagi (attention).

2.4.2. Wykorzystanie atencji przez sieci neuronowe

Badacze na świecie zdołali zastosować atencję w sztucznych sieciach neuronowych i, jak się okazało, rozwiążanie te stało się przełomowe. W głębokim uczeniu mechanizm ten ma za zadanie wspomagać sieć neuronową w działaniu poprzez proponowanie jedynie takich fragmentów z danych wejściowych, które z jej punktu widzenia są istotne (w przypadku opisywanego w tym artykule zadania detekcji - konkretne obiekty na obrazie), a odrzucenie tych, które nie powinny wpływać na jej decyzję (przykładowo tło zdjęcia).

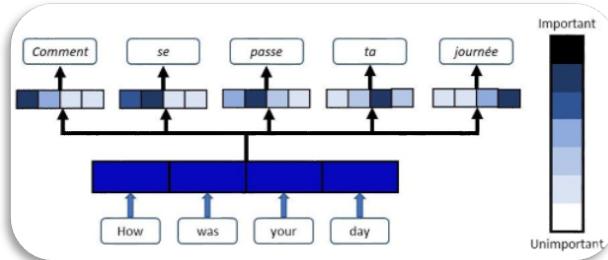
Pojęcie to pierwszy raz ukazało się w przypadku przetwarzania języka naturalnego, w artykule [17], gdzie szybko stało się (na tamten moment) najlepszym rozwiążaniem w zadaniu tłumaczenia maszynowego. Przed atencją, tłumaczenie maszynowe realizowane było przy użyciu sieci rekurencyjnych - architektura ta składała się na takie sieci neuronowe, które przyjmują wejście jako sekwencję danych (w tym przypadku słów w zdaniu) i w pętli produkują słowa zdania wyjściowego (czyli również sekwencję). Takie rozwiązanie określa się mianem autokodera, który składa się na dwie części: koder (ang. encoder) – rekurencyjną sieć neuronową, która nie ma żadnego wyjścia, ma tylko swój stan. Takiej sieci podawane są po kolejne słowa ze zdania źródłowego - oraz dekoder (ang. decoder) – rekurencyjną sieć neuronową produkującą kolejne słowa zdania wyjściowego (wektory), która na samym końcu produkuje wyraz oznaczający koniec zdania (często oznaczany jako EOS - end of sequence).

Aby sieć rekurencyjna w trakcie przetwarzania swojego zdania kumulowała информацию, jaką do tej pory zebrała, w kolejnych iteracjach pętli przekazuje one sobie swój aktualny stan reprezentujący to, co przetworzyła do tej pory. Relacja między koderem a dekoderem w opisywanej architekturze jest taka, że końcowy stan kodera to tak naprawdę początekowy stan dekodera, co oznacza, że ogólną rolę odgrywa tutaj pierwsza z części architektury - kodująca - ponieważ musi ona zamienić zdanie o potencjalnie dowolnej długości (jako liczby słów) w wektor uchwycający cały jego sens. Dodatkowo drugi problem, z jakim mierzy się to rozwiązanie, jest taki, że rola modułu dekodującego zdanie na drugi język jest nieco utrudniona - zwykle gdy człowiek tłumaczy zdanie między językami, robi to fragmentami, ponieważ, przykładowo, nie ma wpływu na tłumaczenie pierwszych słów zdania to, jakie słowo jest na jego końcu - tutaj natomiast niezależnie od miejsca w zdaniu wyjściowym, w którym znajduje się dekoder, ma on dostęp do pełnego zdania tłumaczonego.



Rysunek 2.11. Schemat działania architektury autokodera z sieciami rekurencyjnymi. Można zauważyć, że na początku działa tutaj koder przechodzący po całym zdaniu tłumaczonym i produkujący ostatecznie pojedyncze wyjście - stan po przetworzeniu całego zdania, które otrzymuje dekoder i iteracyjnie produkuje zdanie będące tłumaczeniem. Źródło: [18].

Wyżej wymienione problemy okazały się kluczowe i wymagały poprawy, aby zadanie tłumaczenia maszynowego było lepiej realizowane przez sieci neuronowe. Przełomowym rozwiązaniem okazały się modele z atencją, które w tłumaczeniu maszynowym wprowadziły dwie dodatkowe modyfikacje do poprzedniego rozwiązania. Pierwszą z nich była zmiana sposobu reprezentacji zdania tłumaczonego po przejściu przez koder - zamiast jednego wektora reprezentującego całe zdanie, od teraz istnieje N wektorów reprezentujących N słów wejściowych (tzw. adnotacje). Natomiast druga ze zmian to wprowadzanie mechanizmu skupiania uwagi dekodera na jedynie istotnych fragmentach (słowa) zdania tłumaczonego w trakcie produkowania pojedynczego słowa zdania wyjściowego, co realizowane jest za pomocą modułu, który uczy się jak ważyć wcześniej wyznaczane przez koder adnotacje, by tłumaczone słowo miało jak największy sens. Rozwiązanie to okazało się przełomowe i przez pewien okres zawładnęło tłumaczeniem maszynowym.



Rysunek 2.12. Schemat działania architektury autokodera z atencją. Każde słowo zdania wejściowego przekształcane jest na adnotacje, a te, przy każdej generacji dekodera, są odpowiednio ważone, tak aby tłumaczenie danego słowa zależało jedynie od fragmentów wejścia. Źródło: [19].

Innym przykładem, gdzie zastosowana została atencja, jest wizja maszynowa, a konkretnie, w artykule [20], zadanie opisywania zdjęć (ang. image captioning), polegające na wyprodukowaniu zdania opisującego to, co znajduje się na obrazie. W tym celu konieczne jest odpowiednie przetworzenie obrazu do zagnieżdżenia - wektora reprezentującego go w mniejszej przestrzeni, a następnie zastosowanie na takim wektorze dekodera, który - podobnie jak w tłumaczeniu maszynowym - iteracyjnie generuje słowa opisu. Wprowadzenie atencji do modelu ma podobną rolę jak poprzednio - zamiast generować każdy wyraz zdania na podstawie całego obrazu, można ułatwić sieci dekodującą zadanie, podając jej jedynie fragmenty obrazu, które w danym momencie pętli ma on opisać. W przypadku tego zadania atencja również okazała się rozwiązaniem poprawiającym skuteczność.



Rysunek 2.13. Przykłady zachowania oczekiwanej od mechanizmu atencji w zadaniu generowania opisów do zdjęć - produkując konkretne słowo opisu, dekoder powinien zwracać uwagę na te fragmenty obrazu, które są w danej chwili istotne. Źródło: [21].

2.4.3. Matematyczna teoria atencji

Aby atencja była mechanizmem uczonym w trakcie treningu sieci neuronowej, konieczne jest, aby proces ten posiadał matematyczny zestaw obliczeń go opisujących. Skupienie uwagi to system stanowiący funkcję całego otrzymanego przebiegu danych, gdzie większe wagi przykładowe są do tych fragmentów przebiegu, które faktycznie determinują i opisują oczekiwane wyjście. W tym celu konieczne jest odpowiednie podzielenie całego przebiegu danych na skończoną liczbę fragmentów $h_{1..T}$. Fragment taki może reprezentować, przykładowo w tłumaczeniu maszynowym, pojedyncze słowo, które uległo przetworzeniu przez inną część sieci i stało się adnotacją lub, przykładowo w wizji maszynowej, cechy na obrazie (w postaci zagnieżdżeń), które wykryły filtry sieci konwolucyjnej.

Do fragmentów takich, gdy jest to konieczne, dodawane są znaczniki czasowe reprezentujące ich pozycję w sekwencji danych - szczególnie w przypadku zadań, gdzie kolejność poszczególnych wejść ma znaczenie (jak przykładowo kolejność i szyk słów w zdaniu) i oczekiwane jest od atencji, aby dobrze spozycjonowała sobie te fragmenty w czasie. Znaczniki czasowe wyznaczane są jako funkcje sinusa i cosinusa dla argumentu będącego pozycją fragmentu w ciągu danych podzieloną przez pewną liczbę p , gdzie obliczenie tych wartości dla kilku różnych co do wartości p pozwala wyeliminować problem okresowości tych funkcji trygonometrycznych.

Następnie, atencja ma za zadanie, pracując na fragmentach w postaci adnotacji/zagnieżdżeń, dla każdego z nich wyznaczyć skalar - liczbę mówiącą jaka uwaga powinna zostać przywiązana do tego fragmentu. Realizowane jest to przez acykliczną sieć neuronową b , która poza ciągiem fragmentów otrzymuje również informację o stanie działania ostatniej części architektury - dekodera - v (co zapewnia, że różne wagi są przykładowane w różnych momentach produkowania zdania wyjściowego). Sieć produkuje uwagę dla wszystkich fragmentów jako wektor liczb o długości liczby fragmentów jako:

$$\vec{a}_{1..T} = b(h_{1..T}; v) \quad (1)$$

Na takiej informacji przeprowadzana jest operacja, znana w uczeniu maszynowym jako softmax, która otrzymuje wektor składający się z liczb rzeczywistych $\vec{a}_{1..T}$ i normalizuje go do rozkładu prawdopodobieństwa składającego się z tylu samu liczb-prawdopodobieństw, proporcjonalnych do wykładników liczb wejściowych. Po zastosowaniu funkcji softmax, każda składowa wektora zaczyna znajdować się w przedziale $(0, 1)$, składowe te sumują się do liczby 1, a także większe składowe wejściowe odpowiadają większym prawdopodobieństwom. W ten sposób obliczane są wagi dla poszczególnych fragmentów, na których pracuje atencja, przykładowo dla i -tego fragmentu waga wynosi:

$$w_i = \frac{\exp a_i}{\sum_{j=1}^T \exp a_j} \quad (2)$$

W ostatnim kroku następuje suma ważona wszystkich fragmentów $\vec{h}_{1..T}$ zgodnie z wagami $\vec{w}_{1..T}$ wyznaczonymi przez atencję we wzorze (2):

$$\vec{o} = \sum_{i=1}^T \vec{h}_i \cdot w_i \quad (3)$$

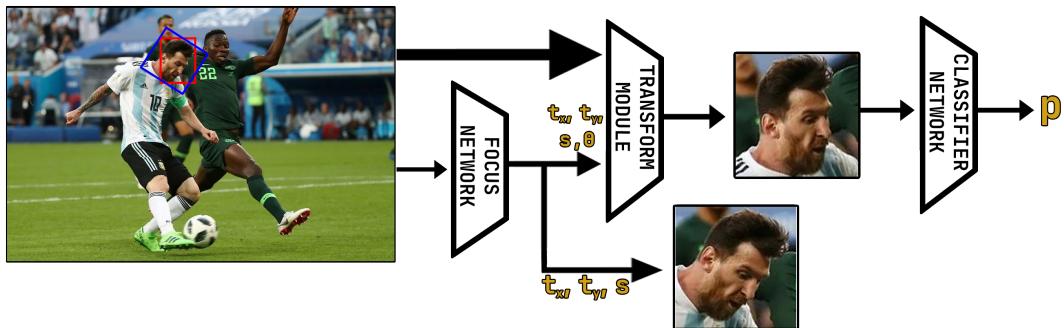
W ten sposób otrzymywane jest ostateczne zagnieżdżenie - \vec{o} - reprezentujące naraz wszystkie fragmenty sieci, do którego kontrybuują konkretne fragmenty zgodnie z tym, za jak istotne zostały uznane przez atencję.

3. Proponowane rozwiązanie

W niniejszym rozdziale zawarty został opis, przedstawiający teoretyczny opis proponowanej architektury do detekcji obiektów - sieci Focus-CNN. Wszelkie aspekty implementacyjne zostały tutaj pominięte i opisane w późniejszej części pracy.

3.1. Ogólna architektura sieci

Proponowana architektura rozwiązania dzieli się na trzy moduły (w tym dwie głębokie sieci neuronowe) spełniające trzy odrębne zadania. Na początku, do pierwszego z nich trafia obraz z obiektem, a następnie każdy z modułów odgrywa swoją rolę - wyjście z jednego stanowi pełne lub pośrednie wejście do kolejnego - do momentu otrzymania współrzędnych kwadratowych ramek otaczających obiekty interesujących klas na obrazie.



Rysunek 3.1. Grafika przedstawiająca schemat działania całej architektury z uwzględnieniem wejść i wyjść każdej części. Na obrazie wejściowym (na lewo) prostokąt czerwony oznacza obwiednię zaanotowaną w ramach zbioru, a niebieski pokazuje oczekiwane przez klasyfikator przybliżenie obiektu. Zauważalna jest różnica między wejściem do klasyfikatora a obrazem wyciętym przez kwadrat będący wyjściem architektury - rotacja używana jest do wspomagania klasyfikatora w jego decyzji i nie ma zastosowania przy ustalaniu ostatecznych koordynatów obiektu.

Pierwszy z modułów - sieć neuronowa Focus (skupiająca) - istnieje osobno dla każdej klasy, na której wykonywana jest detekcja, tzn. jeśli uznajemy, że nasza architektura ma wykrywać obiekty 3 klas na obrazach, to w architekturze istnieje 3 niezależnych sieci Focus poświęconych dla każdej z klas osobno. Pojedyncza taka sieć ma za zadanie stwierdzić, z jakim prawdopodobieństwem obiekt danej klasy występuje na obrazie i, jeśli taki jest, zlokalizować go.

Drugi z nich - moduł Transform (transformujący) - wykorzystuje zwarcane przez sieć Focus liczby do skonstruowania macierzy przekształcenia geometrycznego i przekształca duży obraz (będący wejściem do architektury) na obraz mały, będący przybliżeniem na konkretny obiekt danej kategorii, który został wykryty.

Na obrazie będącym tym przybliżeniem, dokonywana jest klasyfikacja przy użyciu wieloetykietowego klasyfikatora - sieci Classifier - i wynik zwracany przez tę sieć jest ostatecznym stwierdzeniem, czy w danym miejscu obrazu benchmarkowego znajduje się obiekt wybranej klasy.

Jeśli ostateczne prawdopodobieństwo występowania obiektu jest wystarczająco duże, wtedy wykorzystywane są liczby lokalizujące obiekt (generowane przez sieć Focus), następnie odpowiednio przekształcane na kwadrat pełniący funkcję obwiedni obiektu i w takiej formie uznawane za ostateczną lokalizację obiektu.

3.2. Sieć skupiająca - Focus Network

3.2.1. Ogólny opis

Sieć Focus - zwana również Siecią Skupiającą - stanowi pierwszą i najważniejszą część proponowanej architektury. Jej rola może nieco przypominać rolę Sieci Propozycji Regionów (ang. RPN - Region Proposal Network), znaną z opisywanych wcześniej architektur. Jej głównym zadaniem jest lokalizowanie i wskazywanie miejsca, gdzie obiekt danej klasy się znajduje. Pływąca z jej wyjścia informacja jest wykorzystywana później przez moduł transformujący w celu odpowiedniego przekształcenia obrazu (na taki, który jak najlepiej wskazuje obiekt) i przekazania go do sieci klasyfikującej.

Stąd też pierwszą informacją zwracaną przez sieć jest prawdopodobieństwo p , że obiekt danej klasy rzeczywiście jest na obrazie. Informacja ta pełni jedynie rolę analityczną, tzn. na ten moment nie jest wykorzystywana w procesie inferencji sieci. Liczba ta jest z zakresu $[0, 1]$, gdzie liczba bliska 0 oznacza brak obiektu danej klasy na obrazie, a bliska 1 - dużą pewność sieci co do występowania obiektu konkretnej kategorii na obrazie. Pozostałe wyjścia sieci stanowią już informację lokalizującą obiekt, a dokładniej najmniejszy kwadrat zawierający w pełni obiekt w swoim centrum, a jest to:

- t_x - translacja wzduż osi OX - stanowi informację, jak należy obraz przesunąć w poziomie, aby dostać się do centrum obiektu;
- t_y - translacja wzduż osi OY - stanowi informację, jak należy obraz przesunąć w pionie, aby dostać się do centrum obiektu;
- $\log s$ - skala (a dokładniej jej logarytm naturalny, którego łatwiej jest się uczyć sieci), stwierdzająca jak należy przybliżyć obraz, aby został on wypełniony przez obiekt;
- θ - rotacja - mówiąca, o ile radianów należy przekręcić obraz, aby obiekt znajdował się w pozycji prostej.

3.2.2. Działanie sieci

Sieć zobowiązana jest zwracać jak najwyższe prawdopodobieństwo występowania na zdjęciu obiektu danej kategorii i wskazywać najmniejszy możliwy kwadrat, który zawiera w pełni obwiednię obiektu. Sieć otrzymuje również obrazy niezawierające ani jednego obiektu o kategorii mu przypisanej - wtedy powinna zwracać jak najbliższe 0 prawdopodobieństwa (pozostałe zwracane scalary nie mają tu znaczenia). Uwaga: w niniejszym rozdziale, przy opisywaniu matematycznych operacji wykonywanych na obrazie, obowiązuje, charakterystyczny dla wizji maszynowej, lewoskrętny kartezjański układ współrzędnych, gdzie punkt $(0, 0)$ znajduje się w górnej lewej części obrazu i wartości na osi OY rosną w kierunku dolnym - założenie to pozwala zastosować w opisach dokładnie te same wzory, które istnieją bezpośrednio w implementacji, co może ułatwić analizę kodu stworzonego razem z pracą.

