poniedziałek,
3 czerwca
2024

Detekcja dłoni i rozpoznanie wskazanej liczby.

ZASTOSOWANIE UCZENIA MASZYNOWEGO I GŁĘBOKIEGO.

IZABELA PANICZEK (272410)

1. Wprowadzenie

Możemy zauważyć, że na przestrzeni ostatnich lat wzrasta zainteresowanie modelami uczenia maszynowego i głębokiego. Obserwujemy coraz szersze zastosowanie – jednym z nich jest widzenie komputerowe. Obejmuje to rozpoznawanie obiektów, wykrywanie ruchu, segmentację obrazów, analizę scen oraz inne zadania związane z analizą danych wizualnych. Jednym z zastosowań tej technologii jest detekcja ręki i rozpoznawanie pokazywanych przez nią liczb. Tego rodzaju systemy mogą znaleźć szerokie zastosowanie w wielu dziedzinach, takich jak interfejsy użytkownika, gry komputerowe.

Celem projektu jest stworzenie modelu, który będzie wykrywał dłoń, a następnie rozpoznawał wskazaną liczbę zarówno na zdjęciach, jak i w czasie, korzystając z kamery internetowej. Chcemy porównać dwa podejścia, w jednym zamierzamy korzystać z obrazu ręki oraz uczenie głębokie, za to w drugim chcemy dobrać cechy dotyczące pokazywanej ręki – współrzędne odpowiednich punktów palca, kąty pomiędzy palcami i zastosować modele uczenia maszynowego.

Spodziewamy się, że inne podejście będziemy stosować dla modeli uczenia maszynowego i głębokiego. Do uczenia maszynowego ważnym elementem, będzie dobranie odpowiednich cech. W ich wyborze pomoże nam badanie korelacji pomiędzy klasą wskazywanej liczby a daną cechą. W uczeniu głębokim proces nauki będzie dłuższy, model będzie przetwarzał dużą ilość dwuwymiarowych lub trójwymiarowych danych. Dlatego zastosujemy sieć konwolucyjną.

Przewidujemy, że badanie zakończy się sukcesem, gdyż mamy sporą ilość zróżnicowanych danych – zdjęcia mają różne oświetlenie, dłonie, tła. Ważnym elementem dla modelu uczenia maszynowego będzie dobranie odpowiednich cech, za to dla uczenia głębokiego będziemy musieli odpowiednio zadbać o reprezentację obrazu oraz architekturę sieci neuronowej. Dodatkowo wiemy, że wyzwaniem może okazać się dostosowanie modelu do działania w czasie rzeczywistym. Predykowanie wyników może trwać zbyt długo. Dlatego ważne będzie, aby zbalansować stopień wyuczenia modelu oraz jego prostotę.

W trakcie projektu mamy styczność z różnymi metodami przetwarzania obrazów, dowiadujemy się jak najlepiej wydobyć cechy. Porównujemy działanie różnych technik do wykrywania obiektów - wzmacniając kontrast, wykrywanie tła czy analizę gradientów intensywności pikseli. Dodatkowo poznajemy różne biblioteki, które wspomagają detekcję obiektów a także wykorzystują różne modele uczenia głębokiego oraz techniki wykorzystujące mapy cech. Dodatkowo samodzielnie będziemy dobierać cechy dotyczące gestów, co pozwoli zgłębić analizę danych, dobór cech i zmniejszanie złożoności modelu.

2. Opis przeprowadzonych badań

a) Główne narzędzie:

   Projekt realizowaliśmy w środowisku Jupyter Notebook, które umożliwiło nam interaktywne eksperymentowanie z kodem oraz wizualizację wyników w przejrzysty sposób. Jupyter Notebook, w połączeniu z językiem Python, pozwolił na łatwe testowanie różnych konfiguracji modelu oraz prezentowanie wyników w formie graficznej. Korzystamy z 3.9 wersji Pythona. Warto zaznaczyć, że biblioteka mediapipe działa na wersjach pythona 3.8 do 3.10.

b) Akwizycja danych:

   Jednym z pierwszych kroków było pobranie zdjęć i zapisanie ich w formie macierzy, razem z odpowiednią etykietą w tabeli typu DataFrame. Musimy sczytać zdjęcia z folderu, a także zapisywać podaną klasę w pliku csv. Użyłam do tego zadania bibliotekę os oraz open-cv. Łatwe

odczytywanie plików typu csv, zapisywanie bieżących danych oraz zarządzanie przetwarzanymi danymi umożliwiło mi użycie biblioteki pandas.

c) Przetwarzanie obrazów:

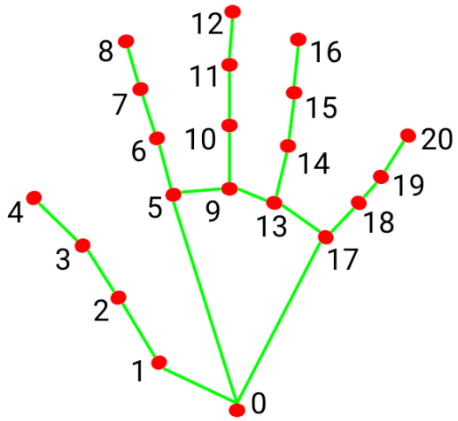
W trakcie badań porównywałam różne sposoby przetwarzania obrazów, aby wydobyć cechy i porównać działanie modelu. Zdecydowałam się na użycie skali szarości, Laplacjana Gaussa, i histogram zorientowanych gradientów.

- Użycie skali szarości umożliwia redukcję złożoności modelu: Obrazy w skali szarości mają tylko jeden kanał w porównaniu do trzech kanałów (RGB) w kolorowych obrazach. To znacząco redukuje ilość danych, które muszą być przetworzone przez model. Jednocześnie zapewniając wystarczającą ilość informacji do skutecznej analizy w wielu zadaniach przetwarzania obrazów.
- Laplacjan Gaussa wykorzystujemy do wygładzanie obrazu za pomocą filtru Gaussa oraz zastosowanie filtru Laplacjana w celu podkreślenia regionów o szybkiej zmianie intensywności, czyli krawędzi. Korzystałam z biblioteki scipy z funkcji: ndimage.gaussian_laplace
- Histogram zorientowanych gradientów koncentruje się na strukturalnych właściwościach obrazu poprzez analizę gradientów intensywności pikseli, co pozwala na skuteczne rozpoznawanie kształtów i konturów. Korzystam z biblioteki scikit-image funkcji hog

d) Uczenie głębokie:

✓ Rozpoznanie ręki:

1. Aby możliwe było rozpoznanie wskazanej liczby musimy wykryć rękę na zdjęciu. Zastosowaliśmy bibliotekę MediaPipe, która ma funkcje stworzone do detekcji rąk. Model śledzenia rąk jest konwolucyjną siecią neuronową (CNN), która jest odpowiedzialna za wykrywanie dłoni w obrazie wejściowym. Jeśli model wykryje dłoń, wyznacza 21 trójwymiarowych punktów charakterystycznych (landmarków) na dłoni. Korzystamy z już wcześniej wytrenowanego modelu. Na podstawie tej detekcji moglismy dokonać wycięcia wykrytej dłoni ze zdjęcia w formie prostokąta – przy pomocy najbardziej wysuniętych punktów wyznaczamy prostokąt



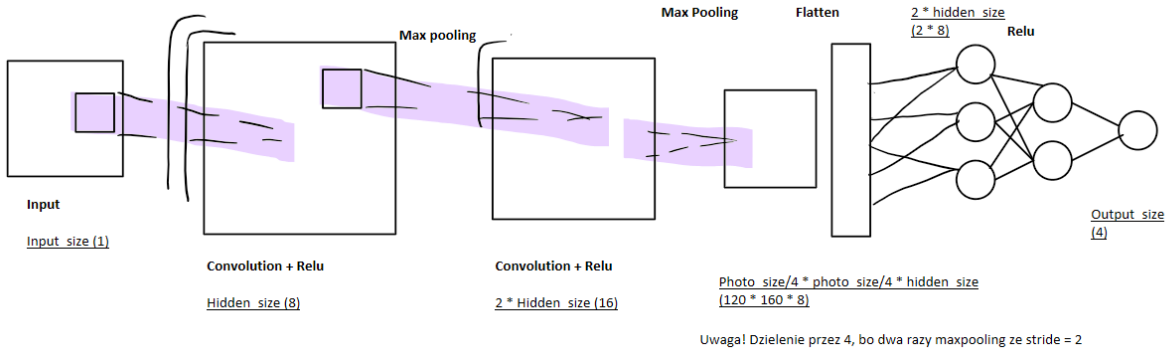
0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Zdjęcie 1 - Punkty wyszukiwane na ręce przez bibliotekę mediapipe

2. Tworzyliśmy także własną funkcję, która przewidywaliśmy jak najlepiej przyciąć zdjęcie. Stworzyliśmy model CNN, korzystaliśmy z labeling studio, które

umożliwiło nam przygotowanie własnego zbioru danych, który miał poprawnie dobrane prostokąty ograniczające wykryty obraz od tła.

a. Architektura sieci neuronowej:



Zdjęcie 2 - schemat prostej sieci konwolucyjnej, znajdywanie prostokąta

✓ Dostosowanie zdjęcia do sieci neuronowej

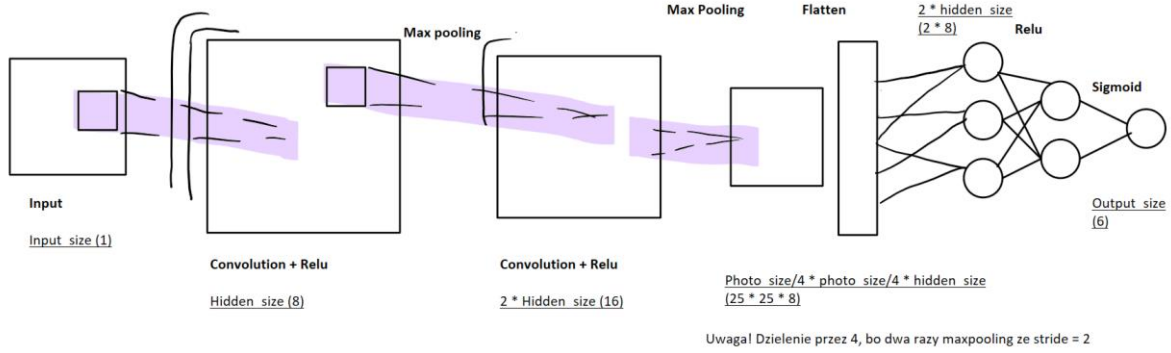
Trzeba pamiętać, że na wejściu sieci potrzebne są te same wymiary – jednak po wycięciu ręki ze zdjęcia, macierze reprezentujące je mają różne wymiary. Dlatego dostosowywaliśmy je do kwadratu (przy pomocy resize_with_crop_or_pad z biblioteki tensorflow), a potem do rozmiar samego zdjęcia do 100 na 100 (przy pomocy thumbnail z biblioteki PIL). To umożliwi zoptymalizowanie czasu nauki i predykowania.

✓ Dataloader

Dane dzielimy na treningowe i walidacyjne, korzystając z biblioteki sklearn. Te dane zmieniamy na tensory, opisując w nim odpowiednie typy danych i tworzymy DataLoadera. Jest to obiekt umożliwiający tasowanie danych, tworzenie batch'y, które przyśpieszają działanie sieci neuronowej. Jest to ważne szczególnie w bardziej wymagających modelach. Do tego zadania wykorzystaliśmy wbudowane funkcje pytorch'a.

✓ Tworzenie modeli

1. Prosta sieć konwolucyjna. Została stworzona przy pomocy biblioteki pytorch. Jej architektura prezentuje się w następujący sposób:



Zdjęcie 3 - schemat prostej sieci neuronowej - rozpoznawanie wskazanej liczby

Opis poszczególnych warstw:

- Warstwa konwolucyjna - zastosowaliśmy dwuwymiarowe filtry kernela, o przesunięciu równym 1 oraz nie dodalismy obramowań. Dzięki przesuwaniu się filtru po obrazie wejściowym uzyskaliśmy mapę cech.
- ReLu – funkcja aktywacyjna, która wartości ujemne skaluje na zero. Dzięki niej nie mamy wartości niedodatnich. Pomaga to we wprowadzeniu nieliniowości do sieci neuronowych – rozwiązuje to problem zanikającego gradientu, co jest problemem szczególnie w rozbudowanej architekturze sieci, pojawia się przy propagacji wstecznej przez kolejne warstwy.
- MaxPool – zmniejsza wymiar mapy cech (przedstawiliśmy to na schemacie)
- Spłaszczanie – Mapa cech jest spłaszczana do wektora jednowymiarowego.
- Warstwa liniowa – przekształcenie wektora do mniejszego wymiaru
- Sigmoida – funkcja, która przekształca wartości na liczby z przedziału 0, 1. Jest to pomocne, gdyż wynik końcowy wybieram poprzez znalezienie maksymalnego prawdopodobieństwa wystąpienia liczby

2. Bardziej rozbudowana sieć neuronowa. Została stworzona przy pomocy pytorcha. Jednak zastosowaliśmy większą ilość warstw i dwa nowe ich typy.
   - Batch Normalization – oblicza średnią i odchylenie, aby standaryzować każdy kanał, bazując na aktywacji poprzedniej warstwy. Również wpływa pozytywnie na zmniejszanie problemu ze znikającym gradientem.
   - Dropout – pomaga w regularyzacji sieci neuronowe, zapobiega nadmiernemu dopasowaniu się modelu do danych poprzez wyłączenie danych neuronów.

e) Uczenie maszynowe:

✓ Ekstrakcja cech

Zdecydowaliśmy się na ekstrakcję cech z podanego zdjęcia. Korzystaliśmy z biblioteki mediapipe, z wcześniej wspomnianego wytrenowanego modelu do detekcji dłoni (zdjęcie 1).

Na podstawie uzyskanych punktów na dłoni wyznaczamy kąt pomiędzy palcami. Obliczamy odpowiednio wektory uzyskanych punktów, iloczyn skalarny, a z niego wyznaczamy kąt.

Dodatkowo wyznaczamy również skalowaną odległość między palcami, rzeczywistą odległość dzielę przez długość dwóch palców. Wtedy niezależnie od wielkości dłoni mamy porównywalne wyniki.

Ostatnią cechą jest ułożenie palców, powinno dać mi możliwość ocenienia czy palec jest zgięty. Mierzę odległość pomiędzy czubkiem kciuka (punkt 4), a punktem 17 oraz reszty palców od punktu 0. Każdą z takich odległości dzielimy przez długość badanego palca, co umożliwi pewną standaryzację, niezależność od wielkości ukazanej na zdjęciu dłoni.

✓ Przetworzenie danych

Zdecydowaliśmy się na przetworzenie tych danych przy pomocy normalizacji stosując drugą normę. W nim każdy wektor (w naszym przypadku wiersz) był przeskalowany i jego norma L2 była równa 1. Powiązaliśmy ze sobą kąty, odległości oraz położenie palców, gdyż te cechy od siebie zależą.

✓ Wybór modelu

Wyboru modelu dokonaliśmy bazując na wykorzystaniu funkcji Grid Search, który porównuje różne modele (SVM, LDA i Decision Tree), z różnymi parametrami, bazując na dokładności. Grid Search jest metodą optymalizacji hiperparametrów, która maksymalizuje wydajność modelu na zbiorze walidacyjnym przeszukując zdefiniowaną siatkę. Korzystaliśmy z biblioteki sklearn.

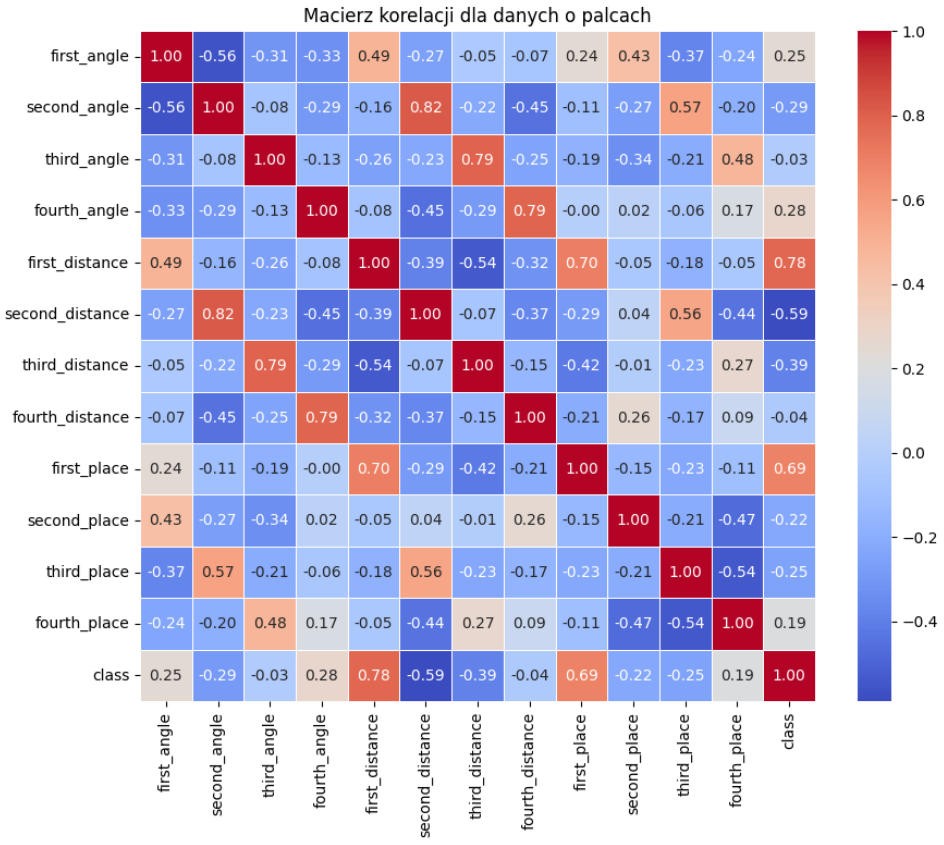
✓ Działanie w czasie rzeczywistym

Zapisany model przy pomocy biblioteki joblib wczytuje się i predykuje w czasie rzeczywistym. Wyświetlamy kamerkę internetową z zapisaną informacją jaką liczbę predykuje model. Do wyświetlania wykorzystujemy bibliotekę pygame. Pobrane dane przez kamerkę z biblioteki OpenCV przetwarzamy odpowiednio, a następnie wyświetlamy sugerowaną odpowiedź. Z trzech klatek wybieramy, która klasa liczby pojawiła się częściej.

