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

Slawomir Serafin

2021-11-26

Contents

Wprowadzenie

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.

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 oświeletniowych panujących na zakładzie. Jeżeli okaże się to niezbędne, koniecznie może być zastosowanie dodatkowych źródeł światła aby uzsykać 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 pomocji 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 uniwesalny, czyli taki który możemy wykorzystać również w innym języku niż Python.

Oczekiwany rezultat

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

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

Figure 1: Zdefiniowanie problemu projektowego

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 umozliwia nam wykrywanie obiektór **TensoFlow Object Deteftion API**.

Cyfrowe przetwarzanie obrazów z wykorzystaniem biblioteki OpenCV

Protokół RTSP

Pierwszym krokiem zanim zaczniemy analizować jakiekolwiek dane musimy uzyskać do nich dostęp. Możemy to zrobić na wiele różnych sposób. Jednym z nich jest fizyczne zrobienie zdjęć nad obiektem, które potem będziemy analizować. Rozwiązanie wydaje się stosunkowo proste jednak niesie ze sobą wiele negatywnych aspektów. Rozdzielczość, którą będziemy aktualnie dysponować oraz inne parametry aparatu mogą się znacznie różnić od docelowego rozwiązania. Co więcej, analiza pojedyńczej 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. Do-

datkowo aby jednocześnie kontrolować aspekt wydajnościowy przekazywany obraz powinien na bieżąco dostarczać kolejne klatki przechwyconego obiektu. W tym miejscu również możemy zastosować dwa podejścia zastosowania 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 ilość miejsca. Problem postanowiono zatem rozwiązać przy pomocy serwera chmurowego. Takie podejście znacznie ogranicza zastosowanie wszelkich fizycznych połączeń. Niestety musimy tutaj zaakceptować pewnie komprosimy. Jednym z nich jest ograniczenie rozdzielczości w taki sposób aby uniknąć braku płynności w przekazywanym obrazie. Ta cecha jak się okaże w kolejnych etapach okaże się 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 przesyłu obrazu. Wybór padł tutaj na rozwiązanie w oparciu o protokół **RTSP** *(Real Time Streaming Protocol)*. Jest to rozwiązanie, które znacznie zyskało na swojej popularności podczas okresu pandemicznego, gdzie większość z nas została zmuszona do przejścia w tryb pracy zdalnej. Całość systemu oparta jest na serwerze, który udostępnia nam dostęp do obrazu 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 transmicji różnego rodzaju konferencji, posiedzeń zarządów, wydarzeń kulturowych, czy nawet wydarzeń sportowych. Poniższa grafika przedstawia uproszczony schemat działania. Platformą obługującą w naszym przypadku będzie biblioteka *OpenCV*.

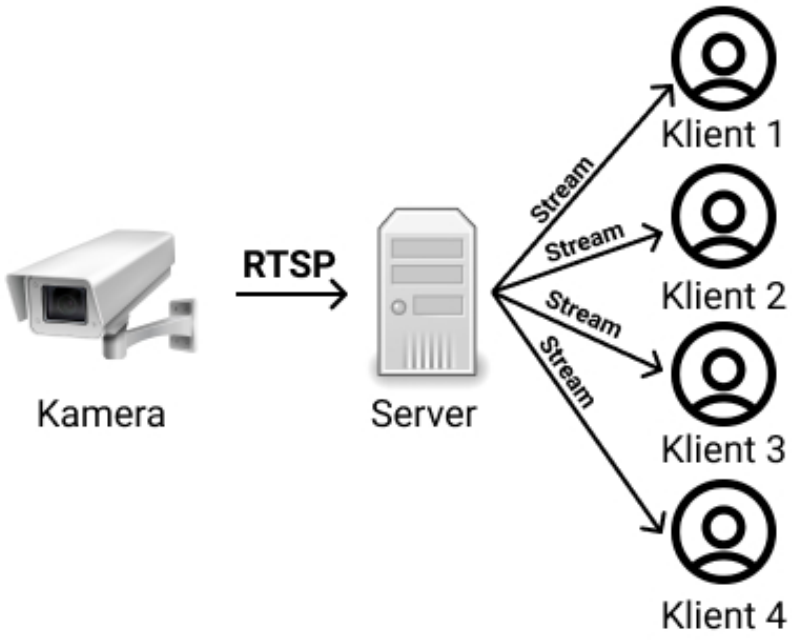


Figure 2: Wykorzystanie protokołu RTSP jako źródło informacji o produkcie

Biblioteka OpenCV

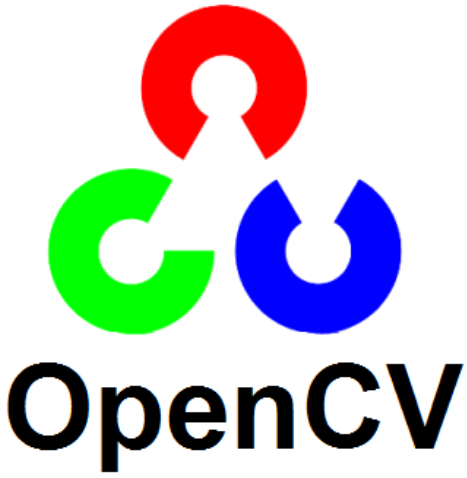


Figure 3: Logo biblioteki OpenCV

OpenCV (Open Source Computer Vision Library) jest biblioteką funckconującą na zasadach licencji otwartnej. Jej głównym zastoswaniem 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 komptybilności 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 najnowoczesniejsze struktury dostarczane przez społeczność, która znacząco przyczynia się do rozwoju produktu. Algorytmy mogą służyć między innymi do wykrywania twarzy na zdjęciach, identyfikacji obiektów, klasyfikacji nastoru w jakim znajduje się użytkownik, śledzenia obiektu w ruchu, redukcji czerownych oczu. W dobie sytuacji penczemicznej program może wykrywać czy wszyscy uczesticzy spotkania mają założene maseczki. Aktualnie nasilony jest również problem migracyjny. Systemy bezpieczeństwa które nieustannie sledzą granice państw są wyposażone również w algorytmy inteligentnego przetwarzania obrazu. Dane na stronie producenta wzkazują, że biblioteka została już pobrana przez ponad 18 milionów użytkowników. OpenCV jest ciągle rozwijana nie tylko przez grupy pastonatów, lecz również jest przedmiotem analiz grup badawczych a nawet organów rządowych. Wsród użytkowników znajdują się rownież 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 użytkowynków 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*.

