5. Program wykrywający

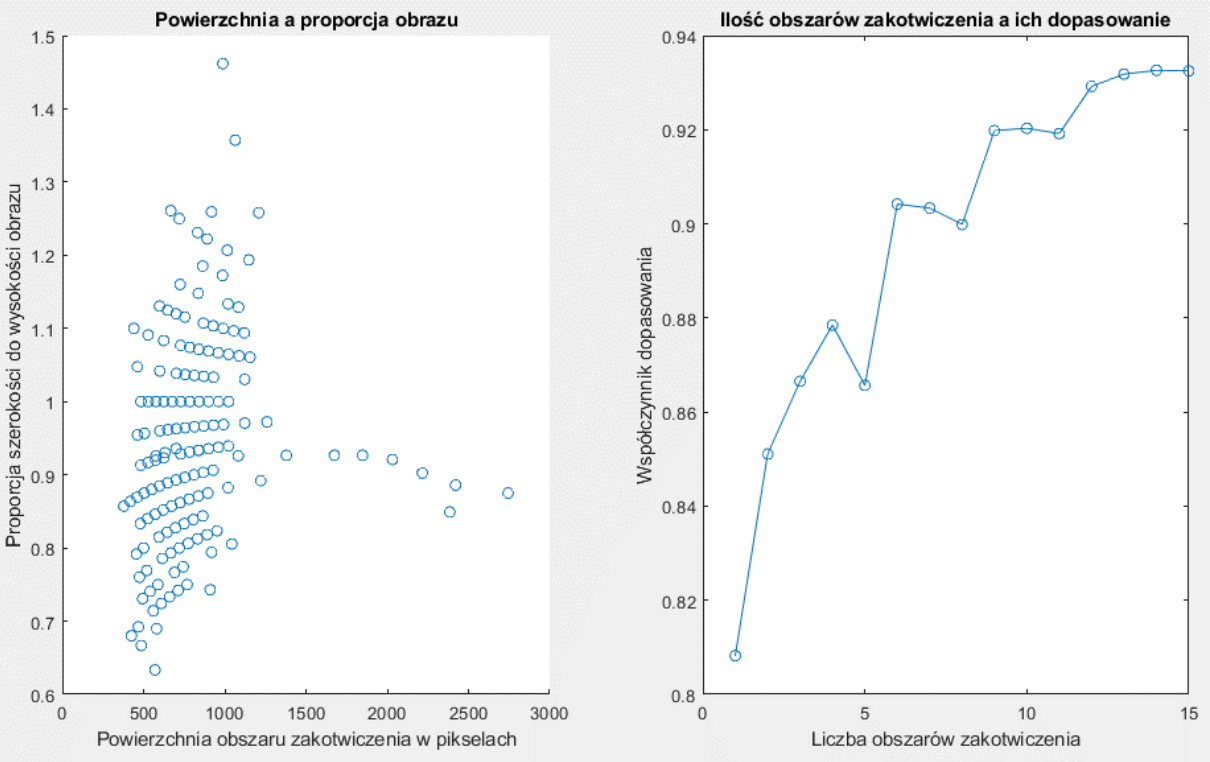
5.1. Tworzenie sieci neuronowej

Sieć neuronowa przygotowana do wykrywania obiektów na obrazie składa się z dwóch części: rozpoznającej oraz wykrywającej. Pierwsza z nich w żaden sposób nie jest w stanie stwierdzić miejsca, w którym znajduje się zadany obiekt. Określa ona prawdopodobieństwo rozpoznania obiektu dla całej przekazywanej sceny. Tak więc może ona świetnie rozpoznawać przykładową piłkę jeśli zajmuje ona cały jej kadr, albo jego zdecydowaną większość [22].

Jeśli na zadanym obrazie tylko jego część stanowi zadany obiekt i ma zostać określone miejsce, w którym się on znajduje to rozpoznająca sieć neuronowa musi zostać uzupełniona o część wykrywającą. Powstało kilka algorytmów realizujących to zadanie i każdy z nich różni się w pewien sposób od pozostałych. Ich częścią wspólną każdorazowo jest natomiast podział początkowego obrazu na prostokątne części o różnych rozmiarach i wykonanie rozpoznawania na każdym z nich.

