

UNIWERSYTET EKONOMICZNY W KATOWICACH

KIERUNEK INFORMATYKA

Rafał Brauner
152865

Aplikacja sterująca dronem przy pomocy gestów ręki

An application that controls the drone using hand gestures

Praca magisterska napisana pod kierunkiem
prof. dr. hab. inż. Marcina Grzegorzek

KATOWICE 2024

Spis treści

Wstęp.....	5
1. Rozdział teoriopoznawczy	7
1.1. Bezzałogowe Statki Powietrzne (UAV).....	8
1.2. Rozpoznawanie gestów ręki	11
1.3. Sztuczna Inteligencja i Głębokie Uczenie w rozpoznawaniu gestów.....	17
1.4. Integracja rozpoznawania gestów z systemami sterowania.....	21
2. Rozdział metodyczny	23
2.1. Opis bibliotek i narzędzi.....	24
2.2. Opis zbioru danych	25
2.3. Przygotowanie danych.....	26
2.4. Wybór architektur sieci neuronowych	26
2.5. Proces trenowania modeli AI	30
2.6. Metody oceny skuteczności modeli	32
3. Rozdział użytkarny (praktyczny).....	33
3.1. Opis aplikacji sterującej dronem	34
3.2. Opis modelu AI	38
3.3. Proces treningu modelu.....	42
3.4. Testowanie modelu i wyniki	47
3.3. Testy i wyniki	50
Podsumowanie i wnioski.....	52
Bibliografia	56
Spis skrótów zastosowanych w pracy	59
Spis rysunków.....	60
Spis tabel.....	61
Spis załączników	63

Wstęp

Sterowanie bezzałogowymi statkami powietrznymi (UAV), znanymi powszechnie jako drony, zyskuje na popularności w wielu dziedzinach, takich jak przemysł, rolnictwo, medycyna, ratownictwo, logistyka oraz rozrywka. Tradycyjne metody sterowania, oparte na kontrolerach radiowych, choć efektywne, mają swoje ograniczenia, szczególnie w sytuacjach wymagających większej precyzji lub gdy konieczny jest szybki czas reakcji operatora. W takich przypadkach zastosowanie technologii rozpoznawania gestów ręki stwarza nowe możliwości w zakresie bardziej intuicyjnej i naturalnej interakcji z urządzeniami. W kontekście dronów, sterowanie za pomocą gestów oferuje nowe perspektywy, zarówno w operacjach cywilnych, jak i komercyjnych. Wprowadzenie bezdotykowych metod komunikacji między człowiekiem a maszyną może przyczynić się do zwiększenia efektywności oraz wygody obsługi urządzeń.

Technologia rozpoznawania gestów ręki, która jeszcze niedawno znajdowała zastosowanie głównie w badaniach akademickich i eksperymentalnych, obecnie zyskuje na znaczeniu w przemyśle, m.in. w produkcji robotów, systemach wirtualnej rzeczywistości (VR) oraz urządzeniach codziennego użytku. Rozpoznawanie gestów może także wspierać operacje, w których szybka reakcja operatora jest kluczowa, np. w działaniach ratunkowych lub inspekcjach w trudno dostępnych miejscach. W takich przypadkach gesty ręki stają się bardziej intuicyjne niż tradycyjne metody sterowania, eliminując potrzebę użycia fizycznych kontrolerów.

Niniejsza praca podejmuje próbę opracowania systemu sterowania dronem DJI Tello za pomocą gestów ręki, wykorzystując nowoczesne techniki sztucznej inteligencji (AI), w szczególności konwolucyjne sieci neuronowe (CNN). W badaniach przeanalizowano, jakie wyzwania stawia implementacja takiego rozwiązania oraz jakie korzyści może przynieść jego zastosowanie w realnych warunkach. Problem badawczy dotyczył możliwości stworzenia systemu, który umożliwia bezdotykowe sterowanie dronem w sposób niezawodny, nawet w dynamicznie zmieniających się warunkach otoczenia. Ponadto badania miały na celu odpowiedź na pytanie, jakie modele sztucznej inteligencji są najbardziej efektywne w rozpoznawaniu gestów ręki na podstawie obrazów z kamery drona.

Zakres pracy obejmuje zarówno część teoretyczną, w której omówiono istniejące technologie oraz rozwiązania, jak i część praktyczną, w której przeprowadzono

implementację aplikacji oraz testowanie systemu. Zastosowano różne techniki przetwarzania obrazu oraz algorytmy głębokiego uczenia, aby zapewnić wysoką dokładność rozpoznawania gestów i stabilność działania systemu w zmiennych warunkach oświetleniowych i środowiskowych. Wykorzystano język programowania Python oraz narzędzia takie jak PyTorch, Torchvision i OpenCV, które umożliwiły efektywne trenowanie modeli AI oraz implementację samej aplikacji.

Głównym celem pracy było stworzenie aplikacji, która w sposób niezawodny umożliwi sterowanie dronem DJI Tello za pomocą gestów ręki, a także analiza skuteczności różnych metod rozpoznawania gestów w rzeczywistych warunkach. Praca ma na celu ukazanie potencjału tej technologii oraz identyfikację przyszłych kierunków badań, które mogą przyczynić się do dalszego rozwoju interakcji człowiek-maszyna.