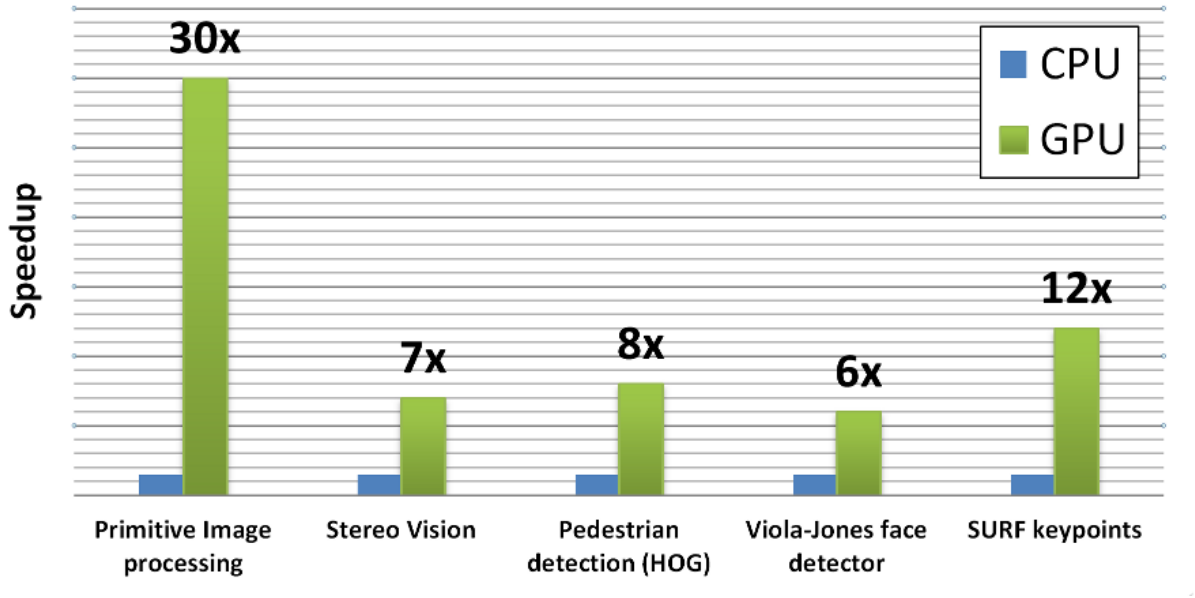


Figure 4: Porównanie czasu obliczeń dla CPU oraz GPU

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 obrębem. 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 uzytkownik nie naciście 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
```

Środowisko pracy

Conda

Figure 5: Conda logo

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 zajmujacych się ptrzetwarzaniem oraz analizą danych. Mogą być to pliki tekstowe jak również multimedialne. Środowisko wspiera również inne popularne jezyki programowania tj. *R*, *Ruby*, *Java*, *C/C++*, *FORTRAN*, *JavaScript*. Conda z łatwością przechowuje, tworzy oraz osbługuje biblioteki, które są instalowane w oparciu o dany język programowania. 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 miejcu. W tym momencie z pomocą przychodzi nam *Wirutalne Środowisko*. Pozwoli nam to wyseparować wersje różnych bibliotek, gdzie zostały już wcześniej zainstalowane na komputerze co w prosty sposób pozwoli nam unikąć możliwej w przyszłości niekompatybilności. Kolejnym ważnym aspektem jest separacja. Dokonując modyfikacji pakietów wyłącznie w obrębie wirtualnego środowiska nie oddziałowujemy w żaden sposób na wersje zainstalowane globalne, co sprawia, że nie wpływamy na kompatybilność wcześnniej wykonanych projeków. Do stworzenia oraz aktywacji wirutalnego 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
```

Jupyter Notebook/Jupyter Lab/Google Colaboratory

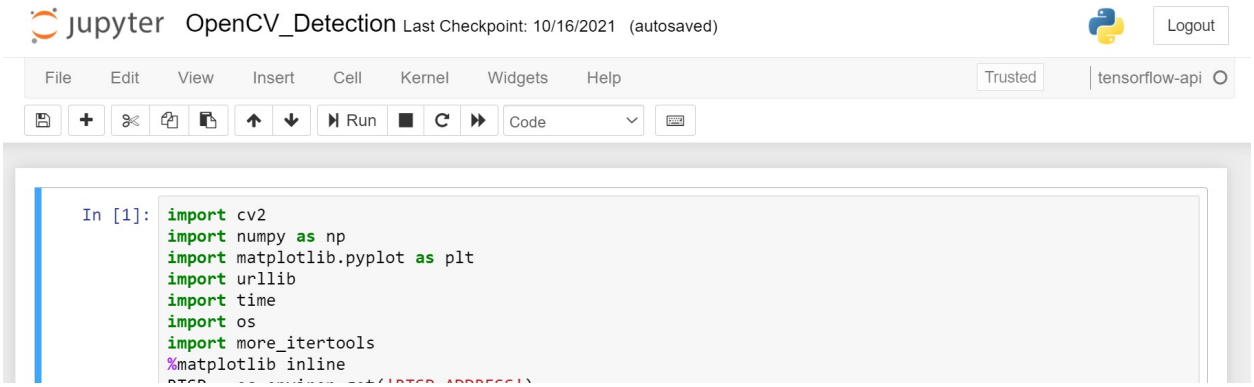


Figure 6: Okno programu Jupyter Notebook

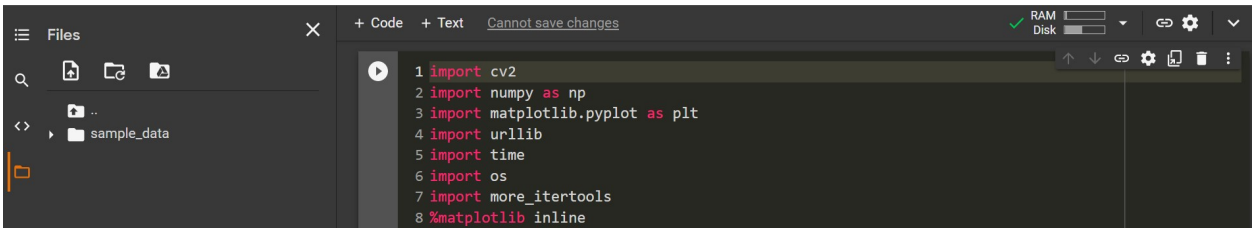


Figure 7: Okno programu Google Colaboratory

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, możemy 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 bądz równań matematycznych zapisywanych zgodnie ze składnią Latex.