Sieć zwraca scalary transformacji, które dla obrazu i oraz obwiedni j obiektu, który jest obrócony o kąt θ , obliczane są według wzorów:

- **translacja x oraz translacja y** (konwersja bazująca na różnicę w pozycjach między środkiem obrazu oryginalnego S_i a środkiem prostokąta otaczającego obiekt S_j i normalizacji do zakresu $(-1, 1)$ poprzez podzielenie przez szerokość w_i lub wysokość h_i obrazu, gdzie 0 oznaczałoby obiekt o centrum na środku obrazu):

$$t_{x_j} = 2 \times \frac{(x_{S_j} - x_{S_i})}{w_i} \quad t_{y_j} = 2 \times \frac{(y_{S_j} - y_{S_i})}{h_i} \quad (4)$$

- **logarytm skali s** (konwersja bazująca na stosunku wielkości obiektu j do wielkości całego obrazu i logarytmizacja w celu ułatwienia treningu sieci):

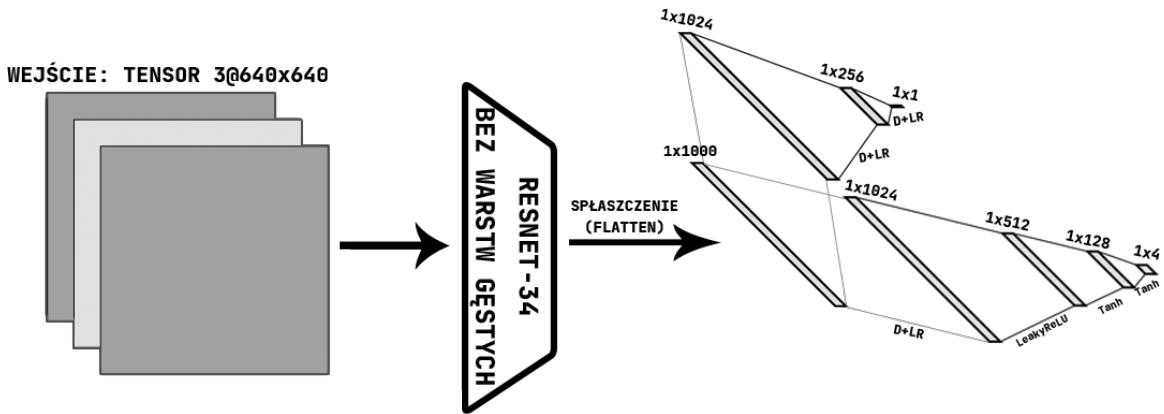
$$\log s_j = \log \frac{\max\{w_j, h_j\}}{w_i} \quad (5)$$

- **rotacja θ'** (konwersja bazująca na przekształcaniu ze stopni do radianów kąta przeciwnego do kąta θ - tego, o który obrócono cały obraz):

$$\theta'_j = -\theta \times \frac{\pi}{180} \quad (6)$$

3.2.3. Architektura sieci

Struktura sieci dzieli się na trzy części, gdzie pierwszą z nich stanowi szkielet (ang. backbone) oparty na opisanej w rozdziale 2.3 sieci rezydualnej. Został tu wykorzystany transfer uczenia modelu ResNet-34, z którego wzięto warstwy ekstrakcji cech z obrazu (warstwy konwolucyjne) i połączono zarówno z zestawem warstw "w pełni połączonych" neuronów (ang. fully-connected layers) realizujących klasyfikację, jak i z zestawem warstw realizujących lokalizację, co najlepiej obrazuje rysunek 3.2.



Rysunek 3.2. Schemat architektury sieci skupiającej, gdzie obraz wejściowy to kwadratowy obraz RGB (3-kanałowy) o wymiarach 640×640 , który ulega zagnieźdzeniu przez warstwy konwolucyjne sieci rezydualnej do wektora, który stanowi wejście do zestawu warstw gęstych przewidujących prawdopodobieństwo wystąpienia obiektu oraz do zestawu warstw gęstych przewidujących skalary transformacji. W ramach rysunku pominięto budowę sieci ResNet-34, składającej się z 34 bloków rezydualnych, których działanie zostało opisane w 2.3. Skrót "D+LR" na połączeniach między warstwami oznacza, że bezpośrednio za warstwą gęstą występuje w architekturze warstwa Dropout oraz aktywacja przy użyciu funkcji LeakyReLU.

3.2.4. Funkcja straty

Jak już zostało to wspomniane w podrozdziale 2.1.2, problem detekcji obiektów wymaga implementacji szczególnego rodzaju funkcji straty (ang. loss function) w procesie optymalizacji modelu (jego treningu) - straty składającej się na część klasyfikującą (do zadania wyznaczania kategorii) i część regresyjną (tutaj do wyznaczania parametrów transformacji). W proponowanym podejściu zdecydowano się na wykorzystanie i niewielkie zmodyfikowanie funkcji użytych w architekturach Fast-RCNN oraz Faster-RCNN.

Proponowana w tej pracy strata - **Focus Loss** - jest ważoną kombinacją czterech niezależnych strat - straty klasyfikującej (\mathcal{L}_{cls}), straty translacyjnej (\mathcal{L}_t), straty skali (\mathcal{L}_s) oraz straty rotacyjnej (\mathcal{L}_{rot}) z wagami stanowiącymi o stopniu kontrybucji w stracie ostateczną (kolejno: w_{cls} , w_t , w_s , w_{rot}). Dla pojedynczego przykładu, gdzie p^* oznacza docelową klasę obiektu, p - pewność modelu co do występowania obiektu, $t^* = (t_x^*, t_y^*)$ - wektor docelowych translacji, $t = (t_x, t_y)$ - wektor przewidywanych przez model translacji, s^* - docelowy logarytm skali, s - przewidywany przez model logarytm skali, θ^* - docelowy kąt obrotu oraz θ - przewidywany kąt obrotu, finalna strata \mathcal{L} prezentuje się następująco:

$$\mathcal{L} = w_{cls}\mathcal{L}_{cls} + w_{obj}\mathcal{L}_t + w_s\mathcal{L}_s + w_{rot}\mathcal{L}_{rot} \quad (7)$$

$$\mathcal{L}_{cls}(p, p^*) = \text{CrossEntropyLoss}(p, p^*) \quad (8)$$

$$\mathcal{L}_t(t, t^*, p^*) = p^*|t^* - t| \quad (9)$$

$$\mathcal{L}_s(s, s^*, p^*) = p^*|s^* - s| \quad (10)$$

$$\mathcal{L}_{rot}(\theta, \theta^*, p^*) = p^*|\theta^* - \theta| \quad (11)$$

3. Proponowane rozwiązanie

W powyższych wzorach istotne jest to, że w obliczaniu straty na skalarach transformacji, ważną rolę odgrywa docelowa klasa obiektu - można zauważyć, że w przypadku, gdy obiekt do wykrycia nie istnieje (wykryte powinno zostać tło), wtedy strata na skalarach transformacji nie powinna być liczona, ponieważ nie ma ona znaczenia - stąd też strata ta jest zerowana, gdy na obrazie nie występuje pożądany obiekt.

Przy obliczaniu straty klasyfikacyjnej, wykorzystywana jest, często używana w klasyfikacji, strata Entropii Krzyżowej (ang. Cross Entropy Loss), która stosowana jest do oceny nietrafności jednego rozkładu prawdopodobieństwa względem drugiego:

$$\text{CrossEntropyLoss}(p, p^*) = -p^* \cdot \log(p) - (1 - p^*) \cdot \log(1 - p) \quad (12)$$

W obliczaniu straty regresyjnej, dla każdego z parametrów transformacji, wykorzystana została tzw. strata L_1 , inaczej zwana Średnim Błędem Bezwględnym (MAE) ze względu na to, że lepiej radzi sobie z małymi ułamkami niż jej bezpośrednia konkurentka do tego zastosowania - strata L_2 , inaczej zwana Średnim Błędem Kwadratowym (MSE).

3.3. Moduł Transformujący - Transform Module

Na początku tego rozdziału konieczne jest odwołanie się do nazewnictwa zawartego w tej pracy - omawiana tutaj część architektury - Moduł Transformujący - nienazwana została siecią, ponieważ z praktycznego punktu widzenia, nie jest ona Siecią Neuronową, gdyż nie posiada ona neuronów - wag, które są optymalizowane. To, co łączy ten moduł z klasyczną Siecią Neuronową to wymóg pełnej różniczkowalności operacji, jakie ona wykonuje. Cechą ta wykorzystywana jest przez architekturę podczas procesu dostrajania architektury (ang. fine-tuning), gdy istotne jest nauczenie sieci Focus wskazywać regiony tak, aby sieć klasyfikująca z jak największą pewnością dokonywała decyzji. Widać więc, że zadanie, jakie realizuje ta część architektury, nie jest proste, ponieważ nie dość, że dokonywana musi być różniczkowalna transformacja afinczna (ang. affine transform), to jeszcze gradient musi być obliczony względem parametrów transformacji - translacji, skali oraz rotacji.

Ze względu na to, że sama implementacja Modułu Transformującego, jak i przetestowanie implementacji, są dosyć złożonymi i istotnymi aspektami pracy, poświęcony dla nich został osobny rozdział pracy - 4.1, natomiast w tym rozdziale zamieszczony został teoretyczny opis zadania, jaki ten moduł realizuje.

Działanie tej części architektury jest działaniem znanym z Grafiki Komputerowej jako przekształcenie geometryczne obrazu. Najprostszym rodzajem przekształcenia geometrycznego jest przekształcenie elementarne, mające na celu transformację obrazu poprzez jego przemieszczenie (translację), przybliżenie/oddalenie (skalę), obrót (rotację) lub pochylenie, w celu otrzymania drugiego obrazu. Bardziej ogólnym pojęciem, rozszerzającym przekształcenie elementarne jest **przekształcenie afiniczne** - do nich należą przekształcenia geometryczne, umiejscowiące zachowywać m.in. stosunki odległości między punktami czy równoległości prostych, czyli zarówno pojedyncze przekształcenia elementarne, jak i również ich połączenia ze sobą. Dokładnie tego typu przekształcenie wykorzystywane jest przez moduł transformujący - opierające się na połączeniu rotacji, translacji oraz skali.

W celu uzyskania takiego przekształcenia afinycznego konieczne jest utworzenie macierzy transformacji - wymaga ona skorzystania ze współrzędnych jednorodnych, gdzie każdy punkt w 2D przedstawiony jest przy pomocy trójki liczb poprzez dodanie czynnika normalizującego, tzn. dla punktu $P = \begin{bmatrix} x & y \\ w & \end{bmatrix}^T$ przy użyciu współrzędnych kartezjańskich, można go przedstawić jako $P = \begin{bmatrix} x & y & w \end{bmatrix}^T$ przy użyciu współrzędnych jednorodnych. Zadanie utworzenia takiej macierzy jest pierwszym z etapów działania sieci transformującej, wykorzystuje ona podstawowe macierze kolejno: przesunięcia o $P = \begin{bmatrix} d_x & d_y \end{bmatrix}^T$, skalowania przez s i obrotu o kąt θ :

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ w \end{bmatrix} \quad (13)$$

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ w \end{bmatrix} \quad (14)$$

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ w \end{bmatrix} \quad (15)$$

Dokonując składania przekształceń, sieć transformująca otrzymuje macierz wykonującą te trzy przekształcenia jednocześnie:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} s & 0 & 0 \\ 0 & s & 1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow \quad (16)$$

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} s \times \cos\theta & s \times -\sin\theta & d_x \\ s \times \sin\theta & s \times \cos\theta & d_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ w \end{bmatrix} \quad (17)$$

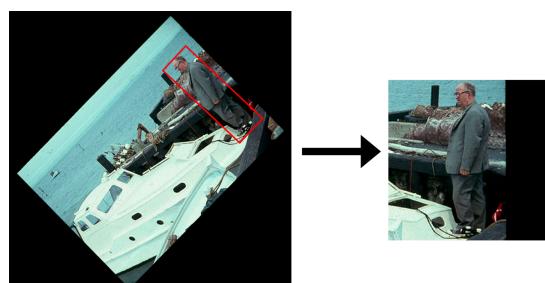
3. Proponowane rozwiązanie

Na końcu, moduł dokonuje przekształcenia aficznego obrazu o skolekcjonowaną uprzednio macierz transformacji. Obraz można interpretować jako siatkę pikseli - punktów posiadających swoje pozycje - tak jak punkty na układzie współrzędnych. Aby przekształcić jedną grafikę w drugą poprzez zastosowanie macierzy transformacji, najpierw tworzony jest obraz rzutowy, a następnie na punkty jego płaszczyzny rzutowane są piksele z obrazu źródłowego. Lokalizacja piksela z obrazu źródłowego na rzutowym otrzymywana jest poprzez przekształcenie jego pozycji w wektor współrzędnych jednorodnych, a następnie przemnożenie przez macierz transformacji. Taka operacja wykonywana jest dla każdego z pikseli obrazu źródłowego, a dzięki temu, że u wszystkich z nich macierz ta jest taka sama, działanie to może być zrównoległe obliczeniowo dla całego obrazu i realizowane przez karty graficzne. Otrzymany obraz rzutowy jest następnie ucinany do zakresu pikseli, które mieścią się w granicach obrazu źródłowego i w ten sposób z tego obrazu i wyjścia sieci Focus otrzymywana jest grafika, stanowiąca wejście dla sieci klasyfikującej.

3.4. Sieć klasyfikująca - Classifier

Sieć ta pełni funkcję wieloetykietowego klasyfikatora neuronowego (ang. multi-label classifier), a jej zadaniem jest wyznaczanie klasy otrzymywanej na wejściu obrazu przedstawiającego przybliżony i potencjalnie obrócony obiekt. Z technicznego punktu widzenia, model zwraca wektor, zawierający prawdopodobieństwa dla każdej z klas. Na podstawie tego wektora, wybierana jest klasa o największym prawdopodobieństwie i, gdy okaże się, że ta klasa zgadza się z kategorią, która jest przypisana konkretnemu modelowi Focus, obiekt uznawany jest za należący do danej klasy.

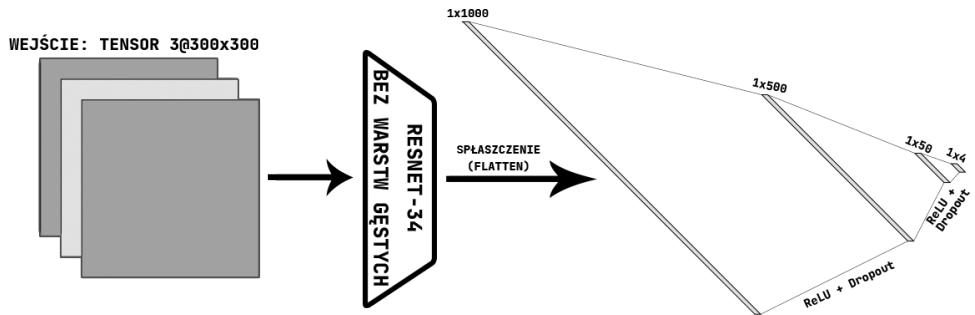
Sieć oczekuje na swoim wejściu tensorów, będących 'wyciętymi', z oryginalnego obrazu, prostokątami reprezentującymi najmniejsze możliwe kwadraty w pełni zawierające obiekty danych klas. Ze względu na fakt, że w środku takiego obrazu wejściowego powinno znajdować się centrum obiektu, którego współczynnik proporcji (ang. aspect ratio) zostaje zachowany, możliwe jest występowanie na obrazach wejściowych do sieci klasyfikującej czarnych pasków reprezentujących marginesy oryginalnego obrazu, co będzie powszechnie w przypadku obiektów występujących przy jego krawędziach. Przykład takiej sytuacji został przedstawiony na rysunku 3.3.



Rysunek 3.3. Przykład sytuacji, gdy niekwadratowy obiekt znajduje się przy krawędzi obrazu.

3.4.1. Architektura sieci

Struktura tej sieci przypominać może architekturę sieci skupiającej. Dzieli się ona na dwie części, gdzie pierwszą z nich stanowi również szkielet w postaci sieci rezydualnej (poprzez transfer uczenia modelu ResNet-34), który ma za zadanie wyekstrahować cechy z obrazu w wektor stanowiący wejście dla zestawu warstw "w pełni połączonych" neuronów.



Rysunek 3.4. Schemat architektury sieci klasyfikującej, gdzie na wejściu otrzymywany jest obraz, a wyjście stanowi 4-elementowy wektor, zwracający prawdopodobieństwo każdej z klas.

3.5. Trening architektury

3.5.1. Trening wstępny sieci Focus

Aby zapewnić jak najlepsze działanie architektury, sieć Focus jest wstępnie trenowana (ang. pretraining). W trakcie tego procesu model jest optymalizowany przy użyciu oryginalnych oraz sztucznie przekreślonych obrazów ze zbioru danych zawierających obiekt danej klasy. Konieczny jest tu odpowiedni preprocessing danych.

3.5.2. Preprocessing danych - sieć Focus

Propozycja w ramach tej pracy informacja zwieracana przez Focus Network ogranicza się do 5 liczb: translacji (t_{x_j}, t_{y_j}) wzdłuż obu osi układu współrzędnych (mówiącej o położeniu obiektu), logarytmu skali $\log s_j$ (określającej wielkość obiektu), rotacji θ_j (mówiącej jak nachylony jest obiekt względem pozycji prostej) oraz prawdopodobieństwa p_j (określającego pewność sieci co do należenia wskazywanego obiektu do danej kategorii).