1. Rozdział teoriopoznawczy

1.1. Bezzałogowe Statki Powietrzne (UAV)

Bezzałogowe statki powietrzne (UAV), potocznie nazywane dronami, to latające urządzenia sterowane zdalnie lub autonomicznie, bez potrzeby obecności pilota na pokładzie. Drony, początkowo rozwijane na potrzeby wojskowe, z czasem znalazły szerokie zastosowanie w zadaniach sektora publicznego. Głównym elementem, który odróżnia drony od tradycyjnych statków powietrznych, jest brak pilota na pokładzie oraz możliwość sterowania nimi na odległość za pomocą sygnałów radiowych, a w przypadku dronów autonomicznych – nawigacja oparta na zaprogramowanych algorytmach (Bukowski et al., 2018).

Historia rozwoju dronów sięga początków XX wieku, kiedy to pierwsze koncepcje i prototypy były testowane głównie w celach militarnych. W miarę postępu technologicznego, drony ewoluowały, stając się bardziej dostępne dla użytkowników cywilnych. Współczesne drony, dzięki postępowi w dziedzinie elektroniki, sztucznej inteligencji oraz systemów nawigacyjnych, mogą wykonywać skomplikowane zadania z wysoką precyzją, co czyni je niezastąpionymi narzędziami w wielu dziedzinach (Bukowski et al., 2018).

Drony można klasyfikować na wiele sposobów, w zależności od ich zastosowania, zasięgu, wielkości, czy też sposobu napędu. Podstawowe klasyfikacje dronów obejmują:

1. Klasyfikacja według zastosowania:

- Wojskowe: Drony wykorzystywane do celów zwiadowczych, szpiegowskich, a także bojowych. Przykłady to MQ-9 Reaper czy MQ-1 Predator.
- Cywilne: Stosowane w różnych dziedzinach, takich jak fotografia, geodezja, inspekcje techniczne, ratownictwo, rolnictwo, czy dostawy. Przykładami cywilnych dronów są DJI Phantom, DJI Mavic, Parrot Anafi.

2. Klasyfikacja według zasięgu:

- Bliskiego zasięgu (do 5 km): Drony używane głównie do fotografii i rekreacji.
- Średniego zasięgu (5–50 km): Drony stosowane w rolnictwie, inspekcjach technicznych, czy nadzorze.

- Dalekiego zasięgu (powyżej 50 km): Wykorzystywane w zastosowaniach wojskowych i specjalnych misjach cywilnych.






3. Klasyfikacja według wagi (European Union Aviation Safety Agency, 2024):

- C0 – całkowita waga urządzenia nie może przekraczać 250g, a prędkość lotu nie może przekroczyć 19m/s. Przykładem są DJI Mini 3 i DJI Mini 4.
- C1 – maksymalna waga nie może przekroczyć 900g.
- C2 – waga urządzenia oscyluje w przedziale od 900g do 4kg.
- C3 i C4 – maksymalna masa startowa to w obu klasach 25kg, a ograniczenie dotyczące maksymalnego typowego wymiaru wynosi 3m.
- C5 i C6 – drony w tych klas służą do lotów w kategoriach szczególnych określonych specjalnymi przepisami.

4. Klasyfikacja według napędu:

- Wielowirnikowe: Drony z napędem opartym na kilku wirnikach, zapewniające dużą stabilność i precyzję lotu (np. quadcoptery).
- Płatowce: Drony przypominające samoloty, charakteryzujące się większym zasięgiem i prędkością, ale wymagające przestrzeni do startu i lądowania.

Na rysunku 1 zamieszczono schemat przydziału klas do kategorii dronów.

Operation			Drone Operator / pilot				
C-Class	Max Take off mass	Subcategory	Operational restrictions	Drone Operator registration?	Remote pilot qualifications	Remote pilot minimum age	
Privately build	<250g 	A1 Not over assemblies of people (can also fly in subcategory A3)	Operational restrictions on the drone's use apply (follow the QR code below)	Yes No if toy or not fitted with camera/sensor 	Read user's manual	No minimum age (certain conditions apply)	
legacy < 250g							
C0							
C1					Yes	Check out the QR code below for the necessary qualifications to fly these drones	16
C2		A2 Fly close to people (can also fly in subcategory A3)					
C3	<25kg 	A3 Fly far from people					
C4							
Privately build							
Legacy drones (art 20)							

Rysunek 1. Schemat przydziału klas do kategorii dronów (źródło: European Union Aviation Safety Agency, 2024).

Bezzałogowe statki powietrzne (UAV), znalazły szerokie zastosowanie w wielu dziedzinach życia. Ich unikalne możliwości, takie jak manewrowość, zdalne sterowanie

oraz możliwość dotarcia do trudno dostępnych miejsc, sprawiają, że są one wykorzystywane w różnorodnych sektorach (K. P. Valavanis, 2008). Poniżej przedstawiono najważniejsze obszary zastosowań dronów:

1. Ratownictwo i działania humanitarne:

Drony odgrywają kluczową rolę w operacjach ratowniczych, umożliwiając szybkie lokalizowanie ofiar wypadków, klęsk żywiołowych czy innych katastrof. Dzięki kamerom termowizyjnym i czujnikom, drony mogą przeszukiwać duże obszary w krótkim czasie, co znacząco zwiększa szanse na uratowanie życia (Naumienko, 2023). W działaniach humanitarnych drony są wykorzystywane do dostarczania leków, żywności i innych niezbędnych zasobów do trudno dostępnych regionów (Dyndal et al., 2017). Przykładem może być wykorzystanie dronów do dostarczania szczepionek w odległych obszarach Afryki.