Google Colaboratory jest chmurowyn 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 wymagamy konkretnej wersji biblioteki musimy 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 możemy cały projekt zsynchornizować 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 nam 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.

Podstawowa klasyfikacja obrazów

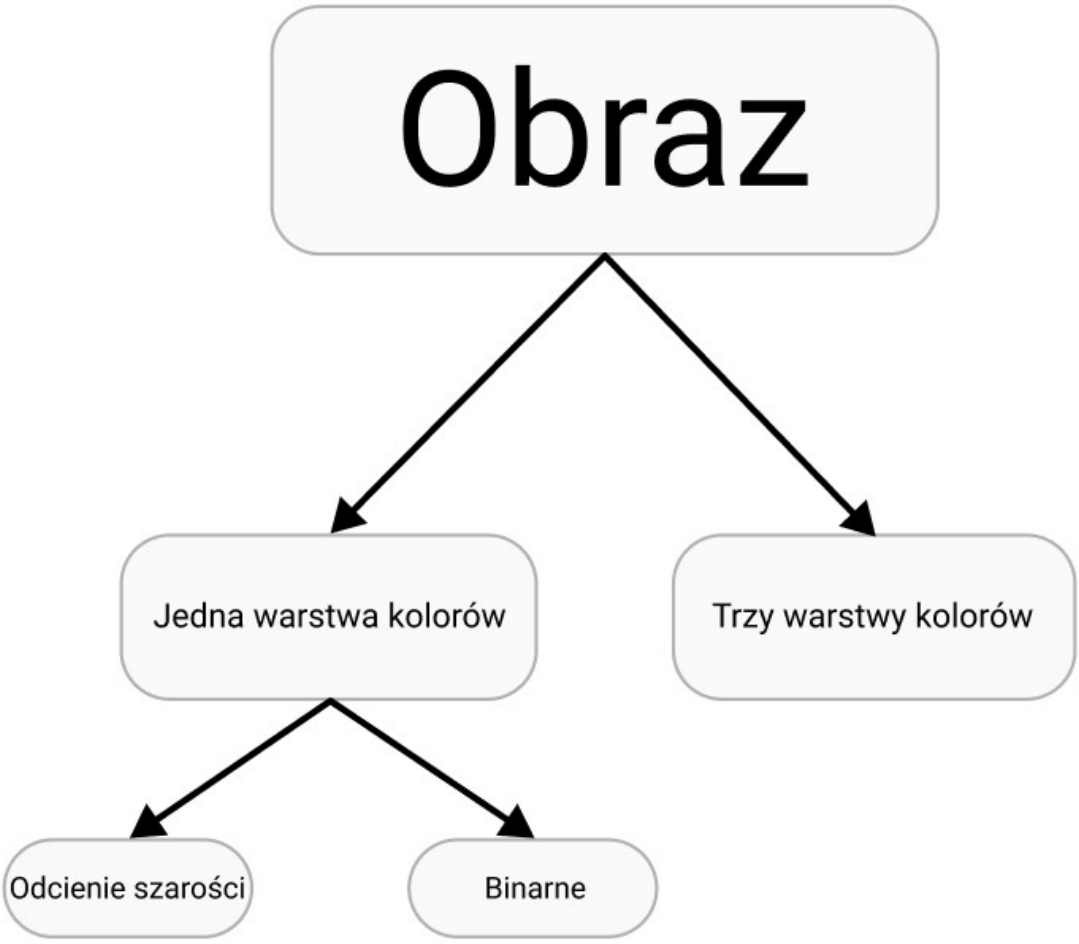


Figure 8: Klasyfikacja obrazów

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* bądź zwyczajnie *wektorami wielowymiarowymi*. Najprostrszym 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 wspopminanym 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 wilokolorowego musimy rozszerzyć istniejącą warstwę o dwie dodatkowe i wykonać na każdej z nich kolejno mapowanie na kolor czerowny (R), zielony (G) oraz niebieski (B). Jeżeli pojedyńcza wartość piksele może znajdować się w zakresie od 0 do 255 oraz uwzględnimy liczbę kanałów to otrzymujemy paletę barw zdolną do odwzorowania niemal *17 mln* kolorów. Ławtwo 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, możemy 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.

