AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Praca dyplomowa

System detekcji wad wyrobów z linii produkcyjnej w oparciu o cyfrowe przetwarzanie obrazów z wykorzystaniem sieci neuronowych

System for detecting product defects from the production line based on digital image processing using neural networks

Autor:	*Sławomir Serafin*
Kierunek studiów:	Elektrotechnika
Opiekun pracy:	*dr inż. Krzysztof Chmielowiec*
Recenzent:	*dr inż. Krzysztof Piątek*

Kraków, 2022

Spis treści:

1 Wprowadzenie

1.1 Cel projektu

Celem projektu jest wykonanie programu, który będzie wykorzystywał cyfrowe przetwarzanie obrazów w celu automatyzacji procesów produkcyjnych w zakładzie przemysłowym specjalizującym się w branży meblarskiej. Dodatkowo w celu zwiększenia precyzji zostanie wykorzystana sieć neuronowa z wykorzystaniem biblioteki TensorFlow.

1.2 Założenia przedsięwzięcia oraz zdefiniowanie problemu

Zadanie projektowe polega na zliczaniu otworów w środkowej sekcji komponentu (kolor czerwony na zamieszczonym rysunku). Obiekt będzie poruszał się dynamicznie po linii produkcyjnej z wcześniej zadaną prędkością. Program przed rozpoczęciem pracy powinien zostać dostosowany do aktualnych warunków oświetleniowych panujących na zakładzie. Jeżeli okaże się to niezbędne, koniecznie może być zastosowanie dodatkowych źródeł światła aby uzyskać jednolite parametry pracy w dłuższej perspektywie czasowej. Dodatkowo środowisko powinno działać z wydajnością, która będzie umożliwiała wykonywanie operacji na obrazie w czasie rzeczywistym. Język programowania, który wykorzystałem jest Python wraz z dodatkowymi biblioteki, które znaczące ułatwiają niektóre etapy przetwarzania. Dobór narzędzi nie może być przypadkowy. System będzie wykorzystywany w komercyjnym zakładzie przemysłowym, zatem wszystkie zastosowane narzędzia będą zdefiniowane przy pomocy licencji o swobodnym dostępie zarówno prywatnym jak i komercyjnym. Docelowo cała struktura na zakładzie oparta jest o język C# zatem obiekt, który zostanie zwrócony przez program musi być charakter uniwersalny, czyli taki który możemy wykorzystać również w innym języku niż Python.

Rysunek 1. Zdefiniowanie problemu projektowego

1.3 Oczekiwany rezultat

Docelowo program powinien zwracać dwie wartości. Pierwszą z nich będzie wartość typu boolowskiego, która określać będzie zgodność analizowanego artykułu z wartością zdefiniowaną przez użytkownika. Druga wartość będzie typu liczbowego, która reprezentować będzie liczbę otworów w zdefiniowanym przez użytkownika fragmencie. Dodatkowo w celu lepszej wizualizacji zostaną zwrócone współrzędne wykrywanego obiektu z wykorzystaniem sieci neuronowych.

1.4 Wykorzystane narzędzia

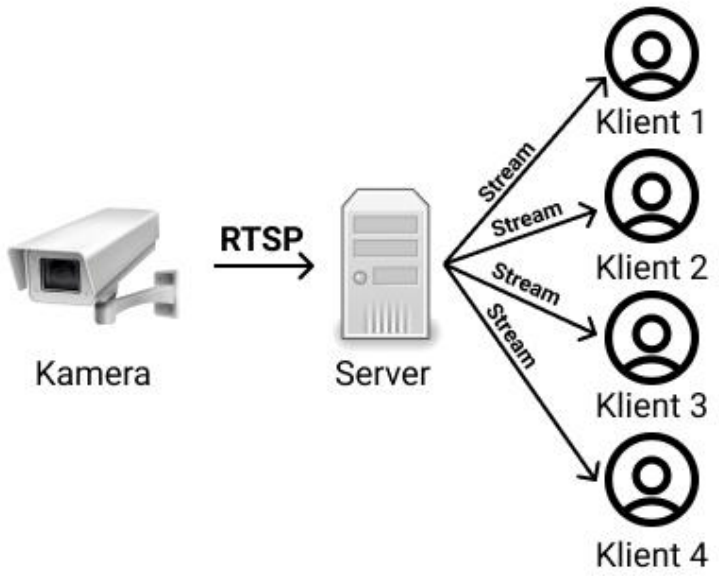
W projekcie wykorzystane zostaną różne biblioteki, jednak wspólnym mianownik w każdym przypadku będzie język programowania Python. Analizując obraz pod kątem jego wstępnej obróbki wykorzystamy bibliotekę OpenCV, która posiada API dzięki czemu możemy swobodnie korzystać z niej bez względu na wykorzystywany język. W przypadku edytora będę korzystał ze środowiska **Jupyter Notebook** jak również z chmurowego odpowiednika oferowanego przez firmę Google **Google Colaboratory**. W tym drugim przypadku, aby zapewnić stałą synchronizacje danych wykorzystamy **Google Drive**. Zdalny dostęp do danych pomiarowych zapewni nam protokół **RTSP**. Natomiast w przypadku sieci neuronowych posłużymy się biblioteką **TensorFlow** oraz nieco jej zmodyfikowaną wersją, która zamiast klasyfikacji umożliwia nam wykrywanie obiektów **TensoFlow Object Deteftion API**.

2 Cyfrowe przetwarzanie obrazów z wykorzystaniem biblioteki OpenCV

2.1 Protokół RTSP

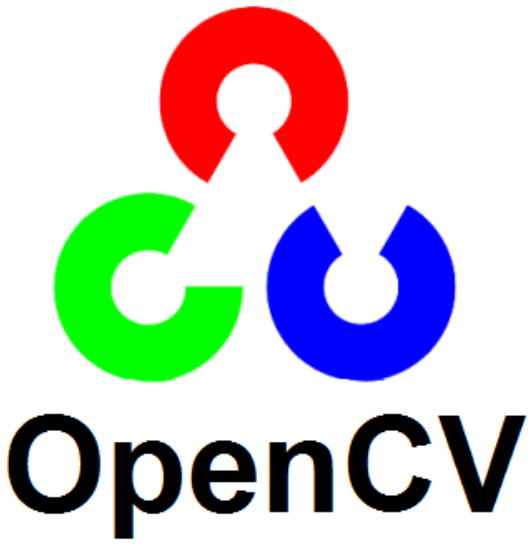
Pierwszym krokiem jest uzyskanie dostępu do danych, które będą analizowane. Można to zrobić na wiele różnych sposób. Jednym z nich jest fizyczne wykonanie zdjęć nad obiektem, które potem zostaną poddane analizie. Rozwiązanie wydaje się stosunkowo proste jednak niesie ze sobą wiele negatywnych aspektów. Rozdzielczość oraz inne parametry aparatu mogą się znacznie różnić od docelowego rozwiązania. Co więcej, analiza pojedynczej klatki nie pozwala nam w żaden sposób sprawdzić wydajności stosowanych algorytmów, które mogą okazać się zbyt wolne podczas pracy nad rzeczywistym obiektem. Zatem poszukiwane rozwiązanie powinno być już wstępnie znormalizowane do warunków, w jakich będzie działać w późniejszym etapie. Dodatkowo, aby jednocześnie kontrolować aspekt wydajnościowy przekazywany obraz powinien na bieżąco dostarczać kolejne klatki przechwyconego obiektu. W tym miejscu również można zastosować dwa podejścia rozwiązania problemu. Jednym z nich jest zastosowanie kamery, która będzie się komunikować z komputerem przy użyciu złącza typu USB. Jest to dość proste rozwiązanie jednak wymaga fizycznego połączenia między kamerą a komputerem. Niestety warunki panujące na zakładzie uniemożliwiły takie podejście ze względu na ograniczoną przestrzeń roboczą. Problem postanowiono zatem rozwiązać przy pomocy serwera chmurowego. Takie podejście znacznie ogranicza zastosowanie wszelkich fizycznych połączeń. Niestety takie podejście wymaga zaakceptowania pewnych kompromisów. Jednym z nich jest ograniczenie rozdzielczości w taki sposób aby uniknąć braku płynności w przekazywanym obrazie. Ta cecha jak się okaże w kolejnych etapach jest bardzo

problematyczna. Kolejnym aspektem jest konieczność zastosowania szybkiego łącza oraz świadomość, że w przypadku tymczasowego braku dostępu do sieci program po prostu nie będzie w stanie funkcjonować. Kolejnym krokiem po zaakceptowaniu wszystkich wad od zalet zastosowania bezprzewodowego jest wybór odpowiedniego narzędzia do przesyłania obrazu. Wybór padł tutaj na rozwiązanie w oparciu o protokół **RTSP** *(Real Time Streaming Protocol)*. Jest to narzędzie, które znacznie zyskało na swojej popularności podczas okresu pandemicznego, gdzie większość została zmuszona do przejścia w tryb pracy zdalnej. Całość systemu oparta jest na serwerze, który udostępnia obraz w czasie rzeczywistym. Dzięki takiemu podejściu wiele użytkowników przy pomocy odpowiedniej platformy może śledzić wydarzenia w tym samym momencie. Takie rozwiązanie jest często wykorzystywane do transmisji różnego rodzaju konferencji, posiedzeń zarządów, wydarzeń kulturowych, czy nawet wydarzeń sportowych. Poniższa grafika przedstawia uproszczony schemat działania. Platformą obsługującą w naszym przypadku będzie biblioteka *OpenCV*.



Rysunek 2. Wykorzystanie protokołu RTSP jako źródło informacji o produkcie

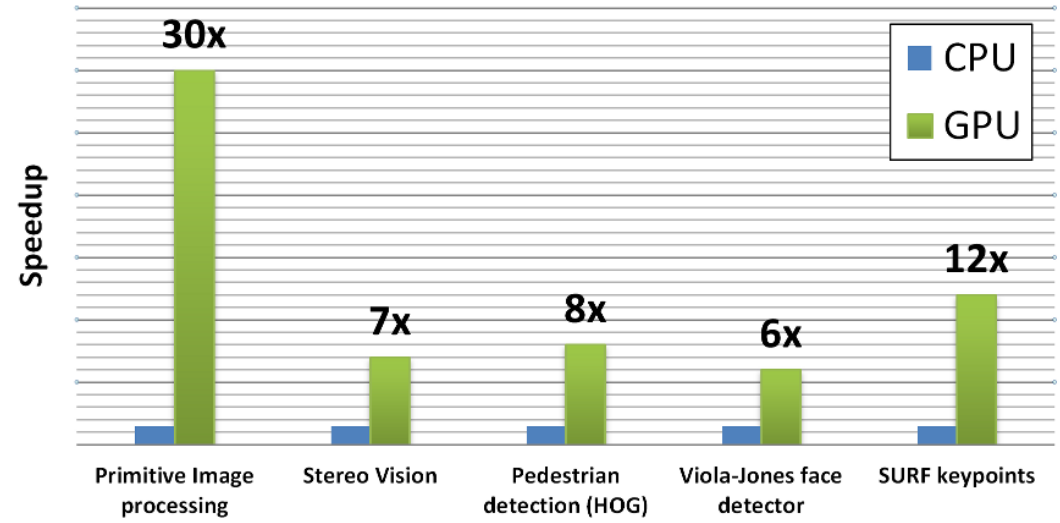
2.2 Biblioteka OpenCV



Rysunek 3. Logo biblioteki OpenCV

OpenCV (Open Source Computer Vision Library) jest biblioteką funkcjonującą na zasadach licencji otwartej. Jej głównym zastosowaniem jest wykonywanie operacji na obrazach w czasie rzeczywistym. W skład biblioteki wchodzą również algorytmy uczenia maszynowego, które znacząco zwiększają zastosowanie oraz wydajność operacji. Kluczowym założeniem przy konfiguracji był aspekt kompatybilności z rozwiązaniami komercyjnymi. Biblioteka jest łatwa w modyfikacji oraz posiada przejrzystą strukturę kodu. W skład pakietu wchodzą zarówno klasyczne rozwiązania stosowane od wielu dekad do analizy obrazów jak i najnowocześniejsze struktury dostarczane przez społeczność, która znacząco przyczynia się do rozwoju produktu. Algorytmy mogą służyć między innymi do wykrywania twarzy na zdjęciach, identyfikacji obiektów, klasyfikacji nastoju w jakim znajduje się użytkownik, śledzenia obiektu w ruchu, redukcji czerwonych oczu. W dobie sytuacji pandemicznej program może wykrywać czy wszyscy uczestniczy spotkania mają założone maseczki. Aktualnie nasilony jest również problem migracyjny. Systemy bezpieczeństwa które nieustannie śledzą granice państw są wyposażone również w algorytmy inteligentnego przetwarzania obrazu. Dane na stronie producenta wskazują, że biblioteka została już pobrana przez ponad 18 milionów użytkowników. OpenCV jest ciągle rozwijana nie tylko przez grupy pasjonatów, lecz również jest przedmiotem analiz grup badawczych a nawet organów rządowych. Wśród użytkowników znajdują się również wielkie korporacje tj. Google, Microsoft, Intel, IBM, Honda, Sony, VideoSurf, Zeitera. Znane wszystkim użytkownikom Google Maps używa właśnie OpenCV w swoich zastosowaniach. Cała składnia języka została zbudowana w oparciu o język *C++*. Wraz

z rozwojem popularności producenci postanowili jednak przyciągnąć nowych użytkowników udostępniając **API**, dzięki czemu z biblioteką możemy się komunikować przy pomocy języka Python, Java oraz dostępne są pewnie funkcjonalne aspekty również w środowisku MATLAB. Interfejs wspiera najważniejsze systemy operacyjne jak: Windows, Mac OS, Linux oraz Android. Dodatkowo implementacja kodu może również być wykonywana z wykorzystaniem mikrokontrolerów z rodziny Arduino oraz Raspberry Pi. Przy wykonywania bardziej zaawansowanych operacjach warto rozważyć akcelerację w oparciu o system **CUDA** oraz **OpenCL**. Jest to jednak możliwe tylko i wyłącznie, jeżeli dysponujemy kartą graficzną o wysokim standardzie. Poniższa grafika ilustruje porównanie czasu wykonywania obliczeń. Na potrzeby symulacji jako GPU została użyta karta *Tesla C2050*, której osiągi porównane zostały z CPU *Core i5-760 2.8Ghz* [1].