2. Fotografia i filmowanie z powietrza:

Jednym z najbardziej popularnych zastosowań dronów jest fotografia i filmowanie z powietrza (*Najlepsze drony do zdjęć – co wybrać?*, n.d.). Drony wyposażone w wysokiej jakości kamery umożliwiają tworzenie spektakularnych ujęć z unikalnych perspektyw, co jest wykorzystywane zarówno w produkcjach filmowych, jak i w marketingu czy turystyce. Drony stały się również narzędziem pracy dla fotografów ślubnych, architektów krajobrazu oraz twórców treści w mediach społecznościowych.

3. Inspekcje techniczne i monitorowanie infrastruktury:

Drony są coraz częściej wykorzystywane do inspekcji technicznych mostów, linii wysokiego napięcia, rurociągów, wież telekomunikacyjnych i innych trudno dostępnych obiektów (Szóstek et al., 2022). Dzięki dronom możliwe jest szybkie i dokładne zidentyfikowanie usterek, co pozwala na bardziej efektywne planowanie prac konserwacyjnych. Drony są również używane do monitorowania postępów budowlanych oraz oceny stanu technicznego infrastruktury, co znacznie obniża koszty i ryzyko związane z tradycyjnymi metodami inspekcji.

4. Dostawy i logistyka:

Drony rewolucjonizują sektor dostaw i logistyki, umożliwiając szybkie i efektywne dostarczanie przesyłek na krótkie i średnie odległości. W miastach drony mogą dostarczać przesyłki bezpośrednio do domów klientów, omijając korki i inne utrudnienia. Przykłady takich zastosowań obejmują pilotażowe projekty firm takich jak Amazon Prime Air (Amazon, 2022) czy UPS, które testują dostawy dronami w wybranych lokalizacjach.

Sterowanie dronami, od ich początków, ewoluowało wraz z rozwojem technologii. Początkowo ograniczone do prostych systemów radiowych, obecnie obejmuje zaawansowane metody, które umożliwiają precyzyjne manewrowanie i automatyzację lotów. Najbardziej tradycyjną metodą sterowania dronami jest wykorzystanie kontrolerów ręcznych, które komunikują się z dronem za pośrednictwem fal radiowych. Kontrolery te zazwyczaj posiadają joysticki, które pozwalają operatorowi na bezpośrednie sterowanie dronem w czasie rzeczywistym. W zależności od modelu drona, kontrolery mogą oferować różne funkcje, takie jak regulacja prędkości, wysokości, a także możliwość wykonywania automatycznych manewrów (np. obrót, powrót do punktu startu). Pomimo pojawienia się bardziej zaawansowanych technologii, ręczne sterowanie nadal jest popularne, szczególnie wśród amatorów oraz w sytuacjach, gdzie wymagana jest pełna kontrola nad urządzeniem.

Wraz z rozwojem technologii GPS oraz algorytmów nawigacyjnych, drony zyskały możliwość automatycznego lotu zgodnie z zaprogramowaną trasą. Operatorzy mogą zdefiniować punkty orientacyjne (ang. *waypoints*), a dron automatycznie przelatuje przez nie, wykonując określone zadania, takie jak fotografowanie czy nagrywanie wideo. Taka technika znalazła swoje zastosowanie m.in. w nalotach fotogrametrycznych, których produktami są mapy i modele 3D (Kossowski, 2022).

Sterowanie dronami za pomocą gestów to jedna z najnowszych i najbardziej innowacyjnych metod, która pozwala na bezpośrednią interakcję człowieka z urządzeniem (Gio et al., 2021). Technologia ta opiera się na rozpoznawaniu gestów ręki przez drona, co umożliwia wykonywanie określonych komend, takich jak start, lądowanie, obrót czy przesunięcie w określonym kierunku. Systemy rozpoznawania gestów zazwyczaj wykorzystują kamery oraz algorytmy przetwarzania obrazu, które analizują ruchy operatora i przekształcają je w polecenia dla drona. Ta technologia ma ogromny potencjał w sytuacjach, gdzie tradycyjne metody sterowania są niewygodne lub niemożliwe do zastosowania.

1.2. Rozpoznawanie gestów ręki

Rozpoznawanie gestów ręki jest jednym z kluczowych elementów interakcji człowieka z maszyną, szczególnie w kontekście sterowania urządzeniami takimi jak drony (Xianghan Wang, Jie Jiangb, Yingmei Wei, Lai Kang and Yingying Gao, 2018). W ciągu ostatnich kilku dekad opracowano wiele metod rozpoznawania gestów, które

można podzielić na dwie główne kategorie: tradycyjne metody przetwarzania obrazu oraz nowoczesne metody oparte na uczeniu maszynowym.

1. Tradycyjne metody rozpoznawania gestów:

- Analiza ruchu i kształtu: Tradycyjne metody rozpoznawania gestów opierają się na analizie ruchu oraz kształtu dłoni przy użyciu algorytmów przetwarzania obrazu. Do najczęściej stosowanych technik należy wykrywanie konturów, analiza kształtu oraz ekstrakcja cech geometrycznych dłoni. Wykorzystując odpowiednie filtry, można zidentyfikować krawędzie dłoni i palców, co pozwala na rozpoznanie określonych gestów.
- Optical Flow: Optical Flow to technika pozwalająca na śledzenie ruchu obiektów w sekwencji obrazów wideo. W kontekście rozpoznawania gestów, Optical Flow może być używany do monitorowania ruchu dłoni, co jest szczególnie przydatne w wykrywaniu dynamicznych gestów, takich jak machanie ręką czy gesty wskazywania. Metoda ta jednak bywa zawodna w warunkach zmieniającego się oświetlenia lub gdy tło jest zbyt skomplikowane.
- Histogramy orientacji gradientów (HOG): HOG to metoda polegająca na analizie gradientów intensywności pikseli w obrazie, która pozwala na wykrywanie charakterystycznych kształtów, takich jak kontury dłoni. HOG jest skuteczny w rozpoznawaniu statycznych gestów i jest często używany w połączeniu z innymi metodami przetwarzania obrazu.

