Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Radioelektroniki i Technik Multimedialnych

Praca dyplomowa

na studiach: Studia podyplomowe
Głębokie Sieci Neuronowe – Zastosowania w Mediach Cyfrowych

Detekcja i śledzenie piłki z wykorzystaniem współczesnych architektur głębokich sieci neuronowych

Jakub Rurak

Promotor mgr inż. Mikołaj Wieczorek

Pod nadzorem prof. dr hab. inż. Władysława Skarbka

Detekcja i śledzenie piłki z wykorzystaniem współczesnych architektur głębokich sieci neuronowych.

Streszczenie

Niniejsza praca dyplomowa podejmuje problematykę detekcji i śledzenia piłki z zastosowaniem głębokich sieci neuronowych. W pracy nad problemem detekcji została wykorzystana konwolucyjna sieć neuronowa posiadająca architektury Faster R-CNN i Retinanet. Na etapie badań przygotowano modyfikacje sieci neuronowej z wykorzystaniem FPN ResNet50 oraz MobileNetV2, z wykorzystaniem biblioteki Detectron2. Przygotowane modele testowano na zbiorach danych volleyball_easy i volleyball_difficult. Do śledzenia piłki został wykorzystany algorytm SORT wraz z modyfikacją oraz Deep Sort.

Słowa kluczowe: głębokie sieci neuronowe, detekcja piłki, Faster R-CNN, Resnet50, MobileNetV2 Sort, Deep Sort

Spis treści

1. Wstęp

Detekcja i śledzenie piłki jest jedną z podstawowych i bardziej użytecznych informacji przy analizie rozgrywek sportowych. Pomimo tego iż istnieją techniki Wizji Komputerowej (ang. Computer Vision), ustalenie dokładnej trajektorii szybko przemieszczającego się obiektów jest skomplikowanym zagadnieniem. Z powodu względnie dużej prędkości piłki możemy mieć do czynienia z rozmyciem w ruchu (ang. motion blur) na poszczególnych klatkach nagrań. W przypadku relacji telewizyjnych mamy również dodatkowo do czynienia z ruchem kamery co dodatkowo potęguje ten efekt. W zależności od ustawienia kamery dochodzi również do zakrycia obiektu lub jego wyjscia z kadru. Z tego względu skuteczność wykrywania piłki nie jest na bardzo wysokim poziomie przy standardowych metodach detekcji.
Najczęściej stosowanym systemem śledzeniem piłki stosowanym w profesjonalnych rozgrywkach sportowych jest system Hawkeye. Hawkeye jest używany między innymi do systemu „Challenge" w tenisie, piłce siatkowej oraz piłce nożnej przy technologii Goal-line. Przy użyciu tego systemu wykorzystywanych jest wiele kamer o wysokiej rozdzielczości i niskim czasie naświetlania. Kamery są ustawione nad boiskiem i ustawione są do niego pod różnymi kątami Obraz z kamer jest łączony i za pomocą triangulacji uzyskiwany jest bardzo dokładna trajektoria lotu obiektu.
System ten jest jednak bardzo drogi, przez co nadaje się on jedynie do profesjonalnych rozgrywek sportowych na najwyższych szczeblach. Motywacją pracy jest sprawdzenie, czy da się osiągnąć zbliżone rezultaty korzystając niskobudżetowych rozwiązań, np. nagrywając rozgrywki kamerą z własnego smartfonu.

Przykładowe zastosowania śledzenia piłki

- **Pomoc w sędziowaniu** – rozstrzyganie spornych decyzji. Dzięki informacji o dokładnej trajektorii piłki arbiter może wspomoć się nagraniem z analizą i wykorzystać tą wiedzę do sprawdzenia outu,
- **Narzędzie trenerskie** – informacje o trajektorii piłki mogą pomóc w analizie oraz korekcji techniki zawodników. Ustalenie tory lotu dodaje obiektywne parametry oceny zawodników przy wystawianiu piłki, zagrywce czy też ataku,
- **Transmisje telewizyjne** – wzbogacenie narzędzi do analizy komentatorskiej meczu, zwiększając ich walory wizualne.

Wykorzystując sztuczne sieci neuronowe (ang. Artificial Neural Networks) i powiązane z nimi techniki sprawdzam skuteczność detekcji oraz śledzenia piłki do piłki siatkowych na nagraniach amatorskich jak i profesjonalnych rozgrywek.

W pracy nad problemem detekcji została wykorzystana konwolucyjna sieć neuronowa posiadająca architekturę Faster R-CNN [5] i Retinanet [7]. Na etapie badań przygotowano modyfikacje sieci neuronowej z wykorzystaniem FPN ResNet50 [13] oraz MobileNetV2 [11] z wykorzystaniem biblioteki Detectron2. Przygotowane modele testowano na zbiorze danych Volleyball_easy i na stworzonym przeze mnie zbiorze Volleyball_difficult. Problem do śledzenia piłki został wykorzystany algorytm Sort [16] wraz z modyfikacją oraz Deep SORT [17] .

Zawartość pracy

W pracy zostały zawarte następujące elementy:

- Rozdział 1 - Wstęp
- Rozdział 2 - Omówienie metryk oceny detekcji,
- Rozdział 3 - Przedstawienie baz danych obrazów wykorzystanych w pracy do trenowania i ewaluacji modeli głębokich sieci neuronowych,
- Rozdział 4 - Przedstawienie zastosowanych metod detekcji obiektów,
- Rozdział 5 – Przedstawienie zastosowanych architerktur CNN do detekcji,
- Rozdział 6 – Przedstawienie zastosowanych metod śledzenia obiektów,
- Rozdział 7 - Opis przebiegu eksperymentu,
- Rozdział 8 - Porównanie wyników modeli i wnioski.

2. Metryki oceny detekcji obrazu

W procesie detekcji obiektu kluczowa jest prawidłowa ocena skuteczności. Metryki skuteczności używane w niniejszej pracy zostały omówione w tym punkcie.

W punkcie 2.5. została przedstawiona miara Coco AP, która została przygotowana do oceny modeli wykorzystanych w niniejszej pracy.

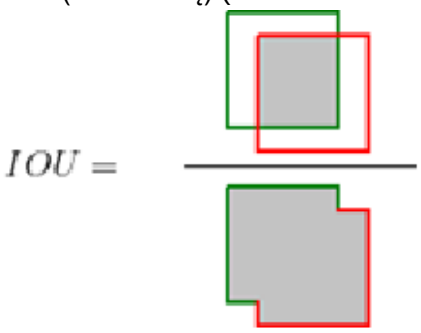
2.1. Intersection over Union

Intersection over Union jest to metryka która pozwala nam na ocenę jak bardzo zbliżony jest predykowanej obwiedni do rzeczywistej obwiedni. Dokładnie jest to stosunek części wspólnej obwiedni (ang. bounding box) do jego pola łącznego:

$$IOU = \frac{pole(B_p \cap B_{gt})}{pole(B_p \cup B_{gt})}$$

gdzie: $\mathbf{B_p}$ oznacza wykrytą obwiednię, $\mathbf{B_{gt}}$ oznacza obwiednię wskazaną

Rysunek poniżej ilustruje IoU pomiędzy wskazanym polem/obwiednią (w kolorze zielonym - ground truth bounding box) a wykrytym polem (obwiednią) (w kolorze czerwonym - detected bounding box).



Rysunek 2.1: IoU pomiędzy wskazanym polem (w kolorze zielonym - ground truth bounding box) a wykrytym polem (w kolorze czerwonym - detected bounding box).

Gdy wartość IoU wynosi 1 to znaczy, że pole wykryte jest równe polu wskazanemu.

2.2. Precision i Recall

Ustalamy pewną wartość progu IoU wedle której oceniamy czy detektor działa prawidłowo. Domyślnie jest to 0,5, ale może to być każda inna wartość w przedziale <0,1>

Podstawowe elementy Precision i Recall:

- **True Positive (TP)** - Wynik w którym model prawidłowo wykryje istniejący obiekt. IoU≥próg
- **False Positive (FP)** – Wynik w .którym model nieprawidłowo wykryje nieistniejący obiekt. IoU<próg
- **False Negative (FN) –** Wynik w którym model nie wykryje istniejącego obiektu
- **True Negative (TN):** Reprezentuje prawidłowy brak detekcji obiektu. Nie jest użyteczny w ocenie pracy detektora.

Precision

Precyzja jest to miara określająca dokładność rozpoznania w obrębie klasy. Określa się, jako stosunek True Positive do wszystkich uzyskanych detekcji dla danej klasy. Miara jest użyteczna gdy koszt False Positive jest wysoki, np w przypadku detekcji spamu. Precyzja definiowana jest podanym wzorem:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{Wszystkie\ Detekcje}$$

Recall

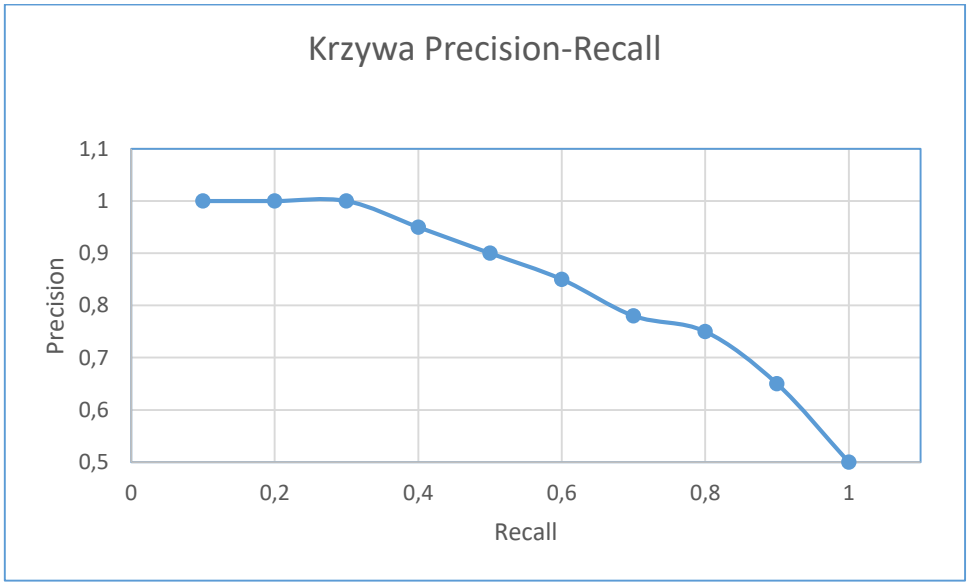
Recall (czułość) jest to miara określająca zdolnosc modelu do wykrycia wszystkich rozpatrywanych obiektów (ground truth). Miara jest użyteczna w przypadku dużego kosztu False Negative – np. w przypadku wykrywania chorób Jest to stosunek True Positive do wszystkich wskazanych obiektów, podany wzorem:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{Wszystkie\ obiekty}$$

2.3. Krzywa Precision-Recall

Krzywa PR to wykres zależności Precision (oznaczany na osi y) od Recall (oznaczany na osi x). Pożądane jest aby algorytm miał jednocześnie zarówno wysoki parametr Precision jak i Recall. Niestety większość algorytmów uczenia maszynowego muszą kompromisować między tymi wartościami. W naszym wypadku, zwiększając próg wykrywania (ang. confidence threshold), zwiększamy precyzję detekcji kosztem parametru recall. Oznacza to że więcej obiektów zostanie wykrytych poprawnie, kosztem niewykrycia niektórych z nich.

Krzywą Precision-Recall możemy ocenić, jak dobry jest nasz detektor. Jednym ze sposobów interpretacji tego wykresu jest policzenie pola pod wykresem (omawiany później Average Precision) – im wyższe jest pole tym detektor jest lepszy. Rysunek 2.2. przedstawia przykładowy wygląd krzywej P-R



Rysunek 2.2: Przykładowy kształt krzywej Precision-Recall

2.4. Average Precision

Średnia precyzja (ang. Average Precision - AP) jest sposobem reprezentacji krzywej Precision-Recall w jedną wartość reprezentującą średnią wszystkich precyzji. AP jest kalkulowane według wzoru poniżej:

$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k+1)] * Precisions(k)$$

Gdzie: $Recalls(n) = 0, Precisions(n) = 1, n = liczba\ prog\acute{o}w(thresholds)$

Podsumowując, AP jest średnią ważoną sum precyzji na każdym progu (ang. threshold), gdzie wagą jest zwiększenie w wartości Recall.

2.5. COCO Mean Average Precision

Detekcja obrazu ewaluowana jest przy różnych poziomach IoU, gdzie każdy z progów może spowodować inne predykcje niż przy innych progach.
Aby unormować precyzję dla różnych progów IoU wymyślono metrykę mAP, stosowana między innymi w konkursie COCO. Z definicji mAP to średnia wartość Averege Precision. W przypadku COCO mAP liczone jest poprzez uśrednianie AP dl wybranych IoU we wszystkich klasach obiektów
Charakterystyczny zapis AP @ [. 5: .95] odpowiada średniemu AP dla IoU od 0.5 do 0.95 z wielkością kroku 0.05.
W konkursie COCO AP to średnia dla 10 poziomów IoU w 80 kategoriach obiektów (AP @ [.50: .05:.95]: zaczynając od 0.5, a kończąc na IoU = 0.95 z krokiem 0.05).
Metryki COCO do detekcji można znaleźć pod linkiem: https://cocodataset.org/#detection-eval

3. Bazy danych treningowych i testowych

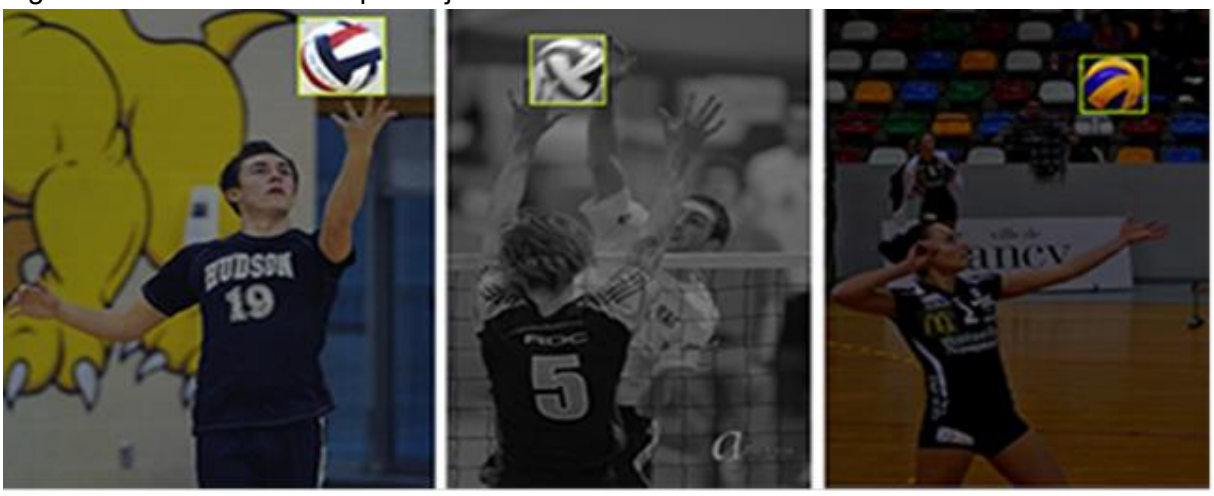
W niniejszej pracy w zagadnieniach detekcji piłki zostały użyte następujące bazy danych:

- **Volleyball_easy** – Dataset dostępny publicznie. Jest to podzbiór datasetu OpenImagesV4, zawierające jedynie piłki do siatkówki..
- **Volleyball_difficult** – Dataset własny, stworzony ze zrzutów ekranu z kilku nagrań siatkówki. Oznaczenie piłki wykonane samodzielnie.
- **Ball_noball** – Dataset stworzony na potrzeby trenowania feature extractora do Deep Sortu.