Rysunek 4. Porównanie czasu obliczeń dla CPU oraz GPU

2.3 Podstawowa składnia

W przypadku gdy źródłem obrazu jest kamera dostarczająca określoną liczbę klatek na dany okres czasowy strukturę programu możemy podzielić na dwa bloki: wnętrze pętli *while* oraz wartości lub funkcje zdefiniowane poza jej obszarem. Pierwszym etapem rozpoczęcia pracy jest import bibliotek, które będą wykorzystywane. W kolejnym kroku definiujemy źródło obrazu. W tym przypadku *cap = cv2.VideoCapture(0)* odnosi się do sprzętu zdefiniowanego przez komputer jako sygnał domyślny, w dalszej części zostanie on zastąpiony poprzez opisany wcześniej protokół **RTSP**. Kolejno zdefiniowane jest pętla *while*, która jest wykonywana dopóki użytkownik nie naciśnie przycisku powodującego przerwanie operacji (w tym przypadku jest to klawisz *q*). Funkcja *cap.read()* zwraca nam dwie wartości: *ret* oraz *frame*. Pierwsza z nich jest zmienna typu boolowskiego informująca nas czy obraz został przechwycony w sposób prawidłowy. Zmienna *frame* natomiast jest obrazem przedstawionym w postaci wektora.

```
import cv2
#Wybor domyslnego źródła
cap = cv2.VideoCapture(0)

while(cap.isOpened()):
    #Wczytaj obraz
    ret, frame = cap.read()
    ---Operacje na obrazie---
    ---Operacje na obrazie---
    #Zamknij okno gdy użytnkownik wciśnie klawisz "q"
    if cv2.waitKey(20) & 0xFF == ord('q'):
    break
```

2.4 Środowisko pracy

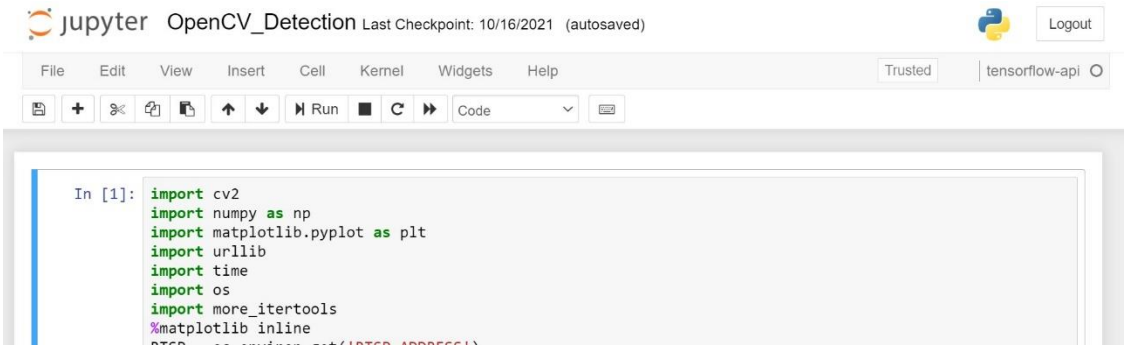
2.4.1 Conda

Rysunek 5. Logo środowiska Conda

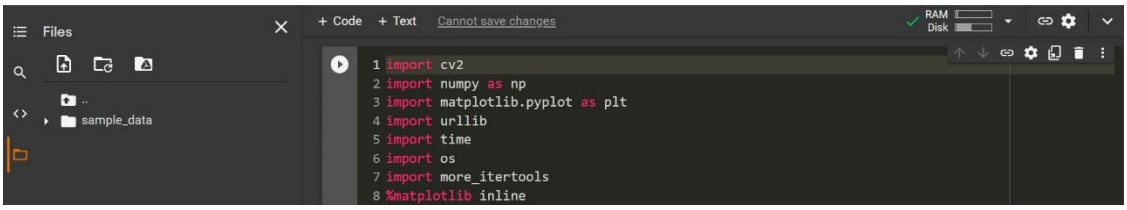
Zarówno środowisko *Python* jak i wszystkie potrzebne pakiety zostały zainstalowane w oparciu o środowisko **Conda**. Jest to system zarządzania pakietami działający na wszystkich popularnych systemach operacyjnych. Najbardziej rozpowszechniony oraz wykorzystywany jest przez użytkowników zajmujących się przetwarzaniem oraz analizą danych. Mogą być to pliki tekstowe jak również multimedialne. Środowisko wspiera również inne popularne języki programowania tj. *R*, *Ruby*, *Java*, *C/C++*, *FORTRAN*, *JavaScript*. Conda z łatwością przechowuje, tworzy oraz obsługuje biblioteki, które są instalowane w oparciu o dany język programowania [3]. Mając na uwadze fakt, że pisany program będzie w przyszłości dystrybuowany na wiele różnych urządzeń potrzebna jest funkcjonalność, która pozwoli zebrać wszystkie potrzebne pliki w jednym miejscu. W tym momencie z pomocą przychodzi *Wirtualne Środowisko*. Pozwoli to wyseparować wersje różnych bibliotek, gdzie zostały już wcześniej zainstalowane na komputerze co w prosty sposób umożliwia uniknięcie w przyszłości niekompatybilności. Kolejnym ważnym aspektem jest separacja. Dokonując modyfikacji pakietów wyłącznie w obrębie wirtualnego środowiska nie oddziałujemy w żaden sposób na wersje zainstalowane globalne, co sprawia, że nie wpływamy na kompatybilność wcześniej wykonanych projektów. Do stworzenia oraz aktywacji wirtualnego środowiska służy poniższa składnia.

```
'''
Tworzenie zmiennej środowiskowej o nazwie engineering-thesis
w oparciu o wersje Pythone 3.10 (jeżeli wersja ta nie zostanie
podana system automatycznie wybierze postać domyślną)
'''

conda create --name engineering-thesis python=3.10

#Aktywowanie zmiennej środowiskowej

conda activate engineering-thesis

'''
Jeżeli komenty zostaną wykonane poprawie w wierszu poleceń
powinny być widoczny następujący tekst
'''

(engineering-thesis) C:\Current\Path

'''
Instalacja biblioteki z wykorzystaniem środowiska Conda
'''
conda install -c conda-forge opencv

'''
Alternatywnie możemy użyć menadżera pip
'''
pip install opencv-python

'''
W celu wylistowania wszystkich zainstalowanych pakietów
stosujemy poniższa składnię
'''
conda list

'''
Powinna sie wyświetlić lista z wszystkimi pakietami,
przykładowo:
'''
# Name                    Version                   Build  Channel
anyio                     3.1.0                     pypi_0
argon2-cffi               20.1.0                    pypi_0
py-opencv                 3.4.2            py36hc319ecb_0
```

2.4.2 Jupyter Notebook/Jupyter Lab/Google Colaboratory



Rysunek 6. Okno programu Jupyter Notebook



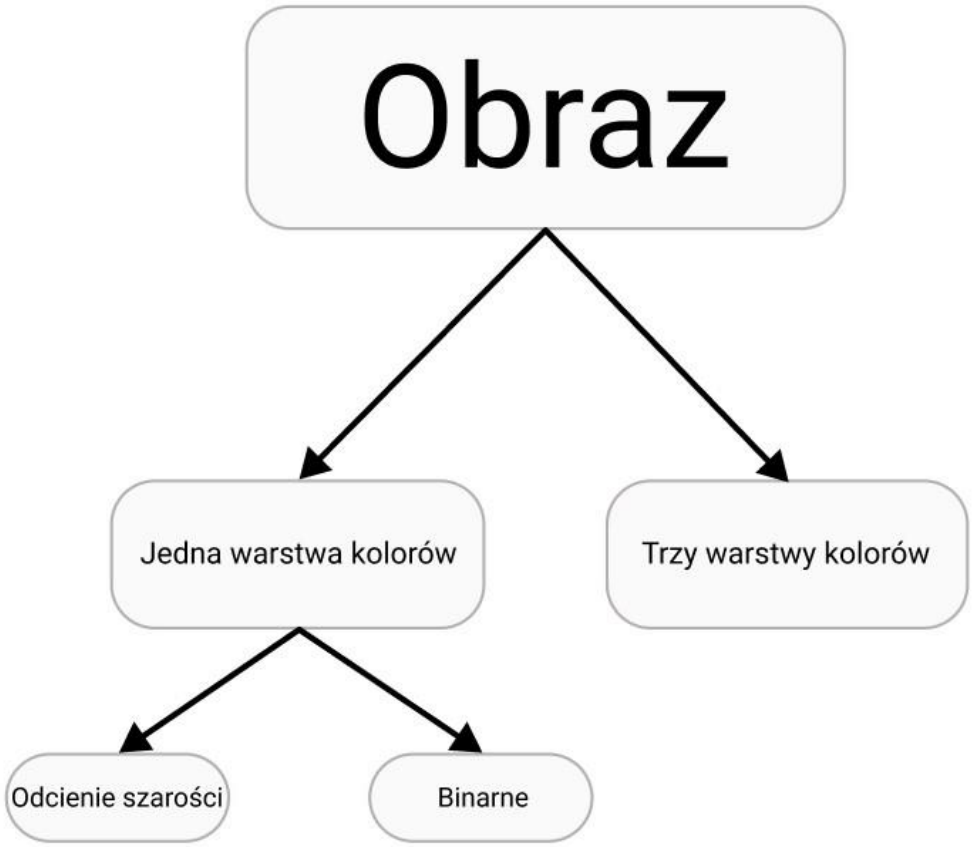
Rysunek 7. Okno programu Google Colaboratory

Jupyter Notebook/Jupyter Lab jest częścią projektu *Jupyter*, który wyewoluował z interaktywnej wersji Pythona *IPython* w roku 2014. Jest to program o charakterze *non-profit* bazujący na bezpłatnej licencji. Środowisko znajduje szerokie zastosowanie w przetwarzaniu danych, obliczeniach numerycznych z wykorzystaniem różnych języków programowania. W odróżnieniu do klasycznego pisania kodu struktura podzielona jest na osobne sekcje zwane potocznie komórkami. Wykonywany program możemy uruchamiać poszczególnymi fragmentami. Po instalacji dodatkowych pakietów, można mieć podgląd do aktualnych wartości zmiennych, które są zdefiniowane w programie na zasadach podobnych do tego co oferuje nam pakiet MATLAB. Nowe elementy mogą przybierać również formę multimedialną w postaci zdjęć bądź równań matematycznych zapisywanych zgodnie ze składnią Latex [5].

Google Colaboratory jest chmurowym odpowiednikiem *Jupytera* dostarczonym przez firmę *Google*. Największą zaletą takiego rozwiązania jest brak konieczności instalowania dodatkowych bibliotek. Wszystkie potrzebne pakiety dostarczone są wraz ze środowiskiem. Program domyślnie używa najnowszych pakietów, dlatego jeżeli jest potrzeba zastosowania konkretnej wersji biblioteki trzeba to wcześniej zdefiniować. Należy mieć również na uwadze, że *Colab* zbudowany jest w oparciu o system Linux, dlatego aby sprawnie poruszać się bo katalogach oraz zarządzać plikami należy posiadać podstawową wiedzę z zakresu pracy z *Terminalem*. Dodatkowo można cały projekt zsynchronizować z zewnętrznymi źródłami co

daje możliwość obsługi plików z lokalnego poziomu komputera. W zależności czy zdecydujemy się na wykupienie wersji *Pro* czy zostaniemy na wariancie darmowym *Colab* oferuje akcelerację GPU, co jest świetnym rozwiązaniem jeżeli mamy ograniczone możliwości sprzętowe. Niestety OpenCV w znacznym stopniu wykorzystuje bibliotekę *Qt* do tworzenia okienek w których znajdują się rezultaty programu, zatem z poziomu przeglądarki nie będziemy mieli do nich dostępu. Istnieją pewnie rozwiązania, które częściowo niwelują ten problem jest nie są na tyle wydajne, żeby w pełni zastąpić wersję stacjonarną. Biorąc pod uwagę wady i zalety takie rozwiązania najlepszym wyjściem okazuje się tutaj praca w trybie hybrydowym tj. cześć obliczeniową przeprowadzić w środowisku chmurowym, natomiast rezultaty przedstawić w oparciu o *Jupytera* w formie stacjonarnej [4].

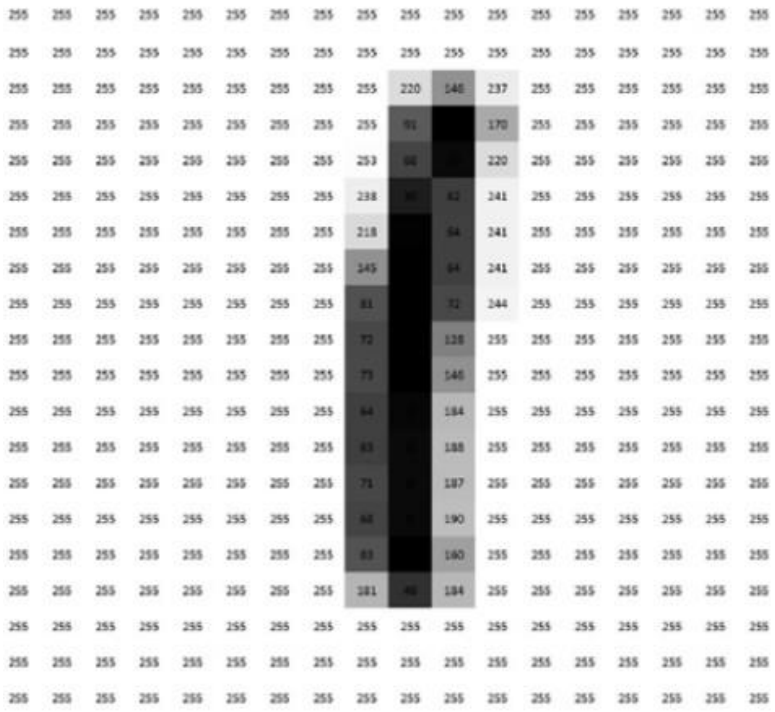
2.5 Podstawowa klasyfikacja obrazów



Rysunek 8. Klasyfikacja obrazów

Ze względu na matematyczną interpretację obrazu zostanie wprowadzona podstawowa klasyfikacja, która będzie odnośnikiem również do dalszej części pracy. W celu odwzorowania rzeczywistości wykorzystuje się wielowymiarowe struktury danych. Zależnie od źródła mogą się one nazywać *macierzami*, *tablicami*, *tensorami*. Najprostszym przypadkiem jest konstrukcja macierzy, która posiada pewną określoną liczbę wierszy oraz kolumn a jej strukturę możemy naszkicować na kartce papieru. Zdjęcie może przykazywać informacje tylko i wyłącznie, kiedy jego wnętrze wypełnione jest danymi. Wszystkie analizowane obiekty zostaną przedstawione w oparciu o rozszerzenie graficzne formatu **jpg**, które umożliwia wykorzystanie *8-bitowej* głębi kolorów. Oznacza to, że wnętrze macierzy może być wypełnione liczbami naturalnymi z przedziału od 0 do 255. Często w kolejnych etapach pracy wykonuje się operacje normalizowania tj. przeskalowania wszystkich liczb znajdujących się we wcześniej wspominanym zakresie na przedział od 0 do 1. Takie podejście umożliwia przedstawienie zdjęcia tylko i wyłącznie w odcieniu jednego koloru. W sytuacji gdy jest potrzeba zastosowania formatu wielokolorowego musimy rozszerzyć istniejącą warstwę o dwie dodatkowe i wykonać

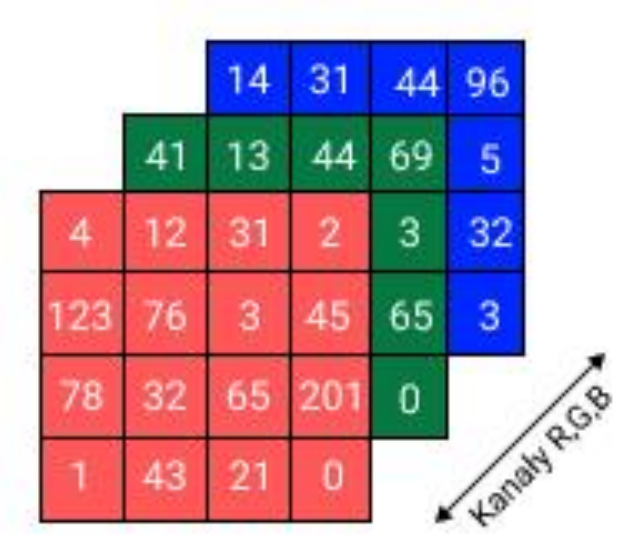
na każdej z nich kolejno mapowanie na kolor czerwony (R), zielony (G) oraz niebieski (B). Jeżeli pojedyncza wartość piksele może znajdować się w zakresie od 0 do 255 oraz uwzględniona zostanie liczba kanałów to można otrzymać paletę barw zdolną do odwzorowania niemal *17 mln* kolorów. Łatwo jednak zauważyć, że zdjęcie wielokolorowe zwiększa ilość dostarczanych informacji trzykrotnie co może nie być pożądane pod kątem wydajnościowym. Tutaj decyzja o formacie leży tylko i wyłącznie po stronie użytkownika, który musi określić czy identyfikacja kolorów pomoże mu w rozwiązaniu problemu czy jest wyłącznie dodatkowymi, zbędnymi informacjami. Analizując zdjęcie monochromatyczne, można przedstawić je w odcieniu jednego koloru (najczęściej głębia koloru szarego) lub zbinearyzować tj. uprościć obraz tylko i wyłącznie do jego warunków brzegowych (brak informacji lub jej obecność). Ta własność okaże się bardzo przydatna w kolejnych etapach przetwarzania.