3. Wyniki

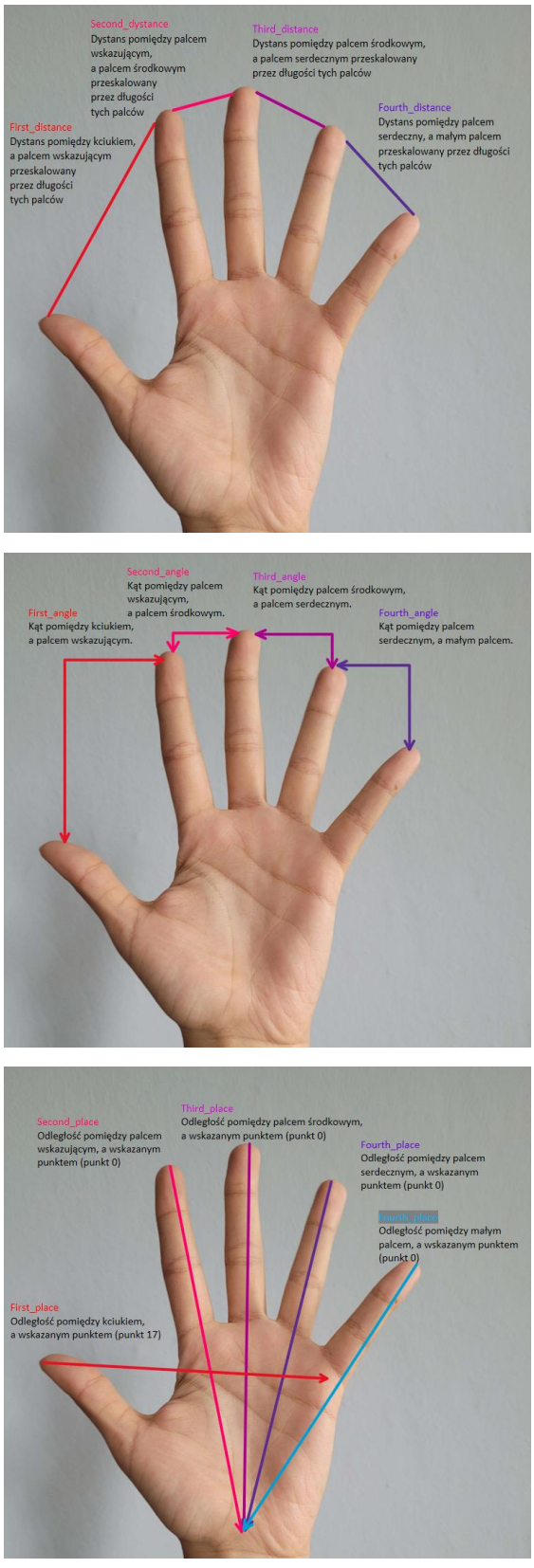
- Uczenie maszynowe:

Macierz korelacji między cechami:



Zdjęcie 4 - macierz korelacji cech

Legenda:



Zdjęcie 4.1 - legenda opisująca zmienne przedstawiona na macierzy korelacji

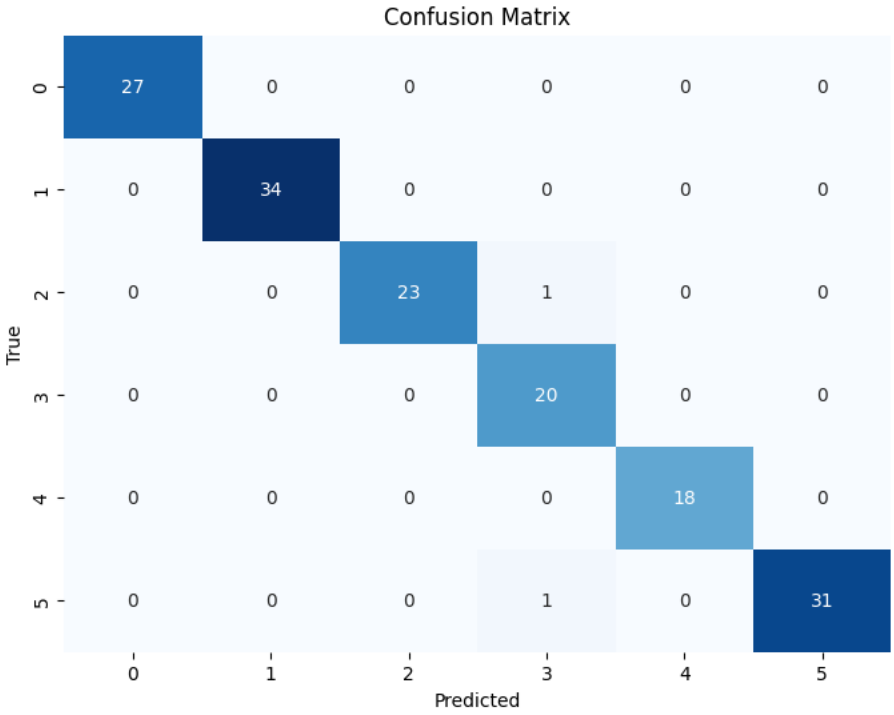
Wyniki algorytmu Grid Search na danych treningowych i walidacyjnych z wykorzystaniem walidacji krzyżowej:

Tabela 1 - porównanie wyników uzyskanych w Grid Search

Nazwa modelu	Parametry	Accuracy
SVM	C: 10, gamma: scale, kernel: poly	0.99
LDA	Solver: svd	0.96
DT	Criterion: gini, max_depth = None	0.98

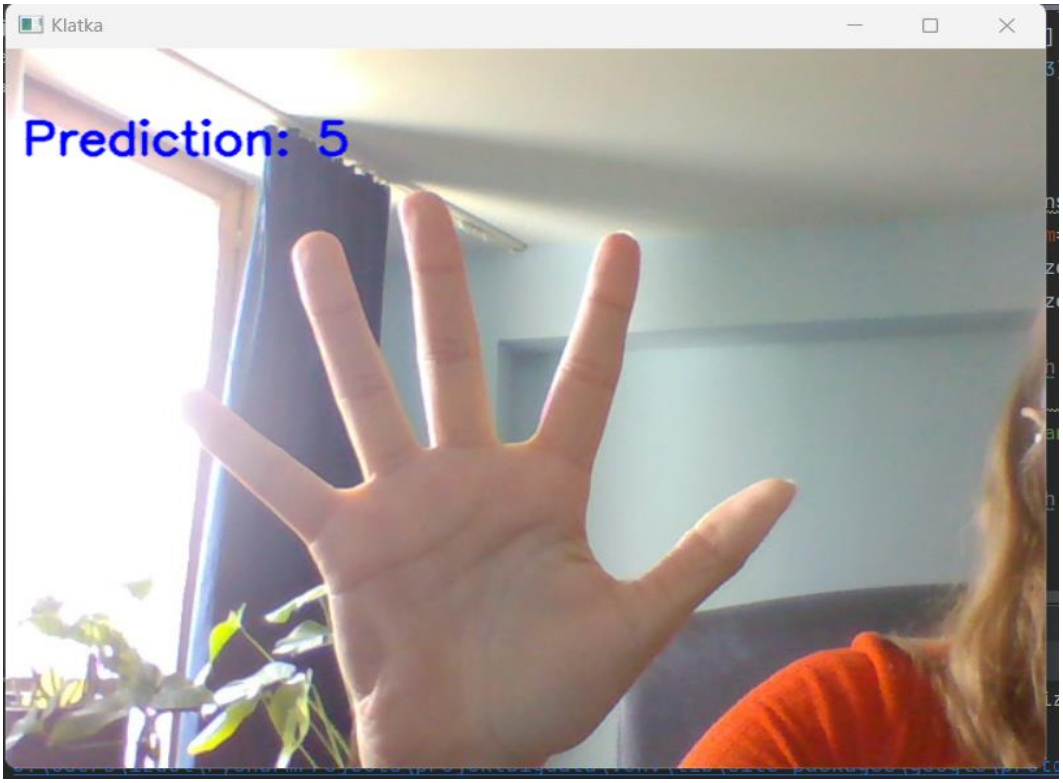
Wyniki algorytmu SVM na danych testowych:

F1 = 0.99



Zdjęcie 5 - tablicę pomyłek dla działania SVM na danych testowych

Wizualizacja działania algorytmu w czasie rzeczywistym:

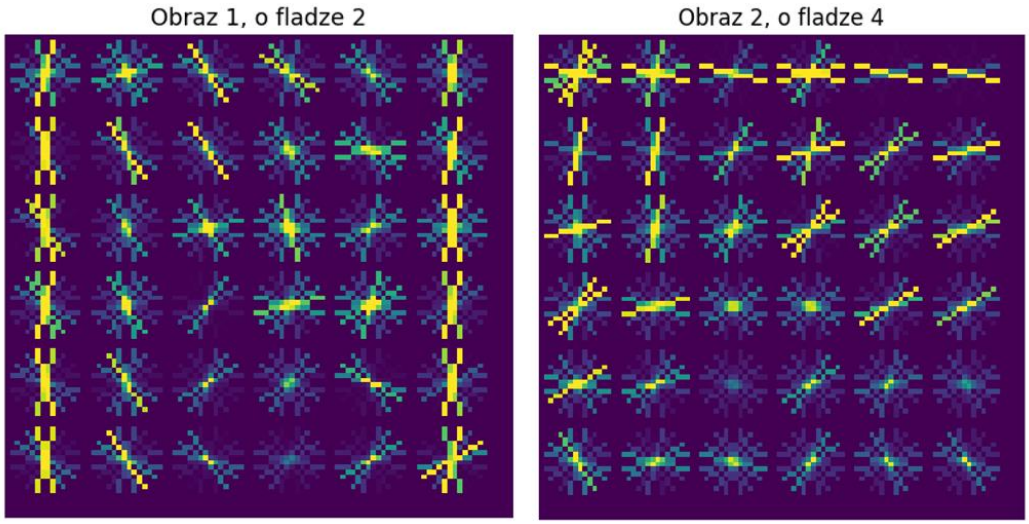


Zdjęcie 6 - zrzut ekranu z działania modelu w czasie rzeczywistym

- Uczenie głębokie:

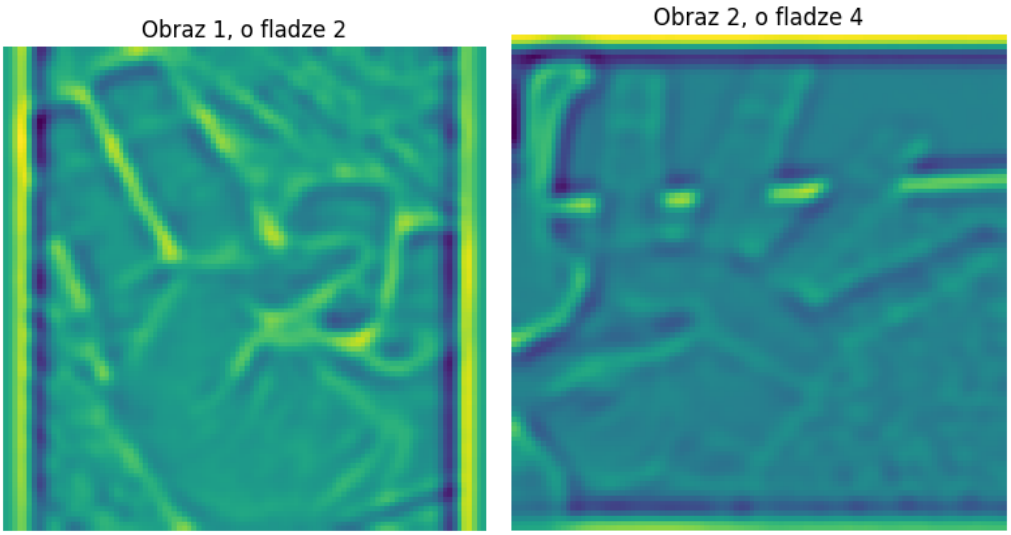
Przetworzone zdjęcia:

Filtr HoG:



Zdjęcie 7 - obrazy w kwadracie, rozmiarze 100x100 i z nałożonym filtrze HoG

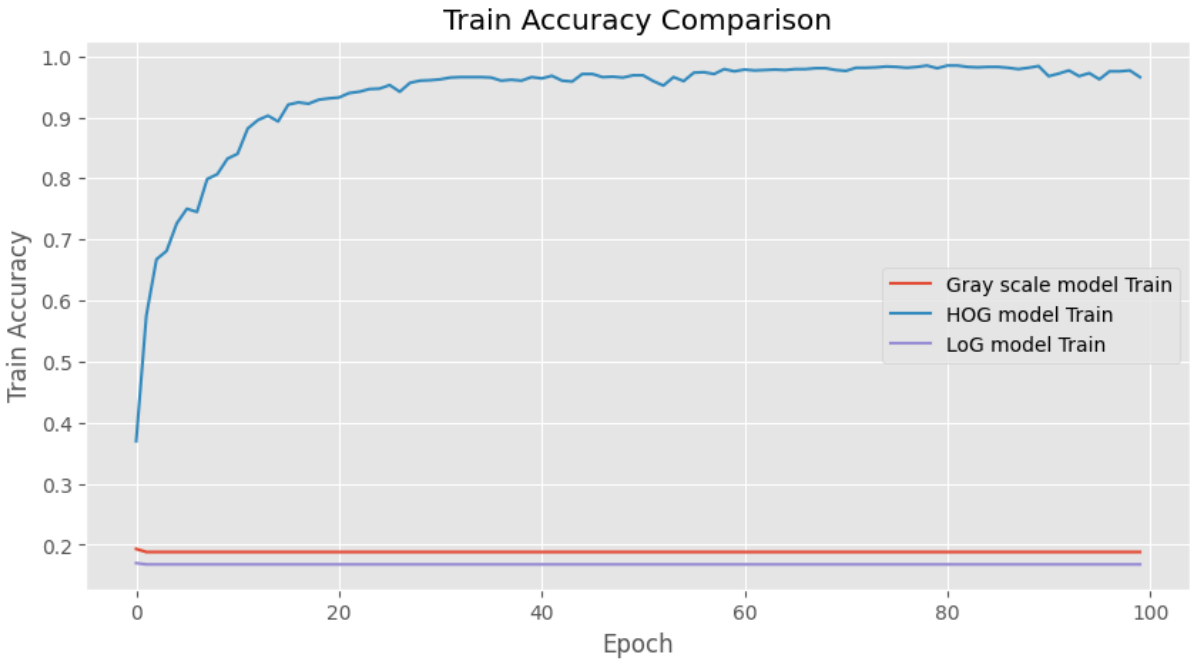
Filtr LoG:



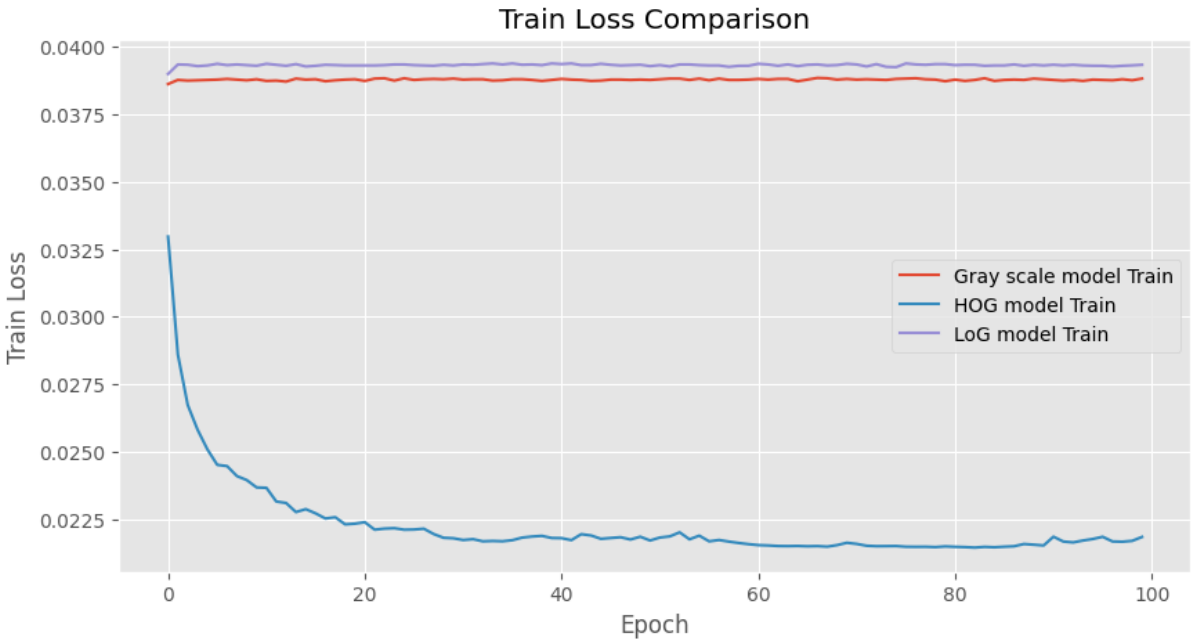
Zdjęcie 8 - obrazy w kwadracie, rozmiarze 100x100 i z nałożonym filtrze LoG

Wykresy przedstawiające proces nauki w sieci neuronowej:

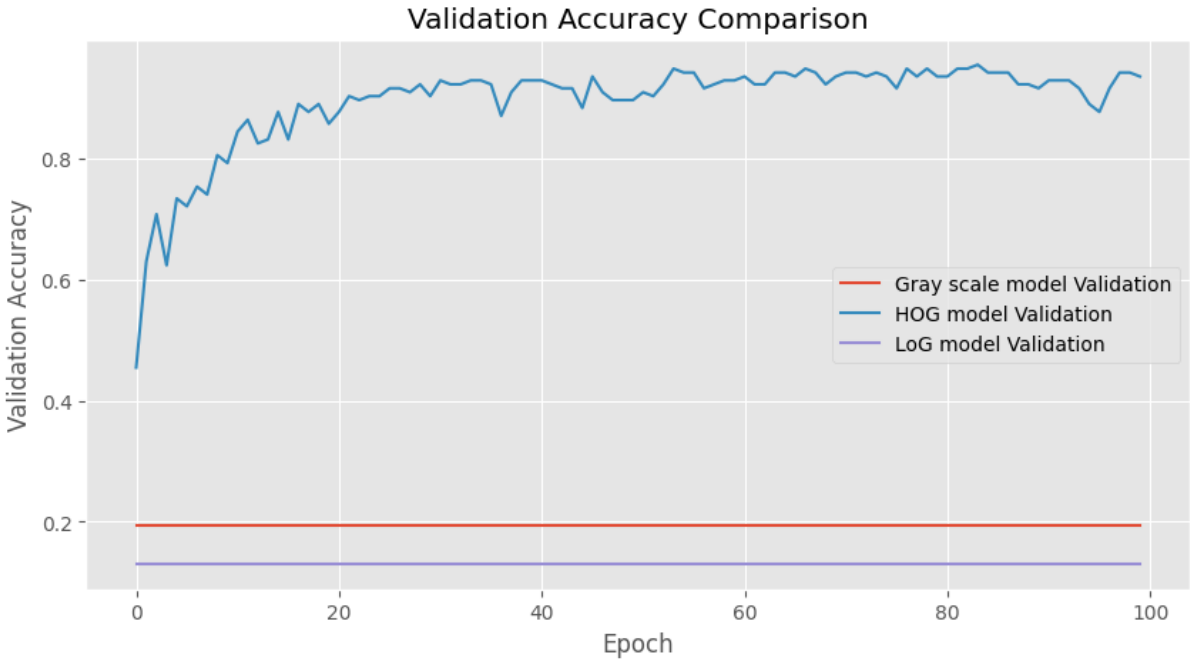
1. Prosta sieć konwolucyjna:



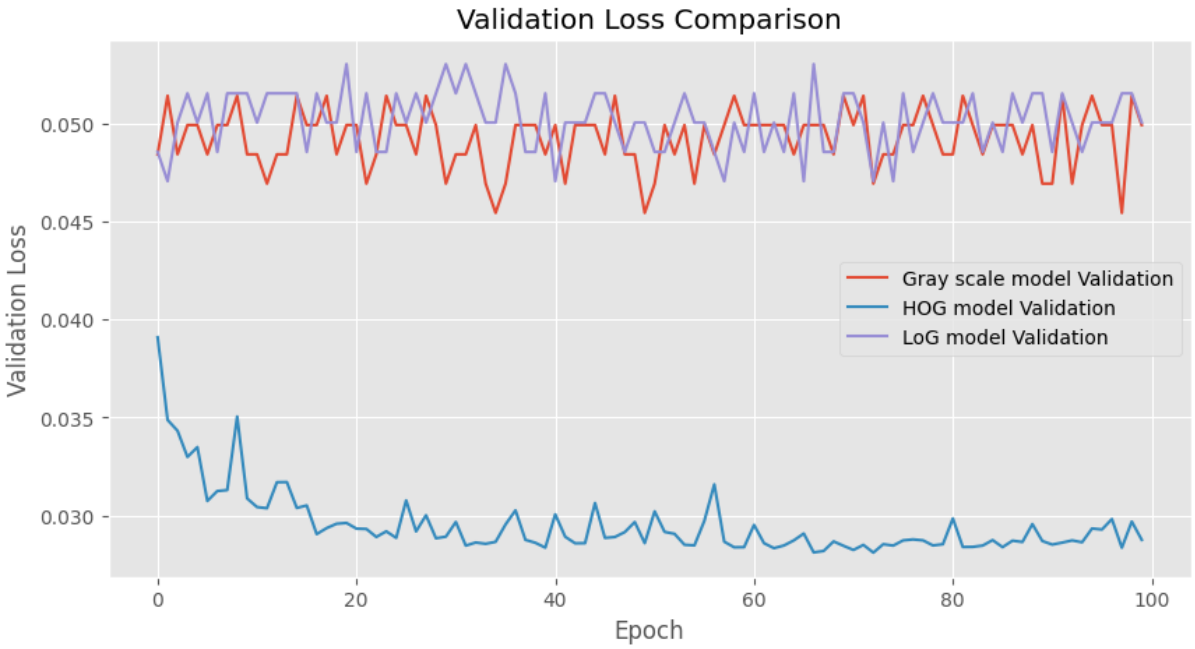
Wykres 1 - wykres dokładności danych testowych modelów w trakcie epok, dla prostszej sieci konwolucyjnej



Wykres 2 - wykres funkcji straty danych testowych modelów w trakcie epok, dla prostszej sieci konwolucyjnej

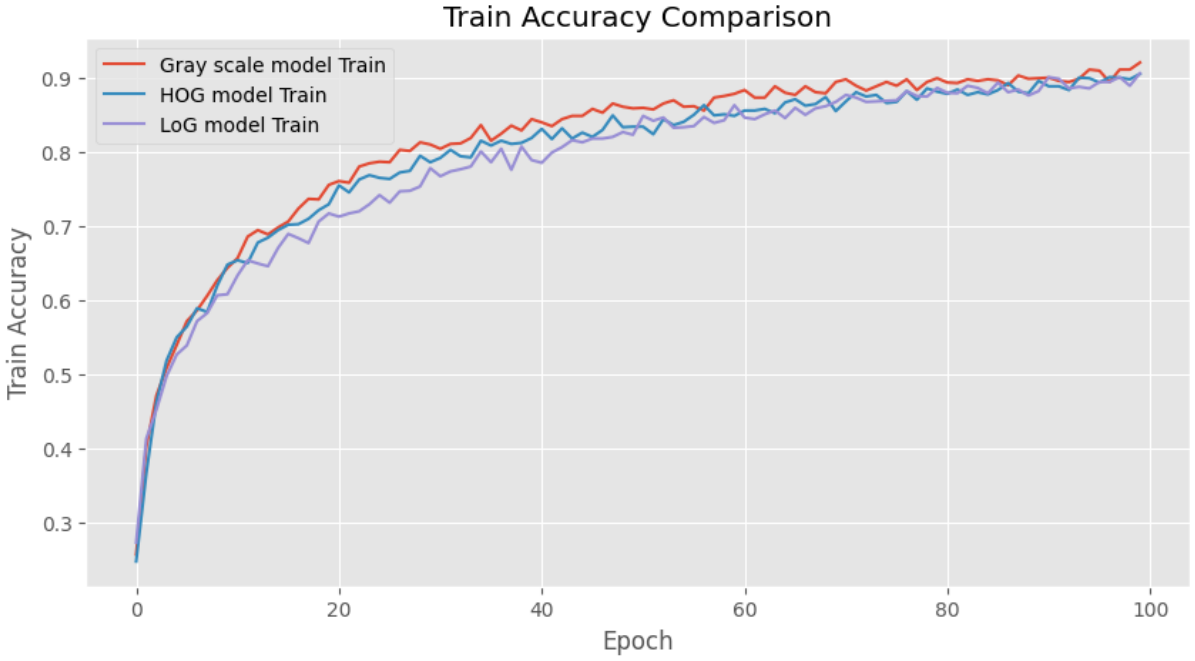


Wykres 3 - wykres dokładności danych walidacyjnych modelów w trakcie epok, dla prostszej sieci konwolucyjnej

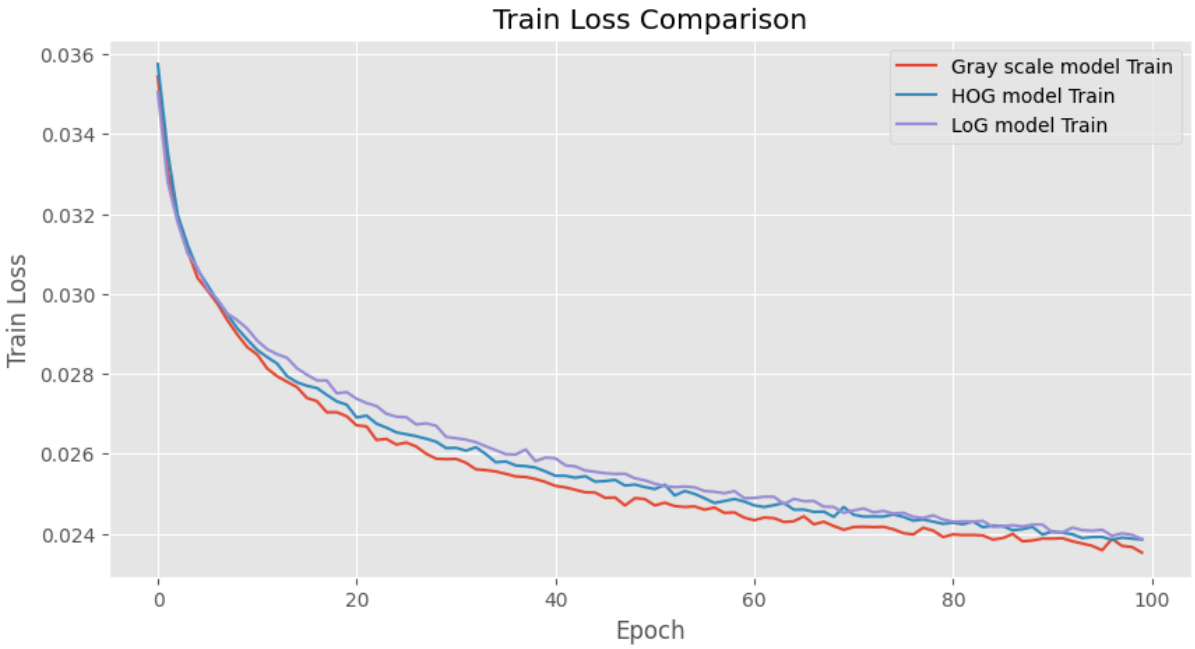


Wykres 4 - wykres funkcji straty danych walidacyjnych modelów w trakcie epok, dla prostszej sieci konwolucyjnej