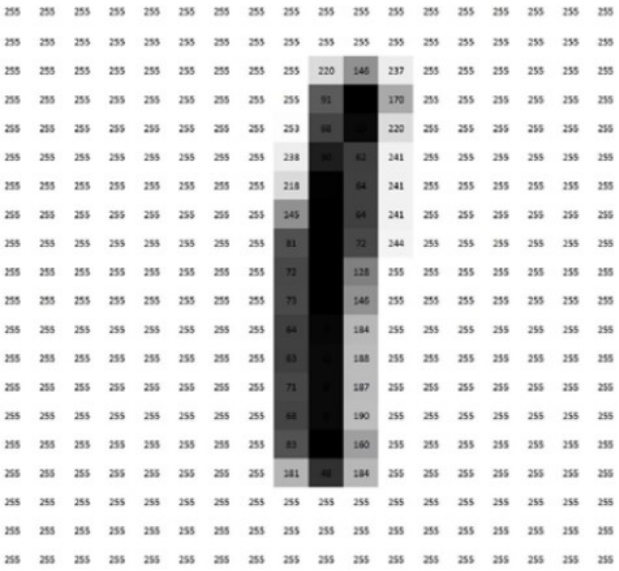


Figure 9: Obraz w różnych odcieniach szarości

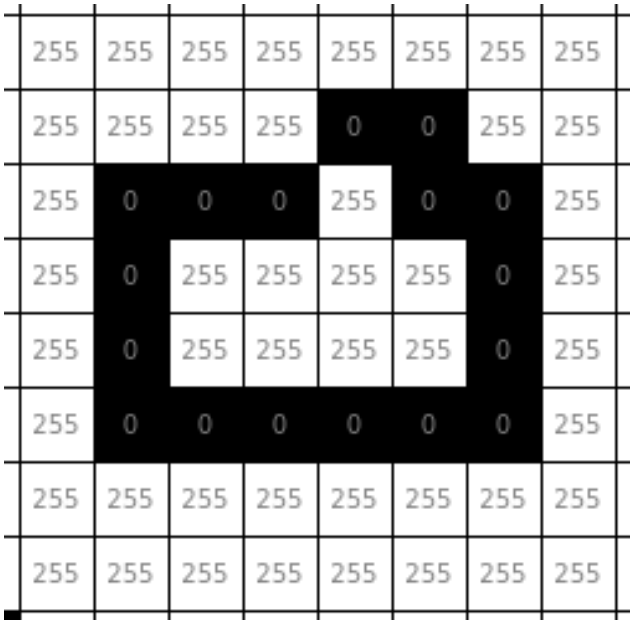


Figure 10: Zdjęcie w postaci binarnej

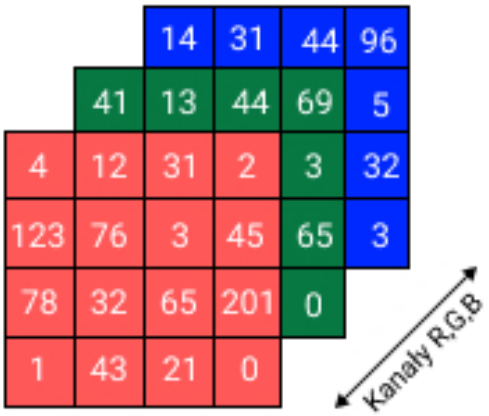


Figure 11: Sposób przedstawienia zdjęcia w formacie kolorowym

Przetwarzanie obrazu

Oczekiwany rezultat

Figure 12: Obiekt do przetworzenia

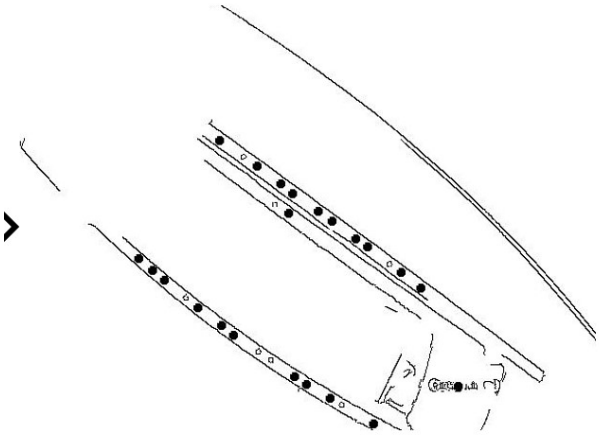


Figure 13: Efekt działania programu

Schemat blokowy toru przetwarzania

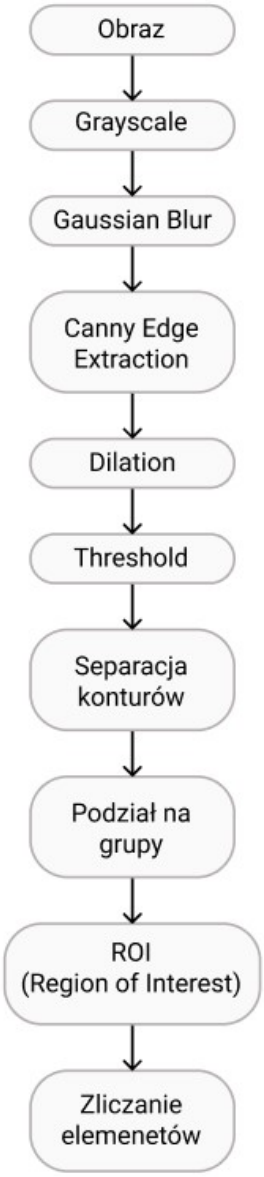


Figure 14: Schemat blokowy przetwarzania obrazu

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 zostany sposób konwersji, w tym przypadku z kanału kolorowego na odcienie szarości.

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

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

([1], n.d.)

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 ([2], n.d.).

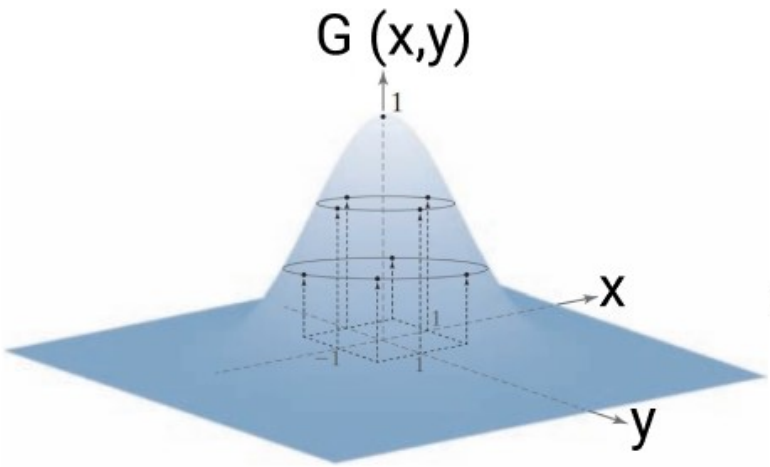
$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l) \tag{2}$$

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 zwizuwalizować przy pomocy okna zawierającego wartości współczynników. Możemy wyróżnić kilka roadzajów filtracji. Najprostszym podejściem jest utworzenie tablicy z wartości średnich w obrębie danego jądra (Richard Szeliski Springer 2010). Poniższa tożsamość ilustruje podstawową implementację.

$$K = \frac{1}{K_{\text{width}} \cdot K_{\text{height}}} \cdot \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ . & . & . & \dots & 1 \\ . & . & . & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \tag{3}$$

W przetwarzaniu obrazów najczęściej wykorzystujemy jednak giltrację w oparciu o okno Gaussa. Warto jednak w tym momencie wzrócić uwagę, że rozwiązanie nie jest najkorzystniejsze pod wzgledem 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.

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



$$\frac{1}{4.8976} \times \begin{array}{|c|c|c|} \hline 0.3679 & 0.6065 & 0.3679 \\ \hline 0.6065 & 1.0000 & 0.6065 \\ \hline 0.3679 & 0.6065 & 0.3679 \\ \hline \end{array}$$

Figure 15: Kształt funkcji gaussa dla dwóch zmiennych parametrów wraz ze współczynnikami dla wymiarów okna 3x3 (Rafael C. Gonzalez, n.d.)