3.1. Volleyball_easy

Open Images [1] to zbiór danych ok. 9 milionów obrazków które zostały oznaczone przez opisy całego obrazka, oraz mają oznaczone obwiednie i relacje wizualne. Zestaw treningowy V4 zawiera 14,6 miliona obwiedni dla 600 klas przy 1,74 milionie obrazków, powodujących że to jeden z największych istniejących zbiorów z annotacjami lokalizacji obiektów. Obwiednie zostały oznaczone manualnie przez profesjonalnych annotatorów żeby zapewnić im dokładność i precyzję. Obrazy są bardzo różnorodne i często zawierają skomplikowane sceny z wieloma obiektami (średnio po 8,4 obiekta na obrazek. Co więcej, ten każdy z obrazków ma własne oznaczenie całego zdjęcia. Open Images można znaleźć pod linkiem: https://storage.googleapis.com/openimages/web/index.html

Volleyball_easy to zbiór obrazów który jest podzbiorem datasetu Open Images V4, zawierający oznaczenia obwiedni piłek do siatkówki. Został tak nazwany, ze względu na to że zadanie detekcji piłki na większości tych zdjęć jest stosunkowo proste. Piłki są wyraźne, w wysokiej rozdzielczości i stanowią dużą część obrazu. Zbiór danych składa się z 438 obrazków do trenowania sieci oraz 50 do testowania sieci. Przykładowe obrazki z adnotacjami piłek z tego datasetu zamieściłem poniżej



Rysunek 3.1. Przykładowe obrazki ze zbioru volleyball_easy

Oznaczenia zdjęć zostały sformatowane do formatu COCO json za pomocą strony: https://roboflow.com/
Na stronie tej również wykonano augmentacje obrazu przy zestawie obrazów trenujących. Wykorzystano następujące techniki augmentacji

- Ilość obrazów wyjsciowych na obraz: 3
- Staruacja: +-15%
- Jasność: +-15%
- Ekspozycja: +-5%
- Rozmycie: do 1px

3.2. Volleyball_difficult

Volleyball_difficult to własny autorski zbiór danych. Zbiór został tak nazwany, gdyż duża część zbioru są to niewyraźne, w czasie najszybszej prędkości lub częściowo zasłonięte. Zdjęcia te mają oddawać realistyczne momenty, z którymi ma się zmagać detektor. Zdjęcia do zbioru zostały wybrane z 4 klipów:

- Autorski z własnej gry nagrany telefonem
- Mecz Polska Francja w turnieju EuroVolley:
    - https://www.youtube.com/watch?v=qhTQ2CHBiqk&ab_channel=EuropeanVolleyball
- Nagrania amatorskie kanału Elavetayourself:
    - https://www.youtube.com/watch?v=mQBBI4nB8gg
    - https://www.youtube.com/watch?v=UoTVZy1-K00&t=82s&ab_channel=ElevateYourself

Zbiór danych składa się z 457 obrazków do trenowania sieci oraz 50 do testowania sieci. Przykładowe obrazki z adnotacjami piłek zamieściłem poniżej:

Rysunek 3.1. Przykładowe obrazki ze zbioru volleyball_difficult

Obrazki zostały wybrane ze zrzutów ekranów z wyżej wymienionych nagrań. Zostały one oznaczone za pomocą aplikacji labelme: http://labelme.csail.mit.edu/Release3.0/, a następnie sformatowane do formatu COCO json za pomocą strony https://roboflow.com/
Na stronie tej również wykonano augmentacje obrazu przy zestawie obrazów trenujących. Wykorzystano następujące techniki augmentacji:

- Ilość obrazów wyjsciowych na obraz: 2
- Staruacja: +-10%
- Jasność: +-10%
- Ekspozycja: +-5%

Ze względu na to że piłka na oryginalnych obrazkach jest słabo widoczna, oraz na doświadczenie empiryczne, zdecydowano się jedynie na delikatną augmentację obrazu. Przy większej augmentacji wyniki trenowania były gorsze niż przy lżejszej.

3.3. Ball_noball

Ball_noball jest to zbiór danych stworzony do trenowania ekstraktora cech głębokich przy stosowaniu algorytmu Deep SORT. Trenowanie ekstraktora następuję poprzez trenowanie na klasyfikacj. Należało stworzyć zbiórt nie składający się tylko z obwiedni, ale prostszy ze zdjęć odpowiadające oznaczeniem na pytanie „piłka czy nie". Obrazki oznaczeniami piłek stworzono na podstawie datasetów volleyball_easy i volleyball_difficult. Oznaczone na obrazach piłki w obwiedni stanowią obrazki pozytywne zbioru Ball_noball. Przeróbki tej dokonano za pomocą prostego kodu:

```
for i,data in enumerate(dataset_dicts):
  im = cv2.imread(data['file_name'])
  for j,n_balls in enumerate(data['annotations']):
    cropped_img = im[n_balls['bbox'][1]:n_balls['bbox'][1]+n_balls['bbox'] [3],
                  n_balls['bbox'][0]:n_balls['bbox'][0]+n_balls['bbox'][2]]
    file_name = 'cropped_ball_dataset' + str(i) + '_' + str(j) + '.jpg'
    cv2.imwrite(cropped_img,file_name)
```

Do obrazków negatywnych został wykorzystany dataset cats_noncats z kursu Andrewa Ng na stronie Coursera: https://www.coursera.org/specializations/deep-learning
Oznaczenie ich nie ma znaczenia, ważne że wśrod tych obrazów nie ma piłek.
Zbiór danych Ball_noball składa się z 2226 piłek i 1500 innych obiektów

4. Przedstawienie zastosowanych metod detekcji obiektów

4.1. Wstęp

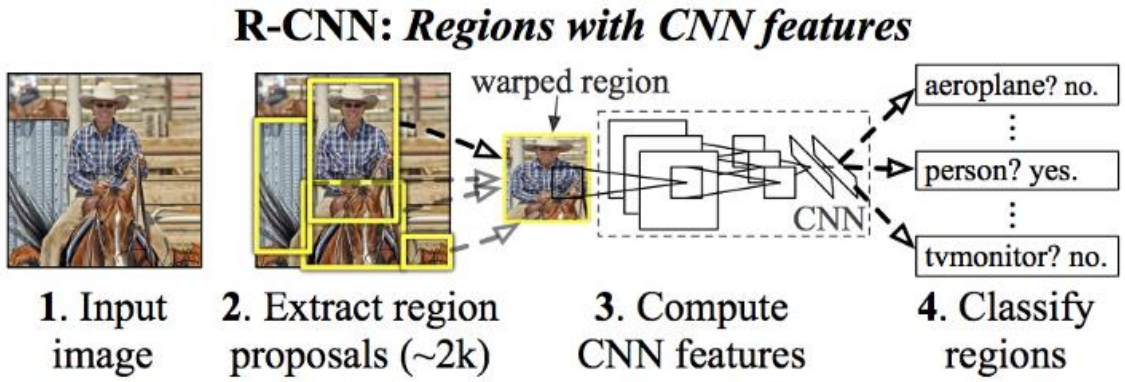
Wizja komputerowa jest to interdyscyplinarna dziedzina, który od kilku lat staje się co raz bardziej popularna. Jedną z centralnych częsci wizji komputerowej jest detekcja obrazów. Detekcja obrazów różni się od problemu ich klasyfikacji. W przypadku detekcji, poza nazwaniem obiektu na obrazie, staramy się dodatkowo narysować rejon ograniczający obiekt (ang. bounding box) by zlokalizować go na obrazie. Dodatkowo w detekcji pojawia się takie zagadnienie, że na jednym obrazie może być wiele rejonów ograniczających różne obiekty które chcemy poddać klasyfikacji. Nie jesteśmy też w stanie stwierdzic ile tych obszarów ma być na obrazku.
Jednym z głownych powodów dla których nie można rozwiązać tego problemu za pomocą standardowej konwolucyjnej sieci neuronowej CNN (ang. Convolutional Neural Network) zakończonej warstwą Fully Connected jest to, że nie da się wcześneij określić rozmiaru ostatniej warstwy. Naiwnym rozwiązaniem tego problemu byłoby podzielenie obrazku na wiele równych części i klasyfikowanie każdego z regionów tego obrazku. Ze względu na ogrom regionów na który trzeba by było podzielić obrazek, jest to absolutnie komputacyjnie nieopłacalne. Z tego powodu wymyślono algorytmy postępowanie takie jak R-CNN, aby usprawdnić detekcje. Architektury te są opisane w niniejszym rozdziale .

4.2. Faster R-CNN

Faster R-CNN jest głęboką siecią konwolucyjną stosowaną do detekcji obiektów. Aby zrozumieć Faster R-CNN, robimy również krótki opis z których wyewoluował, czyli R-CNN i Fast R-CNN

R-CNN

Tradycyjne metody detekcji stosują 3 głowne kroki przedstawione na rysunku 4.1. Pierwszym krokiem jest generowanie wielu propozycji regionów. Te regiony są kandydatami, w których mogą być obiekty.
Aby zapobiec problemowi zbyt dużej ilości regionów, Ross Girshick [2] zaproponował metode w której używamy metody selektywnego wyszukiwania 2000 regionów zainteresowania, co zdecydowanie zmniejsza ilość regionów do klasyfikacji.



Rysunek 4.1: Wyszukiwanie selektywne i regiony zainteresowania [2]

Te 2000 regionów jest generowane za pomocą selektywnego wyszukiwania, czyli:

- Wstępna generacja wielu regionów
- Łączenie wielu podobnych regionów z zastosowaniem algorytmów zachłannych (lokalnie optymalnych)
- Zastosowanie generowanych regionów jako propozycje ostatecznych kandydatów rozpatrywanych regionów

Podobne regiony są łączone, aby utworzyć większe regiony w oparciu o podobieństwo kolorów,tekstur, rozmiarów i zgodności kształtu. Proces jest kontynuowany aż do utworzenia regionówo ostatecznej lokalizacji obiektów Rys. 4.2. Przedstawia powiększanie regionów a niebieskie prostokątyodpowiadające im RoI (region of interest).

Rysunek 4.2: Wyszukiwanie selektywne i regiony zainteresowania [3]

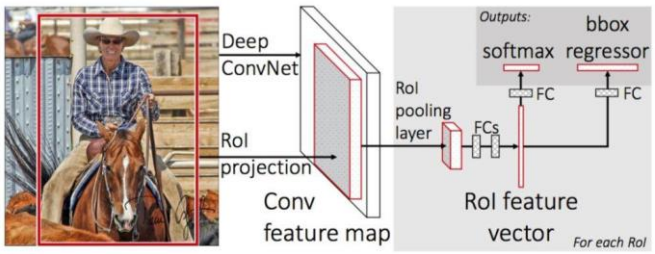
.

Wyodrębnione 2000 regionów mogą zawierać obiekty docelowe i mają różne rozmiary, dlatego są przekształcane na obrazy o ustalonym rozmiarze. Następnie są one przetwarzane przez klasyfikator CNN. W R-CNN do tego celu używany jest model (np. VGG, ResNet) wytrenowany na dużym zbiorze danych o dużej ilości klas (np. ImageNet dataset). Otrzymany wektor cech jest oceniany klasyfikatorem SVM, aby zidentyfikować przynależność do klasy a regresor liniowy przewiduje przesunięcia ramki oznaczenia obiektu względem pierwotnego wskaźnika obiektu.

Głównym problemem z R-CNN jest to, że pomimo zmniejszenia ilości regionów do 2000 to ich trening zajmuje ogromną ilość czasu, więc architektura nie nadaje się do detekcji obiektów w czasie rzeczywistym.. Dodatkowo algorytm selektywnego wyszukiwania nie jest algorytmem który jest trenowany, więc jego skuteczność w trakcie trenowania nie zwiększa się.

Fast R-CNN

Fast R-CNN [4] jest poprawionym algorytmem R-CNN działającym szybciej i zwracającym dokładniejszą detekcję obrazów. Na wejście CNN podawany jest cały obraz. Zestawiając otrzymaną mapę cech z propozycją regionów zewnętrznych realizowaną za pomocą wyszukiwania selektywnego otrzymywane są regiony zainteresowania. Następnie są one przekształcane do określonego rozmiaru przez RoI pooling layer. Warstwa ta generuje wektory cech o stałej długości propozycji regionów. Przesyłane są one do w pełni połączonych warstw (fully connected layers) w celu klasyfikacji z wykorzystaniem funkcji softmax i lokalizacji regresorem liniowym, który zwraca ramki oznaczenia obiektu.

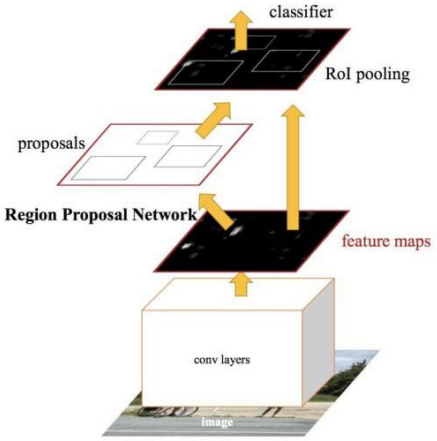


Rysunek 4.3. Projekcja ROI [4]

W porównaniu do R-CNN udoskonalony Fast R-CNN jest znacznie szybszy zarówno w trakcie treningu jak i testu. W czasie treningu z wykorzystaniem VGG16 był 9-krotnie szybszy. 213 razy szybszyw czasie testu. Osiągnął wyższe mAP na PASCAL VOC201

Faster R-CNN

Faster R-CNN [5] składa się z dwóch sieci – sieć propozycji regionów (RPN) do generowania propozycji regionów oraz sieć używająca te regiony do detekcji obiektów. Główną różnicą w stosunku do Fast R-CNN jest to, że Fast R-CNN używa wyszukiwania selektywnego do generacji regionów, co jest zdecydowanie mniej efektywne obliczeniowo niż RPN, gdyż RPN dzieli większość komputacji z siecią detekcji obiektów. W skrócie, RPN ocenia obwiednie regionów (zwane kotwicami – and. anchors) i zostawia te które z największym prawdopodobieństwem obejmują rozpatrywane obiekty.



Rysunek 4.4: Architektura Faster R-CNN [5]

Schemat działania Faster R-CNN

- RPN generuje propozycje regionów
- Dla każdej propozycji regionów w obrazie zwracany jest wektor cech używając warstwy ROI Pooling
- Zwrócone wektory cech są klasyfikowane za pomocą Faster R-CNN
- Zwracane są wyniki klas oraz regiony wykrytych obiektów

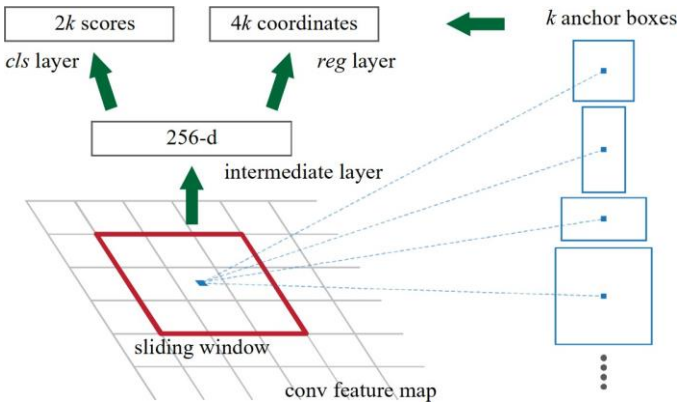
Sieć propozycji regionów (ang. Region Proposal Network - RPN)

R-CNN i Fast R-CNN używają algorytmu wyszukiwania selektywnego do generacji propozycji regionów. Każda z propozycji jest dostarczana do pretrenowanej sieci neuronowej do klasyfikacji, zaś w Faster R-CNN używana jest sieć propozycji regionów.

Zalety RPN:

- Propozycje regionów są generowane przez sieć która może być trenowana i dostosowana do zadania detekcji, co daje lepsze lepsze wyniki niż używanie generycznych metod typu wyszukiwanie selektywne,
- RPN przetwarza obraz za pomocą tych samych warstw konwolucyjnych używanych sieci detekcji Faster R-CNN, w związku z tym oszczędzany jest czas generacji propozycji regionów,
- Ze względu na używanie tych samych warstw konwolucyjnych, RPN i Faster R-CNN mogą być połączone w jedną sieć, co powoduje że trenowanie odbywa się tylko jeden raz.

RPN działa na mapie cech zwróconej przez ostatnią warstwę konwolucyjną dzieloną z Faster R-CNN. Prostokątne okno o rozmiarze o rozmiarach nxn przesuwane jest wzdłuż mapy cech. Dla każdego takiego okna, wytwarzane jest K propozycji regionów. Propozycje te nie są ostateczne, gdyż bedą filtrowane na podstawie „wyniku obiektowości”



Rysunek 4.5: Sieć propozycji regionów RPN [5]

Propozycje te parametryzowane są w odniesieniu do tzw. kotwicy.
Kotwica ma dwa parametry – skala i współczynnik kształtu. Zazwyczaj są 3 skale i 3 współczynniki kształtu, w związku z tym każdy proponowany region ma K=9 kotwic. Innymi słowy, K regionów jest produkowanych przez każdą propozycję regionu, gdzie każdy z nich jest w innej skali lub o innym kształcie.