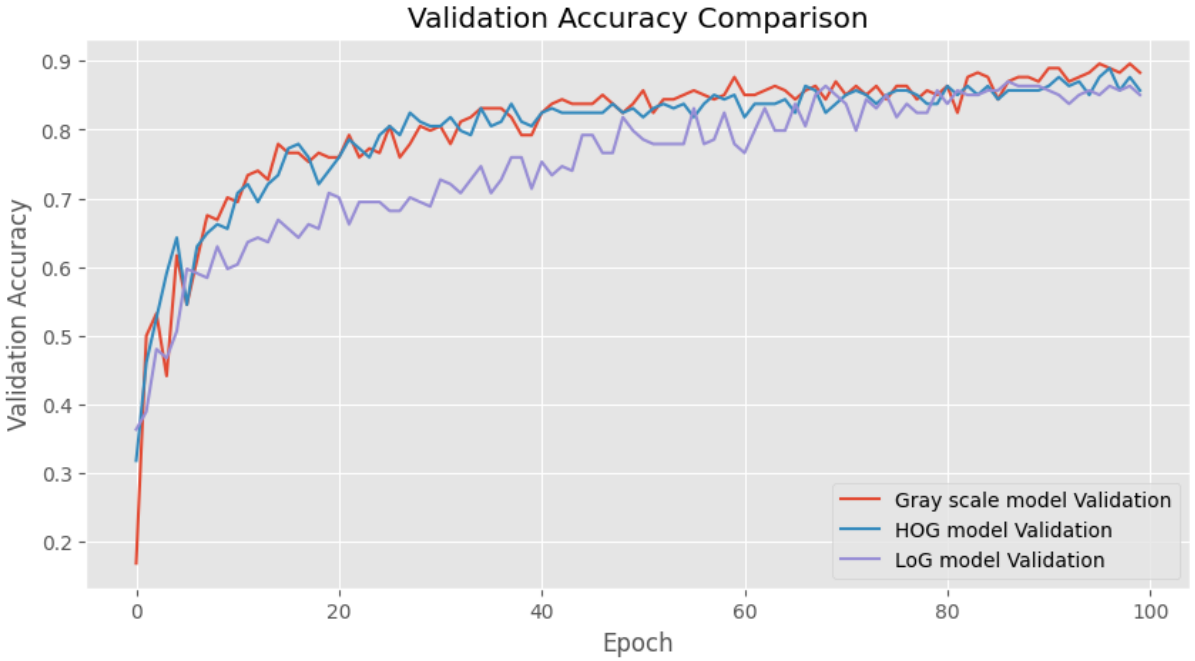
2. Rozbudowana sieć konwolucyjna



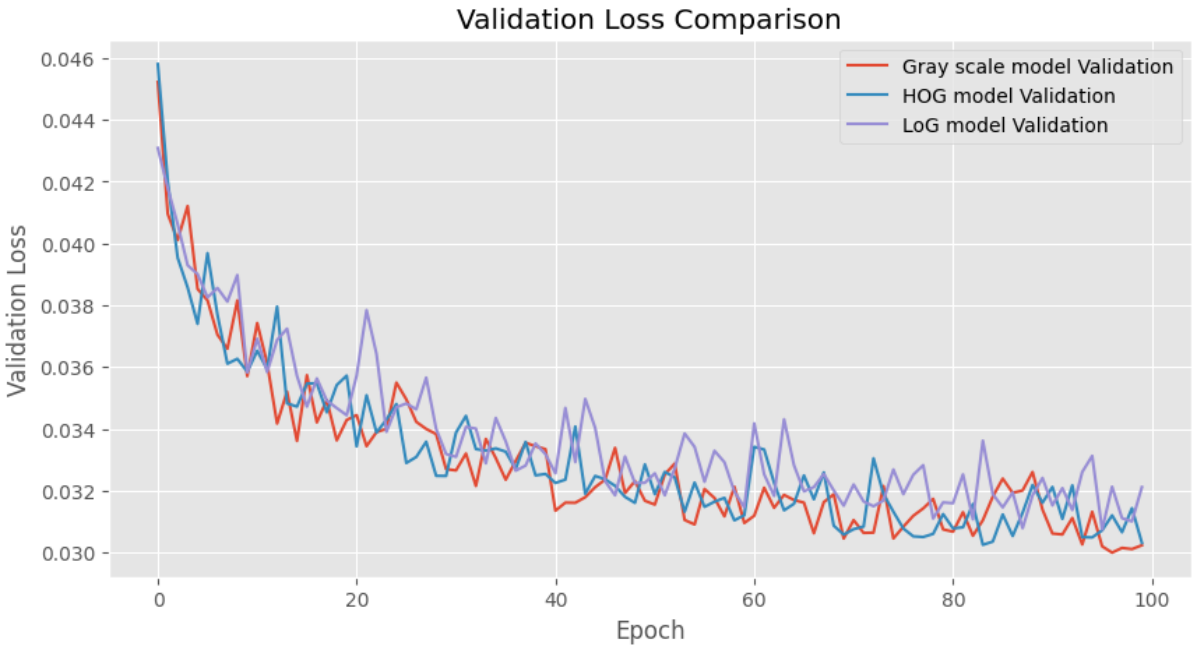
Wykres 5 - wykres dokładności danych testowych modelów w trakcie epok, dla rozbudowanej sieci konwolucyjnej



Wykres 6 - wykres funkcji straty danych testowych modelów w trakcie epok, dla rozbudowanej sieci konwolucyjnej



Wykres 7- wykres dokładności danych walidacyjnych modelów w trakcie epok, dla rozbudowanej sieci konwolucyjnej

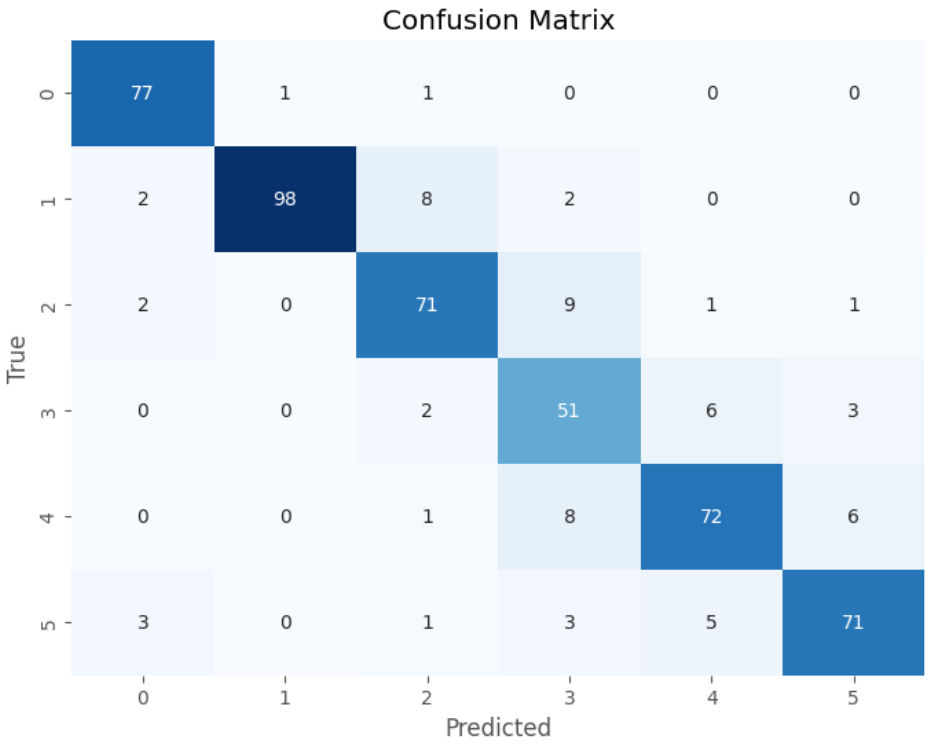


Wykres 8 - wykres funkcji straty danych walidacyjnych modelów w trakcie epok, dla rozbudowanej sieci konwolucyjnej

- Porównanie efektywności sieci konwolucyjnych na testowym zbiorze danych:

Prostsza sieć konwolucyjna na zbiorze danych z nałożonym filtrem HoG:

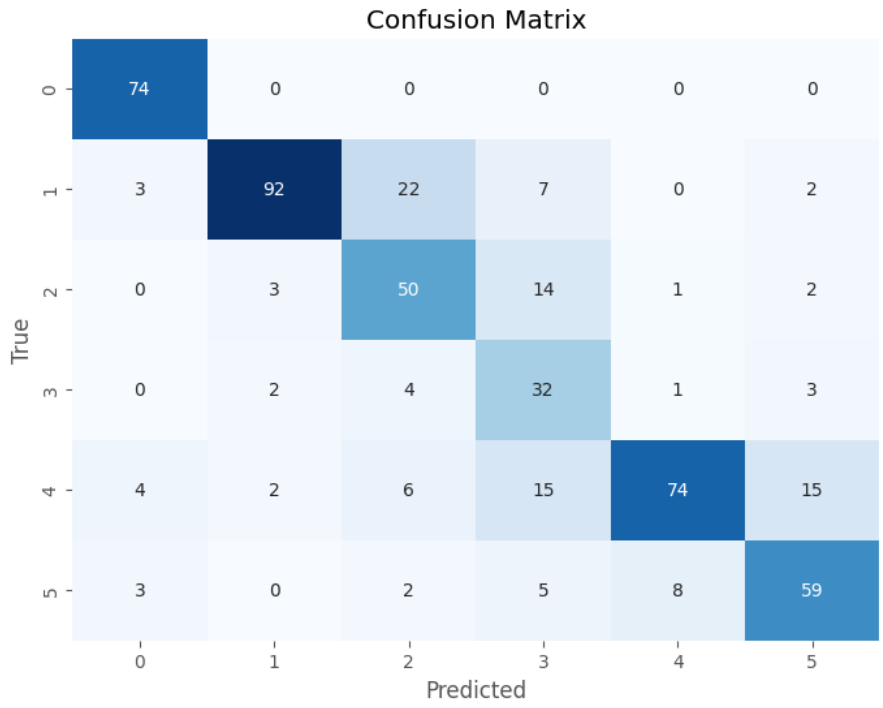
F1-score: 0.87



Zdjęcie 8 – tabela pomyłek dla prostszego CNN, na zbiorzez z filtrem HoG

Rozbudowana sieć konwolucyjna na zbiorze danych z nałożonym filtrem HoG:

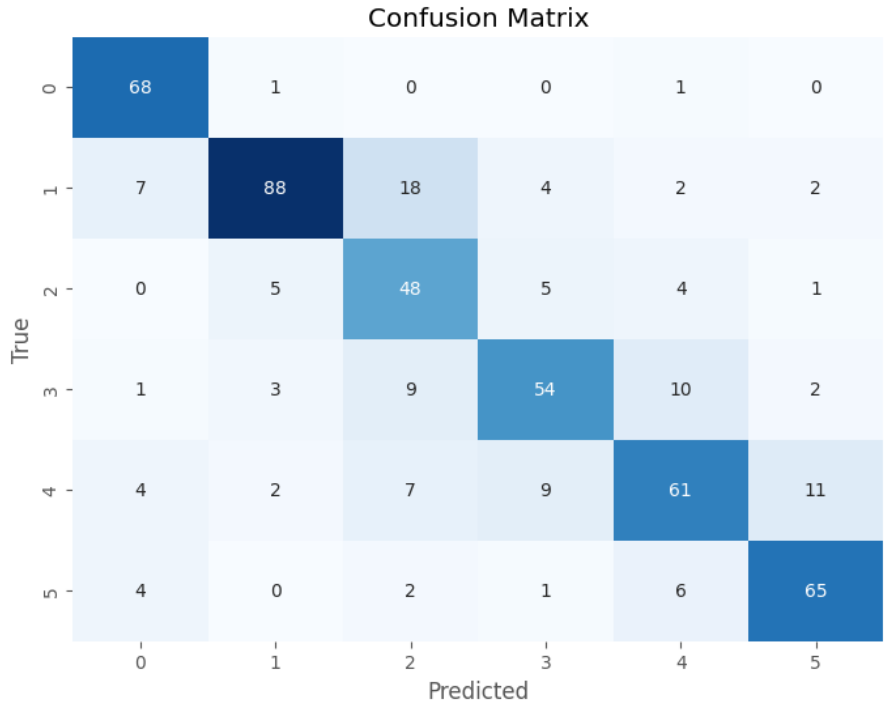
F1-score: 0.76



Zdjęcie 9 – tabela pomyłek dla rozbudowanego CNN, na zbiorze z filtrem HoG

Rozbudowana sieć konwolucyjna na zbiorze danych z nałożonym filtrem LoG:

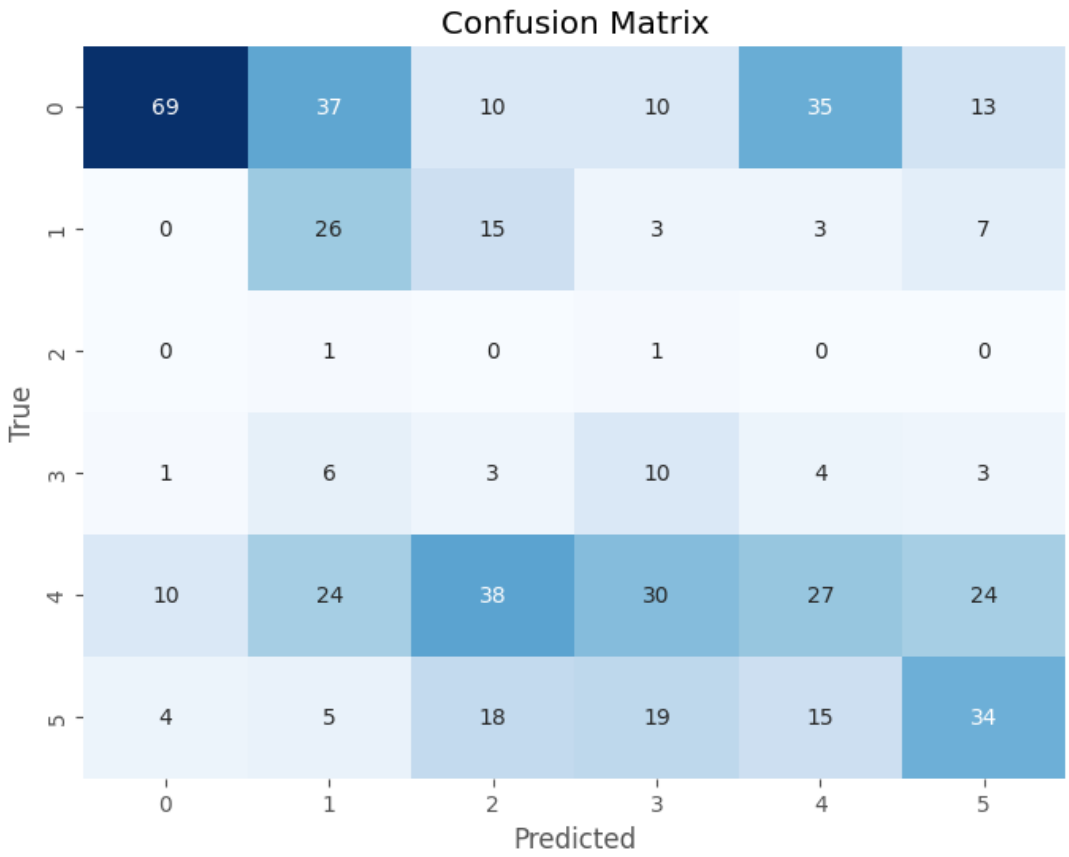
F1-score: 0.76



Zdjęcie 10 – tabela pomyłek dla rozbudowanego CNN, na zbiorze ze skalą szarości

Rozbudowana sieć konwolucyjna na zbiorze danych ze skalą szarości:

F1-score: 0.37



Zdjęcie 11 - Tabela pomyłek dla rozbudowanej sieci w skali szarości

- Porównanie działania sieci neuronowych i algorytmu SVM na zbiorze testowym o wielkości 506 elementów badane 20 razy:

Wykres 9 - wykres działania algorytmów mierzony w sekundach.

3. Analiza uzyskanych wyników

Uczenie maszynowe:

Wybierając i tworząc samodzielnie cechy musieliśmy sprawdzić jak realnie wpływają na działanie modelu oraz jakie zależności mają z predykowaną klasą. Dlatego zdecydowaliśmy się na wizualizację macierzy korelacji, poszukiwania algorytmu oraz hiperparametrów poprzez grid search. A na koniec określenie jakości modelu dokonaliśmy zestawiając dokładność a następnie obliczając współczynnik F1 oraz wyświetlając tablicę błędów.

Uzyskując macierz korelacji (zdjęcie 4) możemy zauważyć, które cechy są od siebie zależne. Najbardziej zależy nam, żeby zaobserwować czy wskazane cechy mają wpływ na przewidywaną liczbę. W ten sposób możemy wyróżnić, że znaczny wpływ położenia kciuka (pierwszy palec) i odległości kciuka od palca wskazującego. Rzeczywiście ten wynik nie zadziwia, gdyż położenie tego palca determinuje większość z pokazywanych liczb. Również mocna korelacja jest pomiędzy odległością palca wskazującego oraz środkowego. Zauważyliśmy, że ona znacznie się różni zależnie od pokazywanego gestu. Wynik pokrywa się z naszymi obserwacjami. Widzimy za to, że wśród obliczanych kątów najmniejszy wpływ ma odległość pomiędzy trzecim, a czwartym palcem. Rzeczywiście zarówno oglądając zdjęcia, jak i bazując na własnym doświadczeniu, kąt między nimi niezależnie od pokazywanej liczby jest podobny. Na koniec warto zauważyć, że zależność pomiędzy różnymi cechami tego samego palca pokrywają się, co jest oczywiście prawdą.

Do wykonania Grid Search'a wybrałam trzy modele – maszynę wektorów nośnych, algorytm liniowej analizy dyskryminującej oraz drzewo decyzyjne. Zastosowałam walidację krzyżową. Pozwala ona na dokładniejszą ocenę wydajności i zdolności modelu do generalizacji poprzez wielokrotne dzielenie danych na zestawy treningowe i testowe. Dzięki temu wyniki są bardziej wiarygodne niż przy jednorazowym podziale danych. Warto zauważyć w tabeli 1., że algorytm wybrał parametr z wysoką regularyzacją (parametr C), co jest ważne przy dużej ilości danych oraz cech. Model może łatwo się przeuczyć. Dodatkowo gamma ma dobrany skalowalny parametr, co zmienia wpływ punktu treningowego w jądrze radialnym (RBF). Jądro wielomianowe przekształca dane wejściowe w wyższą przestrzeń wymiarową. Umożliwia to uchwycenie nieliniowych zależności. Jest to ważne, aby uchwycić zależności pomiędzy wybranymi cechami a wskazaną liczbą.