Implementacja przy użyciu dostarczeniu do funkcji trzech argumentów. Pierwszym z nich jest obraz powstawy w wyniku poprzedniej operacji. Kolejno definujemy rozmiar jądra w postaci listy statycznej. Opcojalnym 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)
```

Canny Edge Detection

Canny Edge Detection to jest z najbardziej rozpowszechnionych algorytmów służących do wyseparowania krawędzi z obrazu. Rozwiązanie zaproponował australijski infomatyk John F. Canny w roku 1986. 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 wdluż osi x oraz y. W tym celu wykorzystywany jest algorytm Sobela. Matemetycznie 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 zależnosci od wyboru poziomu wyzwalania otrzymujemy maksymalną oraz minimalną wartość pochodnej bo wskazuje, że wzdłuż badanego obszaru znajduje się obiekt, który chcieliśmy wykryć.

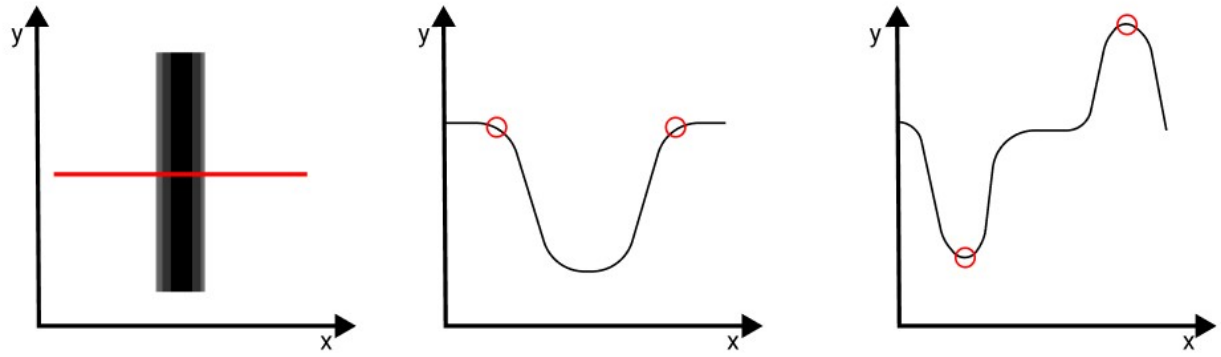


Figure 16: Obliczenia gradientowe

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. Dodatkowo potrzeba określić dwa parametry zastosowanej funkcji, pierwszym z nich jest wejściowy wektor macierzy w postaci binarnej natomiast drugim rozmir 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 porządanym. Poniższe równanie opisuje zastosowaną zależność.

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

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 równieżliczbę iteracji.

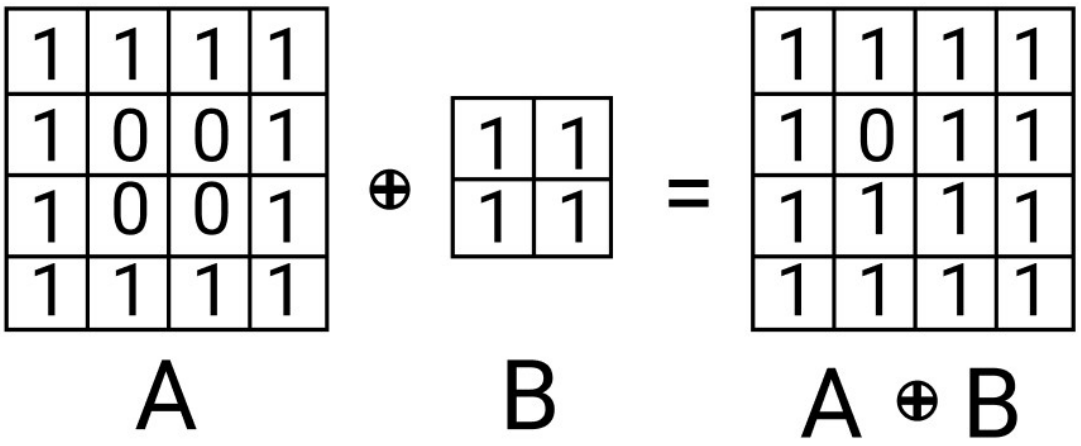


Figure 17: Przykładowy przebieg procesu

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

Threshold

Kolejnym etapem jest **Thresholding**, który polega na ustaleniu wartości granicznych dla których zdjęcie powinno zostać wyprogowane. Proces ten można nazwać również binearyzacją. Przetwarzanie polega na aplikacji dla kazdego 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 róznych 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 ostalają parametry progrowania. 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 naważniejsze z nich w kontekście projektu to *BINARY* oraz *BINARY_INV*.

$$\text{dst}(x,y) = \begin{cases} 0 & \text{if } \text{src}(x,y) > \text{ thresh} \\ \text{maxval} & \text{otherwise} \end{cases} \tag{8}$$

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

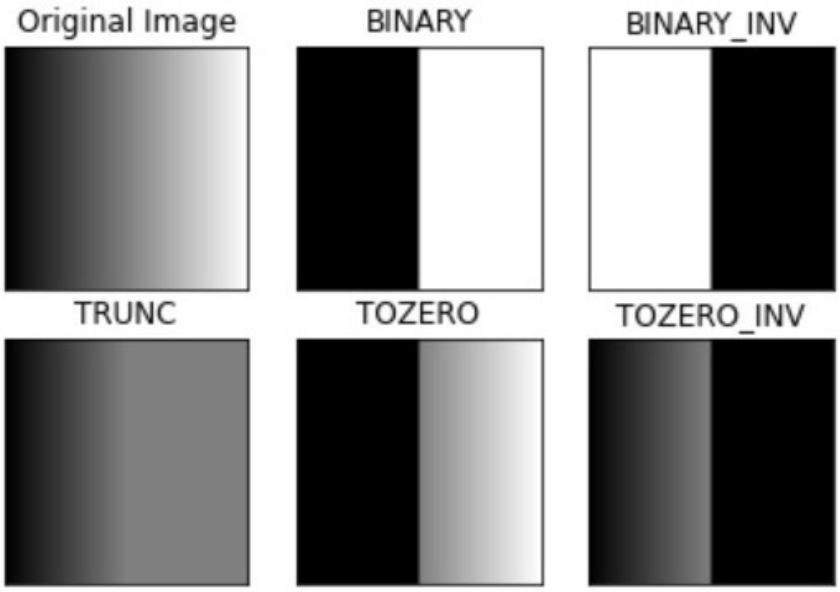


Figure 18: Podstawowe algorytmy progowania

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 algortym *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 kontrur znajduje się we wnętrzu zewnętrznego oraz jego powierzhnia jest mniejsca od zdefiniowanej wartości granicznej następuje graficzna wizualizacja procesu w oparciu o metodę *cv2.drawContours()*. 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. Koncowym etapem jest zwrócenie tych wartości w postacji posortowanej listy. Poniższa grafika prezentuje w uproszczony sposób działanie algorytmu. Środki ciężkości zostały przedstawione w postaci czerwonych punktów.

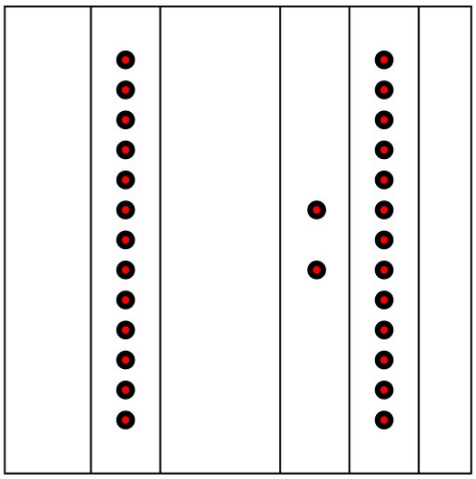


Figure 19: Wysyperowanie konturów oraz wyznaczenie środków ciężkości