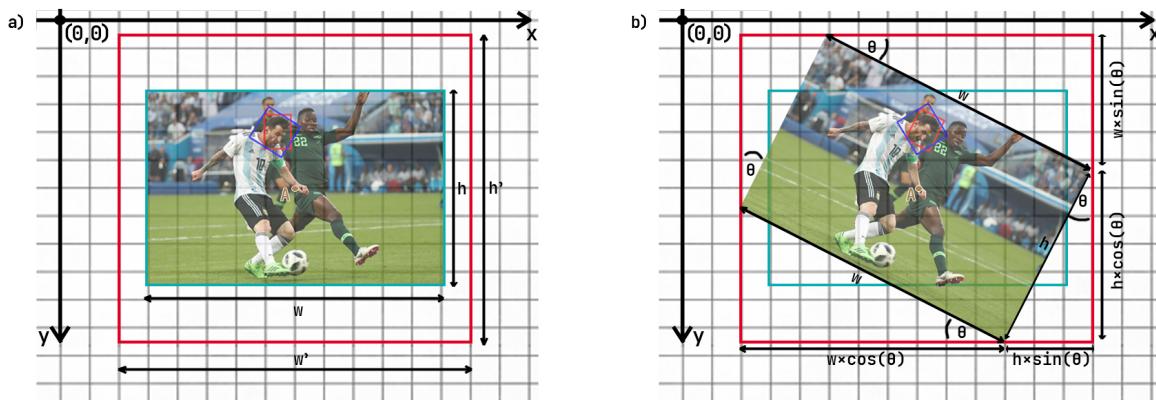
Często jest tak w przypadku zbiorów danych, że obrazy tam zawarte nie są tych samych rozmiarów, co jest problemem dla modeli sieci neuronowych, które zakładają otrzymywanie zawsze tensora tych samych rozmiarów. Aby zapewnić, że model Focus otrzyma zawsze obraz o dobrych wymiarach, zdecydowano się w tej pracy najpierw na sprawdzenie największego możliwego rozmiaru zdjęcia wejściowego ze zbioru benchmarkowego jako (w_{max}, h_{max}) , następnie wzięcie większej z tych liczb: $s_{max} = \max\{w_{max}, h_{max}\}$ i użycie wymiarów (s_{max}, s_{max}) jako wstępnego największego rozmiaru obrazu wejściowego.

3. Proponowane rozwiązanie

Ze względu na dodatkową rotację całych obrazów na tym etapie, konieczne jest obliczenie maksymalnych możliwych wymiarów obrazu po wykonaniu rotacji. Rozmiar prostokąta (w' , h'), który w pełni zawiera obraz obrócony, można otrzymać na podstawie odpowiednich wzorów (gdzie w oznacza szerokość obrazka, h - wysokość obrazka, a θ - kąt obrotu obrazu przy założeniu, że $-90^\circ < \theta < 90^\circ$), których wyprowadzenie zostało przedstawione na rysunku 3.5.

$$w' = (w \times \cos \theta) + (h \times \sin \theta) \quad (18)$$

$$h' = (w \times \sin \theta) + (h \times \cos \theta) \quad (19)$$



Rysunek 3.5. Grafiki przedstawiają wyprowadzenie wzorów na wysokość i szerokość prostokąta w pełni otaczającego obraz obrócony względem jego środka o kąt θ zgodnie z ruchem wskazówek zegara. Grafika (a) przedstawia stosunek wielkości prostokątów otaczających obraz przed i po obrocie, natomiast grafika (b) przedstawia wykorzystanie wzorów trygonometrycznych w celu otrzymania ostatecznych wielkości.

Można zauważyć, że dzięki założeniu, że największy możliwy obraz jest kwadratem o wymiarach (s_{max}, s_{max}), to

$$\max_{rot_w} = \max_{rot_h} = \max_{\theta} (s_{max} \times (\cos \theta + \sin \theta)) \quad (20)$$

Wartość ta największa jest dla $\theta = \pm 45^\circ$. Stąd też ostatecznie, obrazy wejściowe do sieci są kwadratami o wymiarach $w_{max} \times h_{max}$, gdzie:

$$w_{max} = h_{max} = s_{max} \times (\sin 45^\circ + \cos 45^\circ) \quad (21)$$

Po otrzymaniu ostatecznych wymiarów każdy obrazek ze zbioru danych jest otaczany ramką. Aby wyeliminować skłonność sieci neuronowych do przeuczania się na kolorze czarnym, zdjęcia otaczane są ramką będącą odbiciem obrazu połączonym z silnym rozmyciem Gaussa (o odchyleniu standardowym $\sigma = 10$).



Rysunek 3.6. Kolejne etapy dodawania ramki w postaci odbicia z silnym rozmyciem Gaussowskim do obrazów. Najpierw obraz sprowadzany jest do największego możliwego kwadratu w ramach zbioru, następnie występuje rotacja, co wiąże się z dodaniem kolejnej ramki.

Ostatnim krokiem rozszerzania obrazka jest zmiana wartości koordynatów punktu A dla każdej obwiedni na obrazie (jako że wskazują one na odległość górnego lewego rogu obiektu od górnego lewego rogu obrazka, a na etapie preprocessingu dodawana jest ramka m.in. z góry i z lewej strony obrazu). W tym celu wystarczyło obliczyć różnicę między wymiarami obrazu oryginalnego a wymiarami (w_{max}, h_{max}), a jej połowę odpowiednio w pionie i poziomie dodać do punktu A , tzn. dla każdego obrazu i i dla każdej obwiedni j w ramach tego obrazu:

$$A'_j = A_j + 0.5 \times \begin{bmatrix} w_{max} - w_i \\ h_{max} - h_i \end{bmatrix} \quad (22)$$

Po odpowiednim przeniesieniu każdej obwiedni obiektów na obrazach następnym krokiem była zamiana każdego prostokąta wskazującego obiekt na obrazie na 4 skalary jednocześnie identyfikujące obiekt. W tym miejscu dla ułatwienia oznaczeń panuje założenie, że sieć Focus wskazuje maksymalnie jeden obiekt przypisanej mu kategorii, a w przypadku kilku obiektów tej samej klasy na obrazie benchmarkowym, wybierany jest ten najbliższy środka.

Na początku tego etapu konieczne było obliczenie środka obrazu i (dla każdego obrazu ze zbioru jest on taki sam, ponieważ obrazy są tego samego rozmiaru, a ramka jest dodawana wokół) i środka obwiedni j jako odpowiednio:

$$S_i = 0.5 \times \begin{bmatrix} w_{max} \\ h_{max} \end{bmatrix} \quad S_j = A_j + 0.5 \times \begin{bmatrix} w_j \\ h_j \end{bmatrix} \quad (23)$$

3. Proponowane rozwiązanie

Ze względu na fakt, że obrazy wchodzące do uczącej się sieci Focus są sztucznie obracane, a punktem obrotu nie jest środek obiektu, tylko środek obrazu, trzeba obliczyć koordynaty obiektu po dokonaniu obrotu. Ostatecznie, dla celów modelu Focus, wystarczające jest obliczenie zmiany położenia środka obiektu pod wpływem rotacji.

Do wyznaczenia tych koordynatów potrzebna jest macierz rotacji $R'(\theta)$, której postać minimalnie różni się w wizji maszynowej od tej znanej z grafiki komputerowej (ze względu na lewośkrętny układ współrzędnych założony na początku tego rozdziału).

$$R'(\theta) = R(-\theta) = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (24)$$

Jeśli rotacja o kąt θ dokonywana by była względem początku układu współrzędnych - piksela $(0,0)$ stanowiącego górny lewy róg obrazu, wtedy współrzędne te można by było obliczyć przy użyciu przedstawionej wcześniej macierzy:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = R'(\theta) \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \times \cos\theta + y \times \sin\theta \\ -x \times \sin\theta + y \times \cos\theta \\ 1 \end{bmatrix} \quad (25)$$

Natomiast w przypadku tej pracy konieczna jest dodatkowa modyfikacja operacji - ze względu na fakt, że obrót obrazu wykonywany jest względem środka obrazu, wymagana jest najpierw translacja środka obrazu do punktu $(0,0)$ przy użyciu macierzy T_1 (która wykonuje translację w obu osiach poprzez odjęcie połowy rozmiaru obrazu), następnie dokonanie wyżej wymienionej rotacji $R'(\theta)$, a na końcu przywrócenie pozycji pikseli do stanu przed pierwszą translacją za pomocą macierzy T_2 , a więc ostateczna pozycja środka obiektu (a dokładniej prostokąta go otaczającego) j dla obrazu i po wykonaniu rotacji o kąt θ wynosi:

$$\begin{bmatrix} x'_{S_j} \\ y'_{S_j} \\ 1 \end{bmatrix} = T_2 \times R'(\theta) \times T_1 \times \begin{bmatrix} x_{S_j} \\ y_{S_j} \\ 1 \end{bmatrix} \Rightarrow \quad (26)$$

$$\begin{bmatrix} x'_{S_j} \\ y'_{S_j} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_{S_i} \\ 0 & 1 & y_{S_i} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -x_{S_i} \\ 0 & 1 & -y_{S_i} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_{S_j} \\ y_{S_j} \\ 1 \end{bmatrix} \Rightarrow \quad (27)$$

$$\begin{bmatrix} x'_{S_j} \\ y'_{S_j} \\ 1 \end{bmatrix} = \begin{bmatrix} (x_{S_j} - x_{S_i}) \times \cos\theta + (y_{S_j} - y_{S_i}) \times \sin\theta + y_{S_i} \\ -(x_{S_j} - x_{S_i}) \times \sin\theta + (y_{S_j} - y_{S_i}) \times \cos\theta + y_{S_i} \\ 1 \end{bmatrix} \quad (28)$$

3.5.3. Trening wstępny sieci Classifier

Aby zapewnić lepsze działanie sieci klasyfikującej, w ramach pracy zdecydowano się - tak samo jak w przypadku sieci Focus - na dokonanie jej wstępnego treningu (ang. pretraining). Aby jak najlepiej wyznaczać klasy przybliżonych obiektów, sieć jest optymalizowana przy użyciu obrazów będących przybliżeniami na obiekty znajdujące się na obrazach ze zbioru benchmarkowego, co wymaga odpowiedniego przygotowania danych.

Ze względu na to, że przeprowadzane w ramach tej pracy badania ograniczają się do podzbioru klas, jakie mogą przyjąć obiekty w zbiorze benchmarkowym, obwiednie reprezentujące klasy nieuwzględniane w badaniach uznawane są za tło, natomiast pozostałe regiony etykietowane są jako poszczególne klasy.

Kolejną kwestią związaną z przygotowaniem danych pod trening wstępny sieci klasyfikującej jest wycinanie kwadratowych obszarów, w pełni pokrywających obiekty ze zdjęcia, tak jak zostało to ukazane na rysunku 3.3. Proces ten można podzielić na dwa etapy. Na początku konieczne jest znalezienie współrzędnych piksela A' (reprezentującego lewy górny róg kwadratu) oraz piksela D' (reprezentującego dolny prawy róg kwadratu). W tym celu, na podstawie czwórki reprezentującej obwiednię obiektu j ze zbioru benchmarkowego: $(x_{A_j}, y_{A_j}, w_j, h_j)$ (gdzie (x_{A_j}, y_{A_j}) to współrzędne piksela reprezentującego górnego lewego róg prostokąta j , a w_j oraz h_j to kolejno jego szerokość i wysokość) konieczne jest zdobycie środka S_j prostokąta i jego największego możliwego promienia r_j^* .

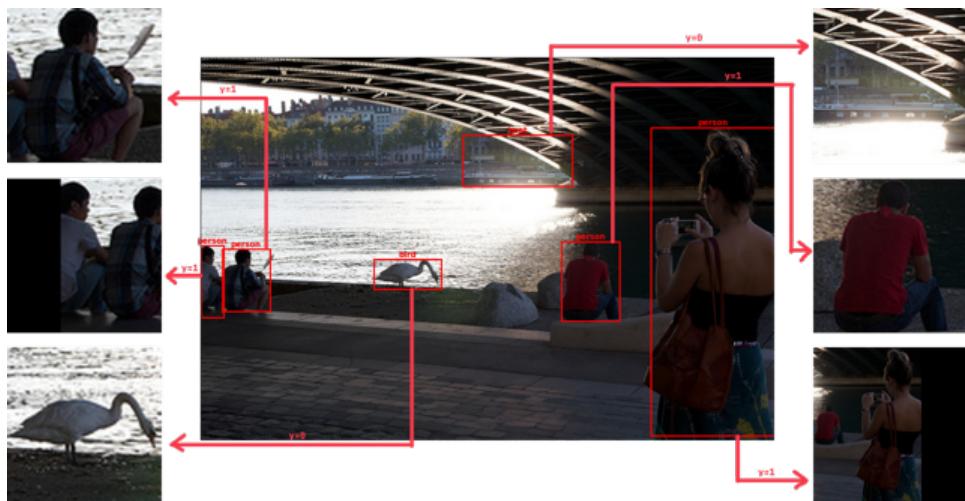
$$S_j = \begin{bmatrix} x_{S_j} \\ y_{S_j} \end{bmatrix} = \begin{bmatrix} x_{A_j} + \frac{w_j}{2} \\ y_{A_j} + \frac{h_j}{2} \end{bmatrix} \quad r_j^* = \frac{\max\{w_j, h_j\}}{2} \quad (29)$$

Mając tę informację, możliwe jest teraz wyznaczenie współrzędnych punktów A' oraz D' , na podstawie których dokonywane jest wycięcie kwadratu z obrazka:

$$A'_j = \begin{bmatrix} x_{A'_j} \\ y_{A'_j} \end{bmatrix} = \begin{bmatrix} x_{S_j} - r_j^* \\ y_{S_j} - r_j^* \end{bmatrix} \quad D'_j = \begin{bmatrix} x_{D'_j} \\ y_{D'_j} \end{bmatrix} = \begin{bmatrix} x_{S_j} + r_j^* \\ y_{S_j} + r_j^* \end{bmatrix} \quad (30)$$

Drugim etapem wycinania jest przeskalowanie otrzymanego kwadratu tak, aby rozmiary obrazu reprezentującego obiekt zgadzały się z rozmiarami tensora oczekiwanej na wejściu sieci klasyfikującej. Ze względu na to, że sieć klasyfikująca na swoim wejściu oczekuje kwadratu, a taki uprzednio został wycięty, skalowanie może zostać wykonane swobodnie (bez większych modyfikacji), ponieważ współczynnik proporcji obrazu zostanie zachowany. Proces przygotowania (ang. preprocessingu) danych, opisany w tym podrozdziale, został wizualnie podsumowany na rysunku 3.7.

3. Proponowane rozwiązanie



Rysunek 3.7. Wizualizacja przygotowania danych dla treningu sieci klasyfikującej. Obwiednia każdego obiektu zamieniana jest w najmniejszy kwadrat w pełni go wypełniający, który następnie jest skalowany do rozmiaru wejścia klasyfikatora. Jeśli klasa obiektu jest rozpatrywana przez architekturę - etykieta przypisywana jest zgodnie z kategorią, a jeśli nie, ustawiana na 0.

3.5.4. Dostrojenie całej architektury

Ostatnim etapem uczenia całej architektury jest połączenie wszystkich trzech modułów ze sobą i "dostrojenie" (ang. fine-tuning) całego modelu poprzez podawanie sieci obrazów ze zbioru benchmarkowego, a następnie ich przetworzenie przez całą architekturę - kolejno sieci skupiące, moduł transformujący i sieć klasyfikującą oraz obliczenie funkcji straty na decyzji klasyfikatora. Z powodu poprawnej implementacji różniczkowalnej transformacji względem translacji, skali i rotacji, możliwy jest pełen przepływ gradientu od początku do końca architektury przez wszystkie moduły i trenowanie sieci skupiącej tak, aby jak najlepiej wspomagała sieć klasyfikującą w decyzji.

Przepływ danych przez architekturę i schemat jej działania dla pojedynczej sieci Focus został przedstawiony już na rysunku 3.1, natomiast należy pamiętać, że proces dostrajania powinien zostać wykonany dla każdej sieci Focus reprezentującej poszczególną kategorię obiektu równolegle (aby sieć klasyfikująca była optymalizowana w kontekście obiektów należących do kilku kategorii i nie preferowała jednej z nich). Dane dla dostrajania wymagają znacznie mniej pracy przy ich przygotowaniu niż dla przetrenowywania poszczególnych sieci. Obrazy jedynie dopełniane są ramkami w celu zapewnienia, że każdy z nich jest tego samego rozmiaru oraz dla każdej z sieci Focus musi istnieć informacja, czy na obrazie znajduje się obiekt klasy jej przypisanej. Proponowane przez sieć Focus parametry przekształcenia przechodzą do dalszej części architektury niezależnie czy prawdopodobieństwo przez nią wskazywane jest dostatecznie wysokie - w ten sposób klasyfikator będzie zmuszany do wskazywania klasy "tło" (czyli braku obiektu), gdy parametry transformacji nie wskazują obiektów danej klasy.

Podsumowując, architektura, obsługująca K klas, w trakcie treningu otrzymuje pakiet (ang. batch) N obrazów, które trafiają do każdej z K sieci wskazujących - te zwracają w sumie $K \cdot N$ zestawów parametrów transformacji, by ostatecznie moduł transformujący zamienił je na $K \cdot N$ obrazów, na których dokonywana będzie klasyfikacja.