Rysunek 9. Obraz w różnych odcieniach szarości

255	255	255	255	255	255	255	255
255	255	255	255	0	0	255	255
255	0	0	0	255	0	0	255
255	0	255	255	255	255	0	255
255	0	255	255	255	255	0	255
255	0	0	0	0	0	0	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

Rysunek 10. Obraz w postaci binarnej

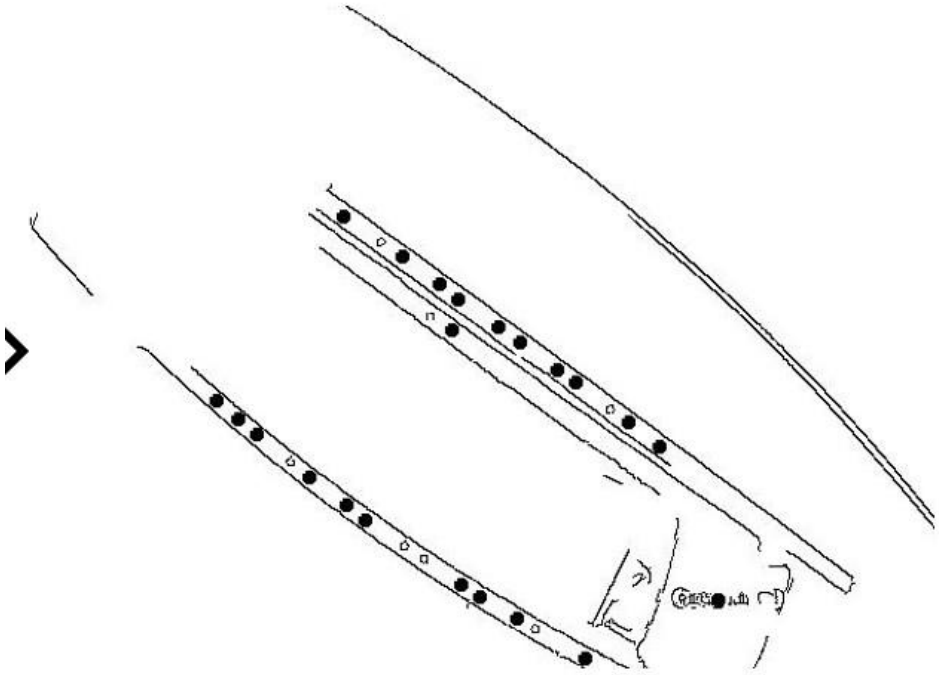


Rysunek 11. Sposób przedstawienia zdjęcia w formacie kolorowym

2.6 Przetwarzanie obrazu

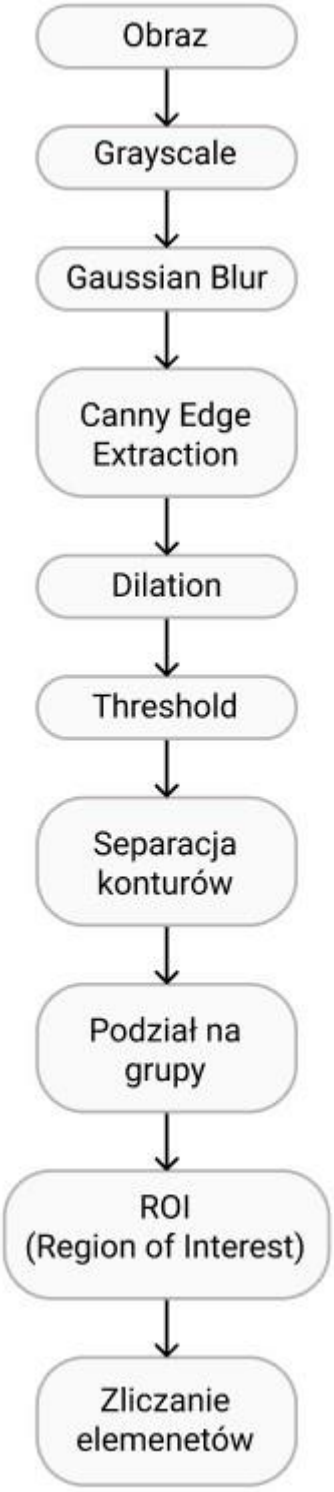
2.6.1 Oczekiwany rezultat

Rysunek 12. Obiekt do przetworzenia



Rysunek 13. Efekt działania programu

2.6.2 Schemat blokowy toru przetwarzania



Rysunek 12. Schemat blokowy przetwarzania obrazu

2.6.3 Grayscale

Zdjęcie dostarczone prze kamerę w trybie domyślnym zawiera trzy kanały danych (R,G,B). W celu usprawnienia wykonywanych operacji oraz możemy wykonać transformacji na obraz w odcieniach szarości. Matematyczną implementację opisuje równanie (1). Korzystając z pakietu OpenCV operację tą możemy wykonać w oparciu o dwa dostarczone argumenty. Pierwszym z nich jest obraz, natomiast w drugim członie funkcji podanie zostaną sposób konwersji, w tym przypadku z kanału kolorowego na odcienie szarości [2].

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

$$y = 0.299R + 0.587G + 0.114B \tag{1}$$

2.6.4 Blurring

Kolejnym krokiem jest usunięcie ze zdjęcia w jak największym stopniu zakłóceń, które mogą w sposób inwazyjny zaburzać dalszą pracę. W tym celu wykonywana jest operacja wygładzania zdjęcia. Innym określeniem stosowanym do tego typu zabiegów jest również rozmazywanie. W ujęciu matematycznym możemy zdefiniować to jako proces filtracji. Wykonanie takiej operacji zostało przedstawione w postaci poniższego równania [6].

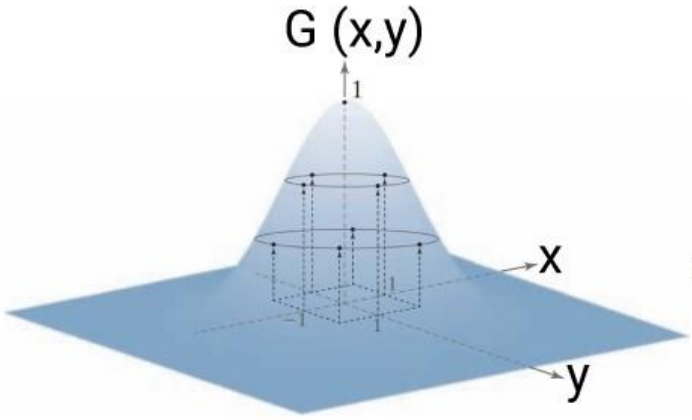
$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l) \tag{2}$$

Wyjściowy wektor *g(i,j)* otrzymywany jest poprzez sumę wartości pikseli w uwzględnieniem wag otrzymywanych przy zastosowaniu jądra. W równaniu został ono przedstawione w postaci *h(k,l)*. Filtr ten można zwizualizować przy pomocy okna zawierającego wartości współczynników. Możemy wyróżnić kilka rodzajów filtracji. Najprostszym podejściem jest utworzenie tablicy z wartości średnich w obrębie danego jądra [15]. Poniższa tożsamość ilustruje podstawową implementację.

$$K = \frac{1}{K_{width} \cdot K_{height}} \cdot \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} \tag{3}$$

W przetwarzaniu obrazów najczęściej wykorzystujemy jednak filtrację w oparciu o okno Gaussa. Warto jednak w tym momencie zwrócić uwagę, że rozwiązanie nie jest najkorzystniejsze pod względem wydajności, zatem jeżeli w kolejnych krokach zaobserwujemy spadek wydajności zmiana rodzaju okna może okazać się kluczową decyzją. Z racji, że obiekt na którym będziemy wykonywać operacje będzie posiadał dwa wymiary, poniższe równanie prezentuje opisywaną funkcje, natomiast poniżej widoczna jest jej graficzna interpretacja.

$$G_0(x,y) = Ae^{\frac{-(x-\mu_x)^2}{2\sigma_x^2} + \frac{-(y-\mu_y)^2}{2\sigma_y^2}} \tag{4}$$



$$\frac{1}{4.8976} \times \begin{bmatrix} 0.3679 & 0.6065 & 0.3679 \\ 0.6065 & 1.0000 & 0.6065 \\ 0.3679 & 0.6065 & 0.3679 \end{bmatrix}$$

Rysunek 14. Kształt funkcji Gaussa dla dwóch parametrów wraz z współczynnikami dla wymiarów okna 3x3 [15].

Implementacja przy użyciu dostarczeniu do funkcji trzech argumentów. Pierwszym z nich jest obraz powstały w wyniku poprzedniej operacji. Kolejno definiujemy rozmiar jądra w postaci listy statycznej. Opcjonalnym krokiem jest zdefiniowanie stanu, tak aby operacja była wykonywana na wartościach znormalizowanych. Domyślnie funkcja przybiera wartość *false*.

```
img_blur = cv2.GaussianBlur(gray,(size,size),1)
```

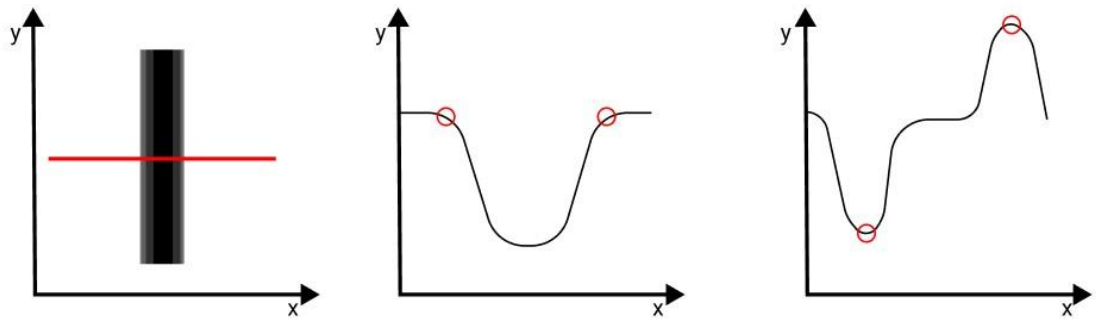
2.6.5 Canny Edge Detection

Canny Edge Detection to jest z najbardziej rozpowszechnionych algorytmów służących do wyseparowania krawędzi z obrazu. Rozwiązanie zaproponował australijski informatyk John F. Canny w roku 1986 [7]. Jest to algorytm wieloetapowy, który wymaga kilku niezależnie związanych ze sobą operacji. Pierwszą z nich jest redukcja szumów, która została wykonana w poprzednim punkcie poprzez operację rozmazywania. Kolejnym krokiem jest przeprowadzenie obliczeń gradientowych wydłuż osi x oraz y. W tym celu wykorzystywany jest algorytm Sobela. Matematycznie wykonaną operacje przedstawia poniższy wzór.

$$EdgeGradient(G) = \sqrt{G_x^2 + G_y^2} \tag{5}$$

$$Angle(\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{6}$$

Powyższą operację można przedstawić w sposób graficzny co prezentuje poniższa ilustracja. Składa się ona z trzech osobnych elementów. Na pierwszej grafice widnieje potencjalny element do wykrycia. Przeprowadzamy przez niego linię odniesienia, która będzie referencją w kolejnych krokach. Następnie liczony jest poziom nasycenia kolorów (grafika 2). Końcowym etapem są obliczenia gradientowe. W zależności od wyboru poziomu wyzwalania otrzymujemy maksymalną oraz minimalną wartość pochodnej bo wskazuje, że wzdłuż badanego obszaru znajduje się obiekt, który chcieliśmy wykryć.



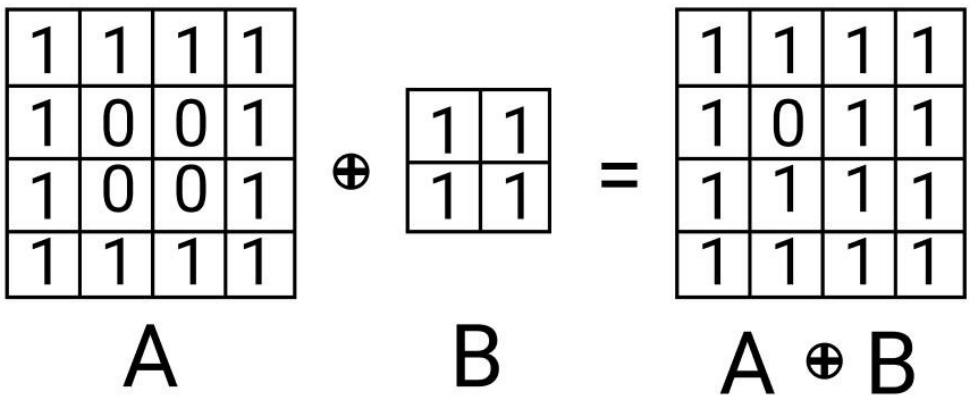
Rysunek 15. Obliczenia gradientowe

2.6.6 Dilation

Dilation jest operacją, która ma na celu zwiększenie objętości wyseparowanych krawędzi. W tym przypadku jest to niezbędny proces gdyż zdjęcie jest o stosunkowo niewielkiej rozdzielczości co przekłada się na niewielką powierzchnię krawędzi. Dodatkowo taki proces jest potrzebny, ponieważ kontury uzyskane w wyniku poprzedniej operacji często nie tworzą obszarów o strukturze zamkniętej, co w dalszym procesie uniemożliwia zastosowanie filtrowania w oparciu o wymieniony parametr [9]. Dodatkowo potrzeba określić dwa parametry zastosowanej funkcji, pierwszym z nich jest wejściowy wektor macierzy w postaci binarnej natomiast drugim rozmiar jądra. W tym momencie należy mieć na uwadze, że czym większy jego wymiar tym krawędzie bardziej zyskają na objętości. Należy jednak dobierać uważnie jago wartość, ponieważ można doprowadzić do sytuacji gdzie kontury zaczną tworzyć jeden większy co nie jest zjawiskiem pożądanym. Poniższe równanie opisuje zastosowaną zależność.

$$dst(x,y) = \max_{(x',y')\text{: element } (x',y')\neq 0} src(x+x', y+y') \tag{7}$$

Z praktycznego punktu widzenia oznacza to, że jeżeli przynajmniej jeden element oznaczający wartość *1* z macierzy wejściowej pokryje się z elementem *1* zastosowanego jądra na wyjściu otrzymujemy *1*. Następnie operację powtarzamy przesuwając wektor jądra w taki sposób aby przejść przez całą macierz wyjściową. Modyfikować możemy również liczbę iteracji.