2. Metody oparte na uczeniu maszynowym:

- Maszynowe metody klasyfikacji: Tradycyjne metody uczenia maszynowego, takie jak SVM (Support Vector Machines) czy KNN (K-Nearest Neighbors), były jednymi z pierwszych technik wykorzystywanych do rozpoznawania gestów. Te metody wymagają ręcznej ekstrakcji cech, takich jak kontury, tekstury czy punkty kluczowe, które następnie są klasyfikowane przez model uczący się na podstawie dostarczonych danych.
- Głębokie uczenie: Obecnie najbardziej zaawansowane metody rozpoznawania gestów opierają się na głębokim uczeniu, w szczególności na wykorzystaniu konwolucyjnych sieci neuronowych (CNN) (Felix Zhan, 2019). CNN są w stanie automatycznie uczyć się

cech obrazu, które są kluczowe do rozpoznawania gestów, eliminując potrzebę ręcznej ekstrakcji cech. Modele te, takie jak AlexNet, ResNet czy GoogleNet, są trenowane na dużych zbiorach danych, co pozwala na osiągnięcie bardzo wysokiej dokładności w rozpoznawaniu gestów.

- Recurrent Neural Networks (RNN): W przypadkach, gdy gesty są sekwencyjne i wymagają analizy czasowej, stosuje się Recurrent Neural Networks (RNN) oraz ich odmiany, takie jak LSTM (Long Short-Term Memory). Te sieci neuronowe są w stanie analizować sekwencje obrazów wideo, rozpoznając zmiany w ruchu dłoni w czasie i klasyfikując je jako określone gesty.
- Transfer Learning: Transfer learning to technika, w której wykorzystuje się wcześniej wytrenowane modele na dużych zbiorach danych, a następnie dostosowuje się je do konkretnego zadania rozpoznawania gestów. Technika ta jest szczególnie przydatna, gdy mamy ograniczoną ilość danych treningowych, ponieważ pozwala na szybsze i bardziej efektywne trenowanie modeli o wysokiej dokładności.

Rozpoznawanie gestów ręki opiera się na skutecznej analizie obrazu, która wymaga zastosowania odpowiednich technik przetwarzania obrazu. Techniki te pozwalają na ekstrakcję istotnych cech z obrazu, co jest kluczowe dla skutecznego rozpoznawania gestów. W kontekście rozpoznawania gestów ręki, przetwarzanie obrazu obejmuje szereg etapów, takich jak wstępne przetwarzanie obrazu, segmentacja, ekstrakcja cech oraz normalizacja.

1. Wstępne przetwarzanie obrazu:

- Normalizacja i skalowanie: Normalizacja obrazu polega na przekształceniu wartości pikseli w sposób umożliwiający uzyskanie bardziej jednolitego zakresu intensywności. Jest to istotne, ponieważ modele uczące się, takie jak sieci neuronowe, mogą być bardziej efektywne, gdy dane wejściowe są znormalizowane. Skalowanie obrazu do odpowiednich rozmiarów, na przykład 200x200 pikseli, również ułatwia przetwarzanie, szczególnie w kontekście stosowania modeli CNN.
- Konwersja do skali szarości: Konwersja obrazu do skali szarości jest powszechnie stosowana w przetwarzaniu obrazu, ponieważ redukuje liczbę kanałów kolorów i upraszcza analizę obrazu. W kontekście

rozpoznawania gestów, konwersja do skali szarości może pomóc w skupieniu się na strukturze i konturach dłoni, co jest kluczowe dla skutecznego rozpoznawania.

- Filtracja: Filtracja obrazu ma na celu redukcję szumów i wygładzanie obrazu. Zastosowanie filtrów, takich jak filtry medianowe lub gaussowskie, pozwala na usunięcie zakłóceń, które mogą utrudniać dokładne rozpoznawanie gestów. Filtracja jest szczególnie istotna w warunkach zmiennego oświetlenia i w przypadku obrazów o niskiej jakości.

2. Segmentacja obrazu:

- Detekcja konturów: Segmentacja obrazu, a zwłaszcza detekcja konturów, pozwala na wyodrębnienie obszarów obrazu, które reprezentują dłonie lub palce. Jest to kluczowe dla dalszej analizy, ponieważ umożliwia izolowanie dłoni od tła i skupienie się na istotnych cechach obrazu.
- Binarizacja: Binarizacja obrazu to proces przekształcania obrazu do postaci dwuwartościowej (czarno-białej). Technika ta jest używana do wyraźnego oddzielenia obiektów od tła, co ułatwia dalszą analizę i rozpoznawanie gestów. W kontekście gestów ręki, binaryzacja może pomóc w wyodrębnieniu kształtu dłoni.

3. Ekstrakcja cech:

- Histogramy orientacji gradientów (HOG): HOG to technika używana do ekstrakcji cech opartych na gradientach pikseli. Analizując kierunki gradientów w obrazie, HOG umożliwia wykrywanie charakterystycznych kształtów, takich jak kontury dłoni i palców, co jest kluczowe dla rozpoznawania gestów.
- Fourier Transform: Transformacja Fouriera pozwala na analizę częstotliwościową obrazu, co może być użyteczne w wykrywaniu powtarzalnych wzorców, takich jak tekstury skóry dłoni. Ta technika jest szczególnie przydatna w przypadkach, gdy gesty ręki mają specyficzne cechy, które mogą być trudne do wykrycia za pomocą innych metod.