3.6. Wyznaczanie parametrów obwiedni na podstawie skalarów transformacji

Aby móc dokonywać ewaluacji modelu na zbiorze benchmarkowym, ostateczna informacja płynąca z architektury musi być zamieniana na czwórkę liczb reprezentujących obwiednie obiektów - (x_A, y_A, w, h) . Zarówno wysokość jak i szerokość prostokąta otaczającego obiekt j na obrazie i można otrzymać przy użyciu, zwracanego przez sieć Focus, logarytmu skali s oraz rozmiarów (w_i, h_i) obrazu:

$$\begin{aligned} w_j &= \exp s \cdot w_i \\ h_j &= \exp s \cdot h_i \end{aligned} \tag{31}$$

Ze względu na fakt, że sieć Focus proponuje skalę jako pojedynczą liczbę, a obrazy wchodzące do sieci skupiającej są kwadratami, wskazywane obwiednie będą zawsze kwadratami. Współrzędne punktu A_j dla obwiedni j otrzymywane są przy użyciu, zwracanych przez sieć Focus, skalarów translacji (t_x, t_y) , rozmiarów (w_i, h_i) obrazu oraz wcześniej otrzymanych rozmiarów (w_j, h_j) obwiedni jako:

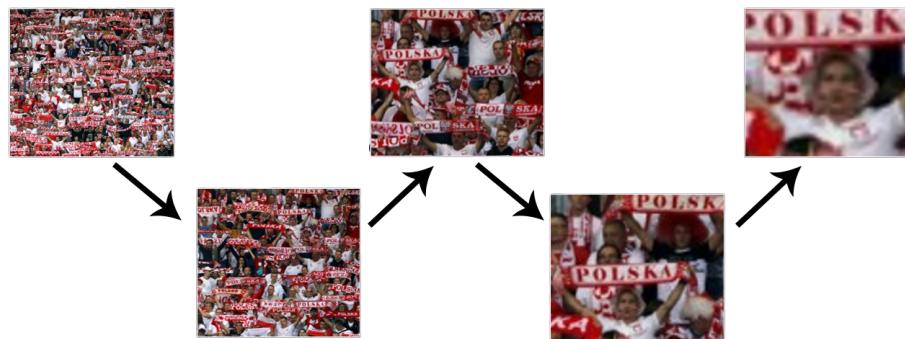
$$\begin{aligned} x_{Sj} &= 0.5 \cdot (w_i + w_i \cdot t_x - w_j) \\ y_{Sj} &= 0.5 \cdot (h_i + h_i \cdot t_y - h_j) \end{aligned} \tag{32}$$

3.7. Mechanizm skupiający

W podrozdziale 3.2.1 można było zauważać rolę mechanizmu skupienia w całej architekturze - ma on proponować takie skalary, których połączenie w macierz przekształcenia i zastosowanie przez moduł transformujący na wielkim obrazie stworzy obraz, na którym podejmowanie decyzji przez klasyfikator będzie prostsze. Można dopatrzyć się tutaj dużego podobieństwa w działaniu do opisanego wcześniej w 2.4 mechanizmu atencji, natomiast ze względu na to, że nie jest to dokładnie to samo działanie, zdecydowano się na określenie "skupienie".

Nie jest to jedyny aspekt, w jakim mechanizm skupienia przysługuje się całemu rozwiązaniu. Wszystkie architektury dotyczące detekcji narażone są na problem skończonej liczby proponowanych przez nie regionów z obiektemi. Architektura Focus-CNN, pomimo wskazywania pojedynczego obiektu przez sieć skupiającą, zastosowana rekurencyjnie może wskazywać dowolną liczbę obiektów na obrazie - poprzez aplikowanie przesuwania i przybliżania obrazu do momentu, aż nie pozostanie na nim pojedynczy obiekt.

3. Proponowane rozwiązanie



Rysunek 3.8. Schemat zastosowania rekurencji na obrazie z dużą liczbą obiektów nas interesujących. Obraz jest odpowiednio przesuwany i przybliżany do momentu aż nie pozostanie na nim pojedynczy obiekt danej kategorii.

3.8. Potencjalne korzyści i ograniczenia rozwiązania

Proponowana architektura swoją siłę głównie pokłada w mechanizmie skupienia, którego prawidłowe działanie może spowodować w przyszłości wysoką skuteczność rozwiązania. Jedną z jej głównych zalet jest to, że z wszystkich do tej pory dostępnych architektur najbardziej w swoim działaniu przypomina intuicję, jaką miałby człowiek rozwiązujący zadanie detekcji - na początku szukałby on obszarów, w których znajdują się obiekty i odpowiednio obracał oraz przybliżał obraz, aby potem jak najłatwiej było mu zdecydować o ich kategorii.

Co więcej, sieć wyróżnia się na tle innych tym, że uwzględnia rotację, aby być bardziej odporną na obrócone obiekty na obrazie. Kolejną korzyścią, o jakiej można wspomnieć, jest fakt, że architektura posiada pełną różniczkowalność obliczeń od początku do końca przez wszystkie moduły architektury, dzięki czemu proces jej dostrajania może zapewnić bardzo dobrą końcową skuteczność. Jeszcze inną zaletą, niebadaną w ramach tej pracy, jest korzyść, jaką nie mogą się pochwalić dotychczasowe rozwiązania w dziedzinie detekcji - zastosowana rekurencyjnie może wykryć potencjalnie dowolną liczbę obiektów danej kategorii na obrazie dzięki stałemu dostępowi do skali i translacji w module transformującym.

Mówiąc natomiast o wadach architektury, należy mieć na uwadze, że implementowane w ramach tej pracy rozwiązanie jest rozwiązaniem wstępny i stanowi bazę dla badań rozwojowych w przyszłości. Dodatkowo większość słabych punktów omawianego podejścia może zostać ominięta poprzez modyfikację aktualnego rozwiązania o rozszerzenia, które zostały zaproponowane w kolejnym rozdziale.

Obecnie największa wada rozwiązań dotyczy zależności liczby sieci skupiających w architekturze od liczby kategorii obiektów przez nią wykrywanych - powoduje to, że przy dużej liczbie klas sieć może osiągnąć gigantyczne rozmiary, przez co być trudną do zmieszczenia na kartach graficznych wykorzystywanych przy jej trenowaniu, a dodatkowo musi zadziałać dla każdej z kategorii, co może znacznie spowolnić jej działanie i utrudnić jej pracę w zadaniach czasu rzeczywistego. Dodatkową wadą może być kształt obwiedni proponowanej przez architekturę - jako że wychodząca z niej skala jest pojedynczą liczbą stanowiącą o wielkości obiektu, można zauważać, że wszystkie otaczające prostokąty proponowane przez sieć muszą być pojedynczym kształtem, co oznacza, że są kwadratami, a nie jak typowo dla tego zadania - prostokątami o różnych stosunkach szerokości do wysokości. Problem ten możliwy byłby do rozwiązania poprzez zwracanie przez sieć Focus dwóch skali - osobno dla osi OX i osobno dla OY , natomiast wyznaczanie jednej z nich nie mogłoby podlegać optymalizacji w procesie dostrajania ze względu na fakt, że klasyfikator oczekuje tensorów tych samych rozmiarów.

3.9. Proponowane rozszerzenia architektury

Aby umożliwić architekturze wykrywanie wielu obiektów danej kategorii na obrazie, konieczna jest modyfikacja obecnej koncepcji, w celu umożliwienia takiego działania sieci. Proponowane zmiany, jakie można by było zastosować w przyszłości, koncentrują się wokół schematu działania sieci wskazującej. Obecnie, moduł ten, w przypadku wielu obiektów tej samej klasy na grafice, ma za zadanie opisywać jeden - ten będący jak najbliżej środka. Tu możliwa jest pierwsza zmiana - zamiast proponować obiekty, sieć może proponować regiony i to w większej ilości (przykładowo 5 regionów - jeden dla środka obrazu i po jednym dla każdego z jego rogów). Region wtedy mógłby nie zawierać żadnego obiektu (byłaby to klasa 0), pojedynczy obiekt (klasa 1) lub najmniejszy kwadrat zawierający kilka obiektów (klasa 2). W przypadku otrzymania regionu opisującego kilka obiektów naraz wykorzystywane byłyby parametry translacji i skali, a na wyniku tego przesunięcia i przybliżenia sieć skupiająca zadziałałaby jeszcze raz (rekurencyjnie). W innej sytuacji, gdyby region wskazywał pojedynczy obiekt, wykorzystywane byłyby parametry translacji, skali i rotacji tak jak dotychczas i przekształcone zdjęcie podawane by było klasyfikatorowi (co kończyłoby rekurencję), natomiast w przypadku, gdyby region nie zawierał żadnego obiektu, dalsze przetwarzanie byłoby porzucone (stanowiłoby również o końcu rekurencji). Ze względu na fakt, że ogromną trudnością dla treningu tej architektury byłoby przygotowanie danych uczących, zdecydowano się nie realizować tej koncepcji w ramach pracy dyplomowej.

3. Proponowane rozwiązanie

Kolejną kwestią problematyczną w tej architekturze jest z pewnością fakt istnienia odrębnych sieci Focus dla każdej z rozpatrywanych klas. Narzucającym się od razu rozwiązańem tego problemu jest wykorzystanie pomysłu, jaki został zastosowany w przypadku opisywanej w rozdziale 2.2.4 architektury Faster R-CNN, gdzie sieć propozycji regionów (RPN) ma za zadanie wskazywać obiekty, nie zwracając uwagi na konkretne klasy, do których należą. Decyzja o kategorii obiektów odwlekana jest tam do sieci detekcji, a w przypadku architektury Focus-CNN wyznaczanie klas mogłoby być dokonywane przez sieć klasyfikującą.

Trzecim pomysłem na poprawę architektury jest rozwiązanie mające na celu zapewnić, że przewidywane przez model obwiednie nie będą kwadratami, a mogą być prostokątami. Z powodu, że sieci neuronowe, ze względu na swoją naturę, oczekują w trakcie treningu na swoim wejściu pakietów danych o tych samych rozmiarach, nie jest możliwe podawanie klasyfikatorowi jednocześnie obrazów o różnych stosunkach szerokości i wysokości. Stąd obrazy wejściowe dla sieci klasyfikującej są przeskalowanymi kwadratami o tych samych rozmiarach, otaczającymi pełne obiekty - zapewniając w ten sposób, że zawsze będą je całkowicie zawierać. W tym przypadku niemożliwe jest, aby pełen przepływ gradientu przez sieć na etapie jej dostrajania umożliwiał optymalizację zarówno wysokości, jak i szerokości obwiedni, natomiast nic nie stoi na przeszkodzie, aby sieć skupiająca zwracała oddziennie skalę w osiach OX oraz OY , a podczas tworzenia macierzy transformacji wykorzystywała tylko większą z nich (tą oznaczającą mniejsze przybliżenie). Obie te skale mogłyby być optymalizowane naraz na etapie treningu wstępnego sieci skupiającej i wykorzystywane do konstruowania wynikowych obwiedni modelu.

4. Implementacja rozwiązania

Niniejszy rozdział opisuje wszelkie kwestie techniczne związane z implementacją architektury. Zapisane zostały tu uwagi i istotne informacje odnoszące się do zadania zaimplementowania rozwiązania, które zostało przedstawione w poprzednim rozdziale.

4.1. Różniczkowalne przekształcenie geometryczne

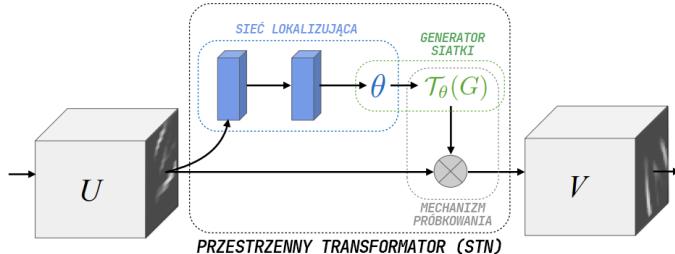
Bardzo istotnym zadaniem, z jakim trzeba było się zmierzyć przy implementacji modelu, było zapewnienie pełnego przepływu gradientu przez cały model. O ile kwestia ta nie była trudna do zrealizowania przy tworzeniu modelu sieci skupiającej i sieci klasyfikującej (są to sieci neuronowe, a ich programowanie jest względnie proste przy pomocy biblioteki PyTorch), o tyle wymagającą była realizacja modułu transformującego.

Jak już zostało to opisane w rozdziale 3.5.4, architektura, po wstępny treningu sieci skupiających i sieci klasyfikującej, ulega dostrajaniu. Proces ten pełni bardzo ważną funkcję dla całego rozwiązania - ułatwia trening sieciom skupiającym, ponieważ ich przetrenowanie na zadaniu regresji współrzędnych obwiedni okazuje się niewystarczające, aby dobrze radziły sobie z zadaniem lokalizacji obiektów, a drugi etap treningu, gdzie uczone są one wskazywać takie regiony, na których dalsza klasyfikacja jest ułatwiona, może poprawić ich optymalizację. Stąd też wymagana jest pełna różniczkowalność operacji przekształcania geometrycznego obrazu o zadane przez sieć Focus parametry.

4.1.1. Inspiracja dla rozwiązania

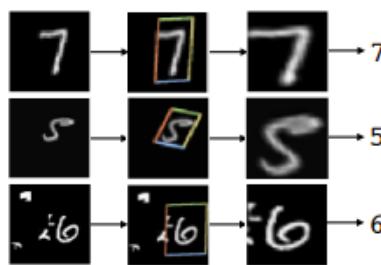
Inspirację wykonanej implementacji różniczkowalnych przekształceń geometrycznych stanowi realizacja, opisanej w [22], Sieci Przestrzennego Transformatora (ang. Spatial Transformer Network). Autorzy artykułu zrealizowali moduł, Przestrzenny Transformator, który, dodany jako warstwa architektury, ma za zadanie uczyć się wyznaczać macierz transformacji, według której przekształcenie obrazu wejściowego zostanie dokonane tak, aby pełna sieć neuronowa mogła łatwiej podejmować decyzję o kategoriach obiektów na obrazie (gdyż sieci konwolucyjne miewają problemy z przekształconymi danymi).

Pierwszy element modułu stanowi sieć lokalizującą - otrzymuje ona mapę cech obrazu U i realizuje zadanie regresji parametrów transformacji - macierzy θ , której trening nie jest przeprowadzony bezpośrednio ze zbioru, a przez optymalizację globalnego zadania całej sieci. Następnie, generowana jest siatka punktów próbujących, reprezentująca rzut koordynatów pikseli z obrazu na przestrzeń będącą przekształceniem oryginalnej przestrzeni o macierz transformacji. W ostatnim kroku, przy użyciu obrazu wejściowego i siatki punktów próbujących, dokonywane jest różniczkowalne próbkowanie obrazu niezależnie w każdym z kanałów i otrzymywany jest obraz przekształcony. Architektura tego modułu została przedstawiona na rysunku 4.1.



Rysunek 4.1. Schemat działania, struktura i kolejne operacje wykonywane wewnątrz modułu Przestrzennego Transformatora (ang. Spatial Transformer Network Module). Źródło: [22].

Autorzy artykułu [22] zastosowali powyższy moduł do zadania klasyfikacji zniekształconego zbioru MNIST (zbioru do kategoryzacji odręcznie pisanych cyfr), osiągając na tym zadaniu najlepsze dotychczas wyniki, a przykład tego zastosowania obrazuje rysunek 4.2.



Rysunek 4.2. Wynik zastosowania Przestrzennego Transformera jako pierwszej warstwy gęstej w architekturze wytrenowanej w celu klasyfikacji zniekształconych grafik odręcznie zapisanych cyfr ze zbioru MNIST (grafiki zostały poddane losowym translacjom, rotacjom i skalom). Kolorową ramką zaznaczone zostały siatki punktów próbujących, które przedstawiają przekształconą przestrzeń obrazu o macierz transformacji. Trzecią kolumnę stanowi wynik zastosowania mechanizmu próbującego siatkę punktów na wejściowej grafice. Na końcu warstwy gęstej działają na przekształconym obrazie i przyporządkowują cyfrę, która się na nim znajduje. Źródło: [22].

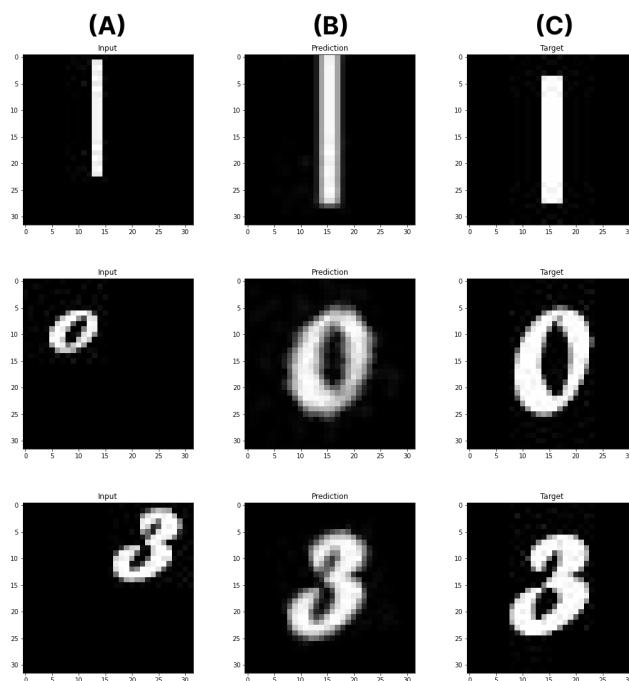
4.1.2. Rozwiązanie przy użyciu PyTorch

Wykorzystując oryginalną implementację Przestrzennego Transformatora, możliwe okazało się zaimplementowanie różniczkowalnych przekształceń geometrycznych dla celów tej pracy. Moduł transformujący składa się na 2 funkcje i przyjmuje tensor o wymiarach $(N, 4)$, gdzie N to rozmiar pakietu (ang. batch size), a każda próbka stanowi 4-elementowy wektor $P = \begin{bmatrix} t_x & t_y & \ln s & \theta \end{bmatrix}^T$, gdzie t_x oraz t_y to proponowane przez sieć wskazującą parametry translacji, a s i θ - proponowane parametry skali i rotacji. Pierwsza z metod modułu ma za zadanie dobierać się do konkretnych elementów wektorów, wyliczyć funkcje \cos i \sin dla kątów rotacji i skali jako $\exp ln s$ i skonstruować macierz transformacji taką jak wzór 17 jako tensora o wymiarach $(N, 2, 3)$ (implementacja nie oczekuje ostatniego wektora macierzy, ponieważ zawsze jest to $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$). Każda instrukcja generująca macierz przekształcenia stanowi operację na tensorach (z biblioteki PyTorch), przez co udało się zapewnić różniczkowalność operacji co do parametrów transformacji.