```
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)
```

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 group 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 produkcynjej. Poniższe grafiki przedstawiają opisany proces zarówno w postaci uproszczonej jak i rzeczywistej.

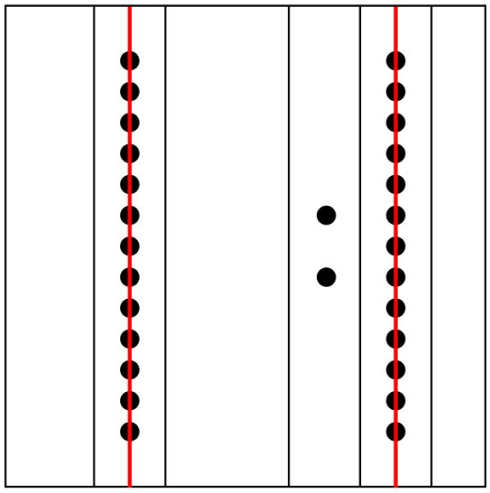


Figure 20: Wyznaczenie grup w postaci uproszczonej

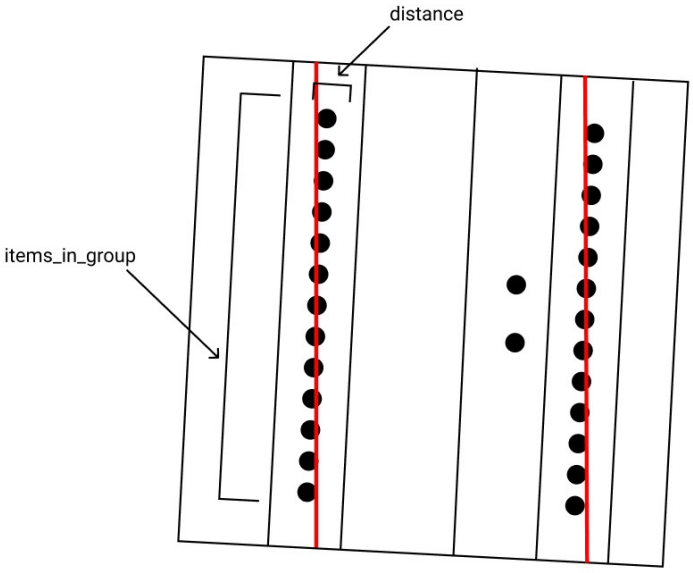


Figure 21: Wyznaczenie grup w postaci rzeczywistej

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
```

Zliczanie elementów

Ostatnim etapem jest zwrócenie liczby otworów w postaci liczbowej. Dysponując grupami z poprzedniej opracji, które przedstawione są w postai stałej wartości rzędnej na obrazie zdefionowany 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 pojedyńczej klatce obrazu. Dla tak wyznaczonego obszaru (kolor niebieski na poniższej grafice) ponowiony został proces wyseparowania krawędzi znany z poprzenich punktów przwtwarzania. Poniżej znaduje sie również fragment kodu realizujący opisane operacje.