4. Normalizacja:

- Znormalizowane obrazy: Znormalizowanie obrazów wejściowych, poprzez ujednolicenie wartości pikseli i rozmiaru obrazu, jest kluczowym

etapem przygotowania danych dla modeli uczących się. Normalizacja zapewnia, że modele CNN mogą skutecznie uczyć się na zróżnicowanych danych bez konieczności dostosowywania parametrów dla każdego obrazu z osobna.

Rozpoznawanie gestów ręki, szczególnie w kontekście komunikacji bezdotykowej, opiera się na odpowiednim zbiorze danych, który umożliwia trenowanie modeli uczenia maszynowego. Istnieje kilka dobrze znanych i szeroko wykorzystywanych baz gestów ręki, które odgrywają kluczową rolę w rozwoju systemów rozpoznawania gestów.

1. ASL Alphabet Dataset (*ASL(American Sign Language) Alphabet Dataset*, 2021):

- Opis: Jednym z najczęściej używanych zbiorów danych w kontekście rozpoznawania gestów ręki jest zestaw danych zawierający amerykański alfabet języka migowego (ASL). Zbiór ten zawiera obrazy dłoni reprezentujące litery alfabetu ASL, co umożliwia modelom uczenia maszynowego naukę rozpoznawania tych liter na podstawie obrazu dłoni.
- Zawartość: Zbiór danych ASL Alphabet zawiera około 3000 obrazów dla każdej litery alfabetu, co daje łącznie około 87 000 obrazów. Każdy obraz przedstawia pojedynczą dłoń wykonującą określoną literę z alfabetu ASL. Obrazy są zazwyczaj wysokiej jakości i przedstawiają różne ujęcia dłoni, co zapewnia różnorodność danych treningowych.
- Zastosowanie: Zbiór danych ASL jest szeroko wykorzystywany do trenowania modeli sieci neuronowych, które mają na celu rozpoznawanie gestów ręki w czasie rzeczywistym. W projekcie wykorzystano ten zbiór danych do trenowania modeli CNN, które były następnie stosowane do rozpoznawania liter języka migowego na podstawie obrazów z kamery drona.

2. Cambridge Hand Gesture Dataset (Kim, 2007):

- Opis: Cambridge Hand Gesture Dataset jest kolejnym popularnym zbiorem danych używanym do rozpoznawania gestów ręki. Ten zbiór zawiera różnorodne gesty dłoni, które są nagrywane w różnych warunkach oświetleniowych i z różnymi tłem, co czyni go wszechstronnym zasobem do badań nad rozpoznawaniem gestów.

- Zawartość: Zbiór zawiera około 900 obrazów dla 9 klas gestów ręki. Każdy gest jest sfotografowany z różnych kątów i w różnych warunkach, co zapewnia dużą różnorodność danych. Cambridge Hand Gesture Dataset jest szczególnie użyteczny w badaniach nad dynamicznymi gestami ręki.
- Zastosowanie: Zbiór ten jest często wykorzystywany w badaniach akademickich oraz komercyjnych, gdzie istnieje potrzeba rozpoznawania złożonych, dynamicznych gestów. Ze względu na jego rozmiar i różnorodność, Cambridge Hand Gesture Dataset jest cennym narzędziem w rozwoju systemów interakcji człowiek-maszyna.

3. MSR Gesture 3D Dataset (Li, 2023):

- Opis: MSR Gesture 3D Dataset został opracowany przez Microsoft Research i jest używany głównie do badań nad rozpoznawaniem gestów w trójwymiarowej przestrzeni. Ten zbiór danych jest szczególnie użyteczny w kontekście rozpoznawania gestów z użyciem sensorów głębi, takich jak Microsoft Kinect.
- Zawartość: Zbiór zawiera sekwencje gestów nagranych w trójwymiarowej przestrzeni z użyciem sensorów głębi. MSR Gesture 3D Dataset obejmuje zarówno proste gesty ręki, jak i bardziej złożone sekwencje ruchów, które są trudne do rozpoznania przy użyciu tradycyjnych metod przetwarzania obrazu.
- Zastosowanie: Zbiór ten jest często stosowany w badaniach nad interakcją człowiek-komputer oraz w systemach rozpoznawania gestów dla aplikacji wirtualnej i rozszerzonej rzeczywistości. Dzięki zastosowaniu technologii głębi, MSR Gesture 3D Dataset umożliwia rozwój systemów, które mogą rozpoznawać gesty w bardziej skomplikowanych warunkach przestrzennych.

4. EgoGesture Dataset (Zhang et al., 2018):

- Opis: EgoGesture Dataset jest jednym z największych i najbardziej różnorodnych zbiorów danych, który zawiera gesty ręki nagrane z perspektywy pierwszej osoby (ang. egocentric view). Zbiór ten jest szczególnie użyteczny w kontekście rozwijania systemów sterowania urządzeniami noszonymi oraz dla aplikacji rzeczywistości rozszerzonej.