Druga z metod natomiast dokonuje już ostatecznego przekształcenia - przyjmuje macierz transformacji oraz obraz i na ich podstawie generuje siatkę punktów próbujących (przy użyciu funkcji realizowanej w STN jako generator siatki), by na końcu dokonać próbkowania pełnego obrazu (w STN był to mechanizm próbkowania) przy wykorzystaniu wyznaczonej siatki, czego wynikiem jest obraz otrzymywany przez sieć klasyfikującą.

4.1.3. Przetestowanie rozwiązania

Aby upewnić się, że wykonana implementacja działa, w ramach pracy zrealizowane zostało poboczne zadanie. Stworzono sztuczne grafiki przedstawiające białe cyfry na czarnym tle, a także ich wersje z tymi cyframi w postaci obróconej, przesuniętej i przeskaliowanej. Zaimplementowano prostą sieć neuronową, która zwraca, dla danego zniekształconego obrazu, parametry transformacji, jakie później moduł transformujący wykorzystuje, by przekształcić zniekształcony obraz w obraz z wyprostowanymi liczbami. Sieć uczona była w taki sposób, by minimalizować kwadrat różnic w pikselach pomiędzy proponowanym obrazem wyprostowanym a tym, jak powinien wyglądać oryginalnie (przed sztucznym zniekształceniem). Ze względu na fakt, że zadanie to jest niezwykle narażone na utknięcie optymalizacji w lokalnym minimum, a także sama optymalizacja silnie zależy od kroku uczenia (ang. learning rate), dla każdego obrazu sieć uczona była niezależnie z różnymi parametrami kroku, a wyniki uczenia można zobaczyć na rysunku 4.3. Wagi modelu zostały zainicjalizowane na wartość 0.0, dzięki czemu proponowane przekształcenie na początku optymalizacji to przekształcenie identyfikujące.



Rysunek 4.3. Wyniki eksperymentów optymalizacji zaimplementowanego modelu. W kolumnie (A) znajdują się obrazy, stanowiące wejście do sieci, w kolumnie (C) obrazy, jakie sieć powinna otrzymać, natomiast w kolumnie (B) - proponowane obrazy przez model.

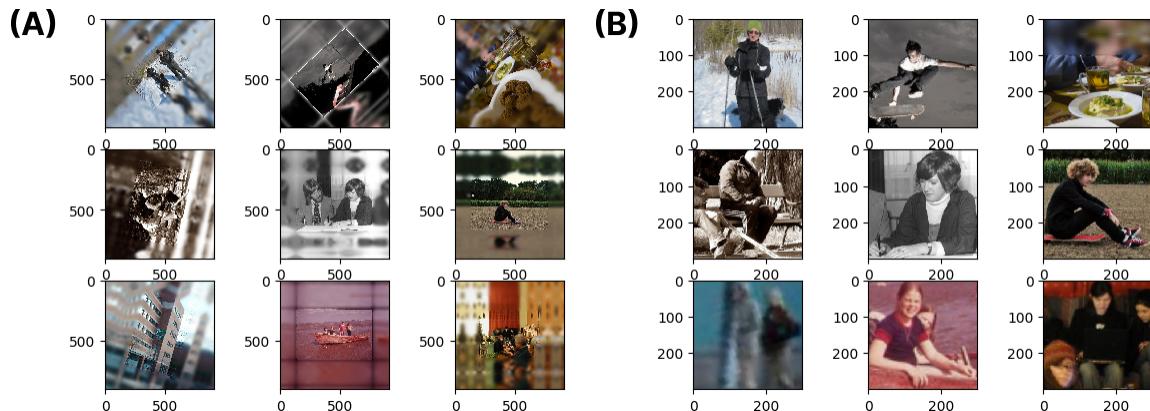
4. Implementacja rozwiązania

Dodatkowo zbadano działanie modelu również na obrazach RGB i osiągnięto satysfakcyjujące wyniki.



Rysunek 4.4. Wyniki eksperymentów optymalizacji zaimplementowanego modelu na obrazie RGB.

Drugi test dla implementacji stanowiło sprawdzenie, czy zastosowanie modułu transformującego na danych, którymi przetrenowywane są sieci skupiające, gwarantuje otrzymanie obrazów, jakie powinien otrzymywać klasyfikator (zawierające całkowicie obiekt prosty w swoim środku), co również zadziałało, jak można zobaczyć na rysunku 4.5.



Rysunek 4.5. Zestawienie grafik dla drugiego testu. W (A) można zobaczyć obrazy wejściowe dla całej architektury (na nich przetrenowywane są sieci skupiające), natomiast w (B) przedstawione są odpowiadające im grafiki będące zastosowaniem na nich transformacji o parametry, które powinny wyznaczać sieci skupiające.

4.2. Struktura kodu

Ze względu na fakt, że praca wymaga licznych eksperymentów, zdecydowano się na odpowiednie strukturalne ułożenie kodu projektowego. Zasadniczo, w ramach projektu dokonać można trzech akcji - przygotować dane pod trening modelu, trenować model na przygotowanych danych lub przetestować działanie wytrenowanego modelu - dla każdej z nich istnieje program ją uruchamiający, napisany w języku Python. Każdą z tych akcji wykonać można zarówno dla sieci klasyfikującej, jak i sieci skupiającej (przy przetrenowywaniu) lub dla całej architektury (przy dostrajaniu). Stąd też pełna implementacja podzielona została na klasy bazowe (reprezentujące typy modułów), z których dziedziczą klasy implementujące moduły pod konkretne zadania. Rodzaje modułów:

- **Trener** - implementuje proces uczenia sieci - przede wszystkim to, jak przetwarzana jest pojedyncza epoka tego procesu.
- **Tester** - implementuje proces testowania sieci - to jak sprawdzana jest skuteczność modelu na zbiorze testowym pod względem metryk.
- **Preprocessor** - implementuje proces przygotowywania danych pod trening i testowanie modeli - wszelkie kroki przekształcające zbiór benchmarkowy w zbiór, na którym model jest w stanie działać.
- **Zbior danych** - implementuje wykorzystywane w bibliotece PyTorch klasy zbiorów danych (ang. DataSets) - to, jak przechowywane są dane wejściowe oraz etykiety danych dla sieci neuronowych w programie.
- **Moduł ładowania danych** - implementuje wykorzystywane w bibliotece PyTorch klasy ładujące dane (ang. DataLoaders) - umożliwia iterację zbiorów danych i układy dane ich w pakiety.
- **Moduł modelu** - implementuje model sieci neuronowej - to jak dane przetwarzane są przez jej warstwy w celu otrzymania predykcji oraz jak liczona jest strata, jaką ponosi model w procesie optymalizacji.

Dla każdego z powyższych typów modułów zaimplementowane zostały klasy je realizujące dla sieci klasyfikującej (w procesie przetrenowania), dla sieci skupiającej (w procesie przetrenowania) oraz dla całej architektury (w procesie dostrajania). Uruchomienie konkretnej akcji (trening, ewaluacja lub przygotowanie danych) realizowane jest przez wywołanie odpowiedniego z programów wraz z plikiem konfiguracyjnym, gdzie zawarte są informacje o wyborze wersji dla każdego z potrzebnych modułów oraz o metrykach, które mają być obliczane na etapie ewaluacji, a także m.in. ścieżki zapisu modeli i danych oraz parametry optymalizatora podczas treningu sieci.

Do zbadania działania modelu Faster R-CNN na danych dla własnych eksperymentów dokonano dostosowania implementacji z oficjalnej dokumentacji biblioteki PyTorch ([23]). Jej implementacja stanowi niezależną, oddzielną część struktury kodu.

5. Przeprowadzone badania

Istotnym elementem każdej pracy w dziedzinie uczenia głębokiego, wprowadzającej nowe rozwiązanie, jest sprawdzenie działania modelu na dobrze zdefiniowanych i znanych w danym obszarze badań zbiorach danych poprzez wyznaczenie wartości popularnych metryk mierzących efektywność podejść.

Głównym celem eksperymentów przeprowadzonych w ramach opisywanej pracy było zbadanie skuteczności działania proponowanej architektury poprzez porównanie wyników jej działania, a konkretne wartości metryki *mAP*, opisanej w 2.1.5, z wynikami działania architektury Faster R-CNN na dwóch najpopularniejszych zbiorach danych w zadaniu detekcji obiektów na obrazie - PASCAL VOC 2012 [24] oraz MS COCO 2017 [25]. Ten eksperiment umożliwił sprawdzenie, jak skuteczne może być zastąpienie mechanizmu łączenia regionów zainteresowania (ang. RoI pooling) w modelu Faster R-CNN przez różniczkowalne operacje przekształcenia geometrycznego, nazwane w przypadku tej pracy realizacją mechanizmu atencji, ponieważ dokładnie ten element stanowi główną różnicę w koncepcjach tych dwóch architektur.

Dodatkowym badaniem przeprowadzonym w ramach tej pracy było sprawdzenie wpływu zastosowania pełnego przepływu gradientu łączącego przetrenowane części architektury w całość na ostateczne wyniki modelu. Pozytywny wynik takiego eksperymentu umożliwić mógł stwierdzenie poprawności zastosowania mechanizmu atencji w architekturze, gdyż początkowe modele całej architektury (sieci skupiące) wpływalyby na skuteczniejsze działanie modelu wyjściowego (sieci klasyfikującej).

Ostatnim z przeprowadzonych eksperymentów było stwierdzenie, jaki jest wpływ uwzględnienia rotacji w wyjściowych parametrach proponowanych przez sieci skupiące, które reprezentują lokalizację obiektów znajdujących się na obrazie, na skuteczność działania zarówno całej architektury jak i samej sieci klasyfikującej. W przypadku pozytywnego wyniku tego eksperymentu (podobnej skuteczności modelu zarówno dla modelu z rotacją jak i bez niej), można by było stwierdzić, że architektura jest odporna na zaburzenia kąta, pod jakim występują obiekty na obrazie.

W niniejszym rozdziale omówione zostały zbiory danych, jakie wykorzystano do badań, wymieniono wartości hiperparametrów i metody, które umożliwiły otrzymanie konkretnych wyników, a także zestawione zostało tabelaryczne porównanie wyników eksperymentów przeprowadzonych dla każdego z modeli. W trakcie wykonywania badań zadbano o ich powtarzalność, poprzez stosowanie konkretnego ziarna generatora liczb losowych, a wszelkie hiperparametry umieszczone w odpowiednich plikach konfiguracyjnych.

5.1. Zbiory danych

Nierożłącznym i bardzo istotnym elementem każdego eksperymentu w uczeniu maszynowym jest dobranie odpowiedniego zbioru danych do rozwiązywanego problemu. W przypadku tej pracy postawiono na dwa bardzo popularne zbiory w przypadku zadania detekcji obiektów na obrazie, czyli PASCAL VOC 2012 [24] oraz MS COCO 2017 [25]. Wybór padł na te zbiory głównie z powodu ich wysokiej pozycji w rankingu najczęściej cytowanych zbiorów do detekcji obiektów na obrazie ([26]), a także, ponieważ dwa te zbiory zostały wykorzystane w artykule [14] wprowadzającym architekturę Faster R-CNN.

5.1.1. Zbiór PASCAL VOC 2012

PASCAL Visual Object Classes (VOC) stanowi jedno z najpopularniejszych, jeśli chodzi o wizję maszynową, wyzwań i było prowadzone między latami 2005 a 2012. PASCAL VOC 2012, ostateczny etap zadania, jest szeroko znany i rozpoznawanym zbiorem benchmarkowym skierowanym na takie zadania jak detekcja obiektów na obrazie, klasyfikacja obrazów czy segmentacja.

Zbiór danych zawiera ponad 11 tysięcy obrazów z prawie 30 tysiącami obiektów zaznaczonych na tych obrazach. Klas, do których należą te obiekty, jest 20, natomiast na potrzebę eksperymentów prowadzonych w ramach tej pracy, zdecydowano się na trzy najpopularniejsze z nich.

5.1.2. Zbiór COCO-2017

Microsoft Common Objects in Context (MS COCO) stanowi kolejny, bardzo ważny zbiór danych w detekcji obiektów na obrazie, który zapewnia środowisko dla kilku rodzajów zadań - rozpoczynając od detekcji obiektów i segmentacji instancji na obrazie, przez wykrywanie punktów kluczowych osób, aż po syntezę opisów obrazów.

Zbiór MS COCO posiada około 200 tysięcy ręcznie zaanotowanych obrazów z około 1,5 milionem zaznaczonych na nich obiektów należących do jednej z 80 możliwych kategorii. Również w tym przypadku, badania zostały ograniczone do trzech najpopularniejszych klas obiektów.

5.1.3. Przygotowanie zbiorów danych

Aby móc zaaplikować stworzoną architekturę do publicznie dostępnych zbiorów danych, koniecznym była ich odpowiednia modyfikacja.

Proces rozpoczął się od odpowiedniego rozszerzenia obrazów, aby miały one wszystkie te same wymiary. Wykonano to zgodnie z algorytmem opisanym w 3.6 i zgodnie z tymi obliczeniami przesunięto wszystkie niezbędne obwiednie.

5. Przeprowadzone badania

Następnie, w ramach danego obrazu, jeśli znajdował się na nim więcej niż jeden obiekt tej samej kategorii, pozostawiona została jedynie ta anotacja, która wskazuje obiekt bliżej. Na końcu, wybrano tylko te anotacje, które opisują kategorie brane pod uwagę w przeprowadzanych eksperymentach, a dokładnie te, które opisują ludzi, samochody oraz rowery.

Działanie to było niezbędne, aby upewnić się, że, stworzona implementacja architektury zostanie zbadana w ograniczonej wersji zadania, a także, żeby móc bezpośrednio na tym zbiorze móc wytrenować i sprawdzić wyniki architektury Faster R-CNN.

5.2. Zbadanie skuteczności działania architektury

Pierwszym, a jednocześnie najważniejszym z przeprowadzonych eksperymentów było sprawdzenie, jak skuteczna jest stworzona architektura w porównaniu do znanej już w detekcji obiektów architektury Faster R-CNN. Wykonano to poprzez porównanie wartości metryk $mAP@0.5$ oraz $mAP@0.5 : 0.05 : 0.95$ (opisanych w 2.1.5) na zbiorach COCO-2017 oraz PASCAL VOC 2012.

Na początku tego zadania, konieczne było przetrenowanie i zbadanie wartości metryk dla architektury Faster R-CNN na przygotowanych zbiorach danych.

5.2.1. Wyniki architektury Faster R-CNN

Wykonano tutaj dwa niezależne procesy uczenia sieci dla każdego ze zbiorów. Treningi rozpoczynały się od wstępnie zainicjalizowanych wag modelu dokładnie w taki sposób jak w [14]. Na przetrenowaną sieć CNN, tworzącą mapę cech obrazu i stanowiącą bazę sieci propozycji regionów, wybrano ResNet-50, natomiast druga część sieci to, znany z Fast R-CNN, predyktor regionów zainteresowania.

Oba treningi wykonywane były przez 10 epok, wyjściowych klas architektury były 4 (3 badane i dodatkowo klasa "0" reprezentująca brak obiektu). W celach augmentacji danych zastosowano dodatkowo odbicie poziome. Tempo uczenia się (ang. learning rate) ustawiono na 0,005, zastosowano optymalizator stochastycznego spadku gradientowego (ang. SGD) z momentum.

Zbiór danych	$mAP@0.5$	$mAP@0.5 : 0.05 : 0.95$
PASCAL VOC 2012	0,420	0,244
MS COCO 2017	0,277	0,169

Tabela 5.1. Zestawienie wyników metryk $mAP@0.5$ i $mAP@0.5 : 0.05 : 0.95$, jakie osiągnął model Faster R-CNN na zbiorach PASCAL VOC 2012 i MS COCO 2017.

Analizując wyniki modelu, można zauważyc, że są one nieco gorsze niż te zaprezentowane w artykule [14]. Powód można dopatrywać się w krócej trwającym procesie nauki, ponieważ zdecydowano się w ramach tych badań na jedynie 10 epok, podczas gdy trening w przypadku cytowanego badania trwał dłużej. Innym powodem może być mniejsza liczba danych, ponieważ autorzy architektury Faster R-CNN łączyli zbiory z różnych źródeł w celu otrzymania jak najlepszych rezultatów. Innym, i również wielce prawdopodobnym powodem niskich wyników, mogło być zachowanie jedynie anotacji najbliższych środka w przypadku kilku obiektów tej samej kategorii na tym samym obrazie. Takie działanie spowodować mogło, że model, pomimo wykonywania poprawnej predykcji, oceniany był negatywnie, co oznacza zwiększenie się liczby przypadków fałszywie poprawnych (false positive) i, zgodnie z definicją metryki mAP , powodowało degradację jej wartości. Wynika to z faktu, że Faster R-CNN potrafi już działać przy wielu obiektach danej klasy na obrazie. Niemniej jednak otrzymane wartości metryki wyznaczają próg, który nowa architektura, Focus-CNN, musiałaby przekroczyć, aby móc rywalizować z najlepszymi detektorami w skuteczności.