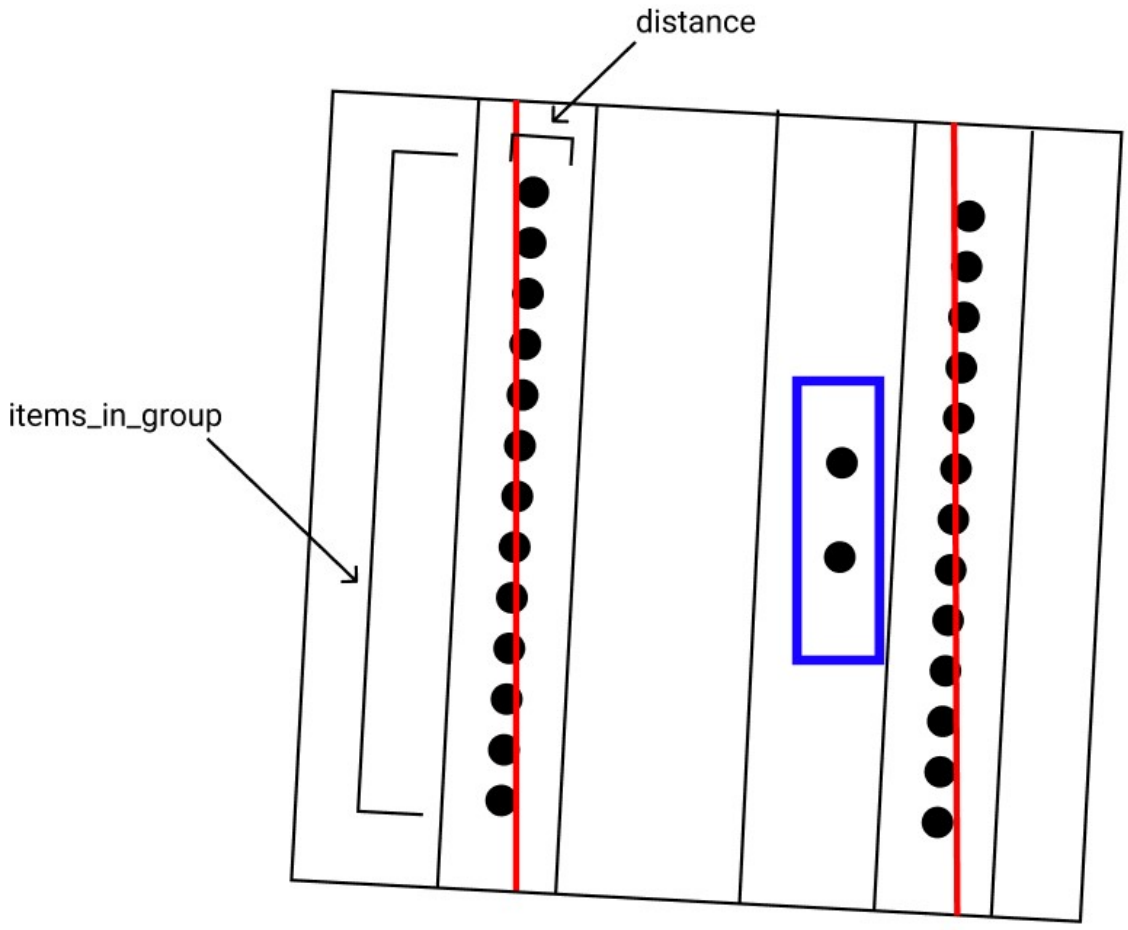


Figure 22: Zdefiniowanie ROI oraz liczanie elementów

```
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!')
```

Wykorzystanie sieci neuronowych

Cel wprowadzenia 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 dokładość 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 konsewwecnji program może zwrócić błąd, co należy uwzględnić w postaci zwrócenia odpowiedniego wyjątku obsługującego taką sytuację. Mając na uwadzę opisany parametr można w łatwy sposób dojść do wniosku, że program odwieża swoje parametry wyjściowe z częstotliwością, która jest w przybliżeniu równa (opóźniona o czas przetwarzania) częsotliwości odświeżania obarazu. 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 postacji liczbowej. Można to porównać do układów cyfrowych cechujących 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.

Informacje wstęne

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**.

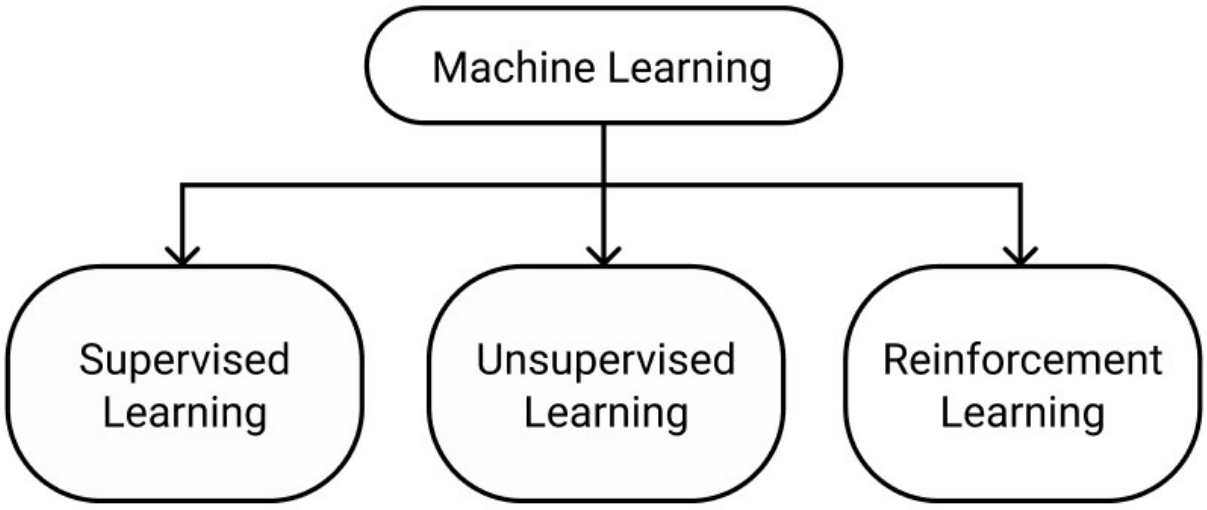


Figure 23: Podział algorytmów uczenia maszynowego

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).

(Soni TowardsDataScience 2018)

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.

(Buds Medium 2020)

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 użytniwnika 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 inteligentych 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 nefralgiczne 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 dopracone, 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.

Budowa neuronu

Początki uczenia głębokiego (*Deep learning*) można datować na lata czterdzieste ubiegłego wieku. Było to możliwe w oparciu o formułę matematycznej implementacji sztuczego neuronu, który z kolei powstał jako inpiracja naturą a w szczególności odkryciu sposobu w jaki nasz mózg przekazuje informacje do reszty organów. Dane w postaci liczbowej reprezentujące wartości wag połączen między poszczególnymi elementami pochodzą z gałęzi zwanych dentrytami, 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 pochodzacej 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 sie linową zależnością, gdy wartość wejściowa jest dodatnia, w przciwnym wypadku na wyjściu układu otrzymamy zero. Ze względu na łatwą implementację oraz oraz wysoką wydajność jest one najczęsciej stosowana w praktycznej implementacji. Możemy wyróżnić również funckję w postaci **sigmoid** oraz **tanh**, których matematyczne implementacje oraz kształty zostały przedstawione poniżej.