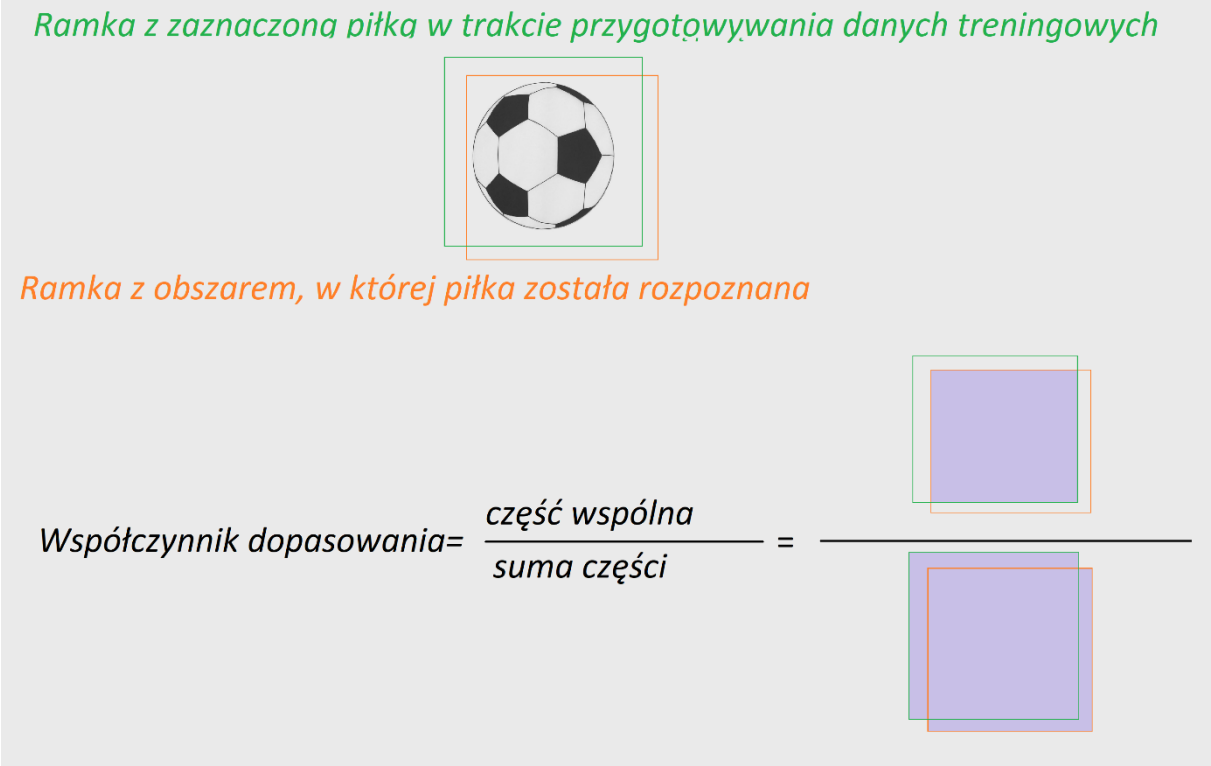
Przy tworzeniu pełnej sieci neuronowej będzie użyta gotowa rozpoznająca oraz opracowany już algorytm detekcji w formie warstw dodanych na końcu tej pierwszej. Kierując się założeniem projektowym, że całe rozwiązanie ma działać w czasie rzeczywistym należy wybrać najszybsze z nich. Będzie ona przygotowywana w oprogramowaniu MatLab, które oferuje wiele z nich dedykowanych do tego rodzaju mechanizmów. Tak więc wybraną siecią neuronową rozpoznającą jest *squeezenet*, gdyż mimo, że cechuje się ona najmniejszą skutecznością względem pozostałych to ma ona najmniejszą ilość warstw, dzięki czemu działa najszybciej [23]. Również algorytm wykrywający został wybrany najszybszy z oferowanych- *YoloV2* [24].

Po połączeniu warstw uzyskuje się sieć neuronową, którą należy jeszcze wytrenować przy użyciu danych treningowych. Do ich stworzenia został użyty filmik, na którym znajduje się cel obserwacji, a więc piłka. Została ona nagrana tą samą kamerką, która będzie potem odpowiadać za jej wykrywanie, a także w takiej samej rozdzielczości obrazu, w której będzie ona później pracować. Specyfikacja działania algorytmu detekcji wymusza zaznaczanie danych treningowych w postaci prostokątów. Przedmiot obserwacji jest okrągły i nigdy nie będzie zajmował całego przeznaczonego dla niego obszaru. Z tego powodu powinien się znajdować na każdym możliwym tle, aby w jak najmniejszym stopniu wpływało ono na wynik części rozpoznającej obraz. Obiekt został zaznaczony w każdej klatce na której był obecny za pomocą aplikacji *Video Labeler* dostępną w środowisku *MatLab*, a dane treningowe zostały wyeksportowane do obsługiwanego przez to środowisko formatu *Ground Truth Data* [25].