5.2.2. Przetrenowanie sieci skupiających

Kolejnym krokiem eksperymentu był, wspomniany już w rozdziale 3.5.1, etap przeuczenia sieci skupiających tak, aby w procesie łączenia i dostrajania model należących do architektury, były one w stanie ze sobą lepiej współpracować. Takich treningów wykonano sześć, tzn. dla dwóch zbiorów po 3 badania działające konkretnych klasach. Istotny jest fakt, że badania w ramach tej sekcji nie uwzględniają parametru rotacji (co za tym idzie, nie działają na sztucznie obracanych obrazach), a spowodowane jest to faktem, że architektura Faster-RCNN nie obsługuje obrazów obróconych, a w ramach tego eksperymentu istotne jest zastosowanie takich samów obrazów wejściowych w celu porównania architektur.

Optymalizacja modeli skupiających wykonywana była w celu otrzymania jak najlepszej straty na zbiorze treningowym i walidacyjnym, gdyż w przypadku tego procesu nie istniała żadna znana metryka, mogącą jeszcze lepiej sprawdzić jak działa model. Najlepszym optymalizatorem okazał się tutaj również algorytm stochastycznego spadku gradientowego (ang. SGD) z momentum wynoszącym 0,9.

Dla zbioru Pascal, rozpoczęto od treningu sieci skupiającej dla klasy osób. Początkowe rozłożenie klas w przygotowanym zbiorze danych to było 6061 obrazów pustych (bez klasy) oraz 7639 obrazów z osobami. Ze względu na dobry rozkład klas, nie była stosowana tu żadna augmentacja danych. Zastosowano tutaj tempo uczenia wynoszące 0.0001. Proces trenowania trwał 50 epok, natomiast najlepszym okazał się model z epoki 42.

5. Przeprowadzone badania

Zbiór	\mathcal{L}_{t_x}	\mathcal{L}_{t_y}	\mathcal{L}_s	\mathcal{L}
Treningowy	0,045	0,028	0,067	2,21
Walidacyjny	0,126	0,059	0,194	5,06

Tabela 5.2. Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali oraz pełnej straty modelu, jakie osiągnął model skupiający w fazie przetrenowania na klasie osób ze zbioru PASCAL VOC 2012.

Na podstawie wyników i faktu, że obrazy wejściowe do sieci były o wymiarach 640×640 można stwierdzić, że przetrenowany model, jeśli chodzi o lokalizację środka obwiedni, w przypadku zbioru treningowego średnio myli się o 14 pikseli poziomo i 9 pikseli pionowo oraz średnio docelowe obwiednie są 1,07 razy mniejsze lub większe, natomiast, w przypadku zbioru walidacyjnego, model średnio myli się o 40 pikseli poziomo i 19 pikseli pionowo w przypadku środka obwiedni oraz 1,21 razy w przypadku jej wielkości.

Kolejnym procesem dla zbioru Pascal był trening sieci skupiającej dla klasy samochodów. Początkowe rozłożenie klas w przygotowanym zbiorze danych było już znacznie gorsze: 13220 obrazów pustych (bez klasy) oraz jedynie 480 obrazów z samochodami. Ze względu na taki bilans klas, zastosowano tutaj metodę usuwania przykładów o klasie 0, co spowodowało, że w zbiorze treningowym było tylko 4 razy więcej przykładów negatywnych. Zastosowano tutaj tempo uczenia wynoszące również 0.0001, wagi dla translacji i skali to odpowiednio 12 i 3. Proces trenowania trwał 70 epok, natomiast najlepszym okazał się model z epoki 51.

Zbiór	\mathcal{L}_{t_x}	\mathcal{L}_{t_y}	\mathcal{L}_s	\mathcal{L}
Treningowy	0,044	0,038	0,086	2,11
Walidacyjny	0,106	0,059	0,157	3,01

Tabela 5.3. Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali oraz pełnej straty modelu, jakie osiągnął model skupiający w fazie przetrenowania na klasie samochodów ze zbioru PASCAL VOC 2012.

W tym przypadku można stwierdzić, że przetrenowany model, jeśli chodzi o lokalizację obwiedni, dla zbioru treningowego, średnio myli się o 14 pikseli poziomo i 12 pikseli pionowo w przypadku koordynatów środka prostokąta otaczającego oraz średnio docelowe obwiednie są 1,1 razy mniejsze lub większe, natomiast, w przypadku zbioru walidacyjnego, model średnio myli się o 35 pikseli poziomo i 19 pikseli pionowo w przypadku środka obwiedni oraz 1,17 razy w przypadku jej wielkości.

Ostatnim treningiem sieci skupiającej dla zbioru Pascal był trening z klasą rowerów. Początkowe rozłożenie klas w tym przypadku było również nie najlepsze: 12670 obrazów pustych (bez klasy) oraz jedynie 1030 obrazów z rowerami. Również tutaj zastosowano metodę usuwania przykładów o klasie 0, co spowodowało, że w zbiorze treningowym było już tylko 3 razy więcej przykładów negatywnych. Zastosowano tutaj również tempo uczenia wynoszące 0.0001, wagi dla translacji i skali takie same: 8. Proces trenowania trwał 40 epok, natomiast najlepszym okazał się model z epoki 20.

Zbiór	\mathcal{L}_{t_x}	\mathcal{L}_{t_y}	\mathcal{L}_s	\mathcal{L}
Treningowy	0,110	0,060	0,169	3,67
Walidacyjny	0,190	0,082	0,340	5,73

Tabela 5.4. Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali oraz pełnej straty modelu, jakie osiągnął model skupiający w fazie przetrenowania na klasie rowerów ze zbioru PASCAL VOC 2012.

W tym przypadku można stwierdzić, że przetrenowany model, jeśli chodzi o lokalizację obwiedni, dla zbioru treningowego, średnio myli się o 35 pikseli poziomo i 19 pikseli pionowo w koordynatach środka prostokąta oraz średnio docelowe obwiednie są 1,18 razy mniejsze lub większe, natomiast, w przypadku zbioru walidacyjnego, model średnio myli się o 61 pikseli poziomo i 26 pikseli pionowo w przypadku środka obwiedni oraz 1,4 razy w przypadku jej wielkości.

W przypadku zbioru MS COCO również rozpoczęto od treningu sieci skupiającej dla klasy osób. Początkowe rozłożenie klas w przygotowanym zbiorze danych stanowiło 53151 obrazów pustych (bez klasy) oraz 64115 obrazów z osobami. Ze względu na dobry rozkład kategorii, dodatkowe bilansowanie i augmentacja danych okazały się zbędne. Zastosowano tutaj tempo uczenia wynoszące 0.0001, a wagi w liczeniu straty dla translacji i skali wynosiły odpowiednio 12 i 3. Proces trenowania trwał 70 epok, natomiast najlepszym okazał się model z epoki 63.

Zbiór	\mathcal{L}_{t_x}	\mathcal{L}_{t_y}	\mathcal{L}_s	\mathcal{L}
Treningowy	0,028	0,021	0,048	1,66
Walidacyjny	0,119	0,068	0,275	6,07

Tabela 5.5. Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali oraz pełnej straty modelu, jakie osiągnął model skupiający w fazie przetrenowania na klasie osób ze zbioru MS COCO 2017.

5. Przeprowadzone badania

Można stwierdzić, że przetrenowany model, w przypadku zbioru treningowego, średnio myli się o 9 pikseli poziomo i 7 pikseli pionowo w przewidywaniu koordynatów środka prostokąta otaczającego oraz średnio docelowe obwiednie są 1,05 razy mniejsze lub większe, natomiast, w przypadku zbioru walidacyjnego, model średnio myli się o 38 pikseli poziomo i 22 pikseli pionowo w przypadku środka obwiedni oraz 1,32 razy co do jej wielkości.

Kolejnym procesem dla zbioru Pascal, był trening sieci skupiającej dla klasy samochodów. Początkowe rozłożenie klas również w przypadku tego zbioru było gorsze: 105015 obrazów pustych (bez klasy) oraz 12251 obrazów z samochodami. Także w tym przypadku zastosowano metodę usuwania przykładów o klasie 0, co spowodowało, że w zbiorze treningowym było tylko 3 razy więcej przykładów negatywnych. Zastosowano tutaj tempo uczenia wynoszące również 0.0001, wagi dla translacji i skali to odpowiednio 7 i 3. Proces trenowania trwał 70 epok, natomiast najlepszym okazał się model z epoki 58.

Zbiór	\mathcal{L}_{tx}	\mathcal{L}_{ty}	\mathcal{L}_s	\mathcal{L}
Treningowy	0,047	0,032	0,087	1,93
Walidacyjny	0,234	0,069	0,454	6,08

Tabela 5.6. Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali oraz pełnej straty modelu, jakie osiągnął model skupiający w fazie przetrenowania na klasie samochodów ze zbioru MS COCO 2017.

Można tutaj stwierdzić, że przetrenowany model, jeśli chodzi o lokalizację obwiedni, dla zbioru treningowego, średnio myli się o 15 pikseli poziomo i 10 pikseli pionowo co do koordynatów środka prostokąta oraz średnio o 1,1 razy co do ich wielkości, natomiast, w przypadku zbioru walidacyjnego, model średnio chybi o 75 pikseli poziomo i 22 pikseli pionowo co do środka obwiedni oraz 1,57 razy w jej wielkości.

Ostatnim treningiem dla sieci skupiających było przetrenowanie modelu na obrazach rowerów ze zbioru COCO. Początkowe niezbilansowanie klas w tym przypadku było drastyczne: 114014 obrazów pustych (bez klasy) oraz jedynie 3252 obrazów z rowerami, więc tutaj zastosowana również została metoda usuwania przykładów, co spowodowało, że w zbiorze treningowym było 9 razy więcej przykładów negatywnych. Zastosowano tutaj także tempo uczenia wynoszące 0.0001 i wagi dla translacji oraz skali jako odpowiednio 8 i 3. Proces trenowania trwał 100 epok, natomiast najlepszym okazał się model z epoki 92.

Zbiór	\mathcal{L}_{tx}	\mathcal{L}_{ty}	\mathcal{L}_s	\mathcal{L}
Treningowy	0,043	0,034	0,070	2,09
Walidacyjny	0,127	0,063	0,232	4,17

Tabela 5.7. Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali oraz pełnej straty modelu, jakie osiągnął model skupiający w fazie przetrenowania na klasie rowerów ze zbioru MS COCO 2017.

Tutaj można stwierdzić, że przetrenowany model, w lokalizacji środka obwiedni, dla zbioru treningowego, średnio myli się o 14 pikseli poziomo i 11 pikseli pionowo, a także średnio docelowe obwiednie są 1,07 razy mniejsze lub większe, natomiast, w przypadku zbioru walidacyjnego, model średnio myli się o 40 pikseli poziomo i 20 pikseli pionowo co do środka obwiedni oraz 1,26 razy w jej wielkości.

5.2.3. Przetrenowanie sieci klasyfikującej

Następnym w kolejności etapem treningu architektury było przetrenowanie klasyfikatora tak, aby dobrze klasyfikował poprawne wycinki obrazów ze zbioru. W tym przypadku zbiór danych przygotowany został zgodnie z opisem w rozdziale 3.5.3. Metryki, jakimi kierowano się przy wyborze najlepszego modelu to standardowe metryki dla zadania klasyfikacji wieloetykietowej, opisane w rozdziale 2.1.5, ale również zwrócono uwagę na to jak klasyfikator radzi sobie z pojedynczymi klasami.

Początkowy zbiór danych dla zadania składał się, dla zbioru PASCAL VOC 2007, z wycinek o odpowiednio 9804 przykładach pustych, 7639 przykładach osób, 480 przykładach samochodów i 1030 przykładach rowerów. Natomiast, dla MS COCO 2017, zbiór danych składał z odpowiednio: 546556 przykładów pustych, 262465 przykładów z osobami, 43867 przykładów z samochodami oraz 7113 przykładów z rowerami. Aby zaradzić dużemu niezbilansowaniu klas, w ramach przetwarzania danych dla obu zbiorów, zastosowano dosztukowywanie przykładów dla obrazów samochodów i rowerów, przy czym dodawane obrazy poddawane były transformacjom w celu uniknięcia przetrenowania modelu, a dokładniej operacjom losowego odbicia poziomego z prawdopodobieństwem $\frac{1}{2}$ oraz losowego usuwania fragmentu obrazu, również z prawdopodobieństwem $\frac{1}{2}$. Z tak otrzymanego zbioru, usunięto również część przykładów pustych tak, aby stanowiły one co najwyżej połowę całego zbioru.

W przypadku procesu trenowania, dla obu eksperymentów zastosowano te same optymalizatory, tzn. algorytm Adam, a także takie samo tempo uczenia się, wynoszące 0,00005. Dodatkowo zastosowano metodę ważenia przykładów przy liczeniu straty klasyfikacji, aby móc jeszcze bardziej zniwelować problem niezbilansowania danych.

5. Przeprowadzone badania

W przypadku zbioru PASCAL, trening trwał 30 epok, natomiast najlepszym modelem okazał się ten z epoki 12.

Zbiór	Dokł.	Mikro			Makro		
		Czułość	Precyzja	F1	Czułość	Precyzja	F1
Treningowy	0,998	0,996	0,996	0,996	0,995	0,989	0,992
Walidacyjny	0,968	0,937	0,937	0,937	0,907	0,924	0,915

Tabela 5.8. Zestawienie wartości metryk klasyfikujących, jakie osiągnął model klasyfikujący w fazie przetrenowania na zbiorze PASCAL VOC 2012.

Klasa	Dokładność	Czułość	Precyzja	F1
Pusta	0,943	0,943	0,945	0,944
Osoba	0,949	0,941	0,932	0,937
Samochód	0,995	0,875	0,923	0,898
Rower	0,987	0,869	0,895	0,882

Tabela 5.9. Zestawienie wartości metryk klasyfikujących, z dokładnością do poszczególnych klas, jakie osiągnął model klasyfikujący w fazie przetrenowania na zbiorze PASCAL VOC 2012.

Natomiast, w przypadku zbioru MS COCO 2017, trening trwał 80 epok, natomiast najlepszym modelem okazał się ten z epoki 72.

Zbiór	Dokł.	Mikro			Makro		
		Czułość	Precyzja	F1	Czułość	Precyzja	F1
Treningowy	0,997	0,994	0,994	0,994	0,996	0,988	0,992
Walidacyjny	0,968	0,936	0,936	0,936	0,902	0,861	0,880

Tabela 5.10. Zestawienie wartości metryk klasyfikujących, jakie osiągnął model klasyfikujący w fazie przetrenowania na zbiorze MS COCO 2017.

Klasa	Dokładność	Czułość	Precyzja	F1
Pusta	0,937	0,931	0,969	0,950
Osoba	0,953	0,949	0,902	0,925
Samochód	0,981	0,893	0,773	0,829
Rower	0,996	0,834	0,750	0,789

Tabela 5.11. Zestawienie wartości metryk klasyfikujących, z dokładnością do poszczególnych klas, jakie osiągnął model klasyfikujący w fazie przetrenowania na zbiorze MS COCO 2017.

5.2.4. Dostrajanie architektury

Ostatnim elementem tego badania było dostrojenie całej architektury poprzez połączenie wszystkich przetrenowanych modeli w jeden i sprawdzenie, jak model radzi sobie w zadaniu detekcji obiektów na obrazie, poprzez wyznaczenie osiąganych przez niego metryk $mAP@0.5$ oraz $mAP@0.5 : 0.05 : 0.95$ na badanych zbiorach.

W obu przypadkach zbiór obrazów był tak przygotowany, aby był identyczny z tym, który użyto do trenowania i walidacji modelu Faster R-CNN, w podrozdziale 5.2.1 i zgodnie z opisem w podrozdziale 5.1.3.

W przypadku badania dla zbioru PASCAL VOC 2012 dystrybucja obiektów na obrazach wyglądała następująco: 7639 obiektów ludzi, 480 obiektów samochodów i 1030 obiektów rowerów. W ramach samego zbioru treningowego postanowiono usunąć przypadki obrazów niezawierających obiektów z żadnych z 3 badanych klas, a także ograniczono liczbę przypadków, gdzie na obrazie występowały tylko ludzie, aby poradzić sobie z problemem niebilansowania kategorii w danych. W eksperymencie zastosowano optymalizator SGD (podobnie jak przy przetrenowywaniu sieci skupiających) z tempem uczenia wynoszącym 0,0001, trening trwał 23 epoki, przy czym najlepszym okazał się model z epoki 16.

Zbiór danych	$mAP@0.5$	$mAP@0.5 : 0.05 : 0.95$
PASCAL VOC 2012	0,545	0,324

Tabela 5.12. Zestawienie wyników metryk $mAP@0.5$ i $mAP@0.5 : 0.05 : 0.95$, jakie osiągnął model Focus CNN na zbiorze PASCAL VOC 2012.

Natomiast w przypadku badania zbioru MS COCO 2017 dystrybucja obiektów na obrazach była następującą: 64115 obiektów ludzi, 12251 obiektów samochodów i 3252 obiektów rowerów. Również tutaj, w ramach jedynie zbioru treningowego, usunięto przypadki obrazów niezawierających obiektów z żadnych z 3 badanych klas, oraz zmniejszono o połowę liczbę przykładów, gdzie występowały jedynie ludzie. W badaniu zastosowano dokładnie ten sam algorytm optymalizujący, SGD, a także tempo uczenia wynoszące 0,0001. Trening w tym przypadku trwał również 23 epoki, przy czym najlepszym okazał się model z epoki 20.