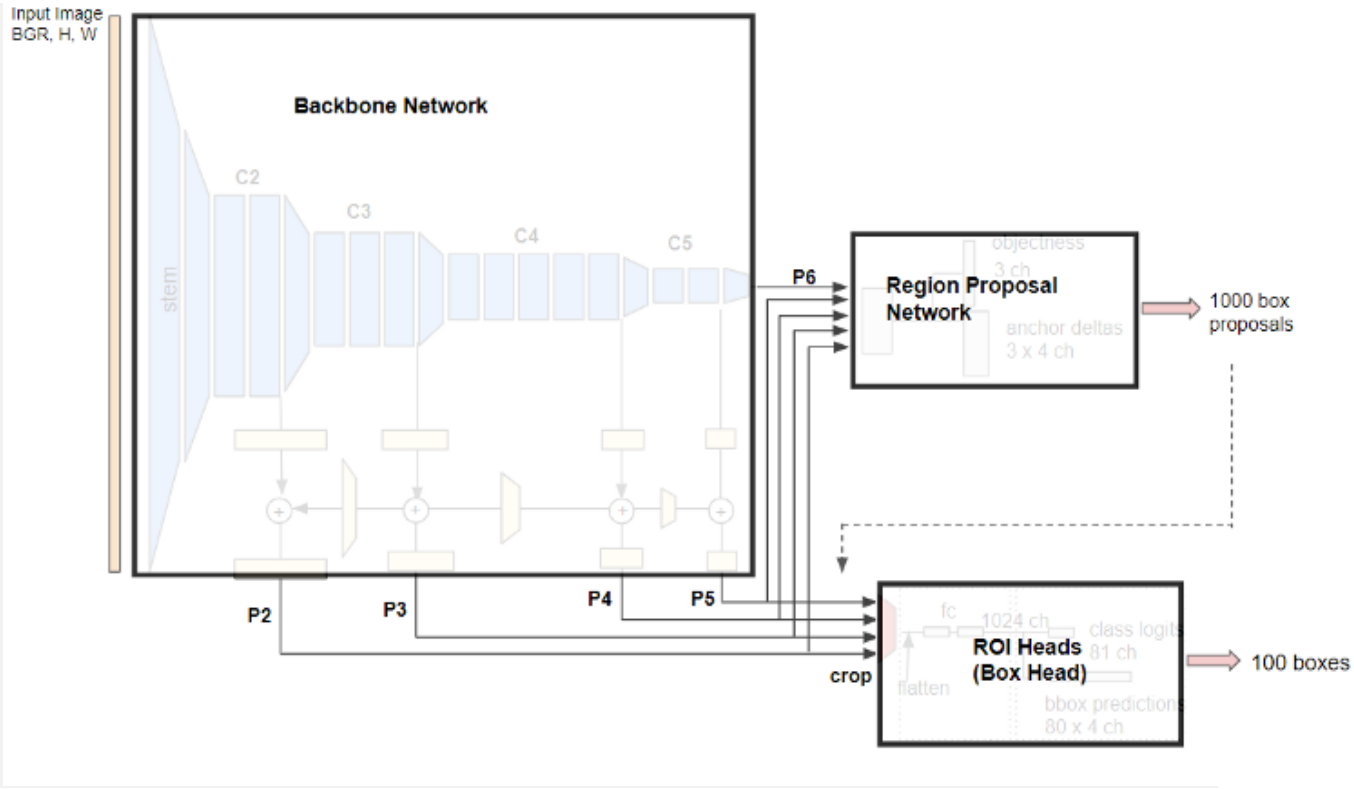
Kotwice używane są po to, by z jednego obrazka o jednej skali można było produkować detektory obiektów odporne na zmiane skali. Dzięki temu unikamy używaniu obrazów i filtrów. Dla każdej propozycji regionu nxn, produkowany jest wektor cech. Wektor ten jest jest przypisywany do 2 warstw w pełni połączonych FC.
Pierwszą warstwą FC jest warstwa nazywana cls i reprezentuje klasyfikator oceniający „wynik obiektowości" dla każdej propozycji regionu. Wynik obiektowości zwraca dwie wartości – jedna klasyfikująca region jako tło a druga klasyfikująca region jako obiekt.
Drugą warstwą FC jest wartstwa nazywana reg, która zwraca wektor definiujące obwiednie regionu.

Warstwa cls produkuje 2 elementowy wektor dla każdej propozycji regionów. W przypadku wektora [1,0], region jest klasyfikowany jako tło, a jeśli wektor wynosi [0,1], to region ten reprezentuje obiekt.
Przy trenowaniu RPN, każda z kotwic dostaje pozytwny lub negatywny „wynik obiektowości" bazując na IoU:

- Gdy IoU>0,7 – przypisanie pozytywnego oznaczenia do kotwicy z największym IoU w stosunku do obwiedni danej (ground truth)
- Jeżeli nie istnieje kotwica o IoU>0,7, przypisywana jest pozytywne oznaczenie do kotwic z największą obwiednią w stosunku do obiedni danej
- Negatywny gdy IoU<0,3
- Kotwicy które nie są negatywne ani pozytywne nie są używane do trenowania

Faster R-CNN w Detectronie z FPN

Bazową siecią Detectrona2 zastosowaną w punkcie 7.2. jest Faster R-CNN z FPN (Feature Pyramid Network). Pomaga on zdecydowanie w skalowaniu obiektów, a więc zwiększa dokładność detekcji zarówno przy obiektach małych jak i dużych. Schemat działania sieci pokazano na rysunku 4.6.



Rysunek 4.6. Architektura Faster R-CNN z FPN [6]

Sieć składa się z 3 głównych bloków

- **Backbone Network**: wydobywa mapy cech z wejsciowego obrazu w różnych skalach. Wyjsciowe cechy Base-RCNN-FPN są nazywane P2 (1/4 skali), P3 (1/8), P4 (1/16), P5 (1/32) and P6 (1/64). Architektura Faster-RCNN bez FPN (‘C4”) wypuszcza jedynie wektory cech ze skali 1/16

- **Sieć propozycji regionów RPN**: Detekcja regionów obiektów z cech wielu skal. Domyślnie przewidzianych jest 1000 propozycji obwiedni z oceną pewności.

- **Box Head (ROI Head)**: Przycina i transformuje mapy cech używając proponowane obwiednie w wiele cech o stałych wymiarach oraz uzyskuje doprecyzowane lokalizacje obwiedni i rezultaty klasyfikacji za pomocą warstw FC. Zwracane jest maksymalnie 100 obwiedni.

Faster R-CNN oblicza wynik dla każdego regionu RPN i przez to definiuje pewność, że ten obiekt jest w tym regionie. Regiony z najlepszymi wynikami są klasyfikowane i zamieniane w predykcje klas, jeżeli wynik jest lepszy od zdefiniowanego minimalnego wyniku.

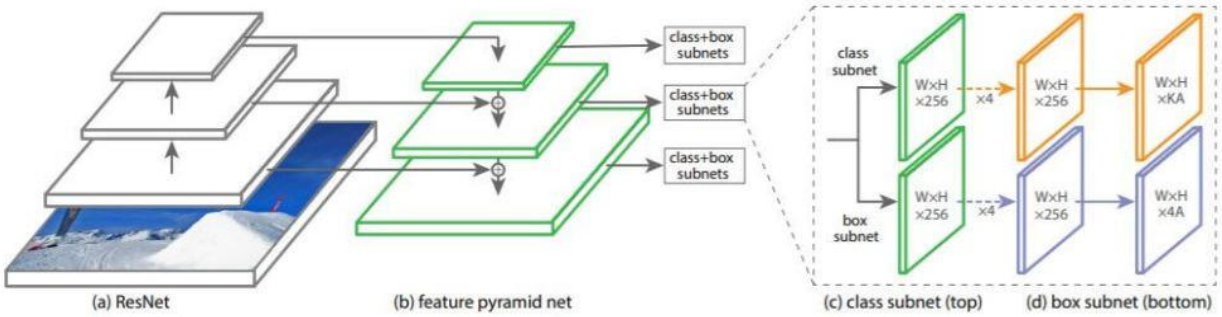
4.3. Retinanet

Retinanet [7] jest to model wprowadzony przez Facebook AI reseach (FAIR) którego celem jest poradzenie sobie z detekcją małych obiektów.Z tego też powodu stał się popularną metodą detekcji obiektów używaną przy zdjęciach lotniczych i satelitarnych.
Retinanet wprowadza 2 poprawki w stosunku do ówcześnie istniejących jednoetapowych detektorów: Feature Pyramid Network oraz Focal Loss

Architektura modelu Retinanet

Architekturę Retinanet można rozbić na 4 podstawowe części:

- Ścieżka oddolna – backbone network np. ResNet, sieć której wynikiem są mapy obiektów w różnych skalach niezależne od rozmiaru obrazu wejściowego,
- Ścieżka odgórna z połączeniami bocznymi– Ścieżka ta upsampluje mapy obiektów z wyższych poziomów piramidy, a połączenia poziome łączą warstwy ścieżki oddolnej i odgórnej o tej samej wymiarowości. Wyższopoziomowe mapy cech mają mniejszą rozdzielczość, ale są lepsze w detekcji dużych obiektów. Mapy niższych poziomów mają wyższą rozdzielczość więc są lepsze w detekcji małych obiektów. Dzięki temu połączeniu, każdy poziom mapy obiektów nadaje się do rozpoznawania innego typu obiektów
- Podsieć klasyfikacji obiektów – sieć konwolucyjna jest połączona z każdym poziomiem FPN do klasyfikacji.obiektów. Podsieć składa się z warstw konwolucyjnych 3x3 z filtrem 256, a następnie kolejną konwolucję 3x3 z ilośćią KxA filtrów. Wyjsciowa mapa cech ma rozmiar WxHxKA, gdzie W i H są proporcjonalne do wymiary wejściowego obrazu a K i A są liczbami klas obiektów i ilości kotwic (anchor boxów). Na końcu używana jest warstwa sigmoid do klasyfikacji obiektów. Ze względu na to że mamy A liczbę kotwic i K klas klasyfikacyjnych, ostatnia warstwa konwolucyjna ma KA liczbe filtrów (każdy anchor boź ma możliwość być klasyfikowanych przez K liczbę klas)
- Podsieć regresji obwiedni – Sieć jest dołączona do mapy cech FPN'a w równolegle do podsieci klasyfikacji. Architektura tej podsieci jest prawie identyczna do tej z klasyfikatora, z jedną różnicą że ostatnia warstwa konwolucyjna 3x3 ma 4 filtry i produkuje mapę cech o rozmiarze WxHx4A. Aby zlokalizować obiekty, sieć regresji produkuje 4 liczby na każdy anchor box aby przewidzieć relatwny offset pomiędzy kotwicą a obwiednią daną. W związku z tym wyjściowa feature mapa posiada 4A filtrów



Rysunek 4.7: Architektura Retinanet [7]

Strata ogniskowa

Strata ogniskowa [8] (ang. Focal Loss – FL) to udoskonalenie w stosunku do funkcji entropii krzyżowej (ang. Cross-Entropy Loss) wprowadzone w celu rozwiązania problemu nierównowagi klas w jednostopniowych modelach detekcji. Parametr ogniskowania γ automatycznie zmniejsza wagę łatwych przykładów podczas treningu, jednocześnie koncentrując trening modelu na trudnych przykładach.

Definiujemy funkcję entropii krzyżowej dla binarnej klasyfikacji następująco:

$$CE(p,y) = \begin{cases} -\log(p) & jeżeli\ y = 1 \\ -\log(1-p) & jeżeli\ y \neq 1 \end{cases}$$

Gdzie $y \in \{\mp 1\}$, specyfikuje klasę obiektu a $p \in [0,1]$ specyfikuje prawdopobieństwo należenia obiektu do klasy. Dla wygody definiujemy:

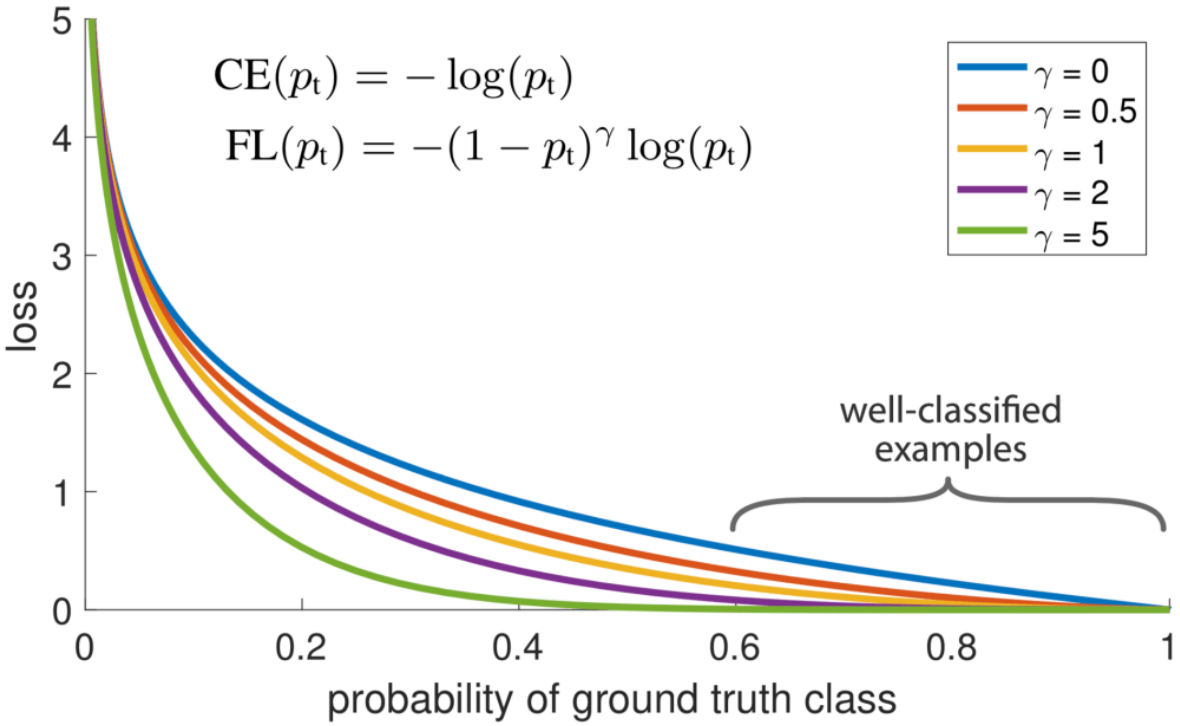
$$p_t = \begin{cases} p & jeżeli\ y = 1 \\ -p & jeżeli\ y \neq 1 \end{cases}$$

Więc $CE(p,y) = CE(p_t) = -\log(p_t)$

Strata ogniskowa to nowatorska funkcja straty, która dodaje czynnik modulujący do funkcji utraty entropii krzyżowej z regulowanym parametrem ogniskowania γ ≥ 0.

Parametr ogniskowania γ automatycznie zmniejsza wagę łatwych przykładów podczas treningu, jednocześnie koncentrując trening modelu na trudnych przykładach.