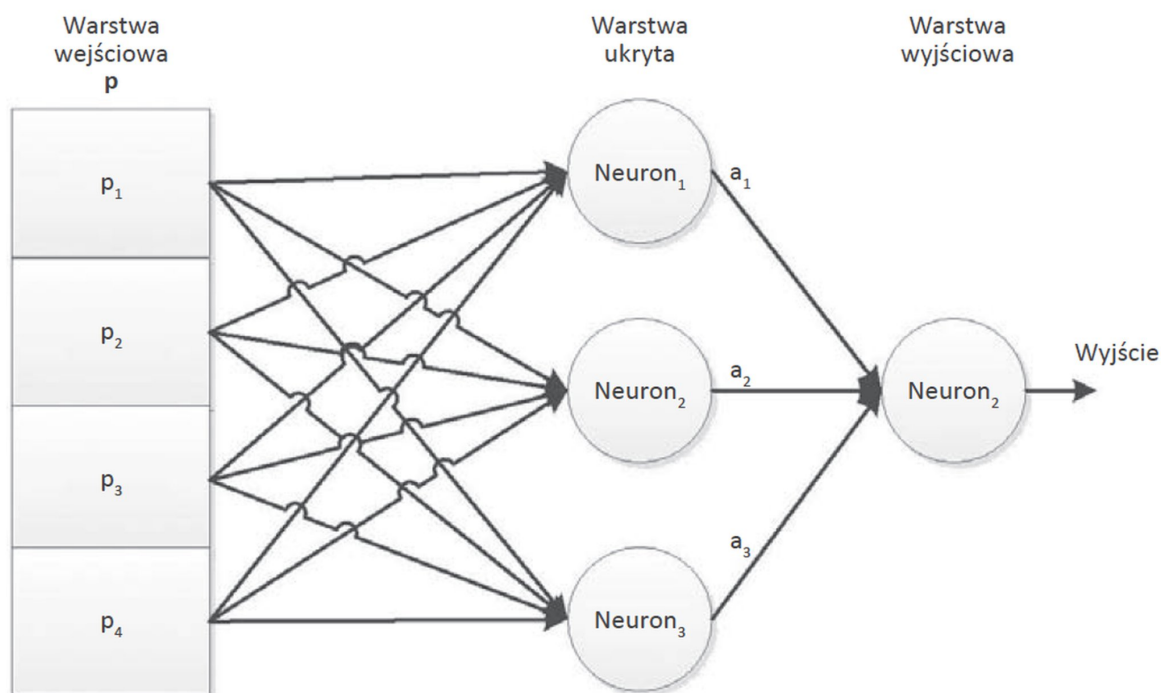
- Zawartość: Zbiór ten zawiera ponad 24 000 sekwencji wideo gestów, nagranych w różnych warunkach oświetleniowych i z różnymi scenariuszami tła. Obrazy są nagrywane z różnych perspektyw, co zapewnia różnorodność danych i umożliwia rozwój systemów, które mogą działać w rzeczywistych warunkach.
- Zastosowanie: EgoGesture Dataset jest często stosowany w badaniach nad rozpoznawaniem gestów w czasie rzeczywistym oraz w systemach interakcji człowiek-komputer, które wymagają analizy obrazu z perspektywy pierwszej osoby.

1.3. Sztuczna Inteligencja i Głębokie Uczenie w rozpoznawaniu gestów

Sieci neuronowe to klasa algorytmów inspirowanych biologiczną strukturą mózgu, które stanowią podstawę nowoczesnych metod uczenia maszynowego. Sieci neuronowe, a w szczególności głębokie sieci neuronowe, zyskały na popularności dzięki swoim zdolnościom do modelowania złożonych nieliniowych zależności oraz skuteczności w zadaniach związanych z przetwarzaniem dużych zbiorów danych, takich jak rozpoznawanie obrazów, dźwięków czy tekstu.

Podstawowa jednostka w sieci neuronowej to neuron, który jest wzorowany na biologicznym neuronie. Neurony są zorganizowane w warstwy: wejściową, ukryte i wyjściową co jest przedstawione na rysunku 2.

- Warstwa wejściowa: Warstwa ta przyjmuje dane wejściowe do sieci. Każdy neuron w tej warstwie reprezentuje jedną cechę z danych wejściowych.
- Warstwy ukryte: Neurony w tych warstwach przetwarzają dane, które przechodzą przez sieć, poprzez przekształcanie i przesyłanie sygnałów. Sieci neuronowe mogą mieć jedną lub wiele warstw ukrytych, co decyduje o ich głębokości.
- Warstwa wyjściowa: Ta warstwa produkuje wynik sieci, czyli przewidywaną odpowiedź na podstawie danych wejściowych. Liczba neuronów w tej warstwie zależy od specyfiki zadania, na przykład w klasyfikacji liczba neuronów odpowiada liczbie klas.



Rysunek 2. Przedstawienie warstw sieci neuronowej (Źródło: <https://controlengineering.pl/sieci-neuronowe-w-sterowaniu-procesami-technologicznymi/>).

Każdy neuron przetwarza swoje wejścia za pomocą funkcji aktywacji, która decyduje, czy neuron powinien być aktywowany (czyli wysłać sygnał dalej). Popularne funkcje aktywacji to na przykład ReLU, sigmoid czy funkcja tangensa hiperbolicznego.

Uczenie się sieci neuronowej polega na dostosowywaniu wag przypisanych do połączeń między neuronami (Goodfellow et al., 2016). Proces ten odbywa się za pomocą metody zwanej wsteczną propagacją błędów (ang. *Backpropagation*), w której błąd wynikający z różnicy między przewidywaniami sieci a rzeczywistymi wartościami jest propagowany wstecz przez sieć, od warstwy wyjściowej do wejściowej. W wyniku tej propagacji wagi są aktualizowane w celu minimalizacji funkcji straty (ang. *Loss Function*), która mierzy, jak daleko przewidywania sieci są od rzeczywistych wyników.