Zbiór danych	$mAP@0.5$	$mAP@0.5 : 0.05 : 0.95$
MS COCO 2017	0,499	0,301

Tabela 5.13. Zestawienie wyników metryk $mAP@0.5$ i $mAP@0.5 : 0.05 : 0.95$, jakie osiągnął model Focus CNN na zbiorze MS COCO 2017.

5.3. Zbadanie wpływu rotacji na wynik sieci

Istotnym badaniem było również sprawdzenie, jaki wpływ ma występowanie rotacji w wyjściach modelu skupiającego na całe działanie architektury. Jako że model Faster R-CNN nie został stworzony do predykcji kąta obrotu, pod jakim obiekty znajdują się na obrazie, to w przypadku tego badania porównane zostało działanie wytrenowanego modelu z rotacją do standardowego modelu Focus-CNN, opisanego w poprzednim eksperymencie, a badania wykonane zostały na zbiorze PASCAL VOC 2012.

Zgodnie z tym, konieczny był najpierw proces przetrenowania części architektury. O ile nie jest potrzebne ponowne przetrenowywanie klasyfikatora i można wykorzystać wagi otrzymane w poprzednim badaniu, to proces ten jest niezbędny w przypadku sieci skupiających. Przed rozpoczęciem, ważne było specjalne przygotowanie danych, poprzez sztuczne obracanie obrazów ze zbioru PASCAL, co zostało opisane w 3.5.2.

Zastosowano tutaj te same metryki co w przypadku poprzedniego badania. Najlepszym optymalizatorem okazał się tutaj również algorytm stochastycznego spadku gradientowego (ang. SGD) i tempo uczenia wynoszące 0,0001.

Proces trenowania dla obrazów z osobami trwał 80 epok, natomiast najlepszym okazał się model z epoki 73. Wagi w stracie dla translacji, skali i rotacji to odpowiednio 12, 12 i 3.

Zbiór	\mathcal{L}_{tx}	\mathcal{L}_{ty}	\mathcal{L}_s	\mathcal{L}_θ	\mathcal{L}
Treningowy	0,026	0,021	0,055	0,058	1,91
Walidacyjny	0,087	0,062	0,221	0,122	5,33

Tabela 5.14. Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali, straty rotacji oraz pełnej straty modelu, jakie osiągnął model skupiający z rotacją w fazie przetrenowania na klasie osób ze zbioru PASCAL VOC 2012.

Na podstawie wyników można stwierdzić, że przetrenowany model, w zadaniu lokalizacji środka obwiedni, w przypadku zbioru treningowego, osiągnął nieco lepsze wartości strat dla translacji i skali niż w przypadku standardowego podejścia (przedstawionego w tabeli 5.2), natomiast, w przypadku zbioru walidacyjnego, wartości straty się nieco pogorszyły. Jeśli chodzi o rotację, w przypadku zbioru treningowego model średnio popełnia błąd o około $3,3^\circ$, podczas gdy, w przypadku zbioru walidacyjnego, o około 22° .

W przypadku obrazów z samochodami trening trwał również 80 epok, natomiast najlepszym okazał się model z epoki 76. Wagi w stracie dla translacji, skali i rotacji to odpowiednio 7, 7 oraz 3.

Zbiór	\mathcal{L}_{t_x}	\mathcal{L}_{t_y}	\mathcal{L}_s	\mathcal{L}_θ	\mathcal{L}
Treningowy	0,034	0,032	0,092	0,112	2,296
Walidacyjny	0,082	0,067	0,148	0,247	3,570

Tabela 5.15. Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali, straty rotacji oraz pełnej straty modelu, jakie osiągnął model skupiający z rotacją w fazie przetrenowania na klasie samochodów ze zbioru PASCAL VOC 2012.

W przypadku tego badania można zauważyc, że model osiąga lepsze wartości strat niż jego odpowiednik w podejściu bez rotacji (w tabeli 5.3). Jeśli chodzi o rotację, w przypadku zbioru treningowego model średnio popełnia błąd o około 20° , podczas gdy, w przypadku zbioru walidacyjnego, jest to już około 44° .

Dla obrazów z rowerami proces trenowania trwał 50 epok, natomiast najlepszym okazał się model z epoki 41. Wagi w stracie dla translacji, skali i rotacji to odpowiednio 8, 8 oraz 3.

Zbiór	\mathcal{L}_{t_x}	\mathcal{L}_{t_y}	\mathcal{L}_s	\mathcal{L}_θ	\mathcal{L}
Treningowy	0,055	0,044	0,123	0,088	2,981
Walidacyjny	0,157	0,089	0,405	0,118	6,395

Tabela 5.16. Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali, straty rotacji oraz pełnej straty modelu, jakie osiągnął model skupiający z rotacją w fazie przetrenowania na klasie rowerów ze zbioru PASCAL VOC 2012.

Można tu stwierdzić, że model z rotacją, dla zbioru treningowego, osiąga lepsze wartości straty, podczas gdy, dla zbioru walidacyjnego, wyniki te są nieco gorsze, szczególnie w wartości skali. Model przewiduje kąt obrotu obiektów, w przypadku zbioru treningowego, ze średnim błędem 15° , podczas gdy, w przypadku zbioru walidacyjnego, myli się o około 21° .

Na końcu, połączono przetrenowane modele skupiające z klasyfikatorem w architekturę, którą dostrojono na tych samych danych, co w przypadku podejścia standardowego. Trening trwał 30 epok, natomiast najlepszym modelem okazał się ten z epoki 23. Osiągnął on jednak gorsze wyniki niż zarówno Focus-CNN bez rotacji, jak i Faster R-CNN.

Zbiór danych	$mAP@0.5$	$mAP@0.5 : 0.05 : 0.95$
PASCAL VOC 2012	0,362	0,106

Tabela 5.17. Zestawienie wyników metryk $mAP@0.5$ i $mAP@0.5 : 0.05 : 0.95$, jakie osiągneły badane modele detekcji obiektów na obrazie na zbiorze PASCAL VOC 2012.

5. Przeprowadzone badania

Interesującym spostrzeżeniem jest to, że klasyfikator w modelu z zastosowaną rotacją, osiąga lepsze wartości pod względem metryk makro-precyzji i makro-f1, niż klasyfikator w podejściu bez rotacji, a z powodu, że oba rozpoczęły dostrajanie z klasyfikatorami o tych samych wagach, można stwierdzić, że lepszy wynik osiągany jest dzięki zawartemu mechanizmowi rotacji.

Model	Makro-precyzja	Makro-F1
Focus-CNN	0,618	0,676
Focus-CNN z rotacją	0,671	0,697

Tabela 5.18. Zestawienie wyników metryk makro-precyzji i makro-F1, jakie osiągnął model Focus CNN z rotacją i bez rotacji na zbiorze PASCAL VOC 2012.

5.4. Zbadanie wpływu pełnego przepływu gradientu na wynik sieci

Za ważne uznane zostało również sprawdzenie, jak dostosowanie sieci wpływa na działanie całej architektury. Korzystne działanie pełnego przepływu gradientu mogłoby być zauważone, poprzez wzrost metryki mAP wraz z kolejnymi epokami, gdyż oznacza to, że działanie klasyfikatora wpływa na proponowanie regionów przez sieci skupiające. Natomiast, wzrost metryk klasyfikujących, oznaczałby, że klasyfikator zyskuje na coraz lepszym działaniu sieci skupiających.

Dlatego też porównano wartości metryk klasyfikujących i metryk do detekcji (mAP) dla modelu przed rozpoczętym treningiem i na końcu treningu.

Epoka	Dokł.	Mikro			Makro			MAP@	
		Czuł.	Prec.	F1	Czuł.	Prec.	F1	0.5	0.5:0.95
0	0,92	0,84	0,84	0,84	0,70	0,68	0,69	0,41	0,22
20	0,94	0,87	0,87	0,87	0,76	0,68	0,70	0,50	0,30

Tabela 5.19. Zestawienie wartości metryk klasyfikujących i metryk detekcji dla modelu Focus CNN przed dostrajaniem i po dostrojeniu na zbiorze MS COCO 2017.

Epoka	Dokł.	Mikro			Makro			MAP@	
		Czuł.	Prec.	F1	Czuł.	Prec.	F1	0.5	0.5:0.95
0	0,90	0,80	0,80	0,80	0,69	0,67	0,68	0,47	0,26
16	0,92	0,84	0,84	0,84	0,78	0,62	0,68	0,54	0,32

Tabela 5.20. Zestawienie wartości metryk klasyfikujących i metryk detekcji dla modelu Focus CNN przed dostrajaniem i po dostrojeniu na zbiorze PASCAL VOC 2012.

5.5. Przykłady działania

W tym podrozdziale zamieszczone zostały przykłady predykcji całej architektury wraz z omówieniem niektórych błędnych i poprawnych decyzji modelu.

Model, potrafi radzić sobie w przypadkach, gdy jest kilka obiektów na obrazie, ale o różnych klasach, nawet jeśli bardzo na siebie nachodzą, co pokazują przykłady z rys. 5.1.



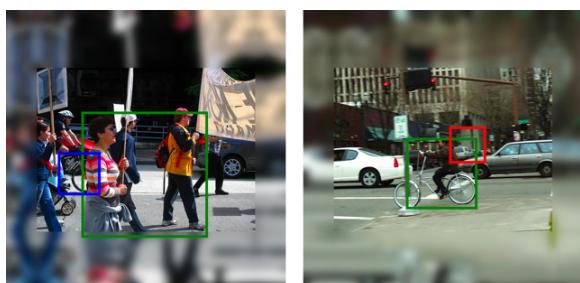
Rysunek 5.1. Przykłady poprawnych detekcji przy kilku obiektach różnej kategorii.

Można również zauważyc, że model radzi sobie dobrze, gdy jest kilka obiektów na obrazie o tej samej klasie i poprawnie wybiera ten, który jest najbliżej środka, co pokazują przykłady z rys. 5.2.



Rysunek 5.2. Przykłady poprawnych detekcji przy kilku obiektach tej samej kategorii.

Model potrafi również wykonywać poprawne predykcje, nawet gdy obiekty są przysłonięte, co pokazują przykłady z rys. 5.3.



Rysunek 5.3. Przykłady poprawnych detekcji przy obiektach przysłoniętych.

5. Przeprowadzone badania

Problematyczne dla modelu wydają się sytuacje, gdy jest wiele obiektów tej samej kategorii bardzo blisko siebie. Wtedy, model ma w zwyczaju zwracać obwiednię otaczającą grupę osób, tak jak to widać w przypadku przykładów z rys. 5.4.



Rysunek 5.4. Przykłady błędnych detekcji przy wielu obiektach tej samej kategorii blisko siebie.

Model posiada też niekiedy problem, gdy obiekty są wyraźnie widoczne, natomiast są wyjątkowo małe, tak jak to widać w przypadku przykładów z rys. 5.5.



Rysunek 5.5. Przykłady błędnych detekcji przy małych obiektach na obrazie.

6. Podsumowanie

Na podstawie przeprowadzonych eksperymentów, zarówno dla zbioru PASCAL VOC 2012 jak i MS COCO 2017, można stwierdzić, że, przedstawiona w ramach tej pracy architektura Focus-CNN, w wersji nieuwzględniającej parametru rotacji, pokonuje w rozwiązywanym zadaniu detekcji obiektów na obrazie swojego bezpośredniego konkurenta, model Faster R-CNN. Focus-CNN, w prostszej wersji, zdołał osiągnąć, w przypadku obu zestawów danych, lepsze wyniki zarówno dla metryki $mAP@0.5$ jak i $mAP@0.5 : 0.05 : 0.95$, co oznacza, że nie tylko wykonuje on więcej poprawnych predykcji, ale także przewidywane przez niego obwiednie są znacznie bliższe prawdziwym, co wskazuje tę architekturę jako potencjalną do stania się jednym z najlepszych detektorów, jednak wymaga to wciąż rozszerzenia jej do obsługiwanego wielu obiektów tej samej kategorii na obrazie. Fakt ten wskazuje, że wykonana w ramach tej pracy implementacja jest poprawna i stanowi dobrą bazę do rozwinięcia przedstawionej koncepcji w przyszłości. Ważne jest zwrócenie uwagi na istotność poprawnego przetrenowania sieci skupiających i sieci klasyfikującej, aby ostateczny model osiągnął dobre wyniki, natomiast wysokie wartości metryk również przy przetrenowywaniu, potwierdzają słuszność idei, jakie zostały w ramach tej pracy wprowadzone.

Dodatkowo eksperiment sprawdzający całkowitą różniczkowalność operacji wykonywanych przez architekturę, udowodnił, że posiadanie pełnego przepływu gradientu gwarantuje wzmacnienie współpracy między poszczególnymi częściami architektury, ponieważ, dzięki procesowi dostrajania, udało się ulepszyć działanie sieci skupiających, bazując na stracie klasyfikatora. Ukazało się to w zwiększeniu się wartości metryk mAP wraz z dłuższym procesem trenowania w przypadku badań dla obu zbiorów obrazów. Wynik eksperymentu udowodnił, że przedstawione nowatorskie podejście w zadaniu detekcji pasuje do rozwiązywanego problemu i może potencjalnie w przyszłości zastąpić dotychczasowe rozwiązania.

Analizując przykłady predykcji sieci, można zauważyć, że architektura Focus-CNN rozwiązuje zadanie, jakie zostało jej postawione. Udaje jej to się, szczególnie gdy działa ona na obrazach, gdzie obiekty są większych rozmiarów. Potrafi ona znajdować obiekty przysłonięte na obrazie i ukryte za obiektami innych klas. W przypadku kilku obiektów tej samej kategorii na obrazie dobrze wskazuje ten, który jest najbliżej środka obrazu, co jest zgodne z przedstawioną początkowo koncepcją. Gorzej radzi sobie, co jest standardowym problemem dla modeli neuronowych w detekcji, w przypadku wielu małych obiektów tej samej klasy, znajdujących się blisko siebie, jak i pojedynczych, bardzo małych obiektów. Nie zmienia to jednak tego, że przykłady pokazujące działanie sieci udowadniają potencjał przedstawionej koncepcji i zgodność wykonanej implementacji z założeniami.

6. Podsumowanie

Analizując wyniki zamieszczonych w ramach tej pracy eksperymentów, nie można wskazać, że uwzględnienie kąta obrotu w wyjściu sieci skupiającej i aplikowanie transformacji obrazu o rotację gwarantuje lepszy wynik całej architektury. W przypadku badań wykonanych na zbiorze PASCAL VOC 2012 nie udało się osiągnąć lepszej skuteczności architektury z rotacją w detekcji niż dla tego samego rozwiązania, ale pomijającego obrót. Otrzymany model osiągnął gorsze wyniki niż podejście Faster R-CNN, co wskazuje, że przedstawiona koncepcja z obrotem nie może na ten moment równać się z najlepszymi detektorami, natomiast nie powinna być zupełnie skreślona, gdyż osiąga wyniki ukazujące poprawne jej działanie, tylko w mniejszej liczbie przypadków. W trakcie przetrenowywania zauważono korzystne działanie rotacji na minimalizację strat translacji i straty skali na zbiorze treningowym (modele z rotacją osiągały tu lepsze wartości straty), natomiast niestety można było spostrzec destrukcyjny wpływ na te wartości w przypadku zbioru testowego, co oznacza, że rozwiązywanie zadanie jest trudne pod względem generalizacji, a kąt obrotu obrazu powiązany jest ściśle z pozostałymi parametrami.

Powodu gorszej dyspozycji podejścia z rotacją można dopatrywać się również w samym procesie przetrenowywania, gdzie konieczne jest sztuczne obracanie obrazów ze zbioru danych, aby móc uczyć model przewidywać kąt obrotu, podczas gdy ostateczny zbiór danych posiada jedynie oryginalne, nieobrócone obrazy. Najbardziej tracą na tym sieci skupiające dla klas, dla których przykładów jest najmniej w zbiorze danych, ponieważ posiadają one relatywnie mało obrazów, które pojawiają się w zbiorze ostatecznym, a mniej więcej połowa z nich zostaje sztucznie przekręcona. Fakt ten potwierdza wcześniej wspomniany duży wpływ procesu przetrenowania na ostateczne działanie całej architektury.

W badaniu zauważono również korzystny wpływ rotacji na decyzje klasyfikatora, gdyż rozwiązanie z obrotem osiągnęło lepszy wynik w metryce makro-precyzji oraz makro-F1. Jest to zgodne z intuicją, gdyż wyższe wartości tych metryk oznaczają dokładniejsze działanie samego klasyfikatora, a taki cel ma stosowanie rotacji. Oznacza to, że aplikowanie obrotu na obrazach ze zbioru, chociaż nie daje lepszych wyników w zadaniu detekcji, to usprawnia działanie samego klasyfikatora, co wskazuje potencjał tego parametru w rozwiązywanym problemie.

Podsumowując, przeprowadzone, w ramach tej pracy inżynierskiej, badania wskazały słuszność zastosowania przedstawionego mechanizmu atencji w rozwiązywanym zadaniu detekcji obiektów na obrazie; pokazały, że wprowadzona architektura może równać się z najlepszymi detektorami, a także potwierdziły, że wykonana implementacja jest poprawna i zgodna z całą opisaną koncepcją.