Retinanet używa wariantu α-balansowalnego FL, gdzie α=0.25, γ=2



Rysunek 4.8: Wykres straty ogniskowej w stosunku do prawdopobienstwa klasyfikacji [8]

Definicja FL:

$$FL(p_t) = \alpha(1-p_t)^{\gamma}\log(p_t)$$

Cechy FL:

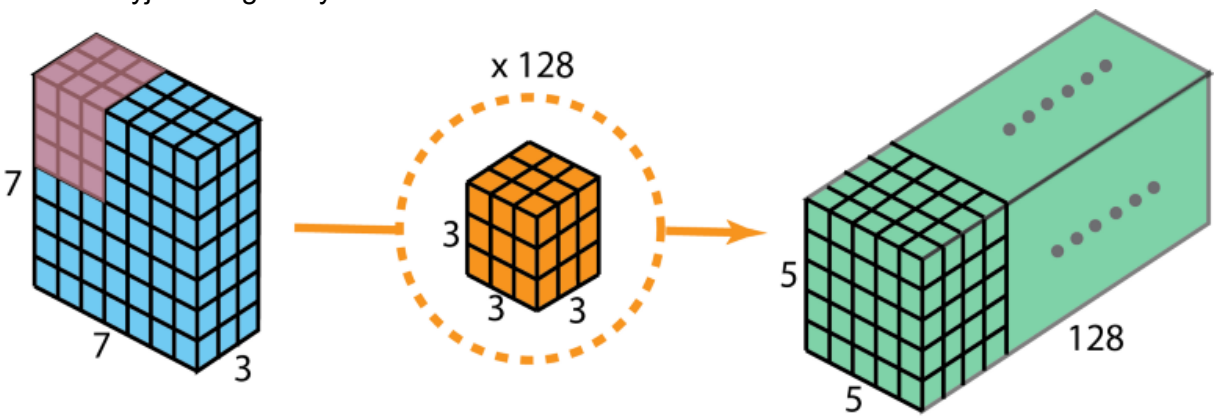
- Kiedy obiekt jest źle sklasyfikowany i p_t jest małe, część modulacyjna jest bliska 1 w związku z tym nie wpływa na stratę
- Gdy p_t zbliża się do 1, cały czynnik zbliża się do zera więc strata dla dobrze klasyfikowanego przykładu jest obniżana
- Parametr ogniskowania, γ, płynnie dostosowuje szybkość, z jaką proste przykłady są zmniejszane

5. Przedstawienie zastosowanych architerktur CNN do detekcji

5.1. MobilenetV1

W 2017 roku, Google wprowadziło MobileNetV1 [10] – rodzina sieci neuronowych służących do klasyfikacji oraz ekstrakcji cech w sieciach neuronowych. Założeniem sieci było zmniejszenie potrzeb na moc obliczeniową, tak aby można było korzystać z sieci nawet z telefonu

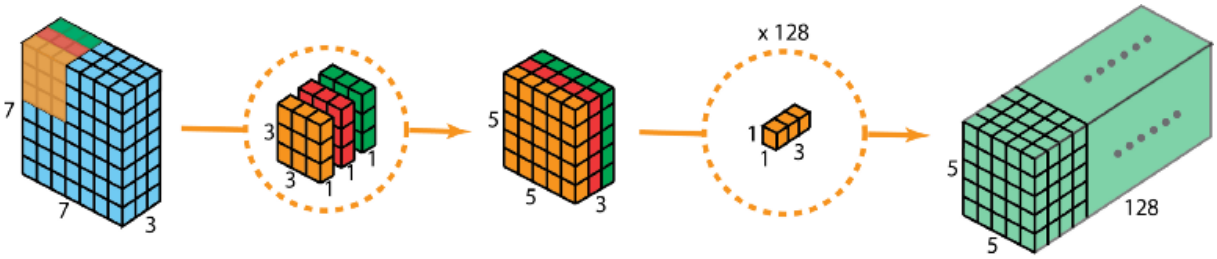
Głównym pomysłem na który opiera się MobileNet jest użycie separowalnych konwolucji aby przyspieszyć obliczenia sieci.
Normalnie przy konwolucji aplikuje maskę na wszystkie kanały wejściowego tensora. Maska jest „przesuwana" po tensorze i przy każdym takiej akcji liczona jest średnia ważona wszystkich składowych tensora objętego jądrem po wszystkich kanałach. Wynikiem każdej z tych operacji jest tylko jedna wartość, zaś liczba operacji jest równa iloczynowi wymiarów tensora wyjściowego i wymiaru maski.



Rysunek 5.1. Standardowa konwolucja 2d tworząca wynik o jednej warstwie, ze 128 filtrami [10]

Mobilenet również stosuje standardową konwolucję, ale tylko raz przy pierwszej warstwie. W pozostałych warstwach stosowana jest separowalna konwolucja. Jest to kombinacja dwóch operacji konwolucji separowalnej po kanałach (depthwise convolution) oraz konwolucji 1x1 (konwolucja pointwise)

Konwolucja separowalna po kanałach - jak sama nazwa wskazuje, konwolucja ta nie łączy wszystkich kanałów tensora wyjściowego, lecz wykonuje konwolucji na każdym z kanałów osobno. Każdy kanał otrzymuje własny zestaw wag.
Po konwolucji ten przeprowadzana jest konwolucja z maską 1x1 , czego wynikiem jest dodanie wartości ze wszystkich kanałów (średnia ważona). Stosując X masek 1x1 otrzymujemy tensor wyjściowy o X kanałach.



Rysunek 5.2. Konwolucja depthwise i pointwise [10]

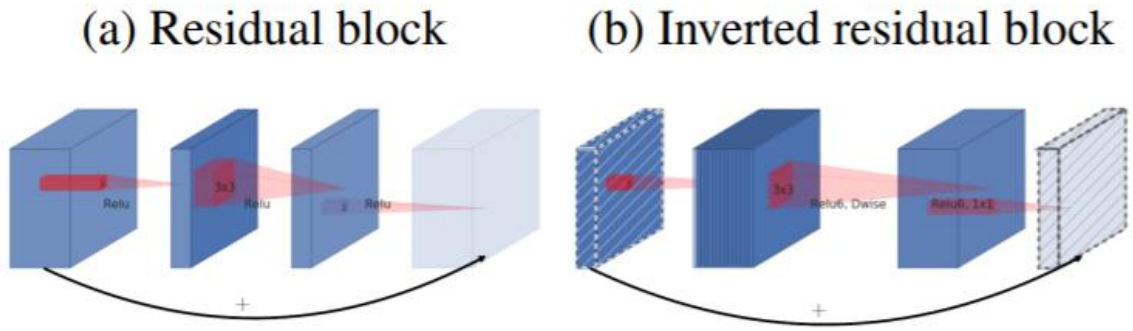
Rezultat konwolucji separowalnej jest bardzo zbliżony do tego ze zwykłej konwolucji, ale ze względu na to że powtarzana jest prosta konwolucja 1x1 w celu uzyskania większej liczy kanałow, oszczędzana jest ogromna ilośc mocy obliczeniowej.

Architektura Mobilenet V1 składa się z standardowej warstwy konwolucyjnej 3x3
a następnie z trzynastu bloków opisanych powyżej. Pomiędzy tymi blokami nie ma warstw pooling, zamiast tego niektóre z warstw mają stride 2 aby zmniejszyć wymiarowość tensorów. Kiedy to się dzieje, podwaja się ilość kanałów odpowiednią ilościa masek 1x1.

Po warstwach kownolucyjnych stosowany jest batch normalization, a funkcją aktywacji jest ReLU6. W klasyfikatorze bazującym na MobileNetach, na końcu jest warstwa average pooling, a za nią warstwa w pełni połączona lub konwolucja 1x1 z softmaxem.

5.2. MobileNetV2

MobileNetV2 [11] wciąż używa warstw separowalnych, ale architektura sieci jest rozwinięta o dwie dodatkowe główne koncepcje, zwane blok **Inverted Residual** i warstwa **Linear Bottleneck**.
Bloki Residual łączą się z początkiem i końcem bloku konwolucyjnego z połączeniem przeksakującym (skip connection). Działa to podobnie jak w ResNecie i istnieje po to, aby pomóc z przepływem gradientów przez sieć. Połączenie rezydualne jest tylko używane gdy ilość kanałów wchodzących i wychodzących z bloku jest jednakowa, przy czym co kilka bloków ilość kanałów jest zwiększana).



Rysunek 5.3. Blok rezydualny i odwrócony blok rezydualny [11]

Patrząc na połączenie przeskakujące w oryginalnym bloku rezydualnym, widzimy że połączenie stosuje podejście szerokie – wąskie – szerokie, pod względem ilości kanałów w warstwie. Wejście ma dużą liczbę kanałów, które są kompresowane lekką obliczeniowo

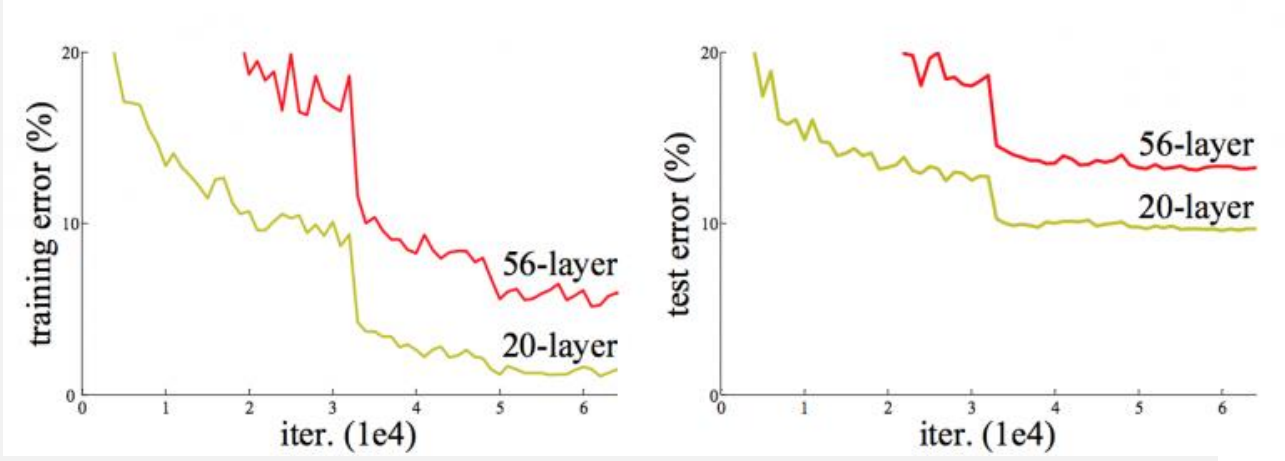
konwolucją 1x1. W ten sposób następująca konwolucja 3x3 ma mniej parametrów. Na końcu liczba kanałów jest ponownie zwiększana korzystając z kolejnej konwolucji 1x1.
MobileNetV2 stosuje podejście wąskie – szerokie wąskie pod względem ilości kanałów w warstwie. W pierwszym kroku zwiększana jest ilość kanałów za pomocą konwolucji 1x1, ponieważ konwolucja depthwise i tak w znaczący sposób redukuje liczbę parametrów. Następnie kolejna konwolucja 1x1 zwęża sieć tak aby liczba kanałów na wejściu i wyjściu się zgadzała. Ostatnia konwolucja 1x1 zwężająca sięc nazywa się warstwą bottleneck
Blok inverted residual łączy wąskie warstwy za pomocą skip connection, podczas gdy warstwy pomiędzy są szerokie. Powoduje to zmniejszenie liczby parametrów w stosunku do zwykłego bloku rezydualnego.

Linear Bottlenecks

Powodem dla którego w sieciach neuronowych używa się nieliniowych funkcji aktywacyjnych jest to, że dzięki nim, wiele operacji mnożenia macierzy nie da się zredukować do jednej operacji numerycznej. Pozowala nam to na budowanie głębokich sieci neuronowych z wieloma warstwami. Jednoczeście funkcja aktywacyjna ReLU, pozbywa się wartości mniejszych niż 0. Z tą stratą informacji radzimy sobie poprzez zwiększanie liczby kanałów w warstwach.
W blokach inverted residual zamiast tego ściskamy warstwy w miejscach gdzie są powiązane połączenia skip, co pogarsza wydajność sieci. Autorzy Mobilenetu V2 wprowadzili przez to pojęcie linear bottleneck, gdzie ostatnie połączenie bloku residual ma liniowe wyjście zanim jest dołączone do wstępnych aktywacji – po to aby przy zmniejszaniu liczby warstw, warstwy nieliniowe nie niszczyły zbyt wiele informacji.

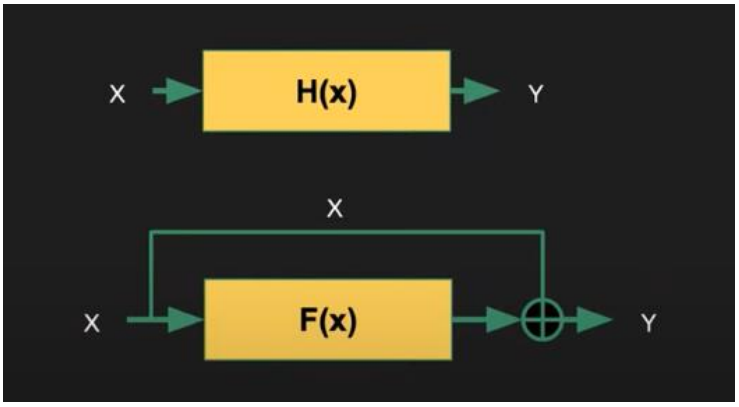
5.3. Resnet-50

Resnet [13], w skrócie sięć rezydualna (ang. Residual Network) jest architekturą sieci używaną w wielu zadaniach wizji komputerowej. Model ten był zwycięzcą konkursu ImageNet w 2015. Resnet spowodował przełom w architekturach sieci neuronowych, z tego powodu że pozwala na trenowanie z sukcesem bardzo głębokie sieci. Wcześniej trenowanie bardzo głębokich sieci było bardzo dużym wyzwaniem ze względu na problem zanikających gradientów.
Jak wiemy, głębokie sieci neuronowe są świetne w identyfikacji zarówno nisko jak i wysokopoziomowych cech obrazów, a dokładnie kolejnych warstw zazwyczaj daje nam lepszą skuteczność. Stąd powstaje pytanie – czy do lepszej wydajności modelu wystarczy dodawać co raz więcej warstw sieci?
Niestety z obserwacji wynika, że niestety co raz większe modele ze zwiększaną ilością warstw nie dawały lepszych rezultatów. Dodatkowo po pewnym czasie zaczyna również spadać skuteczność trenowania.
Początkowo uważano że za tym efektem stoi jedynie problem znikającego/eksplodującego gradientu. Utrudniało to zbieżność modelu, co powodowało że model nie był trenowany efektywnie. Problem ten był w dużej mierze zadresowany przez RRN używające LSTM, normalizacje inicjalizacji i warstwy normalizacyjne. Te techniki pozwalały na zbieżność modelu z większą ilościa warstw przy stochastycznym gradiencie i propagacji wstecznej.
Pomimo tego że został rozwiązany problem zanikających gradientów, dalej obserwowano że przy wystarczająco wysokeij liczbie warstw skuteczność trenowania spadała. Przedstawione jest to na rysunku 5.4.



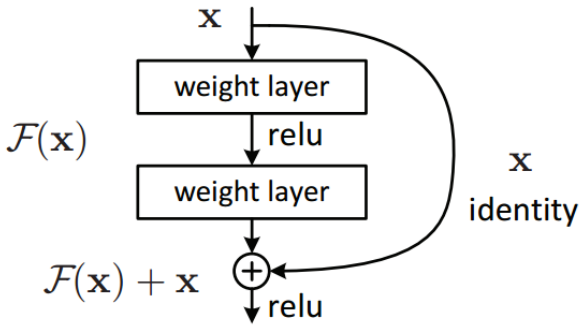
Rysunek 5.4. Porównanie błędu trenowania i testowania zwykłych konwolucyjnych sieci neuronowych [13]

Takie rezultaty nie są związane z overfittingiem, ze względu na to że zwiększa się błąd przy trenowaniu a nie testowaniu. Nazywa to się problemem degradacji. Poźniejsze warstwy nie są w stanie się nauczyć funkcji tożsamościowych, które są wymagane aby przenieść rezultat do wyjścia sieci. W ResNetach, zamiast oczekiwania że warstwy dostosują się do oczekiwanego mapowania, pozwalamy warstwom pasować do rezydualnego mapowania.
Początkowo pożadanym mapowaniem jest H(x). Pozwalamy sieci, aby była dopasowana do rezydualnego mapowania F(x) = H(x)-x, gdyż dla sieći prościej jest dopasować się do rezydualnego mapowania niż oryginalnego.