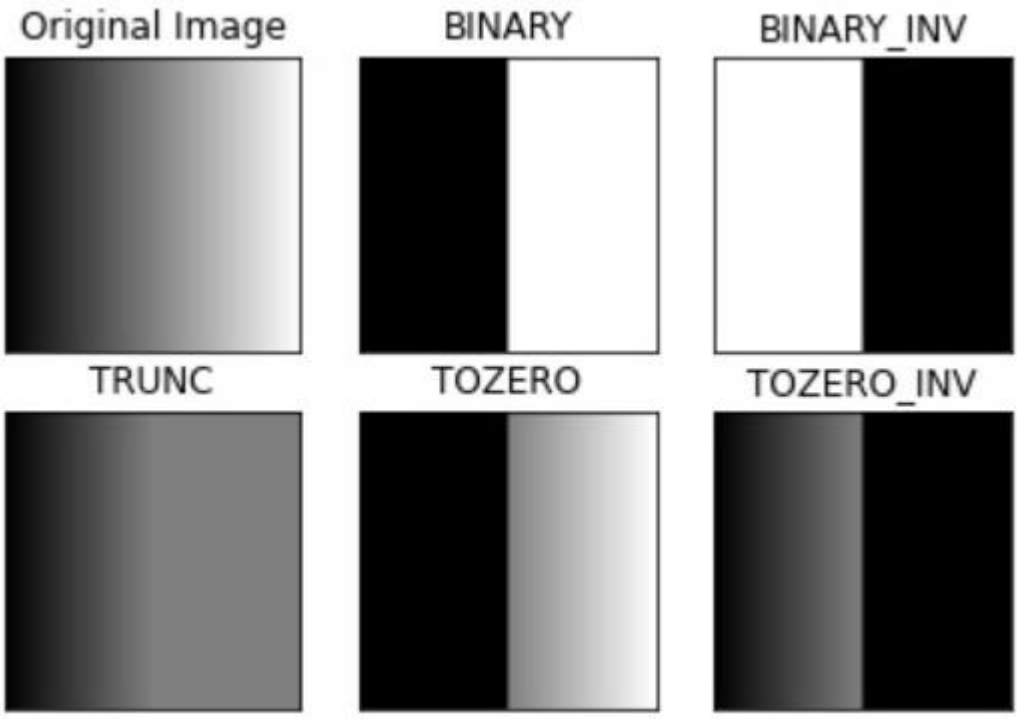
Rysunek 16. Przykładowy przebieg procesu

```
#Dilation process
dsize = cv2.getTrackbarPos("Dilation_kernel","Parameters")
imgDil = cv2.dilate(edges,np.ones((dsize,dsize)),iterations=1)
```

2.6.7 Threshold

Kolejnym etapem jest **Thresholding**, który polega na ustaleniu wartości granicznych dla których zdjęcie powinno zostać wyprogowane. Proces ten można nazwać również binaryzacją. Przetwarzanie polega na aplikacji dla każdego piksela z obrazu funkcji, która w zależności od wartości koloru piksela ustala jego wartość na 0 (kolor czarny) lub 255 (kolor biały). Należy jednak pamiętać że musimy dysponować obrazem przedstawionym w formacie różnych odcieni szarości, aby proces został przeprowadzony poprawnie. Poniżej interpretacja matematyczna oraz przykład użycia funkcji na której wejście należy podać cztery argumenty. Pierwszym z nich jest macierz obrazu. Kolejne dwa ustalają parametry progowania. Jeżeli wartość numeryczna piksela będzie większa niż drugi argument, na wyjściu otrzymamy wartość 255 w przeciwnym wypadku będzie to 0. W zależności od zapotrzebowania możemy zastosować różny algorytm, poniższa grafika prezentuje 5 podstawowych, jednak najważniejsze z nich w kontekście projektu to *BINARY* oraz *BINARY_INV* [7].

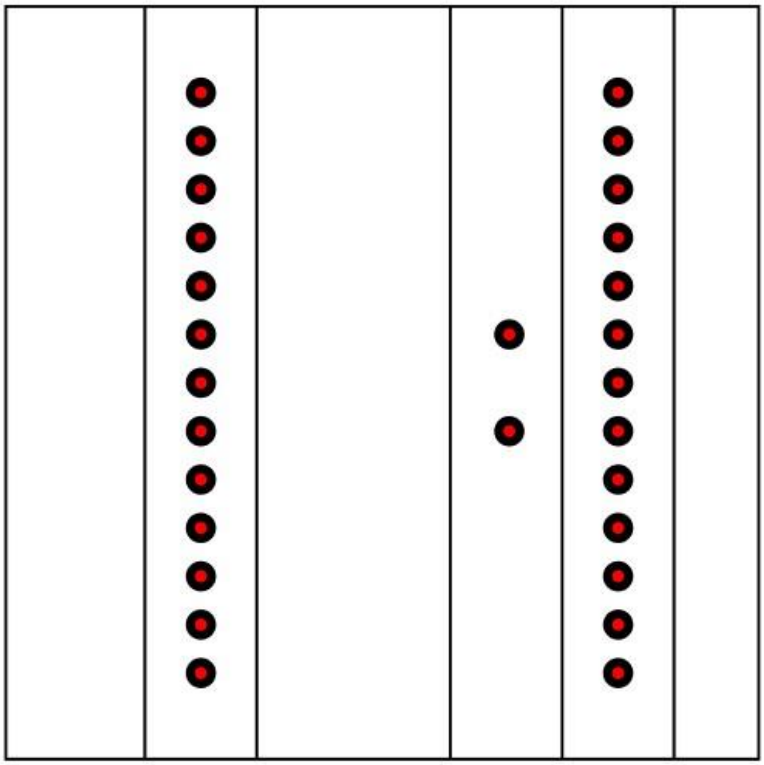
```
#Color inverse
ret,thresh1 = cv2.threshold(imgDil,100,255,cv2.THRESH_BINARY_INV)
```



Rysunek 17. Podstawowe algorytmy progowania [7].

2.6.8 Wyseparowanie konturów

Kontury można przedstawić w prosty sposób jako krzywą łączącą punkty wzdłuż granicy o takim samym kolorze lub intensywności. Przy ich pomocy można w dość prosty sposób analizować wykrywanie obiektów lub ich rozpoznawanie. W celu zapewnienia najlepszych efektów na wejście algorytmu podaje się najczęściej zdjęcie w postaci binarnej, co zostało wykonane w poprzednich punktach. Do implementacji służy funkcja *cv2.findContours()* do której zależy dostarczyć trzy argumenty. Pierwszym z nich jest zbinearyzowany obraz, następnie określa się algorytm separacji krawędzi a jako trzecią wartość podaje się metodę aproksymacji. Zastosowany algorytm *RETR_CCOMP* charakteryzuje się zwracaniem krawędzi w sposób hierarchiczny tworząc dwa poziomy dla konturów zewnętrznych oraz wewnętrznych. Pierwszą warstwę tworzy obramowanie całego wykrytego elementu. Wszystkie kontury znajdujące się wewnątrz niego stanowią jedną grupę i zostają sklasyfikowane jako drugi stopień drabiny zależności. *CHAIN_APPROX_SIMPLE* służy natomiast do zwrócenia punktów granicznych krawędzi, dla których następuje zmiana w kierunku poziomym, pionowym oraz po przekątnej. Przykładowo dla wykrytego prostokąta byłyby to wyłącznie cztery punkty. Tak wykryte krawędzie są następnie filtrowane za pomocą powierzchni, która jest zdefiniowane przez użytkownika przy pomocy trackbara. Jeżeli zostaną spełnione warunki tj. wykrywany kontur znajduje się we wnętrzu zewnętrznego oraz jego powierzchnia jest mniejsza od zdefiniowanej wartości granicznej następuje graficzna wizualizacja procesu w oparciu o metodę *cv2.drawContours()* [8]. Następnie wykonywany jest dodatkowy proces, który umożliwi identyfikacje grup obiektów. Do tego celu następuje zaproksymowanie konturu przy pomocy krzywej a następnie wpisanie jest w równanie prostokąta. Mając te dane można wyznaczyć środek ciężkości figury w postaci pojedynczego punktu zawierającego informacje o jego położeniu. Końcowym etapem jest zwrócenie tych wartości w postaci posortowanej listy. Poniższa grafika prezentuje w uproszczony sposób działanie algorytmu. Środki ciężkości zostały przedstawione w postaci czerwonych punktów.

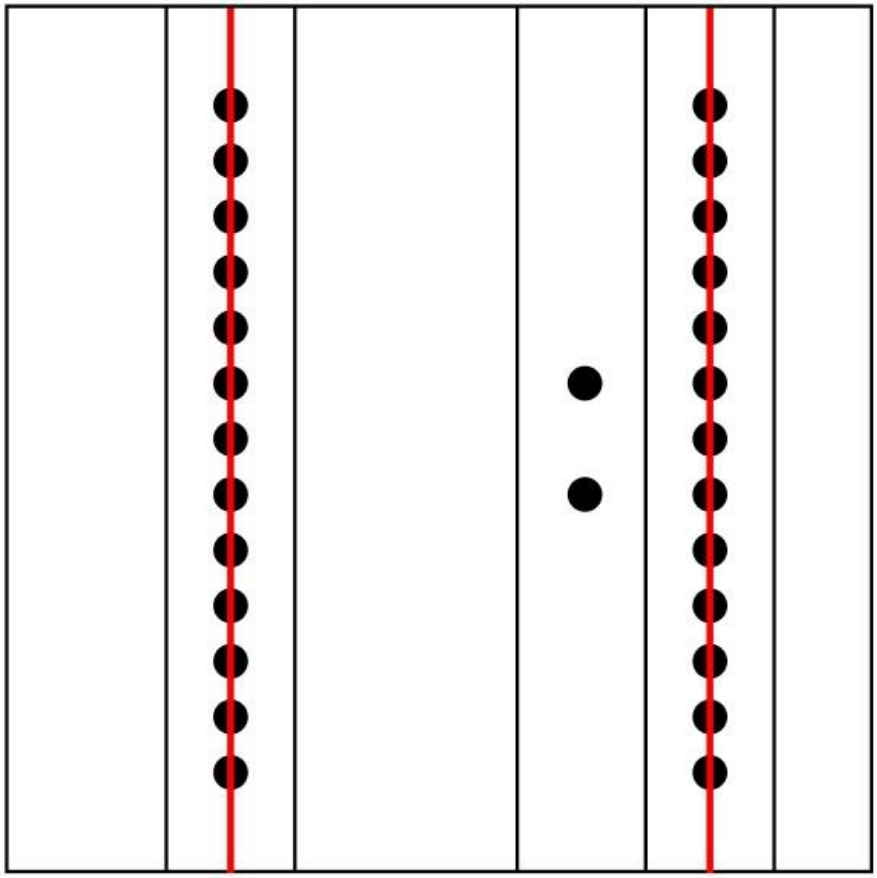


Rysunek 18. Wyseparowanie konturów oraz wyznaczenie środków ciężkości

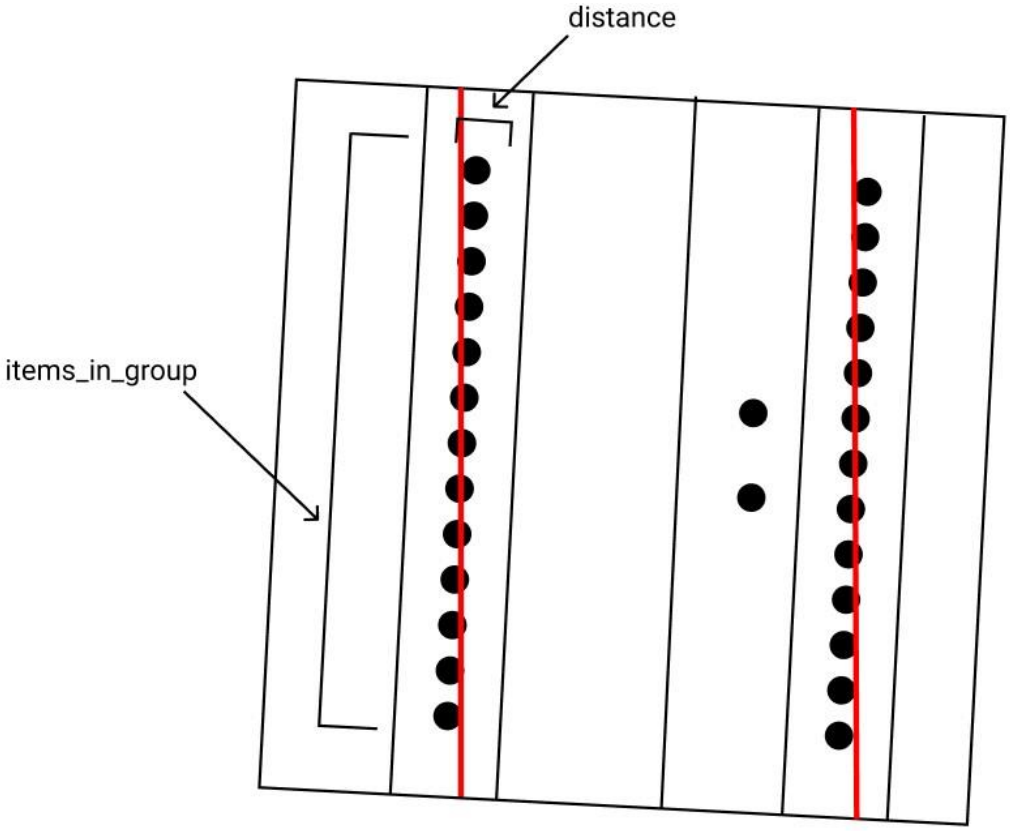
```
def getContours(img,imgContour,ymin=200,ymax=375):
    contours,hierarchy = cv2.findContours(
      img,cv2.RETR_CCOMP,
      cv2.CHAIN_APPROX_SIMPLE)[-2:]
    rectangle_center = []
    for i in range(len(contours)):
        #Filter using area function
        area = cv2.contourArea(contours[i])
        areaMax = cv2.getTrackbarPos("Area","Parameters")
        # If third column value is NOT equal to -1 than its internal
        if hierarchy[0][i][3] != -1 and area < areaMax:
            # Draw the Contour
            cv2.drawContours(imgContour, contours, i, (255,0,255), 2)
            peri = cv2.arcLength(contours[i],True)
            approx = cv2.approxPolyDP(contours[i],0.02*peri,True)
            x,y,w,h = cv2.boundingRect(approx)
            cv2.rectangle(imgContour,(x,y),(x+w,y+h),(0,255,0),1)
            rectangle_center.append(x+w/2)
    return sorted(rectangle_center)
```

2.6.9 Podział na grupy oraz wyznaczenie ROI

Podstawą dalszej analizy jest zdefiniowanie obszaru, w którym będą się znajdować poszukiwane elementy. Można w łatwy sposób zauważyć że znajduje się on wewnątrz dwóch grup otworów zlokalizowanych na krawędziach całego obiektu. Trzeba jednak wziąć pod uwagę, że produkt nigdy nie będzie znajdował się równolegle do obiektywu kamery co skutkuje koniecznością zastosowania dodatkowych parametrów w celach identyfikacji. Znając współrzędne środków otworów, które zostały zwrócone przez poprzednią funkcję można zaproksymować grupę przy pomocy prostej (kolor czerwony), która będzie dynamicznie zmieniała swoje parametry wraz z poruszającym się obiektem na linii produkcyjnej. Poniższe grafiki przedstawiają opisany proces zarówno w postaci uproszczonej jak i rzeczywistej.