$$y = \max\left(0, w^T x + b\right) \tag{9}$$

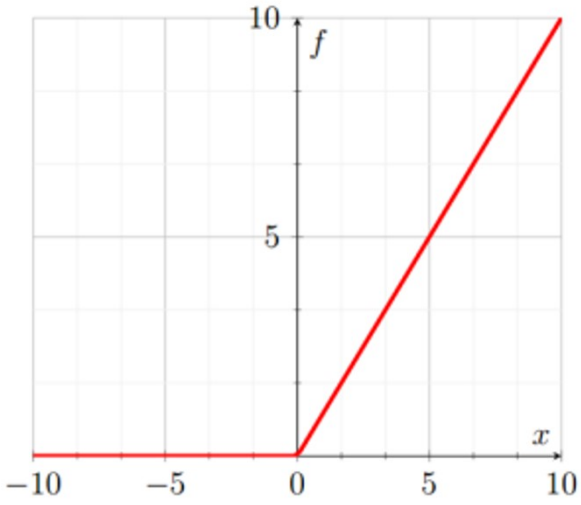


Figure 24: ReLU

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{10}$$

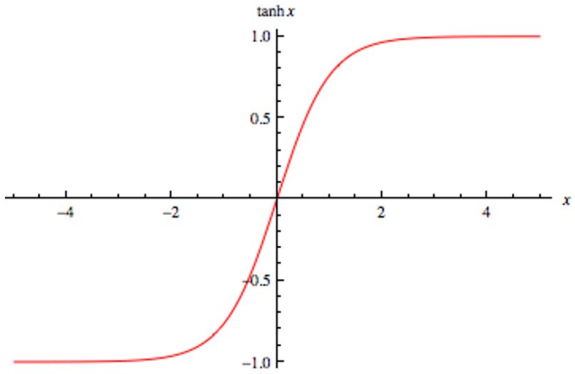


Figure 25: tanh

$$f(z) = \frac{1}{1 + \exp(-z)} \tag{11}$$

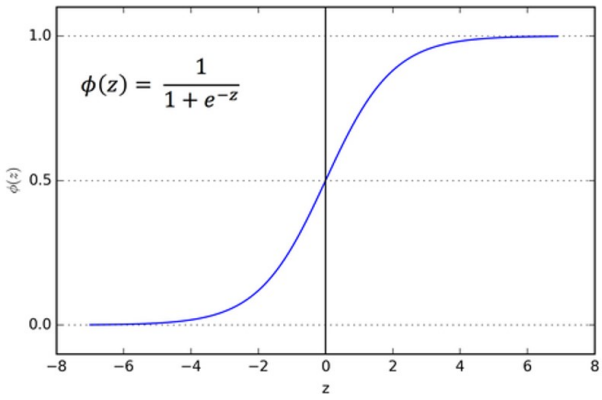


Figure 26: sigmoid

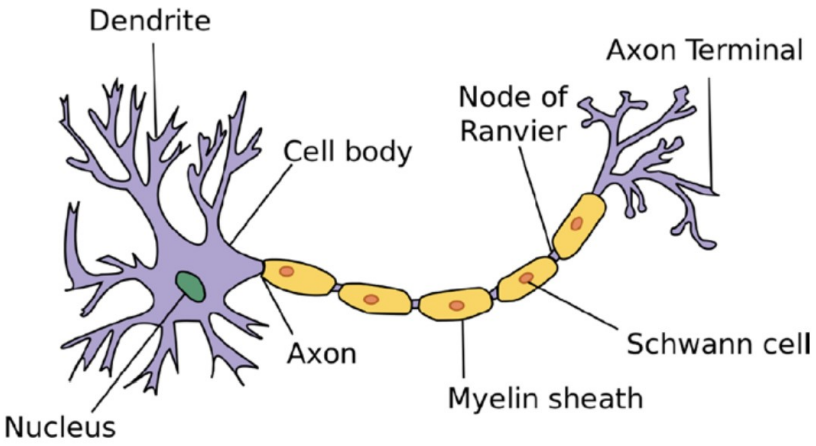


Figure 27: Budowa neuronu inspirowana biologią

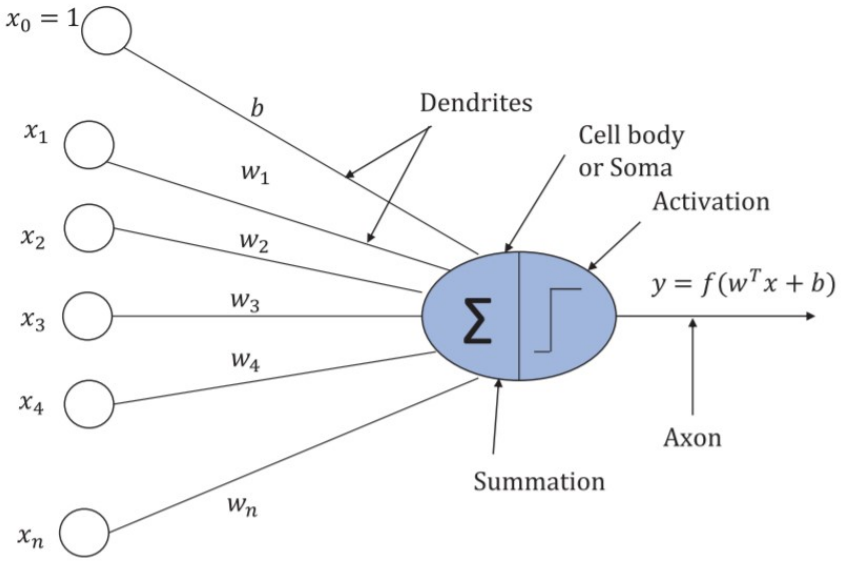


Figure 28: Implementacja neuronu w praktyce

(Santanu Pattanayak Apress 2017)

Określenie dokładności otrzymanego modelu

Tworzenie sieci neuronowej

Schemat

Bibliografia

[1]. n.d. "[1] Image Color Conversions." https://docs.opencv.org/3.4.15/de/d25/imgproc_color_conversions.html.

[2]. n.d. "[2] Tutorial_gausian_median_blur_bilateral_filter." https://docs.opencv.org/4.5.3/dc/dd3/.html.

Buds, Tech. Medium 2020. "Reinforcement." https://medium.com/@techbuds20/supervised-unsupervised-and-reinforcement-learning-58b6b7e6b0dc.

Rafael C. Gonzalez, Richard E.Woods [3]. n.d. "[3] Digital Image Processing 4th."

Richard Szeliski, [4]. Springer 2010. "[4] Computer Vision: Algorithms and Applications."

Santanu Pattanayak, [5]. Apress 2017. "[5] Pro Deep Learning with TensorFlow - a Mathematical Approach to Advanced Artificial Intelligence in Python."

Soni, Devin. TowardsDataScience 2018. "Supervised Vs. Unsupervised Learning." https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d.