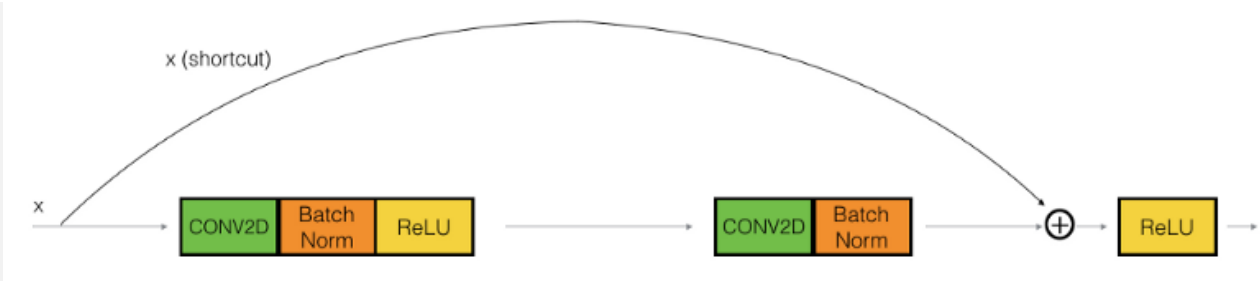
Rysunek 5.5. Omijanie warstwy [14]

Ta metoda omijania danych z jednej warstwy do kolejnej zwana jest połączeniem omijającym (ang skip connection). Ta technika pozwala na łatwy przepływ danych pomiędzy warstwami bez uderzania w możliwość nauki głębokiej sieci neuronowej. Zaletą takiego typu połączenia jest to, że jeśli którakolwiek warstwa przeszkadza w trenowaniu będzie ominięta.

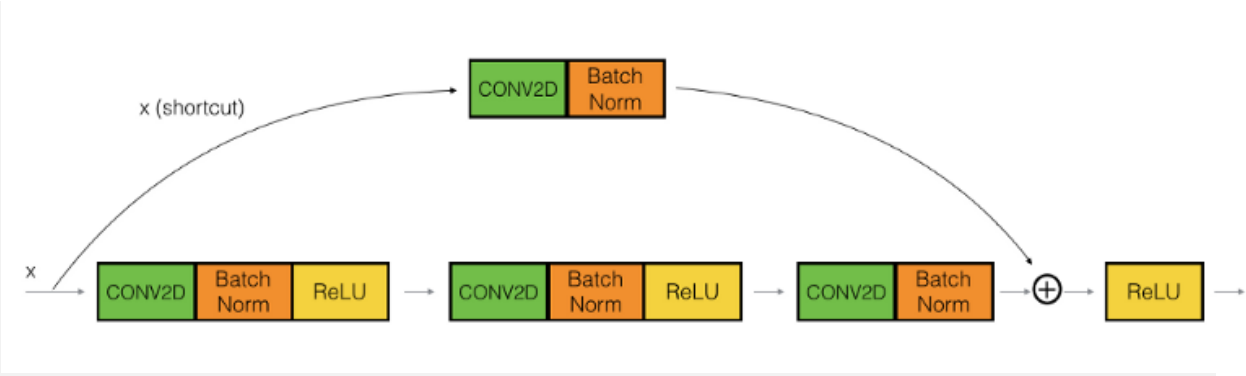


Rysunek 5.6. Połączenie omijające [13]

Za tym typem połączenia stoi intuicja, że łatwiej jest dla sieci nauczyć się zbiegać wartość f(x) do zera tak żeby zachowywała się jak funkcja tożsamościowa, niż nauczyć się zachowywać się jak funkcja tożsamościowa we własnym zakresie, dobierając odpowiednie wagi które dałbyby taki rezultat.ResNet używa dwóch głównych typów bloków w sieci przedstawionych na poniższych rysunkach, czyli blok tożsamościowy i blok konwolucyjny

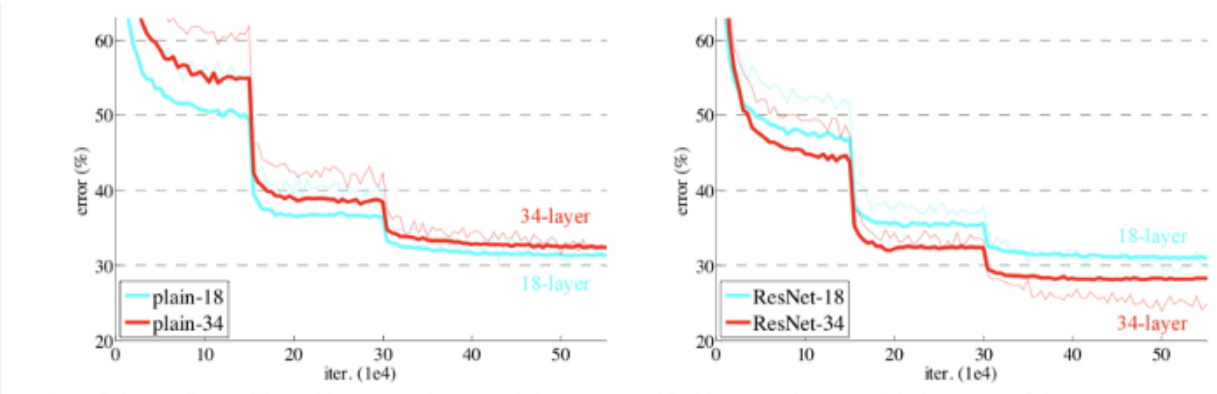


Rysunek 5.7. Blok tożsamościowy



Rysunek 5.7. Blok konwolucyjny

Blok konwolucyjny pomaga modyfikować przychodzące dane, tak aby rezultat pierwszej warstwy miał taką samą wymiarowość jak trzeciej warstwy. Te komponenty pomagają w osiągnięciu lepszej optymalizacji dla skuteczności modelu głębokich sieci neuronowych. Poniżej przedstawione jest efekt używania ResNeta w stosunku do zwykłych sieci konwolucyjnych.



Rysunek 5.8. Porównanie błędów w zależności od ilości warstw w przypadku zwykłych sieci neuronowych i Resnetów [13]

Podsumuwując, ResNet jest architekturą przełomową w dziedzinie deepleraning. Dzięki połączeniom pomijającym, pozwoliła na badanie co raz głębszych sieci neuronowych co daje zdecydowanie lepsze rezultaty.

6. Przedstawienie zastosowanych architerktur CNN do śledzenia obiektów

6.1. Wstęp

Naukowcy poświęcili dużo uwagi zagadnieniom detekcji obiektów w obrazach. Mniej znanym zagadnieniem o szerokich zastosowaniach jest aplikacja śledzenia obiektów w filmach. Wymaga to połączenia wiedzy o detekcji obiektów w nieruchomych obrazach oraz analizą ruchu i używania jej do przewidywania trajektorii obiektów. Przykładami zastosowań mogą być tutaj śledzenie złodziei, automatyzacja mandatów samochodowych lub śledzenie obiektów w wydarzeniach sportowych.
Zagadanienia śledzenia można podzielić na dwie podstawowe kategorie:

- Śledzenie pojedynczego obiektu – zadaniem jest oznaczenie obiektu w obrazie i śledzenie go do momentu wyjścia z kadru,
- Śledzenie wielu obiektów – zadaniem jest oznaczenie wszystkich rozpatrywanych obiektów, oznaczenie indywidualnym id każdego z nich i śledzenia ich do końca filmu.

Rysunek 6.1. Śledzenie pojedynczego obiektu.

Rysunek 6.2. Śledzenie wielu obiektów

Wyzwania
Przy śledzeniu obiektów w rzeczywistych zastosowaniach mamy do czynienia z problemami, które powodują, że zagadnienie to nie jest proste. Poniżej jest lista kilku takich zagadnień.

Zasłonięcie
Przysłanianie się obiektów jest jednym z podstawowych problemów dotyczących śledzenia obrazów. W przypadku gdy jeden z obiektów zostanie zasłonięty przez inne, a potem odsłonięty, naszym zadaniem jest spowodowanie by przez cały film ten sam obiekt miał przypisany ten sam identyfikator.

Różne punkty widzenia
Częstym zagadnieniem przy śledzeniu obiektów jest śledzenie tego samego obiektu przy użyciu różnych kamer. Powoduje to że obiekt wygląda różnie w różnych filmach. W takim wypadku detektow nie może być podatny na zmianę kąta widzenia obiektu.

Niestacjonarna kamera lub szybki obiekt.
W przypadku gdy kamera używana do śledzenia obiektu jest również w ruchu w stosunku do niego, może to powodować niechciane konsekwencje. Jedną z nich jest to, że obiekt może wyglądać inaczej ze względu na ruch kamery (być większy, mniejszy, rozmazany). Ten sam efekt następuje przy obiektach które poruszają się bardzo szybko w stosunku do ich rozmiarów, mamy do czynienia z rozmazaniem obrazu ze względu na ograniczenia kamery. Dobry tracker może ograniczyć problemy związany z tym zagadnieniem.

6.2. Tradycyjne metody śledzenia obrazu

Przesunięcie średniej (ang. Meanshift)
Przesunięcie średniej jest popularnym algorytmem który jest głównie używany w zagadnieniach analizy zgrupowań i innych powiązanych zagadnień uczenia nienadzorowanego. Jest to algorytm podobny do algorytmu centroidów (ang. K-Means), ale zamiast obliczania środków zgrupowań metodą centroidów, używa średniej ważonej która daje większe znaczenie punktów bliższym średniej. Zadaniem algorytmów jest znalezienie wszystkich “modów” przy danej dystrybucji danych.
Algorytm zmiany średniej można wykorzystać do śledzenia wzrokowego. Najprostszy taki algorytm stworzyłby mapę ufności na nowym obrazie w oparciu o histogram koloru obiektu na poprzednim obrazie i wykorzystałby przesunięcie średniej do znalezienia szczytu mapy ufności w pobliżu starego położenia obiektu. Dzięki temu dochodzi do śledzenia obiektu.

Przepływ optyczny
Przepływ optyczny opisuje skomputeryzowane śledzenie ruchomych obiektów poprzez analizę różnic zawartości między ramkami wideo. Na filmie zarówno obiekt, jak i obserwator mogą być w ruchu; komputer może zlokalizować wskazówki, które zaznaczają granice, krawędzie i regiony poszczególnych zdjęć. Wykrywanie ich postępów umożliwia komputerowi śledzenie obiektu w czasie i przestrzeni. Technologia jest wykorzystywana w przemyśle i badaniach, w tym w obsłudze bezzałogowych statków powietrznych (UAV) i systemów bezpieczeństwa.
Przepływ optyczny opisuje śledzienie ruchu obiektu pomiędzy dwiema klatkami spowodowane ruchem obiektu lub kamery. Jest to dwuwymiarowe pole wektorowe, gdzie każdy z wektorów jest wektorem przemieszczenia pokazującym ruch punktów pomiędzy klatkami.

Śledzenie za pomocą przepływu optycznego opiera się na kilku założeniach:

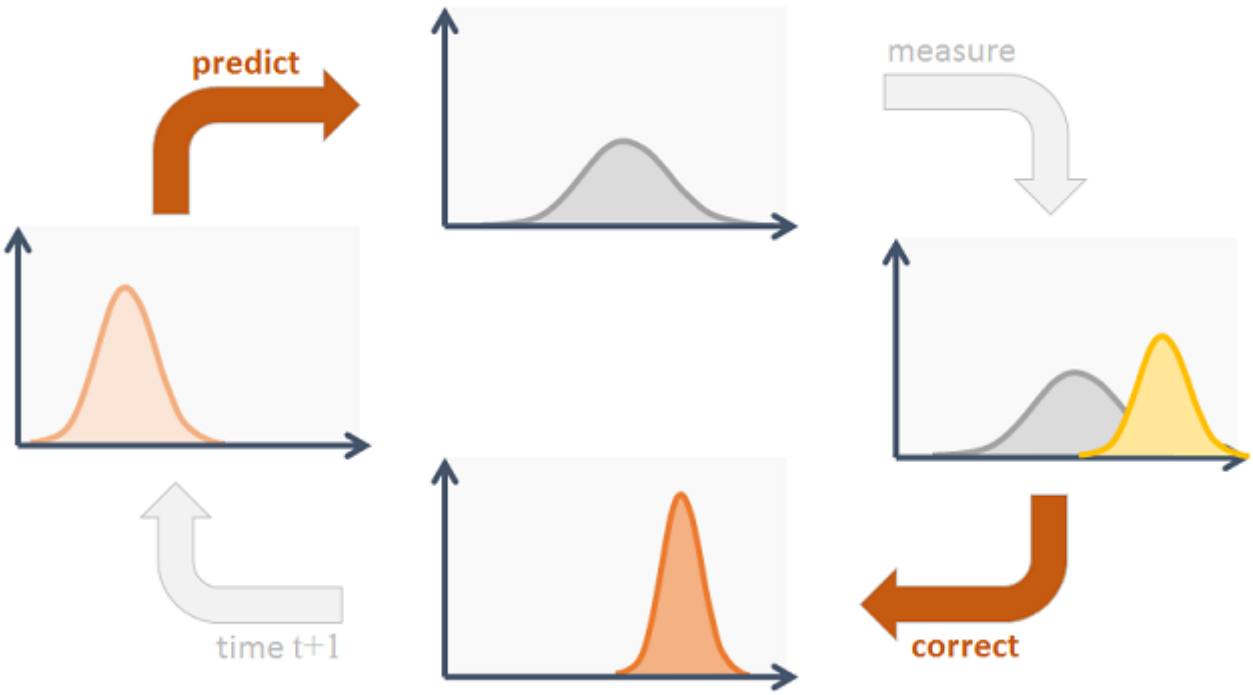
- Stała jasność. Zakłada się, że jasność w obrębie małego określonego regionu jest stała, mimo że region może się zmienić
- Sąsiadujące punkty należą do tego samego obiektu i przez to mają podobny ruch
- Zmiana ruchu jest stopniowa.
- Punkty nie poruszają się bardzo daleko albo chaotycznie

W przypadku gdy te kryteria są spełnione, możliwe jest do użycia metoda Lucas-Kanade aby uzyskać równanie prędkości punktów do śledzenia (zazwyczaj to są łatwo wykrywalne elementy). Za pomocą tych wzorów można śledzić obiekt w filmie.

Filtry Kalmana

Zakładamy że mamy niedoskonały detector – jest podatny na fałszywe pozytywy, oraz fałszywe negatywy, jego detekcje nie są perfekcyjne (skala I pozycja obiektu nie jest dokładna) oraz używaniem detektora jest mocno obciążające pamięć. Rozpatrujemy sytuację detekcji pojedynczej piłki do siatkówki. Po wykryciu obiektu, chcemy użyć znanych informacji o obiekcie w celu śledzenia obiektu w jak najlepszym stopniu. Aby przewidzieć kolejną pozycję piłki, musimy założyć model ruchu piłki (przykładowo model stałej prędkości lub model stałego przyspieszenia). Ze względu na to że detector jest niedoskonały, przy lokalizacji obiektu zmagamy się z szumem nazywanym szumem pomiarowym. Dodatkowo, wybrany model ruchu piłki nie opisuje jego idealnie w związku z tym mamy do czynienia z szumem przetwarzania. Chcemy przewidzieć następną lokalizację piłki wykorzystując te 3 parametry:

- **Model ruchu piłki**
- **Szum pomiarowy** – niedokładności w naszym detektorze
- **Szum przetwarzania** – niedokładność związana z różnicy przyjętego modelu ruchu a rzeczywistości



Rysunek 6.3. Cykl filtru Kalmana [15]

Algorytm, zgodnie z którym działa Filtr Kalmana, dzielimy na dwie fazy (etapy). Pierwszy etap

zwany jest predykcją, a drugi korekcją. Podczas predykcji bazując na poprzedniej wartości stanu x wyznaczana jest nowa wartość stanu x oraz jej kowariancja.. Są one niejako przewidywane na podstawie równań stanu x za pomocą których opisaliśmy nasz system.
W fazie korekcji korzystamy z pomiarów czujników (u nas z naszego detektora) aby uzyskać dane bliższe rzeczywistości niż zgadywanie z zamkniętymi oczami na podstawie matematyki, ale bez informacji ze świata zewnętrznego. Im mniejsze są szumy pomiarowe ym większy wpływ na przewidywaną pozycję ma rozpatrywana detekcja. W przypadku gdy detekcja jest obarczona dużym błędem, większy wpływ na przewidywaną pozycję będzie miał wzór wyliczony z modelu ruchu. Cały cykl może być powtarzany
Filtry kalmana najlepiej działają przy liniowych systemach I przy gaussowskich procesach. W naszym przypadku piłka zdecydowanie należy do przestrzeni liniowej, a procesy należą do gaussowskich.