Rysunek 19. Wyznaczenie grup w postaci uproszczonej



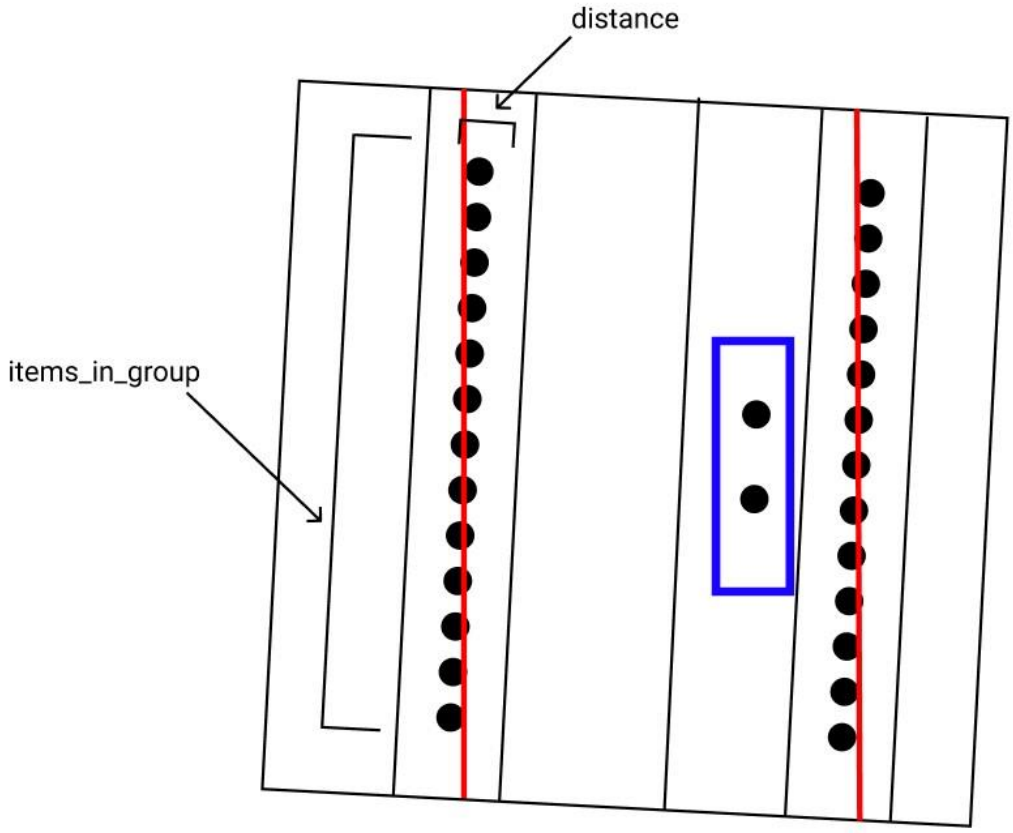
Rysunek 20. Wyznaczanie grup w postaci rzeczywistej

Zgodnie z powyższymi grafikami grupa może być utworzona tylko i wyłącznie wtedy gdy zostaną spełnione dwa podstawowe warunki. Pierwszym z nich jest odległość punktów reprezentujących środki otworów w obrębie jednej grupy. Drugi parametr reguluje minimalną liczbę wykrytych elementów do utworzenia szyku. Jest to bardzo istotne, ponieważ zapobiega traktowaniu potencjalnych zakłóceń jako dodatkowej grupy. Dokładne działanie tego punktu realizuje poniższa funkcja.

```
def group_create(sorted_list, max_distance, items_in_group):
    output = []
    if len(sorted_list) != 0:
        previous = sorted_list[0]
    else:
        previous = 0
    cut = 0
    for i,x in enumerate(sorted_list[1:], start=1):
        if abs(x - previous) >= max_distance:
            if i - cut >= items_in_group:
                output.append((cut, i-1))
            cut = i
        previous = x
    if len(sorted_list) - cut >= items_in_group:
        output.append((cut, len(sorted_list)-1))
    return output
```

2.6.10 Zliczanie elementów

Ostatnim etapem jest zwrócenie liczby otworów w postaci liczbowej. Dysponując grupami z poprzedniej operacji, które przedstawione są w postaci stałej wartości rzędnej na obrazie zdefiniowany został region, w którym znajdować się będą wykrywane elementy. W tym celu została obliczona odległość miedzy dwoma prostymi a następnie przy użyciu wartości procentowych obszar został doprecyzowany do warunków rzeczywistych. Zastosowana metoda posiada pewnie ograniczenia, ponieważ może analizować tylko i wyłącznie jeden przedmiot w pojedynczej klatce obrazu. Dla tak wyznaczonego obszaru (kolor niebieski na poniższej grafice) ponowiony został proces wyseparowania krawędzi znany z poprzednich punktów przetwarzania. Poniżej znaduje sie również fragment kodu realizujący opisane operacje.



Rysunek 21. Zdefiniowanie ROI oraz liczenie elementów

```
def approximate(img,imgContour,rectangle_center,ymin=250,ymax=550):
    groups = group_create(rectangle_center,5,5)
    font = cv2.FONT_HERSHEY_SIMPLEX
    comp = [np.round(np.mean(rectangle_center[groups[i][0]:groups[i][1]]),
2)
            for i,j in enumerate(groups)]
    for i in comp:
        cv2.line(imgContour, (int(i), 0),
                (int(i), 640), (0, 0, 255),
                thickness=1)
    try:
        if len(comp) == 2 and (comp[1]-comp[0]) < 150:
            diff = (comp[1]-comp[0])
            x1 = int(comp[0]+0.85*diff)
            x2 = int(comp[0]+0.99*diff)
            cv2.rectangle(imgContour,(x1,ymin),(x2,ymax),(255,0,0),2)
            contours,hierarchy = cv2.findContours(img[ymin:ymax,x1:x2],
                                cv2.RETR_CCOMP,
                                cv2.CHAIN_APPROX_SIMPLE)[-2:]
            cont =
            [i for i in range(len(contours))
            if hierarchy[0][i][3] != -1 and
            cv2.contourArea(contours[i]) < cv2.getTrackbarPos("Area",
                                            "Parameters")]
            cont_len = str(len(cont))
            cv2.putText(imgContour,
                        text=cont_len,org=(10,100),
                        fontFace=font,
                        fontScale= 4,
                        color=(255,255,255),
                        thickness=2,
                        lineType=cv2.LINE_AA)
            cv2.imshow('Detected holes',imgContour[ymin:ymax,
                                        int(x1):int(x2)])
    except:
        print('Invalid comp dimension!')
```

3 Wykorzystanie sieci neuronowych

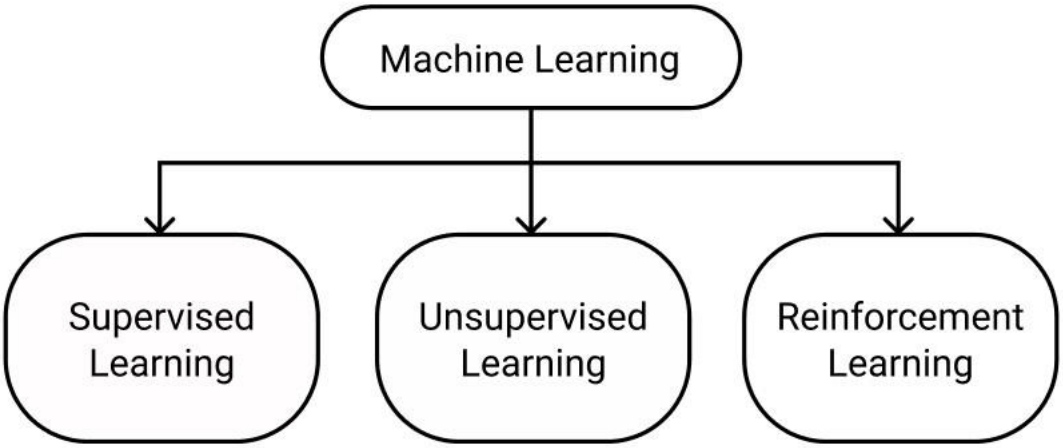
3.1 Cel wprowadzenie sieci neuronowej

Jednym z parametrów opisujących przetwarzanie obrazów jest ilość klatek, które zostają przetwarzane w czasie jednej sekundy. Wartość tą najczęściej ustala się na poziomie serwera obsługującego protokół RSTP, jednak istnieje możliwość wykonania tej operacji również w oparciu o bibliotekę OpenCV. Przy wyborze opisywanej wartości należy kierować się w oparciu o dotychczasową wiedzę na temat dynamiki przetwarzanego obiektu. Jeżeli dokładność jest priorytetem i jednocześnie algorytm będzie pracować na obrazie w czasie rzeczywistym liczba klatek na sekundę powinna minimalnie wynosić około 30. Możemy jednak modyfikować ten parametr w zależności od aktualnej wydajności zaimplementowanych algorytmów. Należy jednak mieć również na uwadze, że mogą występować chwilowe spadki wydajności sieci, co może skutkować przerwami w dostarczaniu obrazów do programu. W konsekwencji program może zwrócić błąd, co należy uwzględnić w postaci zwrócenia odpowiedniego wyjątku obsługującego taką sytuację. Mając na uwadze opisany parametr można w łatwy sposób dojść do wniosku, że program odświeża swoje parametry wyjściowe z częstotliwością, która jest w przybliżeniu równa (opóźniona o czas przetwarzania) częstotliwości odświeżania obrazu. Warto w tym miejscu przypomnieć, że kod programu w swoim działaniu opierał się często na binarnym przetwarzaniu danych, zatem nawet niewielkie przekroczenie wartości progowania skutkuje zwróceniem zupełnie innej wartości wyjściowej. Innymi słowy program ma niewielki margines błędu, ponieważ zwraca jedynie informacje w postaci liczbowej. Można to porównać do układów cyfrowych które cechują się wyłącznie dwoma stanami pracy (brak informacji lub jest występowanie). W celu wprowadzenia wartości pośrednich zostanie wykorzystana sieć neuronowa, która bazować będzie na obrazie nieprzetworzonym (bezpośrednio z kamery). Przy odpowiednim wytrenowaniu sieci możliwa będzie dokładniejsza analiza pozwalająca w sposób procentowy zwracać informacje o występowaniu obiektu, który został zaimplementowany w procesie trenowania. Następuje jednak konieczność wstępnego wyseparowania obszaru z użyciem biblioteki OpenCV, ponieważ wykrywany obiekt będzie poruszał się dynamicznie, więc nie ma możliwości z góry przypisać miejsca w którym będzie się znajdował.

3.2 Informacje wstępne

3.2.1 Podział algorytmów

Wraz z rozwojem technologicznym oraz przechodzeniem z wersji papierowej na cyfrową pojawiła się ogromna liczba danych, które trudno analizować w oparciu o metody tradycyjne. Zastosowanie uczenia maszynowego okazało się dobrym rozwiązaniem w tym przypadku. Algorytmy możemy podzielić na trzy główne grupy: **Supervised Learning**, **Unsupervided Learning** oraz **Reinforecement Learning**.



Rysunek 22. Podział algorytmów uczenia maszynowego

Supervised Learning wymaga od użytkownika znajomości parametrów wyjściowych, jakie powinien otrzymać dostarczając dane na wejście sieci. Głównym zadaniem takiego rozwiązanie jest zdefiniowanie odpowiedniej funkcji, która w sposób najbardziej optymalny będzie aproksymować relacje pomiędzy wyjściem a wejściem. Podstawowe algorytmy stosowane w takiej konfiguracji bazują na klasyfikacji lub regresji.

Unsupervised Learning nie ma zdefiniowanych etykiet wartości wyjściowych. Takie podejście można zastosować w przypadku gdy chcemy znaleźć relacje pomiędzy danymi wyjściowymi oraz w jaki sposób są one między sobą powiązane. Najczęściej algorytmy tego rodzaju wykorzystywane są do operacji grupowanie danych w określone grupy (klasteryzacja) [12].

Reinforcement Learning jest podejściem bazującym na danych pobieranych ze zdefiniowanego środowiska a następnie wybiera opcję dla której posiada maksymalną wartość zysku. Jest to popularne rozwiązanie stosowane w wielu grach komputerowych oraz programach automatyzujących, gdzie po podjęciu konkretnego działania następuje konieczność wyboru opcji optymalnego rozwiązania. Użytkownik musi jednak zdefiniować w fazie projektowania, jakie są jego priorytety i jakie są oczekiwania wobec wykonywanego programu [19].

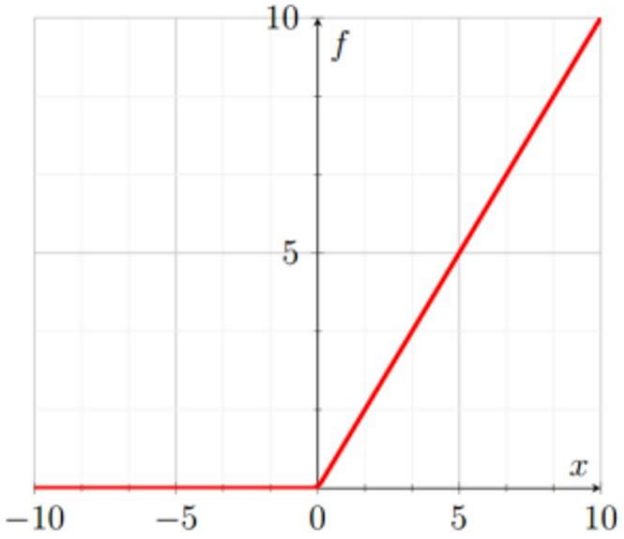
3.2.2 Zastosowanie w życiu codziennym

Obecnie każdy nawet o tym nie wiedząc ma codziennie styczność z algorytmami uczenia maszynowego. Sprawdzając poranną pocztę, dzięki inteligentnemu wykrywaniu spamu nie musimy zaczynać dnia od grupowania wiadomości na ważne oraz spam, dzieje się to automatycznie właśnie w oparciu o uczenie maszynowe. Wiele osób zastanawia się dlaczego reklamy, które są widoczne na jego ekranie tak bardzo wpisują się w aktualne oczekiwania. Akceptując popularne ciasteczka wyrażamy zgodę na przetwarzania danych przeglądania, które są analizowane przez sieć neuronową, która na podstawie algorytmów klasteryzacji dopasowuje treści z tej samej kategorii jak ostatnio przeglądane. Nie trzeba uruchamiać nawet komputera, żeby wejść w świat nowoczesnych technologii. Obecnie urządzenia gospodarstwa domowego również działają w oparciu to najnowocześniejsze rozwiązania techniczne. Przykładem są tutaj odkurzacze, które inteligentnie dopasowują moc ssania do aktualnej powierzchni na której operują. Istnieją również oświetlenia dopasowujące temperaturę barwową oraz moc świetlną w zależności od aktualnego użytkownika lub również warunków świetlnych znajdujących się na zewnątrz. Jednak uczenie maszynowe to nie tylko ułatwianie jego życia ale również jego ratowanie oraz dbałość o bezpieczeństwo. Wiele zabiegów medycznych jest wykonywanych w oparciu o dane, które zostały zebrane na poprzednich pacjentach. Dysponując takimi informacjami można określić aktualne ryzyko oraz najbardziej optymalne rozwiązanie pracując jednocześnie w czasie rzeczywistym. Branża systemów inteligentnych ma jednak swoją kolebkę w przemyśle obronnym oraz militarnym. Zawsze najnowocześniejsze rozwiązanie najpierw zostają testowane w zakresie militarnym a dopiero potem trafiają do zastosowań cywilnych. Dla przykładu mogą posłużyć rakiety typu **LRASM** (*Long Range Anti-Ship Missile*), które same dobierają trajektorię lotu w taki sposób aby uniknąć wykrycia przez radary przeciwnika skracając przy tym czas reakcji do minimum. Dodatkowo wyposażone są one w systemy przetwarzania obrazu aby trafić w najbardziej newralgiczne miejsce. Bez uczenia maszynowego nie skorzystalibyśmy z tłumacza, który z pomocą algorytmów rozpoznaje nie tylko poszczególne słowa ale również składnię oraz gramatykę języka. Obecnie nie jest jeszcze to na tyle dopracowane, aby można było polegać na takim rozwiązaniu bezgraniczne, jednak widoczny jest duży rozwój technologii z perspektywy ostatnich lat. Pozostając przy podobnych rozwiązaniach należy wspomnieć również

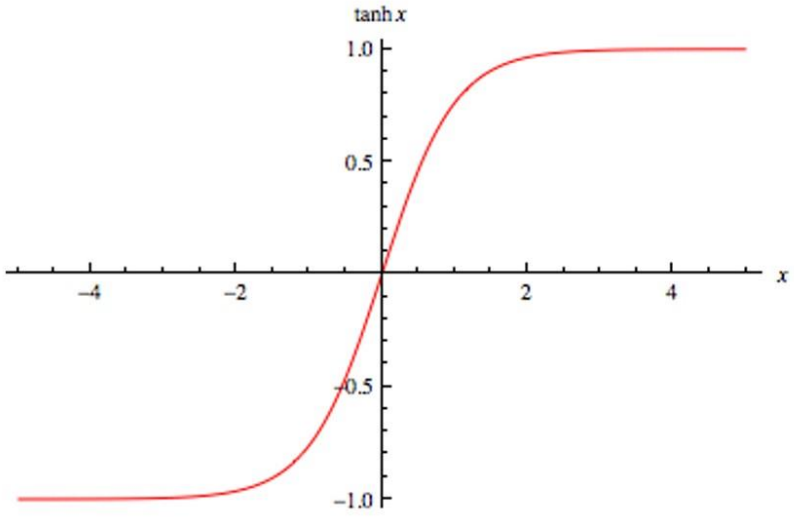
o przetwarzaniu języka naturalnego, co jest szczególnie istotnie dla osób z pewnym stopniem niepełnosprawności werbalnej. W oparciu o rozwiązania techniczne są oni w stanie komunikować się z resztą społeczeństwa co zapobiega izolacji społecznej.