- Funkcja straty: W klasyfikacji często stosowaną funkcją straty jest funkcja entropii krzyżowej (ang. *Cross-Entropy Loss*), która ocenia, jak dobrze przewidywania sieci pasują do rzeczywistych etykiet.
- Optymalizacja: Proces aktualizacji wag jest zarządzany przez algorytmy optymalizacyjne, takie jak stochastyczny gradient prosty (SGD) czy bardziej zaawansowane metody, takie jak Adam.

W zależności od specyfiki zadania i rodzaju danych, istnieją różne architektury sieci neuronowych (Baheti, 2021):

- Sztuczne sieci neuronowe (*ANN*): Podstawowy rodzaj sieci neuronowych, stosowany w zadaniach klasyfikacji i regresji.
- Konwolucyjne sieci neuronowe (*CNN*): Specjalistyczne sieci, które są niezwykle skuteczne w zadaniach przetwarzania obrazów. CNN wykorzystują operacje konwolucji, aby automatycznie wyodrębniać cechy z obrazów, takie jak krawędzie, tekstury czy bardziej złożone wzorce.
- Rekurencyjne sieci neuronowe (*RNN*): Sieci, które są stosowane głównie do przetwarzania sekwencyjnych danych, takich jak tekst czy dane czasowe. RNN potrafią przetwarzać dane sekwencyjne dzięki mechanizmowi pamięci, który pozwala im „pamiętać” poprzednie kroki.

Głębokie sieci neuronowe, a w szczególności konwolucyjne sieci neuronowe (*CNN*), stały się standardem w zadaniach związanych z przetwarzaniem i rozpoznawaniem obrazów. Ich architektura jest zoptymalizowana pod kątem analizy wizualnych danych, co czyni je niezwykle efektywnymi w rozpoznawaniu wzorców, takich jak krawędzie, tekstury, a także bardziej złożone cechy obrazów.

CNN to rodzaj sieci neuronowych zaprojektowany specjalnie do przetwarzania danych w formie siatki, takich jak obrazy. W przeciwieństwie do tradycyjnych sieci neuronowych (Baheti, 2021), CNN automatycznie uczą się wyodrębniać cechy z obrazów, co sprawia, że są one bardziej wydajne w zadaniach rozpoznawania obrazów. CNN posiada następujące warstwy:

- Warstwa konwolucyjna: Podstawowy blok budulcowy CNN. Warstwa ta stosuje zestaw filtrów (jąder konwolucyjnych) do wejściowego obrazu, generując mapy cech, które reprezentują różne aspekty obrazu, takie jak krawędzie, kształty czy tekstury.
- Warstwa poolingowa: Warstwa ta redukuje wymiarowość map cech, zachowując najważniejsze informacje, co zmniejsza złożoność obliczeniową i zapobiega przetrenowaniu modelu. Najczęściej stosowanym rodzajem pooling jest max pooling, który wybiera maksymalną wartość z okna filtru.
- Warstwy w pełni połączone (ang. fully connected layers): W końcowych etapach sieci CNN stosuje się warstwy w pełni połączone, które integrują wyodrębnione cechy i dokonują klasyfikacji obrazu.

Architektury CNN znajdują zastosowanie w szerokim spektrum zadań związanych z przetwarzaniem obrazów:

- Klasyfikacja obrazów: CNN są podstawą większości systemów klasyfikacji obrazów, takich jak rozpoznawanie obiektów na zdjęciach czy klasyfikacja scen.
- Segmentacja semantyczna: W segmentacji semantycznej CNN służą do przypisywania etykiet do każdego piksela obrazu, co jest kluczowe w zadaniach takich jak autonomiczna jazda.
- Detekcja obiektów: CNN są stosowane w systemach detekcji obiektów, takich jak YOLO czy R-CNN, które identyfikują i lokalizują obiekty w obrazach.
- Rozpoznawanie twarzy: W systemach rozpoznawania twarzy CNN są używane do identyfikacji osób na podstawie zdjęć lub nagrań wideo.
- Rozpoznawanie gestów: W kontekście realizowanego projektu, CNN są kluczowe w rozpoznawaniu gestów ręki na podstawie obrazów z kamery, co umożliwia bezdotykowe sterowanie dronem.

Zastosowanie sztucznej inteligencji, a w szczególności sieci neuronowych jest niezwykle przydatne do rozpoznawania gestów rąk. Istotną kwestią jest w tym wypadku rozpoznawanie języka migowego, może znacząco ułatwić i usprawnić komunikację z osobami Głuchymi.

Próba automatyzacji tego procesu została szerzej opisana w artykule dotyczącym rozpoznawania języka migowego z wykorzystaniem konwolucyjnych sieci neuronowych (CNN) i rozpoznawania obrazów (Computer Vision) (Murali et al., 2022). Opisywany przez autorów badania model potrafił rozpoznać nawet 10 różnych amerykańskich alfabetów języka migowego z bardzo wysoką skutecznością na poziomie 90%. Zastosowano w nim obrazy gestów rąk w schemacie kolorystycznym HSV, które posiadały czarne tło.