Rysunek 14. Estymacja obszarów zakotwiczenia

Twórcy algorytmów wykrywających kierując się szybkością ich działania stawali przed pewnymi problemami podczas ich tworzenia. Najważniejsze z nich to na jakie fragmenty podzielić obraz wejściowy do rozpoznawania oraz jak gęsto powinien być nim pokryty. Algorytm wykrywający *YOLOv2* w swoim działaniu używa obszarów zakotwiczenia (ang. *anchor boxes*). Są to przygotowane obwiednie na wykrywane obiekty w kształcie prostokątów o z góry ustalonej szerokości oraz wysokości. Obraz wejściowy będzie potem dzielony właśnie na fragmenty o takich wielkościach, które może przekształcać lub obracać. Powinny one stanowić przybliżony obszar, od którego oczekuje się, że zajmie wykrywany obiekt na obrazie. Im więcej ich będzie wdrożonych tym lepiej dla skuteczności algorytmu kosztem jego prędkości. Ich wielkości oraz ilość może zostać zdefiniowana orientacyjnie przez osobę tworzącą algorytm, ale mogą zostać także estymowane na podstawie danych treningowych. Przy tworzeniu zdecydowano się na ten drugi sposób. Szacowanie tych obszarów zrealizowano dla docelowej ich ilości od 1 do 15 na podstawie przygotowanych już danych treningowych.



Rysunek 15. Sposób obliczania współczynnika dopasowania [26].

Na *Rysunek 14* przedstawiono dwa wykresy. Na wykresie po lewej stronie można dostrzec, że proporcja wysokości do szerokości dla większości z nich zawiera się w przedziale od 0,8 do 1,1. Obszary zakotwiczenia będą zatem bardzo bliskie kształtu kwadratu. Ponadto ich wielkości zawierają się najczęściej między 500 a 1000 pikseli. Na wykresie po prawej stronie widać, że skuteczność algorytmu po zastosowaniu większej ilości niż 6 przygotowanych tutaj obszarów wzrasta nieznacznie, a nawet potrafi się obniżyć. W algorytmie została użyta właśnie taka ich ilość z opisywanych tutaj estym. Po przygotowaniu struktury sieci neuronowej, danych treningowych oraz obszarów zakotwiczenia sieć neuronowa została wytrenowana i jest gotowa do użycia.

5.2. Tworzenie programu wykrywającego.

Program wykrywający został przygotowany w środowisku MatLab w taki sposób, aby mógł zostać przekonwertowany do języka *C++* za pomocą dodatku *MatLab Coder* i skompilowany na urządzeniu docelowym- *Raspberry Pi* [27]. Nie wszystkie elementy składni i funkcje języka *MatLab* mogą zostać bezpośrednio obsłużone przez ten dodatek. Jedną z takich nieobsługiwanych funkcjonalności jest funkcja *find*, która potrafi znaleźć konkretną daną w wektorze zmiennych. Z powodu niemożności jej użycia została użyta klasyczna pętla do wyszukania konkretnej wartości. Ponadto został on przygotowany dla kwadratowego obrazu, którego szerokość i wysokość są takie same.

```
%codegen
function detection()

%załączenie funkcji pomocniczych
coder.cinclude("helpers.cpp");

%Zmienna szerokość/ wysokość obrazka. Skrypt przygotowany dla kwadrato-
wego obrazka.
side=224;

%zmienna przechowująca sieć neuronową
persistent yolov2Obj=
coder.loadDeepLearningNetwork('YoloV2detector-20210117T135802.mat');
%alokacja pamięci dla zmiennej przechowywującej obraz
img=ones(side, side, 3, 'uint8');

%główna pętla programu
while 1
    %pobieranie obrazku ze strumienia wejściowego
    coder.ceval('getFrame', coder.ref(img), side);

    %wykrywanie
    [bboxes,scores,labels] = yolov2Obj.detect(img,'Threshold',0.5);

    %sprawdzenie czy obiekt został wykryty
    if ~isempty(scores) %obiekt wykryty!
        %szukanie obiektu z największym współczynnikiem pewności
        m=max(scores); index=0;
        for i = 1:length(scores)
            if scores(i)==m
                index=i;
            end
        end

        %wyliczanie współrzędnej azymutalnej obiektu
        x=bboxes(index,1)+bboxes(index,3)/2;
        ang=atan2(x-side/2,(side)*tan(62.2*(pi/180)))*180/pi;

        %przygotowywanie i nanoszenie adnotacji
        annotation = sprintf('C:%.1f A:%.1f',scores(index),ang);
        img = insertObjectAnnotation(img,'rectangle',bboxes(index,:),an-
notation);

        %wypisywanie tej współrzędnej na strumień błędu
        coder.ceval('printErr',[ sprintf('%0.0d',int32(ang)) 10 ]);
    else %obiekt nie został wykryty!
        %nanoszenie adnotacji o braku wykrycia obiektu
        img = insertText(img,[1 1], 'no detections', 'FontSize',12, 'Box-
Color',[255 0 0],'BoxOpacity',0.4, 'TextColor',[255 255 255]);
    end
    %przesyłanie przygotowanego obrazu na strumień wyjścia
    coder.ceval('printFrame', img, side);
end
```