6.3. Algorytm Sort

SORT [16] (ang. Simple Online and Realtime Tracking) jest implementacją szkieletu wizualnego śledzenia wielu obiektów bazujące na technikach asocjacji danych i estymacji stanu. Jest zaprojektowany do aplikacji śledzenia, gdzie tylko obecne I przeszłe klatki są dostępne, a obiekty są rozpoznawane natychmiastowe. W momencie pierwszej publikacji w 2016 r. był najlepszym algorytmem śledzenia według wzorca MOT. Zdecydowana część dokładności Sorta jest wynikiem dokładności detekcji.

Jednym z głównych elementów algorytmu Sort jest zastosowanie opisanego pozwyżej **filtru Kalmana**. Stan rozpatrywanego śledzonego obiektu opisywany jest za pomocą 8 zmiennych, $x = [u, v, s, r, u, \dot{v}, \dot{s}] T$, gdzie u and v representują poziome i pionowe współrzędne obiektu, s reprezntuje skalę obiektu a r współczynnik kształtu. Pozostałe zmienne są odpowiednio prędkościami opisanych zmiennych. Zmienne oznaczają jedynie absolutną pozycję oraz prędkość, ze względu na to że zakładamy model liniowy prędkości. Filtr Kalmana pomaga nam dodatkowo zastosować w obliczeniach szum pomiarowy oraz poprzedni stan przy dobieraniu odpowiednich ramek na obiekcie.

Przy przypisywaniu obwiedni istniejących obiektów, geometra tej obwiedni ustalana jest poprzez przewidywanie jej lokalizacji w obecnej klatce filmu. Macierz kosztu przepisania obwiedni obliczana jest jako dystans IOU pomiędzy detekcją a przewidywaną obwiednią istniejącego obiektu. To przypisanie jest rozwiązywane optymalnie za pomocą algorytmu Węgerskiego. Dodatkowo, minimalne IOU jest wymagane aby odrzucić sytuacje, gdzie zachodzenie na siebie przez obwiednie jest mniejsze niż IOUmin. Dodatkowo stwierdzono, że dystans IOU radzi sobie z częściowym zasłonięciem rozpatrywanego obiektu przez inne poruszające się przeszkody.

Gdy obiekty pojawiają się lub opuszczają obraz, do obiektów muszą być przypisane lub zniszczone unikalne id. Przy tworzeniu id obiektów śledzonych, rozpatrywana jest detekcja z zachodzeniem większym niz IOUmin. Śledzenie jest inicjalizowane używając obwiedni detektora z prędkością ustawioną na zero, a kowariancja tej prędkości jest ustawiona na bardzo wysoką wartość aby sygnalizować niepewność tej danej. Dodatkowo nowy obiekt śledzony musi spełniać odpowiednie warunki przez minimum kilka klatek zanim zostanie przypisana mu unikalna tożsamość (aby ograniczyć liczbę fałszywych pozytywów). Tożsamość obiektu jest usuwana w przypadku gdy warunki detekcji nie są spełnione przez

kilka klatek. W przypadku gdy ten sam obiekt znowu pojawi się na obrazach, zostanie przypisany mu nowy id.

Autorzy zdecydowali się zastosować kwadratowy dystans Mahalanobisu (skuteczna metryka w przypadku radzenia sobie z dystrybucjami) aby uwzględnić szumy związane z filrem Kalmana. Stwierdzono, że metryka ta jest bardziej dokładna od metryki Euklidesowej, ze względu na to że mierzony jest dystans między dwiema dystrybucjami.

6.4. Algorytm Deep Sort

Deep Sort [17] jest “dobudówką” do algorytmu Sort, uwzględnia ona dodatkowo metrykę bazującą na wyglądzie obiektu. Pomimo tego jak efektywny jest filter Kalmana, czysty algorytm SORT nie radzi sobie bardzo dobrze z sytuacjami jak przysłanianie obiektów, czy też różne punkty widzenia z kamer. W celu rozwiązania tych problemów, Deep Sort wprowadza dodatkową metrykę bazującą na “wyglądzie” obiektu

Wektor cech obrazu

Koncepcja jest taka, że należy otrzymać wektor który opisuje cechy rozpatrywanego obrazka. Najpierw budowany jest klasyfikator który jest trenowany na naszym datasecie do otrzymania zadawalającej skuteczności. Zakładając klasyczną architecture, zostanie nam warstwa “dense” produkująca wektor cech nadający się do klasyfikacji.
Ten wektor cech stanowi “deskryptor wyglądu” obiektu. Używany jest on do opisywania obrazu objętego obwiednią detektora. Z tego otrzymujemy wektor cech o wymiarach 128x1
Zaktualizowana metryka dystansu w stosunku do Sorta opisana jest wzorem

$$D = Lambda * D_k + (1 - Lambda) * D_a$$

Gdzie Dk to dystans Mahalanobisa, Da jest to dystans kosinusowy pomiędzy wektorami cech, a lambda jest to waga współczynników.
Prosta metryka odległości w połączeniu z techniką z głębokiej sieci neuronowej jest wystarczające aby Deep Sort był jednym z najpowszechniej używanych algorytmów śledzenia obiektuów

7. Opis przebiegu eksperymentu

Do wykonania zadania detekcji i śledzenia piłki wykorzystano środowisko Google Colab, oraz biblioteki Detectron2 Facebooka. Do zadania detekcji wykorzystano następujące modele:

- Model Faster_RCNN_R_50_FPN_3x – Model korzystający z architektury Faster R-CNN, na backbone Resnet50
- Model Faster_RCNN_MV2 - Model korzystający z architektury Faster R-CNN, na backbone Mobilenetu
- Model RETINANET_R_50_ FPN_3x – Model korzystający z architektury Retinanet na backbone Resnet50

Rezultaty i wnioski z trenowania modeli przedstawiono w rozdziale 8. Przebieg trenowania wszystkich 3 modeli jest bardzo podobny, w związku z tym szczegółowo opisany jest tylko jeden z tych modeli.

Notatniki trenowania z wynikami trenowania i testowania powyższych architektur można znaleźć pod poniższymi linkami:

- Model Faster_RCNN_R_50_FPN_3x – https://github.com/KubaRurak/Praca_dyplomowa_Detectron2_Tracking/blob/main/Detectron2_faster_rcnn_resnet50_final.ipynb
- Model Faster_RCNN_MV2 – https://github.com/KubaRurak/Praca_dyplomowa_Detectron2_Tracking/blob/main/Detectron2_mobilenet_final.ipynb
- Model RETINANET_R_50_ FPN_3x – https://github.com/KubaRurak/Praca_dyplomowa_Detectron2_Tracking/blob/main/Detectron2_retinanet_resnet50_final.ipynb

7.1. Opis środowiska Detectron2

Detectron2 jest to platforma do detekcji i segmentacji obiektów wykonana przez FAIR (Facebook AI Research). Środowisko jest PyTorchową implementacją nowoczesnych architektur sieci neuronowych, oraz wstępnie trenowanych modeli do detekcji obiektów, segmentacji obrazów, detekcji punktów kluczowych i innych zastosowań.
Ważną cechą detektrona jest jego struktura licencyjna – sama biblioteka jest licencjonowana za pomocą Apache 2.0 a wstępnie trenowane modele są licencjonowane CC BY-SA 3.0. Oznacza to że dopuszczalne jest modyfikowanie istniejącego kodu, można używać go zarówno do prac badawczych jak i komercyjnych.

Struktura repozytorium detectron2:

- **checkpoint** - checkpointer
- **config** - domyślne konfiguracje
- **data** - programy obsługujące zbiory danych
- **engine** – silnik predyktora i modułu trenujacego
- **evaluation** - ewaluator zbiorów danych
- **export** – konwerter modelu detectrona do ONNX
- **layers** – struktura warstw
- **model_zoo** – modele wstępnie trenowane
- **modeling** - lokalizacja modeli, fpn, roi-head itp.
- **solver** - optymalizatory
- **structures** - klasy struktur np obwiednie, segmentacje
- **utils** - obiekty pomocnicze np wizualizator, logger

7.2. Przebieg trenowania modelu na przykładzie architektury Faster R-CNN z backbone Resnet-50

Instalacja środowiska
Środowisko instalowane jest z repozytorium umieszczonym przez Facebook AI Research na GitHub https://github.com/facebookresearch/detectron2

```
!pip install -U torch torchvision
!pip install git+https://github.com/facebookresearch/fvcore.git
import torch, torchvision
torch.__version__
!git clone https://github.com/facebookresearch/detectron2 detectron2_repo
!pip install -e detectron2_repo
```

Po zainstalowaniu detectrona2 należy uruchomić ponownie środowisko wykonawcze

Import podstawowych modułów Detectrona 2

```
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog, DatasetCatalog
from detectron2.engine import DefaultTrainer
from detectron2 import model_zoo
from detectron2.data.datasets import register_coco_instances
```

Ściagnięcie oraz rejestracja własnych datasetów w formacie COCO

Dzięki temu, że zostały wcześniej przygotowane zbioru danych z opisami w formacie COCO JSON, w Detectronie należy jedynie zarejestrować

Schemat formatu COCO JSON


```
   "info": {
      "year": "2021",
      "version": "5",
      "description": "Exported from roboflow.ai",
      "contributor": "",
      "url": "https://app.roboflow.ai/datasets/volleyball_difficult/5",
      "date_created": "2021-05-19T21:52:57+00:00"
   },
   "licenses": [
      {
         "id": 1,
         "url": "",
         "name": "Unknown"
      }
   ],
   "categories": [

      {
         "id": 1,
         "name": "Ball",
         "supercategory": "Volleball-ball"
      }
   ],
   "images": [
      {
         "id": 0,
         "license": 1,
         "file_name":"VID_20200914_223523
061_jpg.rf.24ddb30db7c4d673bb78de5cb4abd6a9.jpg",
         "height": 720,
         "width": 1280,
         "date_captured": "2021-05-19T21:52:57+00:00"
      },       ...
   ],
   "annotations": [
      {
         "id": 0,
         "image_id": 0,
         "category_id": 2,
         "bbox": [260, 177, 231, 199],
         "segmentation": [...],
```

```
        "area": 45969,
        "iscrowd": 0
    },
  ]
}
```

- **Info** — Opis i informacja o wersji zbioru danych
- **Licenses** — Lista wykorzystywanych licencji
- **Categories** — Klasyfikacja kategorii unikalnym ID. Możliwe dopisanie „superkategorii” która obejmuje wiele klas. W przypadku gdy chcemy korzystać z modeli wstępnie trenowanych na COCO, kategorie muszą się zawierać w klasach COCO
- **Images** — Lista obrazów zbioru z odpowiednimi informacjami takimi jak ID obrazu, ścieżka obrazu, wymiary obrazu i opcjonalnymi atrybutami jak licencje itp
- **Annotations** — Lista anotacji każdy z unikalnym ID i ID obrazu do którego się odnosi. Przechowywane są tutaj informacje o obwiedni/oznaczeniu kategorii.

Dettectron 2 pobiera listę istniejących zbiorów z rejestru, w związku z tym należy zarejestrować własne zbiory do Detectronu 2 w celu treningu i ewaluacji modelów.

```
register_coco_instances("volleyball_easy_train", {},
                        "/content/volleyball_easy/train/_annotations.coco.json",
                        "/content/volleyball_easy/train")
register_coco_instances("volleyball_easy_test", {},
                        "/content/volleyball_easy/test/_annotations.coco.json",
                        "/content/volleyball_easy/test")
register_coco_instances("volleyball_difficult_train", {},
                        "/content/volleyball_difficult/train/_annotations.coco.json",
                        "/content/volleyball_difficult/train")
register_coco_instances("volleyball_difficult_test", {},
                        "/content/volleyball_difficult/test/_annotations.coco.json",
                        "/content/volleyball_difficult/test")
```

```
volleyball_easy_train_metadata = MetadataCatalog.get("volleyball_easy_train")
dataset_dicts = DatasetCatalog.get("volleyball_easy_train")
volleyball_easy_test_metadata = MetadataCatalog.get("volleyball_easy_test")
dataset_dicts2 = DatasetCatalog.get("volleyball_easy_test")
volleyball_difficult_train_metadata = MetadataCatalog.get("volleyball_difficult_train")
dataset_dicts3 = DatasetCatalog.get("volleyball_difficult_train")
volleyball_difficult_test_metadata = MetadataCatalog.get("volleyball_difficult_test")
dataset_dicts4 = DatasetCatalog.get("volleyball_difficult_test")
```

Eksploracja metadanych słowników datasetów.

```
Metadata(evaluator_type='coco', image_root='/content/volleyball_easy/train',
json_file='/content/volleyball_easy/train/_annotations.coco.json',
name='volleyball_easy_train', thing_classes=['Ball'],
thing_dataset_id_to_contiguous_id={1: 0})
```

Dzięki temu można też w łatwy sposób przejrzeć przykładowe zdjęcia ze zbiorów danych:

```
print('Przykładowe zdjęcia z "łatwego" zestawu')
for d in random.sample(dataset_dicts2, 1):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::1],
                            metadata=volleyball_difficult_test_metadata, scale=0.5)
    vis = visualizer.draw_dataset_dict(d)
    cv2_imshow(vis.get_image()[:, :, ::-1])
print('Przykładowe zdjęcia z "trudnego" zestawu')
for d in random.sample(dataset_dicts4, 1):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::1],
                           metadata=volleyball_difficult_test_metadata, scale=0.5)
    vis = visualizer.draw_dataset_dict(d)
    cv2_imshow(vis.get_image()[:, :, ::-1])
```

TRENOWANIE MODELU

Podstawowe konfiguracje modeli pretrenowanych na Coco dataset zamieszczone są w plikach .yaml w katalogu configs/COCO-Detection. W pracy jako wyjściowa została wykorzystanakonfiguracja faster_rcnn_R_50_FPN_3x.yaml.
Zdecydowano się na trenowanie trzyetapowe. W pierwszej kolejności korzystamy ze wstępnie pretrenowanych wagach modelu na datasecie COCO. Wagi wczytujemy bezpośrednio z interfejsu Detektrona.
Następnie każdy z modeli trenowany jest na datasecie **volleyball_easy**. Zbiór ten służy do wstępnego trenowania sieci na łatwiejszych przykładach, w wielu różnych środowiskach. W dalszej kolejności model jest trenowany na datasecie **volleball_difficult**, aby dostosować go realistycznych sytuacji w której moglibyśmy używać detektora. Zdjęcia te pokazują piłkę niewyraźną, w czasie najwyższej prędkości lub częściowo zasłoniętą.
Przy trenowaniu modelu, ze względu na ograniczoną ilość elementów obydwu zbiorów danych zdecydowano się na nie tworzenie pozbioru walidacyjnego.

```
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-
Detection/faster_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("volleyball_easy_train",)
cfg.DATASETS.TEST = ('volleyball_easy_test',)
cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-
Detection/faster_rcnn_R_50_FPN_3x.yaml")
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.005
cfg.SOLVER.MAX_ITER = 4000
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1 # ball class
```

```
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

Tworzenie predyktora i wizualizacja rezultatów:

```
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5  # poziom pewności modelu aby pokazał
detekcje
cfg.DATASETS.TEST = ("volleyball_easy_test", )
predictor = DefaultPredictor(cfg)
```

Sprawdzenie czasu inferencji modelu:

```
import time
times = []
for i in range(20):
    start_time = time.time()
    outputs = predictor(im)
    delta = time.time() - start_time
    times.append(delta)
mean_delta = np.array(times).mean()
fps = 1 / mean_delta
print("Average(sec):{:.2f},fps:{:.2f}".format(mean_delta, fps))
```

Ewaluacja modelu na zbiorze testowym zbiorów danych

```
evaluator = COCOEvaluator("volleyball_easy_test", ("bbox",), False, out
put_dir="./output/")
val_loader = build_detection_test_loader(cfg, "volleyball_easy_test")
print(inference_on_dataset(trainer.model, val_loader, evaluator))
```

Po wykonaniu wstępnego trenowania na datasecie volleyball_easy, dokonujemy inferencji oraz testowania zdjęć na zbiorach testowych zarówno ze zbioru volleyball_easy, jak również i volleyball_difficult. Dzięki temu jesteśmy również w stanie ocenić, czy trenowanie jedynie na obrazkach ogólno dostępnych z OpenImagesV4 jest wystarczające do działania modelu detekcji.
Następnie wykonano kolejne trenowanie na zbiorze **volleyball_difficult**, oraz dokonano inferencji i ewaluacji tego modelu na zbiorze testowym tego datasetu

```
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-
Detection/faster_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("volleyball_difficult_train",)
cfg.DATASETS.TEST = ('volleyball_difficult_test',)
cfg.DATALOADER.NUM_WORKERS = 2
```