3.2.3 Budowa neuronu

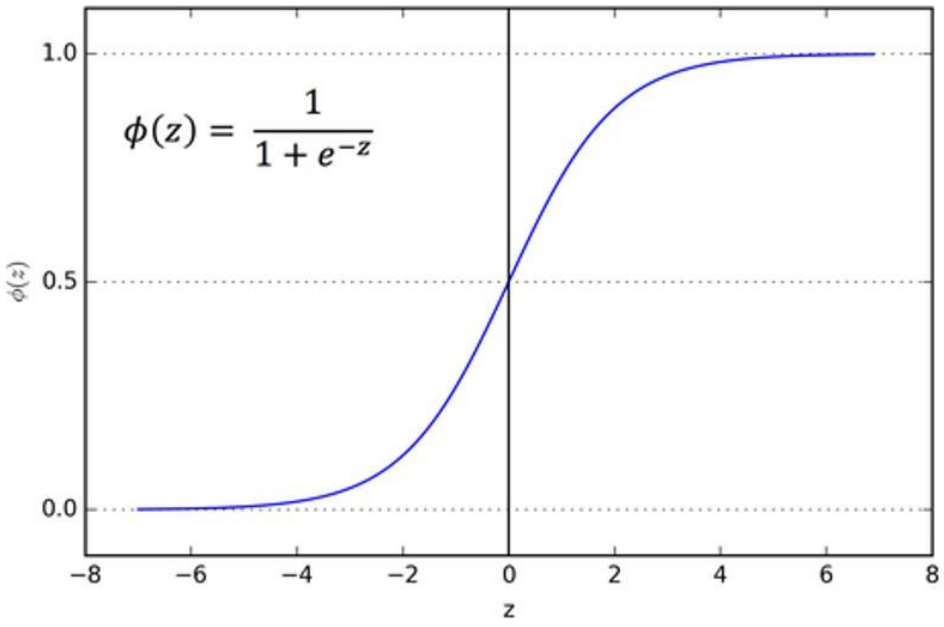
Początki uczenia głębokiego (*Deep learning*) można datować na lata czterdzieste ubiegłego wieku. Było to możliwe w oparciu o formułę matematycznej implementacji sztucznego neuronu, który z kolei powstał jako inspiracją naturą a w szczególności odkryciu sposobu w jaki nasz mózg przekazuje informacje do reszty organów. Dane w postaci liczbowej reprezentujące wartości wag połączeń między poszczególnymi elementami pochodzą z gałęzi zwanych dendrytami, które następnie trafiają do głównego węzła sumującego. W zależności od struktury sieci do układu może zostać dołączone dodatkowe obciążenie (*bias*). W głównej jednostce operacyjnej zwanej również dendrytem następuje wykonanie operacji matematycznej polegającej na wymnożeniu każdej wartości pochodzącej a dendrytu przez jej wagę, zsumowania wszystkich dostarczonych informacji a następnie przepuszczenie ich przez funkcję determinującą wartość na wyjściu (*activation function*). Można wyróżnić trzy podstawowe rodzaje wspomnianych funkcji. Pierwszą z nich jest **RuLU** (*Rectified Linear Unit*), która cechuje się linową zależnością, gdy wartość wejściowa jest dodatnia, w przeciwnym wypadku na wyjściu układu otrzymamy zero. Ze względu na łatwą implementację oraz wysoką wydajność jest one najczęściej stosowana w praktycznej implementacji. Możemy wyróżnić również funkcję w postaci **sigmoid** oraz **tanh**, których matematyczne implementacje oraz kształty zostały przedstawione poniżej [30].



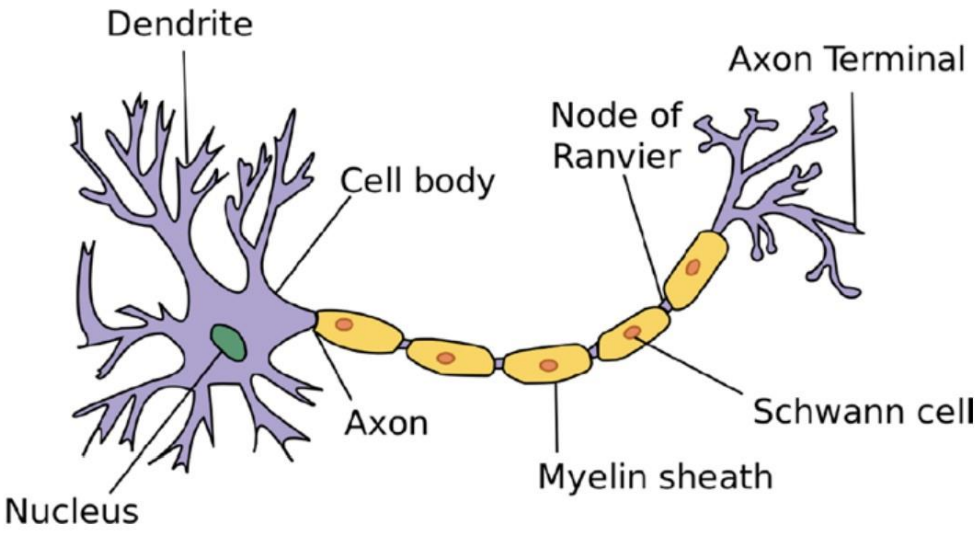
Rysunek 23.. Funkcja aktywacji ReLU



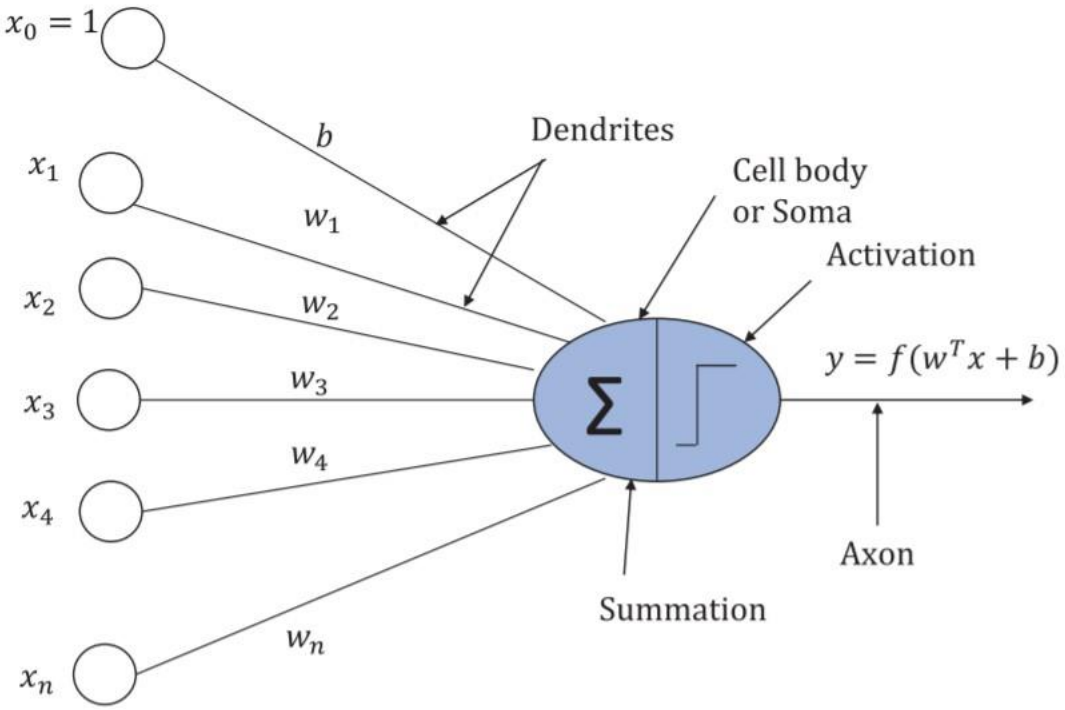
Rysunek 24. Funkcja aktywacji tanh



Rysunek 25. Funkcja aktywacji sigmoid



Rysunek 26. Budowa neuronu inspirowana biologią [15].



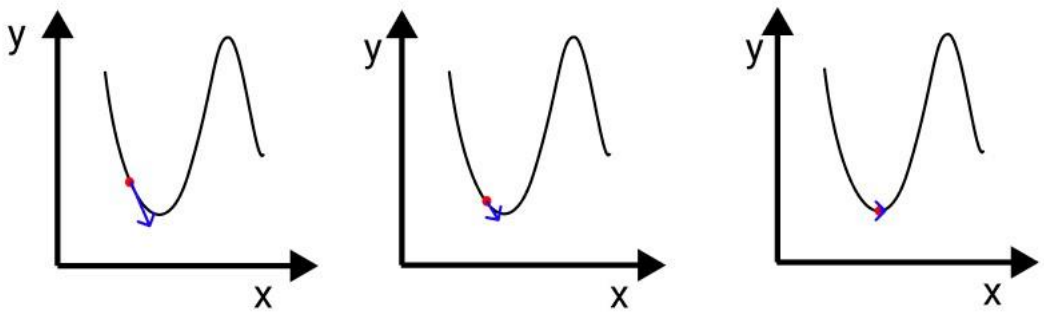
Rysunek 27. Matematyczna implementacja neuronu [15].

3.2.4 Określenie dokładności otrzymanego modelu oraz funkcje strat

Otrzymując współczynniki modelu z punktu widzenia użytkownika istotnym parametrem jest zdefiniowanie różnicy pomiędzy punktami testowymi a przebiegiem poddanemu aproksymacji w postaci modelu. W tym celu wprowadzone zostały funkcje strat. Najpopularniejszym z nich dla regresji liniowej jest estymator **LS** (*Least Squares*), którego równanie zostało przedstawione poniżej.

$$J(\beta_0, \beta_1) = \sum_{i=0}^{n} (y_i - \beta_0 - \beta_1 x_i)^2 \tag{8}$$

Do obliczenia tego parametru potrzebny jest zbiór punktów pomiarowych postaci krotek (x_i, y_i), natomiast poszukiwane wartości to β_0 oraz β_1. Dla uzyskania optymalnej postaci modeli punktem wyjściowym jest określenie takiej wartości pary współczynników dla których funkcja strat będzie przyjmować najmniejszą możliwą wartość. Narzędziem, które to umożliwia są obliczenia gradientowe. Z punktu widzenia sieci neuronowej istotną cechą jest **Propagacja wsteczna** (*Backpropagration*), która umożliwia wrócenie układu współczynników do poprzedniego stanu i wykonywanie operacji dopasowywania, aż do momentu uzyskania minimalnej wartości funkcji strat [30].



Rysunek 28. Wyznaczenie minimum lokalnego w oparciu o obliczenia gradientowe

Przedstawiony problem, który jest do rozwiązania przy pomocy sieci neuronowej potrzebuje innego sposobu na klasyfikację poprawności modelu, ponieważ danymi wejściowymi nie są punkty pomiarowe tylko macierze reprezentujące obraz. Nie można zatem mówić o regresji tylko o klasyfikacji a dokładniej o wykrywaniu obiektów. W celu określenia precyzji modelu zostaną wykorzystane zależności w oparciu o poniższą tabelę.

		Otrzymany wynik	
		Positive	Negative
Wynik Przewidywany	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Rysunek 29. Ocena poprawności modelu dla wykrywania obiektów

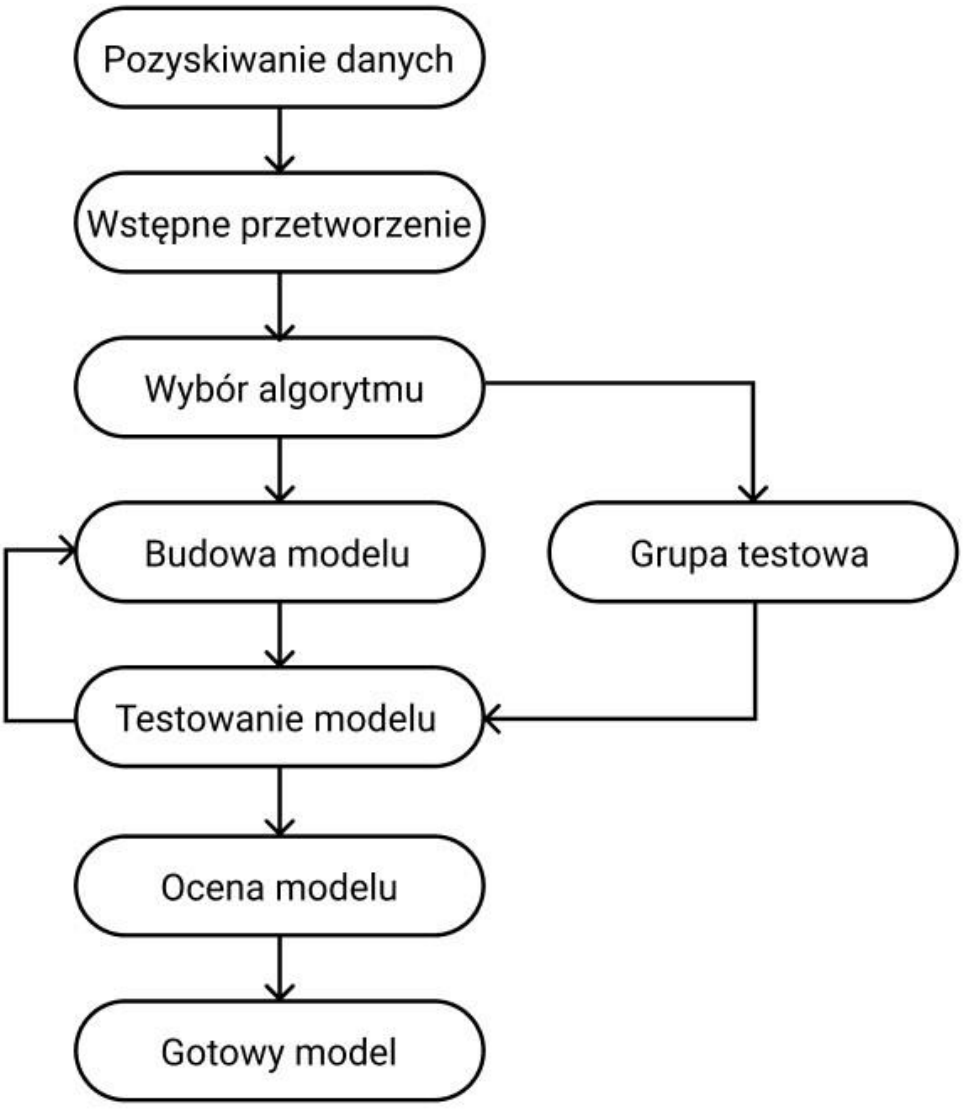
Dwa podstawowe parametry na podstawie powyższej tabeli to *Accuracy* oraz *Recall*. Pierwszy z nich określa w jakich proporcjach wykrywanie obiektów zostało wykonane w sposób poprawny, natomiast *Recall* określa stosunek poprawnych detekcji do sumy przypadków w których wykrywany obiekt znajduje się w przestrzeni roboczej jednak nie został on wykryty oraz przypadków w których detekcja przebiegła pomyślnie. Dla wszystkich przypadków możemy określić również straty modelu w oparciu o informacje z poprzedniego punktu.