Ocenę modelu dokonałam na danych testowych, korzystałam ze wskaźnika F1, który łączy precyzję i czułość, jest to średnia harmoniczna. Wynik wskazuje na fakt, że model jest dobrze nauczony i nieprzeuczony. Pokazuje to, że cechy zostały właściwe dobrane. Warto wspomnieć, ze model przyjmuje wiele cech, dlatego taki wynik jest pożądany. Nie sczytuje on całego zdjęcia a punkty, które nie różnią się tak od siebie niezależnie od zdjęcia, filmu czy działania na żywo. Nie ma różnego oświetlenia, kolorów lub tła. Na tablicy pomyłek (zdjęcie 5) widzimy wiele poprawnych predykcji, ale również dwie pomyłki, które wskazują na niewłaściwe rozpoznanie liczby dwa na liczbę trzy oraz liczby pięć na liczbę trzy. Analizując wykonane testy jakości modelu spodziewamy się dużej skuteczności w czasie rzeczywistym.

Uczenie głębokie:

Aby możliwe było uczenie modelu, trzeba dostosować dane treningowe do wspólnego wymiaru. Zastosowałam rozmiar 100 na 100, co jak widać na zdjęciach (zdjęcie 7. i 8.) sprawia, ze zdjęcia są mniej wyraźne, dla ludzi czasem nieczytelne, ale model na nich się uczy. Wyniki nauki na dwóch modelach, stosując różne filtry możemy zobaczyć na zaprezentowanych wykresach.

W modelu używamy danych treningowych i walidacyjnych, co pokazuje nam zarówno działanie modelu w czasie nauki, jak i na nowych danych w każdej epoce.

Wizualizacja wyników pierwszego modelu widoczne na wykresach 5., 6., 7. I 8. Ukazuje, że model uczy się jedynie na danych ze zastosowaniem filtru histogramu zorientowanego gradientu. Możliwe, że

wynika to z faktu, że ta sieć ma mało warstw, dlatego model nie znajduje sam odpowiedniej mapy cech. W rezultacie HoG efektywnie przechwytuje krawędzie, kontury i inne istotne cechy strukturalne, które są ważne dla rozpoznawania obiektów. Surowe obrazy zawierają dużo zbędnych informacji (np. tło), które mogą utrudniać proces uczenia się. Z kolei HoG wyodrębnia tylko te informacje, które są kluczowe dla klasyfikacji i zawiera je w macierzy. Dlatego pomaga to we stworzeniu mapy cech przez sieć CNN. Najważniejsza jest zmiana funkcji straty, chcemy ją minimalizować. Zapisujemy model na podstawie wyników danych treningowych, w tym przypadku jest to około 80 epoki. Później korzystamy z niego do predykowania liczb w czasie rzeczywistym.

Drugi model jest bardziej złożony. Możemy zauważyć, ze zarówno dla danych treningowych i walidacyjnych zmienia się wartość funkcji straty (wykresy 10. I 12.). Ponadto wyniki jeśli chodzi o dokładność są zbliżone, niezależnie od wykorzystanego filtra (wykresy 9. I 11.) – mimo to, zauważamy minimalną przewagę modelu uczonego na danych ze zastosowanym filtrem HoG. W porównaniu do poprzedniego modelu, mamy mniejsze wyniki maksymalne, może to być spowodowane zastosowaniem dodatkowo normalizacji batch'y oraz wyłączania neuronów – prowadzi to do mniejszego przeuczenia się modeli. Tutaj zauważymy, że model uczy się dłużej, w pierwszym modelu szybciej była stagnacja. Najlepszy model zapisujemy po 80 epoce – około 85 epoki.

Następnie zdecydowaliśmy się na porównanie działania modeli na danych testowych. Porównaliśmy jakość modeli przy pomocy wskaźnika f1 oraz zwizualizowaliśmy tablicą pomyłek.

Zaczynając od prostszej sieci konwolucyjnej dla zbioru danych z filtrem HoG, dla niego mamy najwyższe wyniki we wskaźniku f1 score, dodatkowo na tabeli pomyłek również widzimy najlepsze wyniki (zdjęcie 8.) Co ciekawe, możemy zauważyć że dla najczęstsze błędy wynikają dla predykcji jedynki jako dwójki, dwójki jako trójki. Jest to rzeczywiście możliwe szczególnie przy mniej dokładnym pokazaniu liczby. Ciekawy jest fakt, że błędnie predykowana dwójka jest cztery razy częściej jedynką niż trójką. Warto podkreślić, że wyniki są bardzo dobre, błędy drobne.

Wyniki wskaźnika F1 dla rozbudowanej sieci konwolucyjnej są takie same zarówno dla filtra HoG, jak i LoG. Różni się za to tabela pomyłek. W pierwszym najwięcej błędów zachodzi dla klasy trzeciej, za to dla drugiego modelu jest to klasa druga. Najbardziej godne uwagi jest działanie skali szarości. Mimo dobrego działania na danych treningowych, możemy zauważyć że model musiał być przeuczony. Na danych treningowych radzi sobie lepiej niż losowe wybieranie klasy (to byłby poziom dokładności równy 0.17).

Porównanie czasu działania:

Warto zauważyć, że algorytm SVM ma krótszy czas działania na pięciuset rekordach danych. Jednak przy tej ilości danych te różnice czasowe nie są tak znaczne. Warto wspomnieć, że oprócz porównywania działania algorytmów, porównuję również czas przetworzenia danych do odpowiedniej formy. Dla algorytmu SVM zawiera to obliczanie odległości oraz kątów palca. Dlatego właśnie możliwe jest, że model bazujących na danych z filtrem HoG ma najdłuższy czas działania. Oznacza to, że w porównaniu do innych filtrów jego implementacja jest bardziej czasochłonna.

Te wyniki pomogą nam dobrać odpowiednio algorytm do działania w czasie rzeczywistym.

4. Wnioski

- Te badanie pokazało nam, że warto rozważyć różne modele. Mimo, że na przestrzeni lat coraz bardziej popularne staje się uczenie głębokie, to odpowiednie zastosowanie uczenia maszynowego, gdy możliwe jest cech, może być korzystniejsze. Porównując wyniki najlepiej radziła sobie maszyna wektorów nośnych. Było to w pewien sposób zaskakujące, gdyż spodziewaliśmy się lepszych wyników na sieci konwolucyjnej.

- Obserwując modele widać znaczną przewagę modelu SVM. Zarówno wykazuje się lepszymi wynikami predykcji, jak i czasu działania. Cechy zostały odpowiednio przygotowane, co umożliwia szybkie i dokładne przewidzenie wskazanej liczby.
- Dodatkowo ważny jest sposób przetwarzania obrazów, gdyż może to znacznie wpłynąć na wyniki, co szczególnie było zauważalne przy prostszej sieci neuronowej. Mimo, że prostszy model CNN nie zawsze samodzielnie skutecznie mapował cechy, to użycie histogramu zorientowanych gradientów umożliwiło wytrenowanie dobrze działającego narzędzia.
- Wśród zastosowanych sieci konwolucyjnych najlepiej radzi sobie ta najprostsza, do której użyłam filtr HoG. Za to dane ze zastosowaną skalą szarości na żadnej z sieci nie uzyskały pożądanych wyników. Na pierwszej z nich nawet na modelu treningowym osiągały wyniki zbliżone do losowych, a na drugim model został przeuczony, a na danych testowych wyniki były nie wystarczające.
- Na postawie powyższych wyników zdecydowaliśmy się na zastosowanie modelu SVM do działania w czasie rzeczywistym. Jest spowodowane lepszymi wynikami na danych testowych, ale także na krótszym czasie działania.
- Warto zauważyć pewne bariery, które ma mój projekt. Modele uczyły się na konkretnych danych, więc swoje parametry ma dostosowane pod konkretne cechy. Rozpoznaje on najbardziej popularne formy pokazania danej cyfry. Jest to element, który jest możliwy do dopracowania przy pomocy dodatkowych danych, zmiany zbioru danych lub douczania modelu w trakcie jego działania.
- Na koniec sprawdziliśmy swój model w grze Snake, gdzie odpowiednia liczba zmienia kierunek ruchu węża. Model jest wystarczająco precyzyjny i szybki, aby wykorzystać go do gry. Było to jednym z elementów, które oceniło praktyczną skuteczność modelu.
- W czasie projektu poznaliśmy wiele bibliotek. Najwięcej nowości napotkaliśmy przy przetwarzania obrazów, aby wydobyć kontury, cechy ze zdjęć. Do tego stosowaliśmy głównie bibliotekę PIL, OpenCV oraz skimage. Poznaliśmy również metody detekcji obiektów, szczególnie przy pomocy mediapipe, np. segmentacji obrazów, zmniejszania wymiarowości i filtrów kernela.
- Dodatkowo nowym aspektem dla nas była praca z modelem w czasie rzeczywistym. Łączenie kamerki, pobierania klatki, a następnie wyświetlania wyniku podanego przez model.