Kod źródłowy 2. Program wykrywający w środowisku MatLab.

MatLab Coder bardzo dobrze radzi sobie z procesem zmiany kodu na *C++* i kompilowaniu go na urządzeniu takim jak *Raspberry Pi*. Jednak gdy rozwiązanie wykorzystuje w swoim działaniu sieci neuronowe to kompilacja ich wymaga wskazania rodzaju biblioteki do tego celu używanej oraz musi być ona zainstalowana na urządzeniu docelowym. Biblioteką wybraną do tego celu jest *Compute Library* przygotowaną przez firmę *ARM*. Została ona zainstalowana na *Raspberry Pi* w wersji 19.02, którą dodatek *MatLab Coder* w pełni obsługuje [28].

Kod przygotowywany do zamiany na język *C++* oraz późniejszej kompilacji jest przygotowany w formie funkcji, w której zawiera się główna pętla programu. Takie podejście ułatwia cały proces, gdyż nie trzeba wskazywać oraz przygotowywać głównego pliku z główną funkcją *main*, która jest nierozłącznym elementem każdego programu w nim napisanego i wykonuje się po jego uruchomieniu. Bez jego wskazania taki plik jest generowany automatycznie, a w głównej funkcji programu wykonuje się jednokrotne wywołanie przygotowanej funkcji, która ma zostać docelowo skompilowana.

Program konwertowany z *MatLab* do języka *C++* może wykorzystywać funkcje napisane w tym drugim języku za pomocą *coder.ceval* [29]. Taka możliwość została wykorzystana dla trzech z nich, które łatwiej było zaimplementować, gdyż obsługa strumieni wejścia, wyjścia oraz błędu jest wówczas łatwiejsza:

- *getFrame*- pobiera obraz ze standardowego wejścia i zapisuje go do zmiennej przekazywanej przez referencję za pomocą *coder.ref*. Wymiar boczny kwadratowego obrazu jest podawany jako drugi argument.
- *printFrame*- wypisuje obraz na standardowe wyjście. Pierwszym argumentem jest obraz, a drugim jest wymiar boczny kwadratowego obrazu.
- *printErr*- przyjmuje tylko jeden argument. Jest nim tekst, który zostaje wypisywany na strumień błędu.

Działanie programu po zainicjowaniu zmiennych początkowych oraz załączeniu funkcji pomocniczych zawiera się w głównej pętli *while*. W każdym jej przejściu pobierany jest najpierw obraz z kamery, a następnie na nim wykonywana jest próba wykrycia zadanego obiektu. Gdy zostanie on rozpoznany w ilości większej niż jeden zostaje wybrany ten, dla którego współczynnik pewności ma największą wartość i zostaje on ujęty w ramkę. Dodatkowo na strumień błędu jest wysyłana jego współrzędna azymutalna. Gdy obiekt nie został rozpoznany na obrazie to w rogu ujęcia zostaje umieszczona informacja o braku jego wykrycia.