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

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

3.3 Tworzenie sieci neuronowej

3.3.1 Schemat



Rysunek 30. Schemat tworzenia sieci neuronowej

3.3.2 Pozyskanie danych

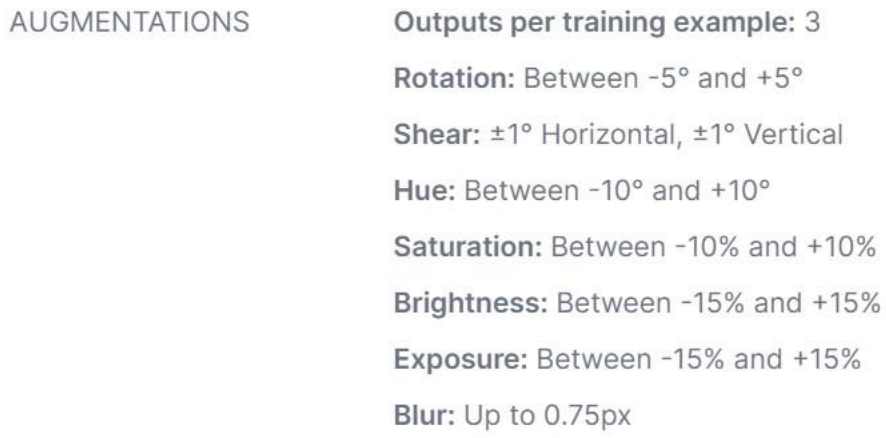
W celu automatyzacji gromadzenia danych wejściowych wykorzystany został skrypt w języku Python. Pierwszym krokiem jest zaimportowanie potrzebnych bibliotek oraz określenie nazwy etykiety, która będzie identyfikowana w procesie wykrywania (zmienna *detect*). Niezbędnym parametrem jest również określenie ilości przechwyconych obrazów, którą reprezentuje zmienna *image_batch_size*. Każdy obraz powinien posiadać swój identyfikator co można otrzymać w oparciu o moduł *uuid*. Niezbędne będzie również wykorzystanie wyrażeń regularnych to poszukiwania wzorca tekstowego. Moduł do zarządzania czasem posłuży do uzyskania niezbędnego interwału czasowego pomiędzy kolejnymi przechwyceniami. Wszystkie operacje zostały wykonane na obiekcie statycznym natomiast współrzędne obszaru, w którym znajdują się poszukiwane otwory dobrane zostały w sposób eksperymentalny. Każde zdjęcie powinno posiadać plik zapisany w notacji XML zawierający informacje o współrzędnych obiektu, który będzie wykrywany. Skrypt został przystosowany do wielokrotnego użytku co oznacza, że w każdej chwili możemy powiększyć istniejącą bazę. Kod programu oraz przykładowe zdjęcie z naniesionymi etykietami przedstawione zostały poniżej.