Bibliografia

- [1] *StradVision - strona domowa*, Dostęp zdalny 06.09.2023, Dostępny w Internecie: <https://stradvision.com/sv/en>.
- [2] *Fighting Crime and Insecurity in Nigeria: An Intelligent Approach - Scientific Figure on ResearchGate*. Dostęp zdalny 06.09.2023, Dostępny w Internecie: https://www.researchgate.net/figure/Facial-Recognition-Surveillance_fig3_351022846.
- [3] A. Mansour, A. Hassan, W. M. Hussein i E. Said, “Automated vehicle detection in satellite images using deep learning”, *IOP Conference Series: Materials Science and Engineering*, t. 610, nr. 1, s. 012 027, wrz. 2019. DOI: 10.1088/1757-899X/610/1/012027. adr.: <https://dx.doi.org/10.1088/1757-899X/610/1/012027>.
- [4] *Respo.Vision - strona domowa*, Dostęp zdalny 06.09.2023, Dostępny w Internecie: <https://respo.vision/>.
- [5] *Python - strona domowa*, Dostęp zdalny 06.09.2023, Dostępny w Internecie: <https://www.python.org/>.
- [6] *PyTorch - strona domowa*, Dostęp zdalny 06.09.2023, Dostępny w Internecie: <https://pytorch.org/>.
- [7] *Albumentations - strona domowa*, Dostęp zdalny 06.09.2023, Dostępny w Internecie: <https://albumentations.ai>.
- [8] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid i S. Savarese, “Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression”, 2019. arXiv: 1902.09630 [cs.CV].
- [9] H. Jain i S. Nandy, “Incremental Training for Image Classification of Unseen Objects”, 2019. DOI: 10.13140/RG.2.2.10266.47046.
- [10] J. Redmon, S. Divvala, R. Girshick i A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection”, 2016. arXiv: 1506.02640 [cs.CV].
- [11] R. Girshick, J. Donahue, T. Darrell i J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation”, 2014. arXiv: 1311.2524 [cs.CV].
- [12] T. H. Phan i K. Yamamoto, “Resolving Class Imbalance in Object Detection with Weighted Cross Entropy Losses”, arXiv, 2020. DOI: 10.48550/ARXIV.2006.01413. adr.: <https://arxiv.org/abs/2006.01413>.
- [13] R. Girshick, “Fast R-CNN”, 2015. arXiv: 1504.08083 [cs.CV].
- [14] S. Ren, K. He, R. Girshick i J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, 2016. arXiv: 1506.01497 [cs.CV].
- [15] K. He, G. Gkioxari, P. Dollár i R. Girshick, “Mask R-CNN”, 2018. arXiv: 1703.06870 [cs.CV].
- [16] K. He, X. Zhang, S. Ren i J. Sun, “Deep Residual Learning for Image Recognition”, arXiv, 2015. DOI: 10.48550/ARXIV.1512.03385. adr.: <https://arxiv.org/abs/1512.03385>.

6. Bibliografia

- [17] D. Bahdanau, K. Cho i Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate”, 2016. arXiv: 1409.0473 [cs.CL].
- [18] T. Tracey, *Language Translation with RNNs*, Dostęp zdalny 06.09.2023, Dostępny w Internecie: <https://towardsdatascience.com/language-translation-with-rnns-d84d43b40571>, 2019.
- [19] G. Loyer, *Attention Mechanism*, Dostęp zdalny 06.09.2023, Dostępny w Internecie: <https://blog.floydhub.com/attention-mechanism/>, 2019.
- [20] Z. Zohourianshahzadi i J. K. Kalita, “Neural attention for image captioning: review of outstanding methods”, 5, t. 55, Springer Science i Business Media LLC, list. 2021, s. 3833–3862. DOI: 10.1007/s10462-021-10092-2. adr.: <https://doi.org/10.1007%2Fs10462-021-10092-2>.
- [21] K. Xu, J. Ba, R. Kiros i in., “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, arXiv, 2015. DOI: 10.48550/ARXIV.1502.03044. adr.: <https://arxiv.org/abs/1502.03044>.
- [22] M. Jaderberg, K. Simonyan, A. Zisserman i K. Kavukcuoglu, “Spatial Transformer Networks”, arXiv, 2015. DOI: 10.48550/ARXIV.1506.02025. adr.: <https://arxiv.org/abs/1506.02025>.
- [23] *PyTorch - TorchVision object detection finetuning tutorial using faster-rcnn*, Dostęp zdalny 06.09.2023, Dostępny w Internecie: https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html.
- [24] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn i A. Zisserman, *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*, Dostęp zdalny 06.09.2023, Dostępny w Internecie: <http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html>.
- [25] *COCO - Common Objects in Context dataset*, Dostęp zdalny 06.09.2023, Dostępny w Internecie: <https://cocodataset.org/#home>.
- [26] tasq-ai, *Raport on Top 15 Public Datasets for Object Detection in 2023*, Dostęp zdalny 06.09.2023, Dostępny w Internecie: <https://www.tasq.ai/blog/top-15-public-datasets-for-object-detection/>.

Spis rysunków

1.1 Przykładowy widok z kamery samochodu autonomicznego. Obiekty są lokalizowane, a także kategoryzowane przy użyciu koloru prostokątów. Grafika pochodzi z [1].	10
1.2 Przykłady zastosowań dla zadań detekcji obiektów na obrazie. Kolejno: rozpoznawanie twarzy, analiza zdjęcia satelitarnego, śledzenie obiektów. Źródła obrazów: [2], [3], [4].	10
2.1 Przykład obrazu z trzema obiektami i ich obwiedniami. Źródło: [7].	14
2.2 Schemat liczenia miary IoU dla dwóch dowolnych obwiedni.	16
2.3 Przykład zastosowania techniki tłumienia niemaksymalnego (NMS). Źródło: [9].	16
2.4 Przykład konstruowania tabeli z predykci modelu w metryce <i>mAP</i> liczonej dla obiektów psów. Na grafice (A) przedstawiono 3 obrazy uwzględnione w przykładzie, gdzie na zielono zaznaczono prostokąty opisujące prawdziwe obiekty, a na czerwono - predykcje modelu (obwiednie i pewność co do klasy obiektów); natomiast na grafice (B) przedstawiono tabelę skonstruowaną na podstawie tych predykcji.	19
2.5 Kontynuacja przykładu liczenia miary <i>AP</i> dla pojedynczej klasy. Najpierw (grafika A) predykcje sortowane są względem wartości pewności malejąco i wyliczana jest precyza oraz czułość dla każdego z progów pewności. Na końcu (grafika B) rysowany jest wykres zależności precyzji od czułości i wyznaczane jest pole pod tym wykresem.	19
2.6 Porównanie działania architektur jednostopniowych i dwustopniowych do detekcji obiektów na obrazie. W podpunkcie (b) przedstawione zostało działanie sieci jednostopniowych, gdzie jednocześnie produkowane są kategorie i lokalizacje obiektów na obrazie. W podpunkcie (a) zostało przedstawione działanie sieci dwustopniowych, zakładających pracę w dwóch etapach - propozycję regionów, w których mogą być obiekty, a następnie pracę na tych regionach w celu skategoryzowania obiektów. Źródło rysunku: [12]. . .	20
2.7 Schemat działania modelu R-CNN.	22
2.8 Schemat działania modelu Fast R-CNN.	23
2.9 Schemat działania modelu Faster R-CNN.	24
2.10 Schemat bloku rezydualnego, gdzie do wyniku operacji warstw splotowych dodawane jest przekształcenie tożsamościowe wejścia do bloku. Źródło rysunku: [16].	25
2.11 Schemat działania architektury autokodera z sieciami rekurencyjnymi. Można zauważyć, że na początku działa tutaj koder przechodzący po całym zdaniu tłumaczonym i produkujący ostatecznie pojedyncze wyjście - stan po przetworzeniu całego zdania, które otrzymuje dekoder i iteracyjnie produkuje zdanie będące tłumaczeniem. Źródło: [18].	27

2.12 Schemat działania architektury autokodera z atencją. Każde słowo zdania wejściowego przekształcane jest na adnotacje, a te, przy każdej generacji dekodera, są odpowiednio ważone, tak aby tłumaczenie danego słowa zależało jedynie od fragmentów wejścia. Źródło: [19].	28
2.13 Przykłady zachowania oczekiwanej od mechanizmu atencji w zadaniu generowania opisów do zdjęć - produkując konkretne słowo opisu, dekoder powinien zwracać uwagę na te fragmenty obrazu, które są w danej chwili istotne. Źródło: [21].	28
3.1 Grafika przedstawiająca schemat działania całej architektury z uwzględnieniem wejść i wyjść każdej części. Na obrazie wejściowym (na lewo) prostokąt czerwony oznacza obwiednię zaanotowaną w ramach zbioru, a niebieski pokazuje oczekiwane przez klasyfikator przybliżenie obiektu. Zauważalna jest różnica między wejściem do klasyfikatora a obrazem wyciętym przez kwadrat będący wyjściem architektury - rotacja używana jest do wspomagania klasyfikatora w jego decyzji i nie ma zastosowania przy ustalaniu ostatecznych koordynatów obiektu.	30
3.2 Schemat architektury sieci skupiającej, gdzie obraz wejściowy to kwadratowy obraz RGB (3-kanałowy) o wymiarach 640×640 , który ulega zagnieżdżeniu przez warstwy konwolucyjne sieci rezydualnej do wektora, który stanowi wejście do zestawu warstw gęstych przewidujących prawdopodobieństwo wystąpienia obiektu oraz do zestawu warstw gęstych przewidujących skalary transformacji. W ramach rysunku pominięto budowę sieci ResNet-34, składającej się z 34 bloków rezydualnych, których działanie zostało opisane w 2.3. Skrót "D+LR" na połączeniach między warstwami oznacza, że bezpośrednio za warstwą gęstą występuje w architekturze warstwa Dropout oraz aktywacja przy użyciu funkcji LeakyReLU.	33
3.3 Przykład sytuacji, gdy niekwadratowy obiekt znajduje się przy krawędzi obrazu.	36
3.4 Schemat architektury sieci klasyfikującej, gdzie na wejściu otrzymywany jest obraz, a wyjście stanowi 4-elementowy wektor, zwracający prawdopodobieństwo każdej z klas.	37
3.5 Grafiki przedstawiają wprowadzenie wzorów na wysokość i szerokość prostokąta w pełni otaczającego obraz obrócony względem jego środka o kąt θ zgodnie z ruchem wskazówek zegara. Grafika (a) przedstawia stosunek wielkości prostokątów otaczających obraz przed i po obrocie, natomiast grafika (b) przedstawia wykorzystanie wzorów trygonometrycznych w celu otrzymania ostatecznych wielkości.	38

3.6 Kolejne etapy dodawania ramki w postaci odbicia z silnym rozmyciem Gaussowskim do obrazów. Najpierw obraz sprowadzany jest do największego możliwego kwadratu w ramach zbioru, następnie występuje rotacja, co wiąże się z dodaniem kolejnej ramki.	39
3.7 Wizualizacja przygotowania danych dla treningu sieci klasyfikującej. Obwiednia każdego obiektu zamieniana jest w najmniejszy kwadrat w pełni go wypełniający, który następnie jest skalowany do rozmiaru wejścia klasyfikatora. Jeśli klasa obiektu jest rozpatrywana przez architekturę - etykieta przypisywana jest zgodnie z kategorią, a jeśli nie, ustawiana na 0.	42
3.8 Schemat zastosowania rekurencji na obrazie z dużą liczbą obiektów nas interesujących. Obraz jest odpowiednio przesuwany i przybliżany do momentu aż nie pozostanie na nim pojedynczy obiekt danej kategorii.	44
4.1 Schemat działania, struktura i kolejne operacje wykonywane wewnątrz modułu Przestrzennego Transformatora (ang. Spatial Transformer Network Module). Źródło: [22].	48
4.2 Wynik zastosowania Przestrzennego Transformerera jako pierwszej warstwy gęstej w architekturze wytrenowanej w celu klasyfikacji zniekształconych grafik odręcznie zapisanych cyfr ze zbioru MNIST (grafiki zostały poddane losowym translacjom, rotacjom i skalom). Kolorową ramką zaznaczone zostały siatki punktów próbujących, które przedstawiają przekształconą przestrzeń obrazu o macierz transformacji. Trzecią kolumnę stanowi wynik zastosowania mechanizmu próbującego siatkę punktów na wejściowej grafice. Na końcu warstwy gęste działają na przekształconym obrazie i przyporządkowują cyfrę, która się na nim znajduje. Źródło: [22].	48
4.3 Wyniki eksperymentów optymalizacji zaimplementowanego modelu. W kolumnie (A) znajdują się obrazy, stanowiące wejście do sieci, w kolumnie (C) obrazy, jakie sieć powinna otrzymać, natomiast w kolumnie (B) - proponowane obrazy przez model.	49
4.4 Wyniki eksperymentów optymalizacji zaimplementowanego modelu na obrazie RGB.	50
4.5 Zestawienie grafik dla drugiego testu. W (A) można zobaczyć obrazy wejściowe dla całej architektury (na nich przetrenowywane są sieci skupiające), natomiast w (B) przedstawione są odpowiadające im grafiki będące zastosowaniem na nich transformacji o parametry, które powinny wyznaczać sieci skupiające.	50
5.1 Przykłady poprawnych detekcji przy kilku obiektach różnej kategorii.	65
5.2 Przykłady poprawnych detekcji przy kilku obiektach tej samej kategorii.	65
5.3 Przykłady poprawnych detekcji przy obiektach przysłoniętych.	65

5.4	Przykłady błędnych detekcji przy wielu obiektach tej samej kategorii blisko siebie.	66
5.5	Przykłady błędnych detekcji przy małych obiektach na obrazie.	66

Spis tabel

5.1	Zestawienie wyników metryk $mAP@0.5$ i $mAP@0.5 : 0.05 : 0.95$, jakie osiągnął model Faster R-CNN na zbiorach PASCAL VOC 2012 i MS COCO 2017.	54
5.2	Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali oraz pełnej straty modelu, jakie osiągnął model skupiający w fazie przetrenowania na klasie osób ze zbioru PASCAL VOC 2012.	56
5.3	Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali oraz pełnej straty modelu, jakie osiągnął model skupiający w fazie przetrenowania na klasie samochodów ze zbioru PASCAL VOC 2012.	56
5.4	Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali oraz pełnej straty modelu, jakie osiągnął model skupiający w fazie przetrenowania na klasie rowerów ze zbioru PASCAL VOC 2012.	57
5.5	Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali oraz pełnej straty modelu, jakie osiągnął model skupiający w fazie przetrenowania na klasie osób ze zbioru MS COCO 2017.	57
5.6	Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali oraz pełnej straty modelu, jakie osiągnął model skupiający w fazie przetrenowania na klasie samochodów ze zbioru MS COCO 2017.	58
5.7	Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali oraz pełnej straty modelu, jakie osiągnął model skupiający w fazie przetrenowania na klasie rowerów ze zbioru MS COCO 2017.	59
5.8	Zestawienie wartości metryk klasyfikujących, jakie osiągnął model klasyfikujący w fazie przetrenowania na zbiorze PASCAL VOC 2012.	60
5.9	Zestawienie wartości metryk klasyfikujących, z dokładnością do poszczególnych klas, jakie osiągnął model klasyfikujący w fazie przetrenowania na zbiorze PASCAL VOC 2012.	60

5.10 Zestawienie wartości metryk klasyfikujących, jakie osiągnął model klasyfikujący w fazie przetrenowania na zbiorze MS COCO 2017.	60
5.11 Zestawienie wartości metryk klasyfikujących, z dokładnością do poszczególnych klas, jakie osiągnął model klasyfikujący w fazie przetrenowania na zbiorze MS COCO 2017.	60
5.12 Zestawienie wyników metryk $mAP@0.5$ i $mAP@0.5 : 0.05 : 0.95$, jakie osiągnął model Focus CNN na zbiorze PASCAL VOC 2012.	61
5.13 Zestawienie wyników metryk $mAP@0.5$ i $mAP@0.5 : 0.05 : 0.95$, jakie osiągnął model Focus CNN na zbiorze MS COCO 2017.	61
5.14 Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali, straty rotacji oraz pełnej straty modelu, jakie osiągnął model skupiający z rotacją w fazie przetrenowania na klasie osób ze zbioru PASCAL VOC 2012.	62
5.15 Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali, straty rotacji oraz pełnej straty modelu, jakie osiągnął model skupiający z rotacją w fazie przetrenowania na klasie samochodów ze zbioru PASCAL VOC 2012.	63
5.16 Zestawienie wartości strat, odpowiednio: straty translacji poziomej, straty translacji pionowej, straty logarytmu skali, straty rotacji oraz pełnej straty modelu, jakie osiągnął model skupiający z rotacją w fazie przetrenowania na klasie rowerów ze zbioru PASCAL VOC 2012.	63
5.17 Zestawienie wyników metryk $mAP@0.5$ i $mAP@0.5 : 0.05 : 0.95$, jakie osiągneły badane modele detekcji obiektów na obrazie na zbiorze PASCAL VOC 2012. . .	63
5.18 Zestawienie wyników metryk makro-precyzji i makro-F1, jakie osiągnął model Focus CNN z rotacją i bez rotacji na zbiorze PASCAL VOC 2012.	64
5.19 Zestawienie wartości metryk klasyfikujących i metryk detekcji dla modelu Focus CNN przed dostrajaniem i po dostrojeniu na zbiorze MS COCO 2017. . .	64
5.20 Zestawienie wartości metryk klasyfikujących i metryk detekcji dla modelu Focus CNN przed dostrajaniem i po dostrojeniu na zbiorze PASCAL VOC 2012. . .	64