```
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final_easy.pth")
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.003
cfg.SOLVER.MAX_ITER = 4000
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1 # ball class


os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)

os.rename('/content/output/model_final.pth','/content/output/model_final_difficult_
easy.pth')
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final_difficult_easy.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = # poziom pewności modelu aby pokazał
detekcje
cfg.DATASETS.TEST = ("volleyball_difficult_test", )
predictor = DefaultPredictor(cfg)
```

Wyniki trenowania przedstawiono w rozdziale nr 8

7.3. Wizualizacja modelu z wykorzystaniem algorytmu Sort

SORT (ang. Simple Online and Realtime Tracking) jest implementacją szkieletu wizualnego śledzenia wielu obiektów bazujące na technikach asocjacji danych i estymacji stanu. Jest zaprojektowany do aplikacji śledzenia, gdzie tylko obecne I przeszłe klatki są dostępne, a obiekty są rozpoznawane natychmiastowe.
Zastosowano istniejącą implementacje algorytmu Sort prze Alexa Bleweya:
https://github.com/abewley/sort

Aby zastosować algorytm, w pierwszej kolejności należy zainicjalizować predyktor. Jako że naszym predyktorem jest Faster R-CNN w interfejsie Detectron2, do inicjalizacji predyktora należy wykonać kroki zgodnie z opisem punktu 7.2 (z pominięciem trenowania, ewaluacji i ekspolarcji danych).
Notatnik z wizualizacją można znaleźć pod linkiem:
https://github.com/KubaRurak/Praca_dyplomowa_Detectron2_Tracking/blob/main/Sort_visualisation_final.ipynb

Inicjalizacja algorytmu wykonywana jest w następujący sposób:

```
#Tworzenie instacji Sort
mot_tracker = Sort(max_age=3, min_hits=2, iou_threshold=0,3)

# Pobranie detekcji
...

# aktualizacja sortu
track_bbs_ids = mot_tracker.update(detections)

# track_bbs_ids is a np array where each row contains a valid bounding box and
track_id (last column)
```

Track_bbs_ids jest to tablica, w której każdy wiersz zawiera obwiednie obiektu oraz identyfikator numeryczny obiektu.
Przy inicjalizacji instacji musimy zdefiniować 3 wartości:

- **Max_age** – maksymalna ilość klatek bez detekcji po której wygasa przypisany do obiektu identyfikator
- **Min_hits** - minimalna ilość klatek pod rząd w których jest detekcja obiektu, tak aby przypisać do obiektu identyfikator. Pomaga w filtracji fałszywych pozytywów. Zdefiniowano tę wartość jako 2
- **Iou_threshold** – minimalna wartość IoU pomiędzy detekcjami wymagana, aby dektor uznał że to jest ten sam obiekt. Wartość zdefiniowano jako 0,3

W ramach pracy wykonano wizualizator filmu przy inicjalizacji algorytmu Sort oraz wcześniej wytrenowanego modelu detekcji obiektu. Wizualizator ten ma możliwość rysowania pozycji i wykrytej średnicy piłki z ostatnich dziesięciu klatek oraz pokazywania toru lotu piłki w ostatnich 10 klatkach.

Dodatkowo zastosowano modyfikację która sztucznie zwiększa obwiednie piłki w celu poprawy wyników. Pełne uzasadnienie tego zabiegu opisano w punkcie 8.2.

Główna pętla po klatkach filmu:

```
def runOnVideo(video, maxFrames,make_big_bbox,draw_line,draw_circle,draw_current_bb
ox):

    readFrames = 0
    count_sort=0
    count_detectron=0
    bbox_size = 400 # rozmiar "oszukanego bboxa do pomocy przy sorcie"
    mot_tracker = Sort()
    frame_array = np.zeros((10,10,5))
    while True:
        hasFrame, frame = video.read()
        if not hasFrame:
            break
        outputs = predictor(frame) # sczytanie detekcji z detectrona
        if outputs['instances'].pred_boxes.tensor.shape[0]>0:
          count_detectron+=1

        pred_bbox = np.array(outputs['instances'].pred_boxes.tensor.cpu())
        scores = np.array(outputs['instances'].scores.cpu())
        bbox_scores, centers, radiuses = switch_to_bigbbox(pred_bbox,outputs,scores
,bbox_size,make_big_bbox)

        # apply SORT algo
        track_bbs_ids = mot_tracker.update(bbox_scores)
        if track_bbs_ids.size > 0:
          count_sort+=1
```

```
        if draw_current_bbox:
          for bbox in bbox_scores:
            draw_bbox(frame,bbox)

        frame_array = ball_position_history(frame,frame_array,
                                        track_bbs_ids,centers,radiuses)
        draw_line_circle(frame,frame_array,draw_circle=False)

        yield frame
        readFrames += 1
        if readFrames > maxFrames:
          break

    print(Ilość klatek z wykryciem z detectrona:{},
              Ilość klatek z wykryciem z Sorta:{}'.format(count_detectron,count_sort))
```

Krótki opis poszczególnych funkcji

Funkcja zmieniająca obwiednie obiektów na duże obwiednie zadane wcześniej w celu poprawienia skuteczności algorytmu. Musimy zapisać również promienie oraz lokalizacje środków obwiedni do dalszej wizualizacji (w przypadku gdybyśmy tego nie zrobili pokazywane byłyby koła ze średnicą „dużą” zadaną przez użytkownika).

```
def switch_to_bigbbox(pred_bbox,outputs,scores,bbox_size,make_big_bbox):
  centers=[]; big_bboxes=[]; radiuses=[];
  for i in range(outputs['instances'].pred_boxes.tensor.shape[0]):
    center = [pred_bbox[i][0]+(pred_bbox[i][2]-pred_bbox[i][0])/2,
              pred_bbox[i][1]+(pred_bbox[i][3]-pred_bbox[i][1])/2]
    centers.append(center)
    if make_big_bbox:
      big_bbox = [center[0]-bbox_size/2,center[1]-
bbox_size/2,center[0]+bbox_size/2,center[1]+bbox_size/2]
    else:
      big_bbox = pred_bbox[i]
    big_bboxes.append(big_bbox)
    radius = min(abs((pred_bbox[i][2]-pred_bbox[i][0])/2),abs((pred_bbox[i][3]-
pred_bbox[i][1])/2))
    radiuses.append(radius)

  if pred_bbox.size==0:
    bbox_scores = np.empty((0,5))
  else:
    bbox_scores = np.append(big_bboxes, np.array([scores]).transpose(),axis=1)

  return bbox_scores, centers, radiuses
```

Funkcja rysująca obwiednie wynikającą z detektora lub wynikająca z zadanej dużej obwiedni:

```
def draw_bbox(frame,bbox,draw_current_bbox=True):
  frame = cv2.rectangle(frame, (int(bbox[0]),int(bbox[1])),(int(bbox[2]
),int(bbox[3])), (0,0,255), 2) # bbox predyktora
  frame = cv2.putText(frame, str(round(bbox[4],2)),
                      (int(bbox[0]+20), int(bbox[1]+20)),
                      cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255,0,0), 2)
  return frame
```

Funkcja zwracająca tablicę historii pozycji piłki z ostatnich 10 klatek:

```
def ball_position_history(frame,frame_array,track_bbs_ids,centers,radiuses):
  frame_array = frame_array[1:10] #usuwamy tablice 10 frame'y temu
  zeros = np.zeros((1,10,5))
  frame_array = np.concatenate((frame_array,zeros)) #dodajemy pusty frame na koniec
  for i in range(track_bbs_ids.shape[0]):
    coord_x = track_bbs_ids[i][0]+(track_bbs_ids[i][2]-track_bbs_ids[i][0])/2
    coord_y = track_bbs_ids[i][1]+(track_bbs_ids[i][3]-track_bbs_ids[i][1])/2
    ball_id = track_bbs_ids[i][4]
    frame = cv2.putText(frame, str(ball_id), (int(coord_x-20), int(coord_y-20)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255,0,0), 2)
    for center,radius in zip(centers,radiuses):
      if (abs(coord_x - center[0])<20) and (abs(coord_y - center[1])<20):
        frame_array [9][i] = [coord_x,coord_y,radius,0,ball_id]
  return frame_array
```

Funkcja rysująca koła o średnicy wykrytej piłki lub tor lotu z ostatnich 10 klatek:

```
def draw_line_circle(frame,frame_array,draw_line=True,draw_circle=True):
  end_points=[(0,0),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0)]
  end_ids=[0,0,0,0,0,0,0,0,0,0]
  for one_frame in frame_array:
    start_points=end_points.copy()
    start_ids=end_ids.copy()
    for i,row in enumerate(one_frame):
      if draw_circle:
        cv2.circle(frame, (int(row[0]),int(row[1])), int(row[2]), (0,255,255), -1)
      end_ids[i] = row[4]
      end_points[i] = (int(row[0]),int(row[1]))
      if draw_line:
        if end_ids[i]!=0:
          for id,start_point in zip(start_ids,start_points):
            if id==end_ids[i]:
              cv2.line(frame,start_point,end_points[i],(255,0,0),8)
  return frame
```

Następnie stosujemy funkcję która zbiera potrzebne właściwości zadanego filmu i inicjalizuję funkcję z pętlą główną runOnVideo

```
def writeVideo(video_name:str,video_output_name:str,make_big_bbox=True,draw_current
_bbox=False,draw_line=True,draw_circle=False):
  video = cv2.VideoCapture(video_name)
  width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
  height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
  frames_per_second = video.get(cv2.CAP_PROP_FPS)
  num_frames = int(video.get(cv2.CAP_PROP_FRAME_COUNT))

  # Initialize video writer
  video_writer = cv2.VideoWriter(video_output_name, fourcc=cv2.VideoWriter_fourcc(*
"mp4v"), fps=float(frames_per_second), frameSize=(width, height), isColor=True)
  mot_tracker = Sort()

  # # Inicjalizacja wizualizacji

  num_frames = num_frames
  for frame in tqdm.tqdm(runOnVideo(video, num_frames,make_big_bbox,draw_line,draw_
circle,draw_current_bbox), total=num_frames):

      video_writer.write(frame)

  video.release()
  video_writer.release()
  cv2.destroyAllWindows()
```

7.4. Wizualizacja modelu z wykorzystaniem algorytmu Deep Sort

Deep Sort jest “dobudówką” do algorytmu Sort, uwzględnia ona dodatkowo metrykę bazującą na wyglądzie obiektu.

Wykorzystano zmodyfikowaną implementację Deep Sorta z linku:
https://github.com/sayef/detectron2-deepsort-pytorch

Wykonany wizualizator można znaleźć pod linkiem:
https://github.com/KubaRurak/Praca_dyplomowa_Detectron2_Tracking/blob/main/Deepsort_visualisation_final.ipynb

Opis plików repozytorium

- **detection.py**: - klasa reprezuntująca detekcję 1 klatki
- **kalman_filter.py**: Implementacja filtru kalmana
- **linear_assignment.py**: - Moduł dopasujący funkcją minimalnego kosztu
- **iou_matching.py**: Moduł dopasowujący IoU
- **nn_matching.py**: - Metryka najbliższego sąsiada (ang. nearest neighbour)
- **track.py**: Klasa track zawierająca dane dla pojedynczego obiektu jak np. stan Kalmana, ilość detekcji, wektor cech itp.
- **tracker.py:** Klasa wielu obiektów track
- **demo_detectron2_deepsort.py** – Inicjalizacja całego algorytmu Deep Sort
- **detectron2_detection.py** – Inicjalizacja predyktora
- **demo_detectron2_deepsort.py** – Łączenie predyktora i algorytmu Deep sort, tworzenie wizualizacji
- **util.py** – Moduł tworzenia obwiedni na wizualizacji

Trenowanie Feature Extractora

Trenowanie Feature extractora przeprowadzono metodą transfer learning. Zastosowano standardowy wstępnie wytrenowany na imagenecie ResNet-50. Następnie zamrożono wagi sieci i dodano 2 liniowe warstwy. Model wytrenowano do klasyfikacji obiektu na autorskim zbiorze **ball_noball**.
Notatnik trenowania Feature Extractora znajduje się pod linkiem: https://github.com/KubaRurak/Praca_dyplomowa_Detectron2_Tracking/blob/main/training_feature_extractor_final.ipynb

Ze względu na to, że feature extractor stosujemy w celu badania dystansu między wektorami cech, a nie do klasyfikacji obiektu, pobierane będą wartości wynikowe wektora cech o wymiarach 128x1.

```
model = models.resnet50(pretrained=True).to(device)

for param in model.parameters():
    param.requires_grad = False

model.fc = nn.Sequential(
               nn.Linear(2048, 128),
               nn.ReLU(inplace=True),
               nn.Linear(128, 2)).to(device)
```

Aby sieć zwracała ten wektor zastosowano następującego hooka:

```
        def hook(module, input, output):
            return output
        self.net.fc[0].register_forward_hook(hook)
```

Inicjalizacja detektora oraz wizualizacja detekcji:

Tak jak w przypadku Sorta, naszym detektorem jest predyktorem jest Fastern R-CNN w interfejsie Detectron2, w związku z tym należy zainicjalizować predyktor.
Odbywa się to w pliku detectron2_detection.py

W celu porównania wyników pomiędzy Sortem a Deep Sortem, parametry części Sorta wpływające na ilość detekcji są takie same jak w punkcie 7.3 czyli

- **Min_hits** - minimalna ilość klatek pod rząd w których jest detekcja obiektu, min_hits=2
- **Iou_threshold** – minimalna wartość IoU pomiędzy detekcjami wymagana, aby dektor uznał że to jest ten sam obiekt, Iou_threshold = 0,3

Dodatkowo przyjęto również taki sam poziom pewności przy predyktorze czyli:
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.3

Tak samo jak w przypadku Sorta do wizualizatora dodano również możliwość rysowania wykrytego toru lotu piłki w ostatnich 10 klatkach filmu

8. Porównanie wyników modeli oraz wnioski

8.1. Wyniki i porównanie modeli detekcji obiektów

Na decydującym zbiorze testowym najlepsze wyniki mAP, AP50 i APs osiągnął model **Faster_RCNN_50R_FPN_3x**, natomiast najsłabszy był model Faster R-CNN z backbone mobilenetu.
W przypadku prędkości inferencji wyniki są odwrotne – najlepszą ilość inferencji na sekundę uzyskał model **Faster_RCNN_MV2_FPN_3x,** a najsłabszy wynik osiągnął model **Faster_RCNN_50R_FPN_3x.**

Otrzymane wyniki zostały zaprezentowane w poniższych tabelach.

Wyniki ewaluacji po trenowaniu na zbiorze volleyball_easy przedstawiono w tabelach 8.1. i 8.2. Wyniki ewaluacji po trenowaniu na zbiorach volleyball_easy, a następnie na zbiorze volleyball_difficult przedstawiono w tabeli 8.3.

Model	AP	AP50	APs
Faster_RCNN_50R_FPN_3x	51,236	82,150	31,403
Faster_RCNN_MV2_FPN_3x	45,823	77,811	26,746
Retinanet_50R_FPN_3x	47,477	79,270	28,231

Tabela 8.1. Ewaluacja modelu na zbiorze testowym volleyball_easy, po trenowaniu na zbiorze volleyball_easy

Model	AP	AP50	APs
Faster_RCNN_50R_FPN_3x	8,566	32,676	8,258
Faster_RCNN_MV2_FPN_3x	5,087	17,731	4,348
Retinanet_50R_FPN_3x	3,427	11,074	1,922

Tabela 8.2. Ewaluacja modelu na zbiorze testowym volleybal_diffcult, po trenowaniu na zbiorze volleyball_easy

Model	AP	AP50	APs
Faster_RCNN_50R_FPN_3x	40,074	69,396	40,737
Faster_RCNN_MV2_FPN_3x	31,976	64,076	33,480
Retinanet_50R_FPN_3x	34,371	68,933	35,147

Tabela 8.3. Ewaluacja modelu na zbiorze testowym volleybal_diffcult, po trenowaniu na zbiorze volleyball_easy oraz volleyball difficult.

Wyniki inferencji modeli

Pomiary czasu inferencji wykonano na Colabie z wykorzystaniem karty graficznej NVidia Tesla K80.

Model	Czas inferencji 1 obrazka	FPS
Faster_RCNN_50R_FPN_3x	0,268	3,731
Faster_RCNN_MV2_FPN_3x	0,097	10,256
Retinanet_50R_FPN_3x	0,238	4,201

Tabela 8.4. Szybkośc inferencji na zbiorze volleyball_easy

Model	Czas inferencji 1 obrazka	FPS
Faster_RCNN_50R_FPN_3x	0,402	2,487
Faster_RCNN_MV2_FPN_3x	0,141	7,058
Retinanet_50R_FPN_3x	0,359	2,785

Tabela 8.5. Szybkośc inferencji na zbiorze volleyball_difficult

Otrzymane wyniki doskonale pokazują sens wykonania autorskiego zbioru obrazów **volleyball_difficult**. Widać, że detektory wytrenowane jedynie na datasecie COCO i publicznie dostępnych obrazach z OpenImages (dataset **volleyball_easy**) kompletnie sobie nie radzą przy detekcji piłki przy realistycznych sytuacjach. Dopiero po dodatkowym trenowaniu na realistycznym zbiorze treningowym predyktor nadaje się do wykorzystania z algorytmami śledzenia piłki.

8.2. Wyniki i porównanie sposobów śledzenia obiektów

W tej pracy nie zdecydowano się na zastosowanie standardowych metryk śledzenia obiektów MOT (Multiple Object Tracking). Metryki takie jak Multiple Object Tracking Precision czy Multiple Object Tracking Accurac wymagają posiadania filmu „ground truth", którego nie posiadam.

W pracy metody śledzenia obiektów są porównywane głównie w sposób wizualny. Jedyną metryką liczbową jaką zastosowano jest zestawienie ilości klatek w rozpatrywanym filmie, do ilości klatek z detekcją Faster R-CNN oraz ilości klatek z wykrytym obiektem przez algorytm śledzenia obiektów. Ze względu na fałszywe pozytywy nie jest to bardzo dokładna metoda porównawcza, nie mniej pokazuje skale różnicy pomiędzy detektorami.

Porównywane są 3 metody śledzenia obiektów:

- Sort bez modyfikowania obwiedni
- Sort z powiększoną obwiednią
- Deep Sort bez modyfikowania obwiedni

Ze względu na najlepsze wyniki, do algorytmów Sort i Deep Sort zastosowano jedynie predyktor architektury **Faster_RCNN_50R_FPN_3x**.

Uzasadnienie zastosowania Sort'a z powiększoną obwiednią

Jednym z głównych parametrów w algorytmie Sort który wpływa na skuteczność detekcji jest IoU_threshold czyli poziom minimalnego IoU dla którego wykonywana jest detekcja. Ze względu na to że piłka jest małym obiektem, który porusza się bardzo szybko, często się zdarza sytuacja gdzie pomiędzy dwiema klatkami filmu IoU wynosi 0. Poza skutecznością detektora, jest to główny powód dla którego nasz tracker „gubi" rozpatrywany obiekt. Dzięki sztucznemu powiększeniu obwiedni na potrzebę algorytmu, problem ten znika i algorytm zyskuje na skuteczności (jak w przypadku większych i wolniejszych obiektów).

Rys. 8.1. Detekcja obwiedni „realnej”

Rys 8.2. Obwiednia powiększona

W Deep Sorcie zmiana obwiedni nie ma sensu ze względu na rolę feature extractora. W przypadku zmiany obwiedni na większą, feature extractor rozpatrywałby nie tylko podobieństwo wektora cech piłki ale również i całego tła w obwiedni, w związku z tym to jedynie pogarsza „głęboką” część algorytmu pomagając częsci „Sort”. Z tego względu nie stosowano tej metody.

Zestawienie detekcji Faster R-CNN ze śledzeniem trackera.

Porównania dokonano na 5 filmach, wyniki zaprezentowane są w tabeli poniżej.

Nr Klipu	Ilość klatek klipu	Ilosc klatek z predykcją detektora	Ilość klatek z detekcją Sort	Ilość klatek z detekcją Sort – powiększona obwiednia	Ilość klatek z detekcją Deep Sort
1	458	448	403	434	438
2	300	155	43	126	68
3	1824	1274	475	1150	764
4	1514	894	342	745	697
5	2847	1555	484	1318	822

Tabela 8.6. Porównanie algorytmów śledzenia

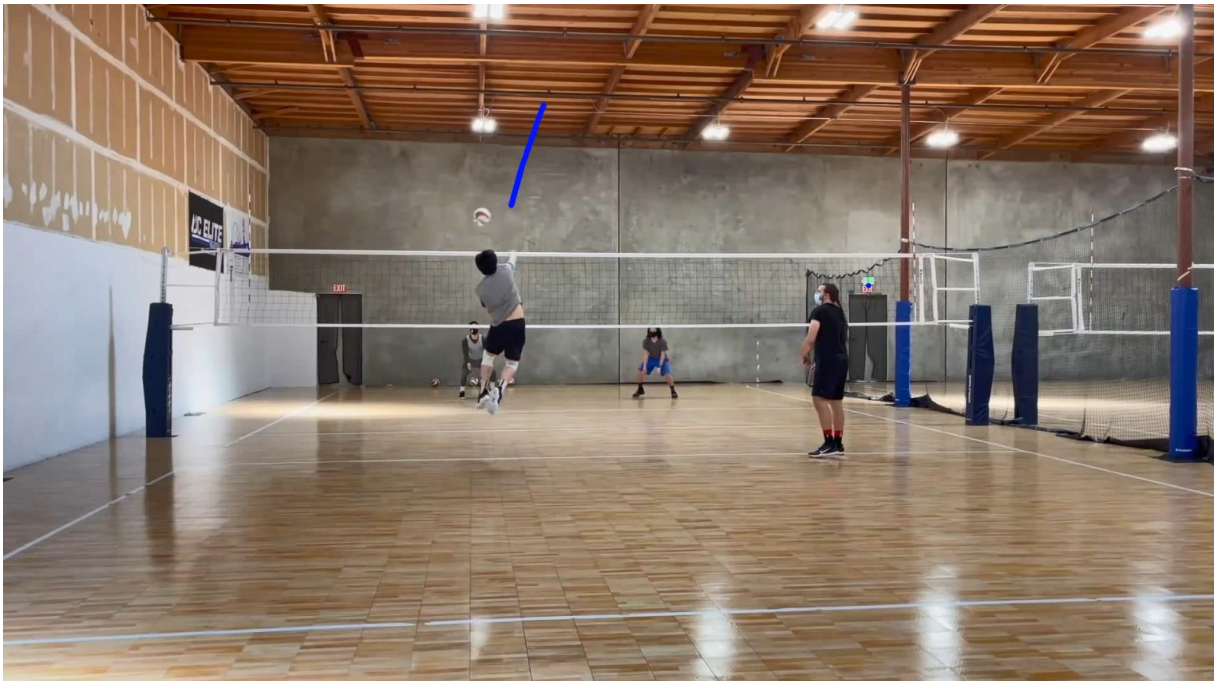
Zestawienie przykładowych obrazów wizualizacji

Porównania wizualne filmów.

Rys 8.3. Sort – tracker nie wykrywa piłki

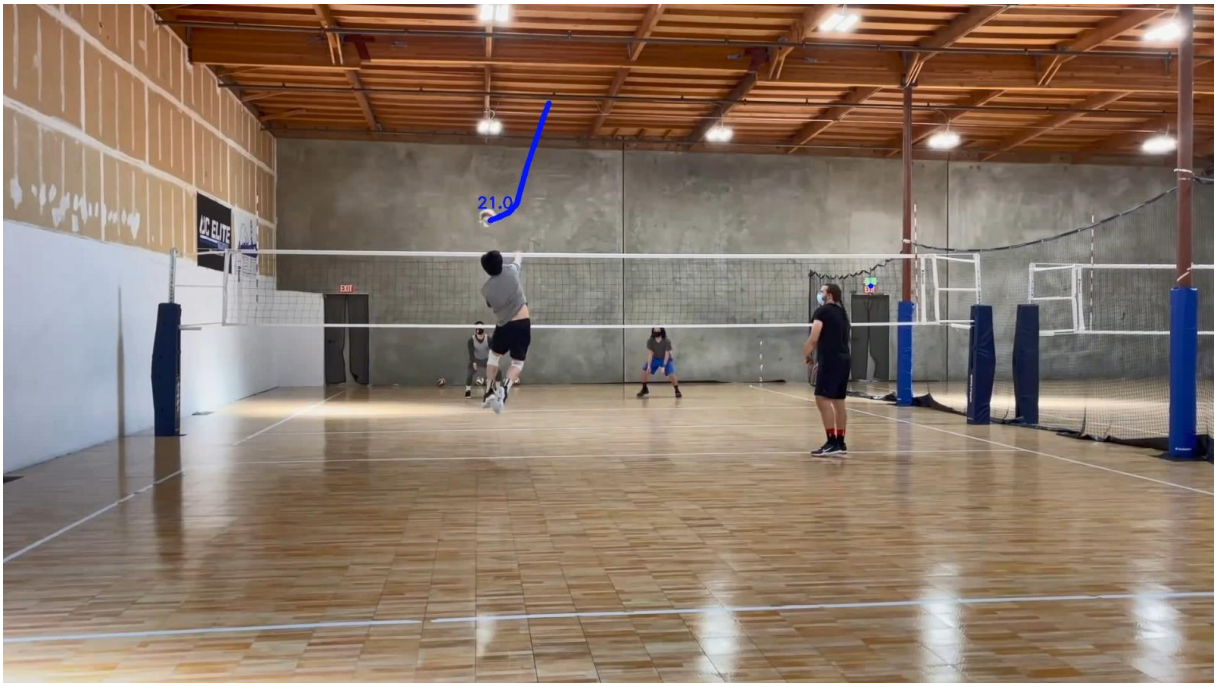
Rys. 8.4. Deeport – tracker wykrywa piłkę

Rys. 8.5. Sort zmodyfikowany – tracker wykrywa piłkę



Rys. 8.6. Sort – ucięcie detekcji w momencie ataku

Rys. 8.7. Deep Sort – ucięcie detekcji w momencie ataku



Rys. 8.8. Sort zmodyfikowany – ciągłość detekcji

W przypadku Sorta ze zmodyfikowaną obwiednią wykonano również inferencje które pokazują koła w ostatnich 10 klatkach. Dzięki temu widać, że nie da się realistycznie sczytać stosunku wielkości piłki gdy jest blisko i gdy jest daleko. W przypadku takiej dokladności predyktora nie da się określić prędkości piłki na podstawie zmiany jej rozmiaru.


Rys. 8.9. Pomimo że piłka leci prostopadle do kamery jej wykryty rozmiar pomiędzy klatkami dużo się rózni

Wszystkie wizualizacje filmów można znaleźć pod linkiem:
https://drive.google.com/drive/folders/1CS_1OMSb-b3g_DdGdmbutmehGqSs2Le2?usp=sharing

Zarówno przy porównywaniu „skutecznych” ilości klatek ze śledzeniem obiektu jak i na obrazach i filmach, widać że najskuteczniejszą metodą jest Sort z powiększeniem obwiedni. W skuteczności zdecydowanie odstają Deep Sort oraz Sort bez modyfikacji. Deep Sort od zwykłego Sorta jest lepszy dzięki zastosowaniu feature extractora, nie mniej wyniki Deep Sorta nie są zadawalające.
Według tabeli 8.6. Deep Sort powinien się wizualnie lepiej sprawować od niezmodyfikowanego Sorta, jednak na wizualizacjach przewaga Deep Sorta nie jest bardzo znacząca. Stwierdzono, że algorytm DeepSort dłużej utrzymuje wykryty obiekt, ale z tego powodu również dłużej utrzymuje wykryte fałszywe pozytywy (co przyczynia się do dobrego wyniki na klipie nr 1). Na klipie nr 2 nie stwierdzono wielu fałszywych pozytywów a wynik Deep Sorta jest jest lepszy od niezmodyfikowanego Sorta, w związku z tym da się stwierdzić poprawienie wyników dzięki zastosowaniu głębokich metryk.

Nawet w przypadku najlepszej metody śledzenia, tracker nie zawsze sobie radzi w przypadku piłek atakowanych w linii prostopadłej do kamery (w szczególności przy rozgrywkach profesjonalnych gdzie prędkość piłki osiąga do 120km/h). Jest to jednak wina detektora, ze względu na to że zawsze sobie on radzi z rozmazanymi piłkami przez efekt motion blur.

8.3. Wnioski

W niniejszej pracy opracowano i przeanalizowano modele umożliwiające detekcję piłki w oparciu o architekturę Faster R-CNN (na backbone Resnet-50 oraz MobileNetV2) i RetinaNet. Sieci wytrenowano na dwóch zbiorach danych volleyball_easy oraz autorskim datasecie volleyball_difficult. Dodatkowo w pracy sprawdzono algorytmy śledzenia Sort oraz DeepSort do zastosowania przy śledzeniu piłki do siatkówki, co można uogólnić do problemu śledzenia szybko poruszających się małych obiektów.

W przypadku architektur sieci detekcji obiektów, z problemem najlepiej sobie poradził Faster R-CNN na backbone Resneta-50. Algorytmy detekcji porównano metrykami COCO AP.

W przypadku algorytmów śledzenia piłki, najlepiej sobie poradził z zadaniem Sort z modyfikacją wielkośći obwiedni obiektu. Ze względu na to że między pojedynczymi klatkami nagrań piłka mocno zmienia swoją pozycję oraz zmienia swój wygląd (ze względu na niedoskonałości kamery) z zadaniem słabo sobie poradziły algorytm Sort i Deep Sort.
Najbardziej wrażliwym momentem przy śledzeniu piłki są momenty uderzenia piłki. Algorytm Sort ze względu na zastosowanie filtrów Kalmana nie zawsze jest w stanie sobie poradzić z obiektem gwałtownie zmieniającym wektor ruchu.

Stwierdzono do weryfikacji kilka możliwych usprawnień w stosunku do zastosowanej metody:

- Ze względu na to, że głównym „zatorem” w skuteczności śledzenia piłki jest detektor, należałoby sprawdzić czy nastąpiłaby zdecydowana poprawa wyników przy większych datasetach treningowych.
- Rezygnacja z tego, że tracker dokonuje obliczeń „na żywo” i przewiduje obecną lokalizacje piłki na podstawie klatek poprzednich i obecnej detekcji. Oprócz tego rozpatrzyć możliwość wypełnienie luk z brakiem detekcji na podstawie przyszłej pozycji piłki.

Bibiliografia

1. Alina Kuznetsova, Hassan Rom, et. al „The Open Images Dataset V4. Unified image classification, object detection, and visual relationship detection at scale” Pre-print accepted to IJCV 2018
2. Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.
3. Van de Sande, Koen EA, et al. "Segmentation as selective search for object recognition." 2011 International Conference on Computer Vision. IEEE, 2011.
4. Girshick, Ross. "Fast r-cnn." Proceedings of the IEEE international conference on computer vision. 2015.
5. Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." Advances in neural information processing systems. 2015.
6. Hiroto Honda, “Digging into Detectron 2”, URL: https://medium.com/@hirotoschwert/digging-into-detectron-2-47b2e794fabd 2020.
7. Lin, Tsung-Yi, et al. "Feature pyramid networks for object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.
8. Lin, Tsung-Yi, et al. "Focal loss for dense object detection." Proceedings of the IEEE international conference on computer vision. 2017.
9. Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 2017.
10. Kunlun Bai. „A Comprehensive Introduction to Different Types of Convolutions in Deep Learning”, URL: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215 2019.
11. Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.
12. Hollemans, Matthijs. "MobileNet version 2." URL http://machinethink.net/blog/mobilenet-v2 2018.
13. Kaiming He, Xiangyu Zhang et. al. „Deep Residual Learning for Image Recognition” IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2016.
14. Masoud Hoveidar , „Deep Residual Learning for Image Recognition | AISC URL: https://www.youtube.com/watch?v=jio04YvgraU&ab_channel=MLExplained-AggregateIntellect-AI.SCIENCE
15. Darko Jurić, „Object Tracking: Particle Filter with Ease”, URL: https://www.codeproject.com/Articles/865934/Object-Tracking-Particle-Filter-with-Ease
16. Alex Bewley, Zongyuan Ge et. al. „Simple Online and Realtime Tracking” arXiv:1602.00763v2 2017.
17. Nicolai Wojke, Alex Bewley, et al. „Simple Online and Realtime Tracking with a Deep Association Metric” arXiv:1703.07402v1 2017.