Rysunek 31. Zdjęcia wraz z etykietą w postaci XML

```
import uuid
import cv2
import re
import shutil
import time
import os

detect = 'hole'
image_batch_size = 300

PROJECT_DESTINATION = os.path.join('TensorflowObjectDetectionAPI',
                                   'mydataset')
XML_SAMPLE_PATH = os.path.join('TensorflowObjectDetectionAPI',
                               'sample.xml')
# Generate UUID list
generate_uuid_list = [str(uuid.uuid4().int)
                      for i in range(image_batch_size)]

# CreateXML file
for n in range(image_batch_size):
    destination = os.path.join(PROJECT_DESTINATION,
          detect+'.'+'{}.xml'.format(generate_uuid_list[n]))
    shutil.copy(XML_SAMPLE_PATH,destination)

# Check if defined model name exist
#(Useful when we collect extra data after first collecting)
if not os.path.exists(PROJECT_DESTINATION):
  !mkdir {PROJECT_DESTINATION}

RTSP = os.environ.get('RTSP_ADDRESS')
cap = cv2.VideoCapture(0)

for n in range(image_batch_size):
    print(f'Nr. {n}')
    ret, frame = cap.read()
    imgname = os.path.join(PROJECT_DESTINATION,
      detect+'.'+'{}.jpg'.format(generate_uuid_list[n]))
    cv2.imwrite(imgname, frame[320:370,310:600])
    cv2.imshow('frame', frame[320:370,310:600])
    time.sleep(1)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

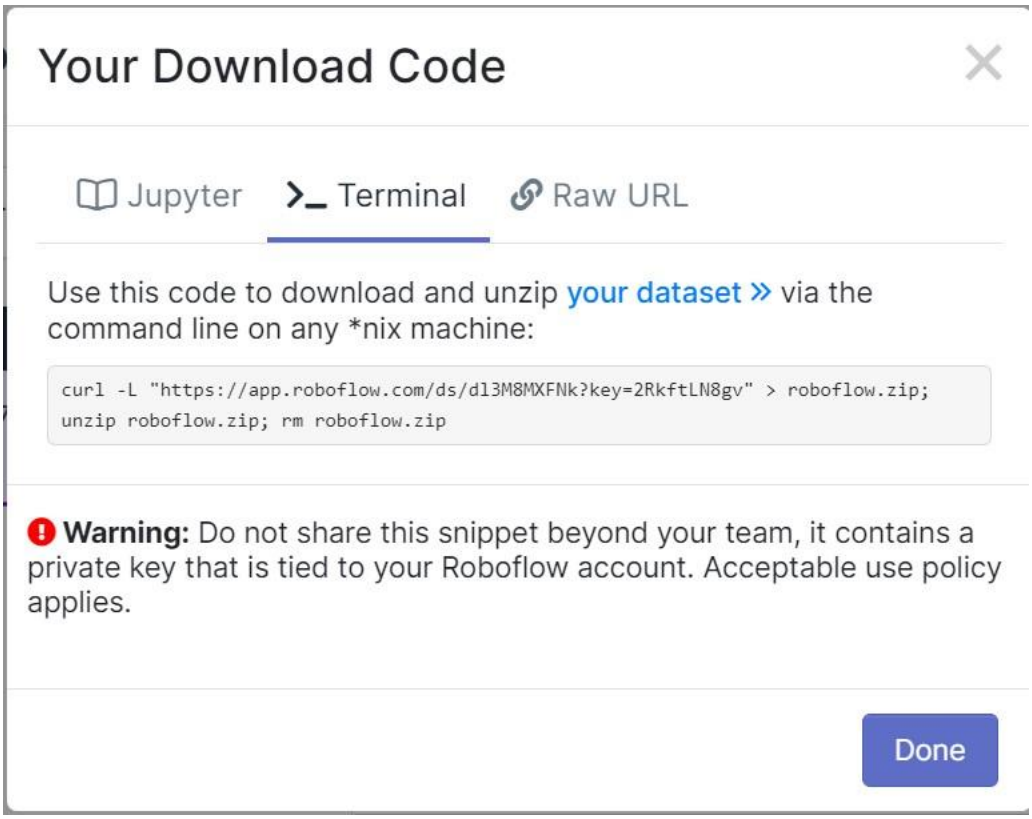
3.3.3 Wstępne przetworzenie

Model może być wygenerowany prawidłowo tylko i wyłącznie w przypadku gdy dane podczas całego procesu uczenia będą zgeneralizowane co oznacza, że muszą przedstawiać stan obiektu w jak największej liczbie wariantów oraz kombinacji. Wyzwanie polega na tym, że podczas procesu gromadzenia danych, wszystkie zgromadzone informacje różnią się od siebie nieznacznie co może powodować możliwość wystąpienia zjawiska opisywanego w literaturze jako *overfitting*. Oznacza to, że model może być dobrze dopasowany do danych na których bazował podczas nauki, jednak w przypadku gdy na jego podejście zostanie podany obraz w nieco innym oświetleniu lub orientacji otrzymany wynik będzie znacząco różny od oczekiwanego. W celu uniknięcia opisanego zjawiska stosuje się zabieg znany jako *Data Augmentation* polegający na sztucznym zwiększeniu dostępnych informacji poprzez dodatkowe operacje wykonywane na obrazie. Szeroki pakiet rozwiązań w tym zakresie oferuje platforma Roboflow, która w darmowej wersji posiada ograniczenie do trzykrotnego zwiększenia dostępnych informacji oraz posiada górny limit w postaci obsługi maksymalnie tysiąca obrazów co jest wystarczające do otrzymania modelu, który będzie wykrywał obiekty w sposób poprawny. W ramach darmowej licencji możemy wykonywać operacje takie jak: zmienianie poziomu nasycenia kolorów, translacje obrazu o wektor, dodanie ziarnistości, zmiana ekspozycji świetlnej. Poniżej widnieje okno z operacjami wykonanymi dla dostarczonych zdjęć wraz z adnotacjami w postaci plików XML [17].



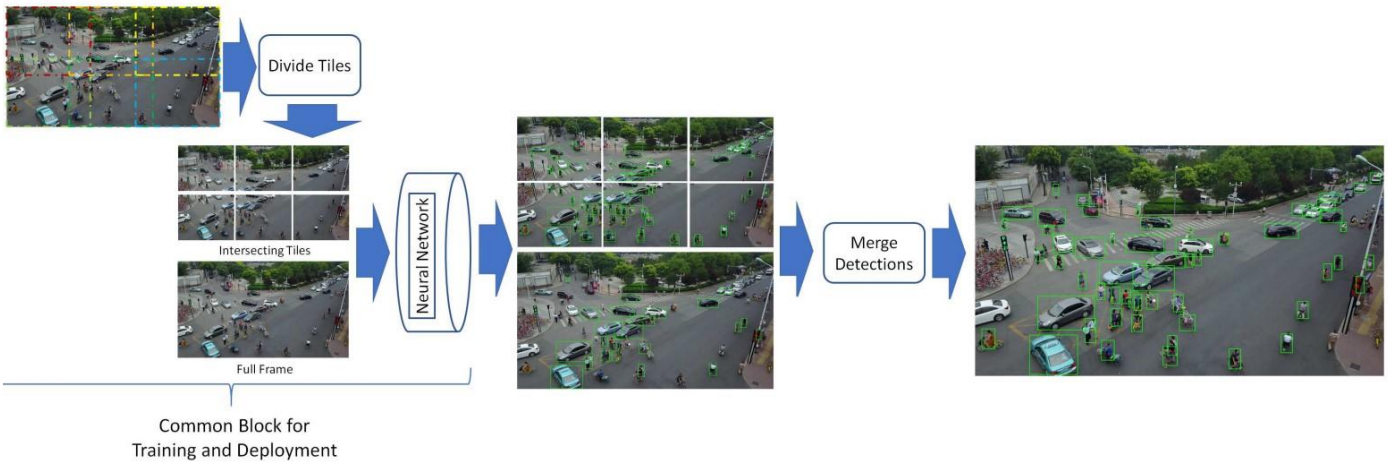
Rysunek 32. Data Augmentation w oparciu o platformę Roboflow

Dodatkową zaletą wykorzystanej platformy jest możliwość eksportowania danych z wykorzystaniem pakietu *curl* do postaci w formacie *TFRecords*, który jest podstawową jednostką nośną informacji przy wykorzystaniu zarówno biblioteki TensorFlow jak i TensorFlow Object Detection API. Gotowe pliki zawierają wszystkie potrzebne informacje do odtworzenia zarówno wykrywanych obiektów jak i ich etykiet.



Rysunek 33. Eksportowanie danych z platformy Roboflow

Dodatkowym wyzwaniem jest konieczność wykrywania otworów, które na zdjęciu można wpisać w prostokąt o bokach 10x10 pikseli. Identyfikacja niewielkich elementów jest szeroko znanym problemem występującym między innymi podczas analizowania zdjęć satelitarnych czy podczas wykrywania osób w tłumie. Przez ostatnie lata zostało powstało kilka sposób, które w znaczy sposób mogą poprawić rezultat działania programu. Najprostszą czynnością jaką można wykonać jest zwiększenie rozdzielczości kamery, która przechwytuje obraz. To rozwiązanie niesie jednak za sobą poważne konsekwencje w postaci możliwych opóźnień w relacji program-kamera spowodowanych niewystarczającą przepustowością łącza. Parametr rozdzielczości został dobrany na samym początku i nie ma możliwości jego zwiększenia bez negatywnego wpływu na kolejne etapy przetwarzania. Innym sposobem na rozwiązanie takiego problemu jest podzielenie zdjęcia na mniejsze fragmenty oraz analizowanie ich osobno po czym na końcu należy złączyć wszystkie informacje w całość. Poniżej przedstawiona zostaje idea pomysłu [16].



Rysunek 34. Wykrywanie niewielkich elementów metodą dzielenia obrazu [13].

Wykonanie takiej operacji jest jednak dość skomplikowane pod względem implementacyjnym, ponieważ dla poszczególnych fragmentów układu powstają nowe układy współrzędnych, dla których wartości etykiet nie pokrywają się z obszarami zdefiniowanymi w plikach XML. Można jednak wyciągnąć wniosek z takiego podejścia, że maksymalne ograniczenie obszaru powoduje procentowy wzrost udziału powierzchni, która podlegać będzie wykryciu co z kolei podnosi skuteczność stosowanych algorytmów. Wiedza wstępna o obiekcie pozwala jednak stwierdzić, że wykrywane elementy znajdują się w niewielkiej odległości od siebie, więc należy wyłącznie maksymalnie ograniczyć obszar wykrywania. Biorąc pod uwagę przytoczone właściwości, fragment kodu odpowiadający za przetwarzanie obrazu z kamery (sekcja z OpenCV) został dobrany w taki sposób aby maksymalnie ograniczać przestrzeń pracy. Do sieci neuronowej zwracane są zatem wyłącznie wyseparowane fragmenty zawierające najważniejsze elementy.

3.3.4 Wybór modelu bazowego w oparciu o framework TensorFlow Object Detection API

Rysunek 35. TensorFlow Object Detection API

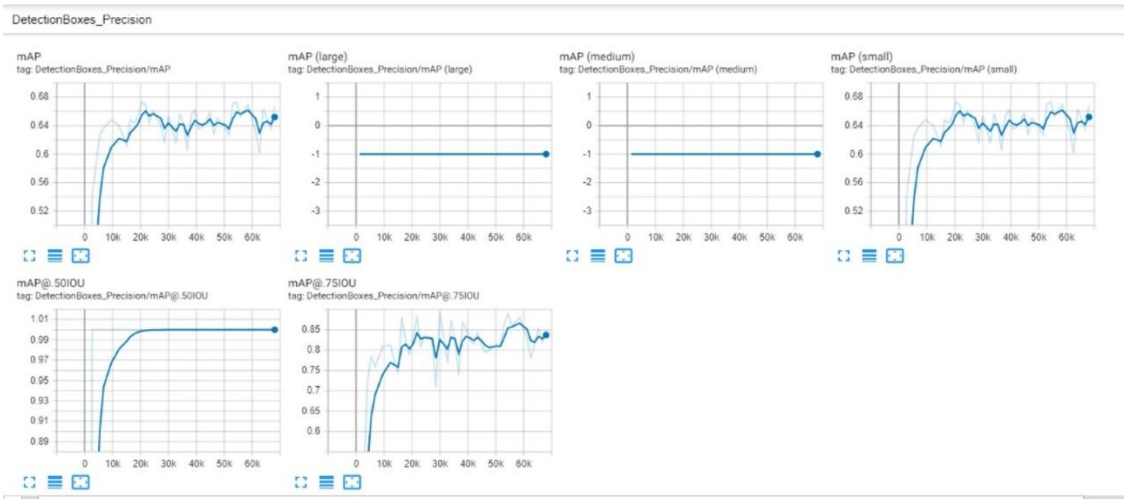
TensorFlow Object Detection API jest biblioteką działającą w oparciu o framework *TensorFlow*, która umożliwia tworzenie, trenowanie oraz wystawianie modeli do fazy produkcyjnej w oparciu o wykrywanie oraz identyfikację wielu obiektów z pojedynczego zdjęcia. Wykrywane elementy mogą być zdefiniowane różnych kategoriach. Pakiet umożliwia pełną współpracę z bibliotekami typu *Computer Vision* (w tym *OpenCV*). Użytkownik otrzymuje dostęp do zaimplementowanych algorytmów (*SSD*, *Faster R-CNN*), które wystarczy jedynie przetrenować dla własnych danych otrzymując w ten sposób wagi połączeń między neuronami dla zdefiniowanego problemu. Głównym aspektem przy wyborze odpowiedniego modelu jest kompromis między wydajnością a dokładnością. W przypadku gdy chcemy wykrywać obiekty w czasie rzeczywistym w oparciu o sprzęt z niskimi parametrami należy pogodzić się z faktem, że precyzja może znacząco odbiegać od wartości oczekiwanej. Programista określając strukturę sieci powinien posiadać informację na jakim sprzęcie aplikacja będzie wykonywana co pozwoli na znalezienie odpowiedniego kompromisu w relacji wydajność-dokładność.

3.3.5 Budowa modelu

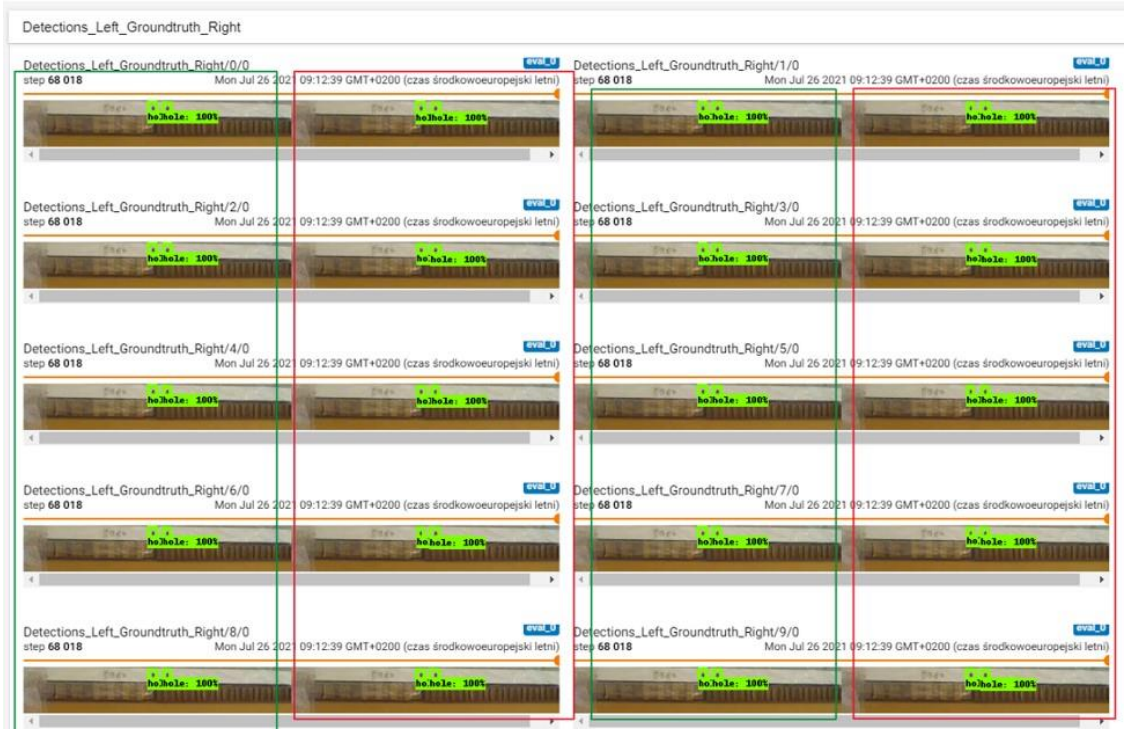
Ze względu na ograniczenia sprzętowe operacja trenowania modelu wykonana została w oparciu o środowisko *Colab*, które jest chmurowym odpowiednikiem oprogramowania *Jupyter*. Aplikacja umożliwia bezpośrednie połączenie z dyskiem oferowanym przez firmę Google, co przy odpowiedniej konfiguracji umożliwia automatyczną synchronizację z plikami znajdującymi się lokalnie na komputerze. Należy jednak mieć na uwadze, że wszystkie operacje zarządzania plikami są wykonywane w oparciu o serwer chmurowy co skutkuje koniecznością zastosowania poleceń zgodnych z formatem systemu Linux. Po odpowiedniej konfiguracji oraz zainstalowaniu wszystkich potrzebnych bibliotek. Funkcje oraz klasy stosowane podczas całego procesu wykorzystane zostały w oparciu o materiały dostarczone przez producenta.

3.3.6 Ocena modelu

TensorBoard jest interaktywną nakładką działającą w oparciu o bibliotekę TensorFlow. Umożliwia użytkownikom śledzenie na bieżąco parametrów takich jak funkcje strat, dokładność badanego modelu. Dla problemu klasyfikacji oraz wykrywania narzędzie wybiera losowo wybrane zdjęcia z części testowej oraz prezentuje aktualne rezultaty otrzymywane przez program. W oparciu o dodatkowe funkcjonalności możemy projektować sieć zmieniając liczbę warstw oraz obserwować zmiany na bieżąco w przyjazny dla użytkownika sposób graficzny. Istnieje również możliwość testowania algorytmów polegających na klasteryzacji poszczególnych danych grupując je odpowiednio w grupy bazując na algorytmach, które możemy modyfikować równolegle do wykonywania programu. Platforma umożliwia eksplorację danych różnego rodzaju wliczając w to dane tekstowe, obrazy czy nawet pliki dźwiękowe [17]. Poniżej zaprezentowane zostały zrzuty ekranu z podsumowania wygenerowane w oparciu o *TensorBoard* [20].



Rysunek 36. Ocena modelu w postaci liczbowej



Rysunek 37. Ocena modelu w postaci graficznej

Ocena poprawności wykonanego modelu została wykonana w oparciu o proces dwuetapowy. W pierwszej części zostały przedstawione wartości w postaci numerycznej uzyskane podczas procesu uczenia. Wartość iteracji wynosi ponad 50 tysięcy i została ona wybrana w sposób eksperymentalny poprzez ostawienie dużej wartości wykonywanych pętli a jednoczenie zwracanie odpowiedzi układu co określoną liczbę iteracji. W tym celu wykorzystany również został parametr postępu modelu (*learning rate*) określający progres jaki został uczyniony od czasu ostatniej wykonywanej oceny. Jeżeli wartość ta będzie już niewielka można zatrzymać wykonywaną symulacje oraz wyeksportować otrzymany model. Na podstawie otrzymanych grafik można stwierdzić, że największa dynamika występuje podczas pierwszych 20 tysięcy iteracji, natomiast potem model uczy się już wolniej. Ważniejszym aspektem jest jednak ocena w oparciu o praktyczne wykrywanie obiektów co zostało przedstawione powyżej. Kolorem czerwonym zostały zaznaczone elementy, które zostały dostarczone do procesu uczenia. Zielona obwiednia oznacza wykryte elementy z bazy testowej, która nie miała dostępu do sieci w trakcie uczenia co pozwoliło zachować generalizacje modelu. Zgodnie z załączonymi grafikami wszystkie elementy zostały wykryte w sposób poprawny co oznacza, że sieć została poprawnie skonfigurowana oraz odpowiednio reaguje na dane, które wcześniej nie były dostarczone na jej wejście.

4 Podsumowanie oraz możliwości dalszego rozwoju aplikacji

Obecnie aplikacja jest w pełni funkcjonalna wyłącznie w postaci desktopowej przy konieczności wcześniejszego instalowania wszystkich potrzebnych bibliotek oraz plików od nich zależnych. W sytuacji, gdy program byłby wykonywany tylko i wyłącznie na jednym komputerze taka sytuacja byłaby w pełni akceptowalna. Jednak w przyszłości może się okazać, że różne oddziały firmy będą potrzebować tego samego oprogramowania co może być kłopotliwe jeżeli zaszłaby jakaś zmiana lub aktualizacja. Kluczem do rozwiązania problemu może okazać się aplikacja w postaci przeglądarkowej, gdzie użytkownik będzie wyłącznie personalizował adres wejścia protokołu RTSP dla odpowiednio skonfigurowanej kamery. Kolejną niedogodnością, która powinna zostać rozwiązania jest czas wykonywania analizy pojedynczej klatki. Na aktualnym etapie problem jest rozwiązywany poprzez wykorzystywanie wydajnej karty graficznej co jednak znacząco zwiększa koszty eksploatacji. Wykorzystanie mniej dokładnego modelu nie daje zadowalających rezultatów co jest obiektem stanowiącym podstawę do dalszego optymalizowania układu. Domyślnie po rozwiązaniu wspomnianych utrudnień aplikacja będzie składać się z dwóch części: warstwy graficznej (*front end*) oraz logicznej (*back end*). Pierwsza z nich zbudowana zostanie w oparciu o framework *React*. Warstwa zarządzająca częścią logiczną będzie jednak dużo większym wyzwaniem, ponieważ będzie musiała łączyć ze sobą wiele dodatkowych bibliotek i zwrócić odpowiedź układu w postaci pliku JSON jako API, który będzie wizualizowany w oparciu o język JavaScript. Pracę z wystawieniem modeli w proces produkcyjny ułatwia narzędzie *Tensorflow Servings*, które wymaga jednak znajomości takich narzędzi *Kuternates* oraz obsługę *Google Cloud Platform*. Aplikacja pobiera plik zdjęciowy a następnie zwraca odpowiedź w postaci JSON z wykrytymi elementami oraz wartościami procentowymi określającymi prawdopodobieństwo wykrytych obiektów. Jako środowisko obsługujące dodatkowe rozwiązania zastosowany zostanie framework *Django* oraz *Django REST* który umożliwia łącznie baz danych z zewnętrznymi API w oparciu o język Python.

5 Bibliografia

[1] “OpenCV About” https://opencv.org/about/

[2] “Image Color Conversions.” https://docs.opencv.org/3.4.15/de/d25/imgproc_color_conversions.html

[3] “Conda Environment.” https://docs.conda.io/en/latest/

[4] “Google Colaboratory.” https://colab.research.google.com/?utm_source=scs-index

[5] “Jupyter_notebook.” https://jupyter.org/about

[6] “Tutorial_gausian_median_blur_bilateral_filter.” https://docs.opencv.org/4.5.3/dc/dd3/.html

[7] “Computer Vision: Canny Edge Detection.” https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html.

[8] “Computer Vision: Contours.” https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html

[9] “Computer Vision: Dilation.” https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html

[10] “Computer Vision: Thresholding.” https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

[11] Bonnin, Rodolfo. Packt Publishing 2016. “Building Machine Learning Projects with TensorFlow-Packt”

[12] Buds, Tech. Medium 2020. “Reinforcement.” https://medium.com/@techbuds20/supervised-unsupervised-and-reinforcement-learning-58b6b7e6b0dc

[13] F. Ozge Unel, Turkey, Burak O. Ozkalaycı Aselsan Inc. Open Access Workshop Paper. “The Power of Tiling for Small Object Detection.”

[14] Huang J, Sun C, Rathod V. CVPR 2017. “Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors.”

[15] Rafael C. Gonzalez, Richard E.Woods “Digital Image Processing 4th.”

[16] Richard Szeliski, Springer 2010 “Computer Vision: Algorithms and Applications.”

[17] “Roboflow” . https://app.roboflow.com/

[18] Santanu Pattanayak, Apress 2017. “Pro Deep Learning with TensorFlow - a Mathematical Approach to Advanced Artificial Intelligence in Python.”

[19] Soni, Devin TowardsDataScience 2018. “Supervised Vs. Unsupervised Learning.” https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d

[20] “TensorBoard” https://www.tensorflow.org/tensorboard

[21] “Image Kernels” https://setosa.io/ev/image-kernels/

[22] “Image Processing Tutorial” https://www.pyimagesearch.com/category/image-processing/

[23] “Image Processing Tutorial – ipynb” https://colab.research.google.com/github/xn2333/OpenCV/blob/master/Image_Processing_in_Python_Final.ipynb

[24] “Feature Extraction in 2D Color Images” https://www.youtube.com/watch?v=lv8-YgWYtU4

[25] “ROI Segmentation, Color Detection and Image Thresholding Using OpenCV”

https://medium.com/swlh/roi-segmentation-contour-detection-and-image-thresholding-using-opencv-c0d2ea47b787

[26] “Small Object Detection using SSD Model”

https://github.com/tensorflow/models/issues/3196

[27] “Training Faster R-CNN Object Detection on a Custom Dataset”

https://colab.research.google.com/drive/1U3fkRu6-hwjk7wWIpg-iylL2u5T9t7rr#scrollTo=uQCnYPVDrsgx

[28] “Tacking the Small Object Problem in Object Detection”

https://blog.roboflow.com/detect-small-objects/

[29] “Deploying a TensorFlow Model to Production made Easy”

https://towardsdatascience.com/deploying-a-tensorflow-model-to-production-made-easy-4736b2437103

[30] “Tensorflow 2.0 Complete Course – Python Neural Networks for Beginners Tutorial”

https://www.youtube.com/watch?v=tPYj3fFJGjk&t=16848s&ab_channel=freeCodeCamp.org

[31] “TensorFlow Object Detection – Python Course with Projects”

https://github.com/nicknochnack/TFODCourse

6 Załączniki

https://github.com/margolek/summer-practise-2021/blob/master/OpenCV_Detection.ipynb

https://github.com/margolek/summer-practise-2021/blob/master/Collect_data.ipynb

https://github.com/margolek/summer-practise-2021/blob/master/Model_training.ipynb

https://github.com/margolek/summer-practise-2021/blob/master/Run%20trained%20model%20with%20OpenCV.ipynb