Podobne podejście zostało zastosowane również przez innych badaczy (Chavan et al., 2021). Zaprezentowali oni podejście oparte na głębokim uczeniu do rozpoznawania znaków wykonywanych w amerykańskim języku migowym (ASL) na podstawie obrazu wejściowego. System ten potrafi rozpoznawać cyfry od 0 do 9 pokazywane przez użytkownika. Dzięki zastosowaniu przetwarzania obrazów, które przekształca dane RGB na obrazy w skali szarości, osiągnięto efektywną redukcję wymagań dotyczących przechowywania danych oraz czasu treningu konwolucyjnej sieci neuronowej (CNN). Celem eksperymentu było znalezienie kombinacji przetwarzania obrazów i architektury głębokiego uczenia o mniejszej złożoności, aby umożliwić wdrożenie systemu w aplikacjach mobilnych. Autorzy porównali dokładność

rozpoznawania przy użyciu różnych sieci, takich jak LeNet-5, AlexNet, Vgg16 i MobileNet v2. Ostatecznie wybrana architektura miała jedynie 10 warstw, w tym warstwę dropout, co podniosło dokładność treningu do 91,37%, a dokładność testów do 87,5%.

Przykładem jest również praca, w której użyto konwolucyjnych sieci neuronowych do opracowania modelu rozpoznawania znaków alfabetu języka arabskiego (Aldhahri et al., 2023). Wykorzystano w niej zestaw danych Arabic Alphabets Sign Language Dataset (ArASL2018), zawierający obrazy przedstawiające poszczególne znaki odpowiadające literom alfabetu. Przeprowadzona analiza eksperymentalna wykazała, że opracowany model osiągnął dokładność rozpoznawania na poziomie 94,46%. Co więcej, analiza porównawcza z wcześniejszymi badaniami wykazała, że zaproponowany model przewyższał je pod względem dokładności rozpoznawania.

Automatyzacja rozpoznawania języka migowego może również złagodzić barierę językową pomiędzy pacjentami Głuchymi a pracownikami służby zdrowia. Badania zostały przeprowadzone w Indiach (Venugopalan et al., 2023). Ponieważ gesty dłoni są najbardziej ekspresyjnymi składnikami słownictwa języka migowego, zaproponowano nowy zestaw danych dynamicznych gestów dłoni dla indyjskich słów języka migowego (ISL) powszechnie używanych do komunikacji alarmowej przez głuchych pacjentów z pozytywnym wynikiem testu na COVID-19. Hybrydowy model głębokiej splotowej sieci pamięci krótkotrwałej został wykorzystany do rozpoznawania proponowanych gestów dłoni i osiągnął średnią dokładność 83,36%.

1.4. Integracja rozpoznawania gestów z systemami sterowania

Rozpoznawanie gestów ręki staje się coraz bardziej popularnym rozwiązaniem w interakcji człowiek-maszyna, szczególnie w kontekście sterowania urządzeniami na odległość. Integracja rozpoznawania gestów z systemami sterowania wymaga nie tylko skutecznego modelu do identyfikacji gestów, ale także niezawodnych metod przekładania tych gestów na konkretne polecenia dla sterowanego urządzenia (Chen et al., 2020).

Systemy sterowania gestami są stosowane w różnych dziedzinach, gdzie umożliwiają bardziej naturalną i intuicyjną interakcję z urządzeniami (Robin R. Murphy, Carlos Xavier Soto, 2012). Poniżej przedstawiono przegląd takich systemów wraz z przykładami ich praktycznego zastosowania.

- Sterowanie robotami: Gesty ręki są często wykorzystywane do kontrolowania ruchów robotów, takich jak poruszanie się, manipulowanie obiektami, czy wykonywanie złożonych zadań. Przykładem może być sterowanie robotami w fabrykach, gdzie gesty mogą zastąpić tradycyjne kontrolery, zwiększając efektywność i bezpieczeństwo pracy.
- Interakcje w rzeczywistości rozszerzonej (AR) i wirtualnej (VR): W tych środowiskach gesty ręki służą do manipulowania wirtualnymi obiektami, nawigacji w interfejsach użytkownika oraz interakcji w grach. Na przykład w systemach AR/VR gesty mogą być wykorzystywane do przenoszenia przedmiotów, rysowania w przestrzeni czy wchodzenia w interakcje z postaciami wirtualnymi.
- Sterowanie dronami: Drony coraz częściej są sterowane za pomocą gestów, co umożliwia operatorom bardziej precyzyjną kontrolę i elastyczność w sytuacjach, gdzie użycie tradycyjnych kontrolerów jest niewygodne lub niemożliwe. Przykładowo, w fotografii lotniczej gesty mogą być używane do sterowania kamerą i kierunkiem lotu drona.
- Asystenty medyczne: W medycynie gesty ręki pozwalają na sterowanie narzędziami chirurgicznymi oraz innymi urządzeniami medycznymi bez potrzeby ich dotykania, co jest szczególnie ważne w kontekście sterylności na sali operacyjnej. Systemy te są wykorzystywane również w rehabilitacji, gdzie gesty mogą monitorować i wspierać postępy pacjenta.
- Sterowanie multimediami i inteligentnymi domami: W kontekście domowym gesty są wykorzystywane do sterowania urządzeniami multimedialnymi, takimi jak telewizory czy systemy audio, a także do zarządzania inteligentnymi domami, w tym oświetleniem, ogrzewaniem czy systemami bezpieczeństwa.
- Interfejsy w samochodach: Nowoczesne samochody coraz częściej korzystają z systemów gestów do sterowania funkcjami pojazdu, takimi jak nawigacja, regulacja klimatyzacji czy obsługa multimedii, co zwiększa bezpieczeństwo i komfort kierowcy.

Implementacja systemów sterowania gestami niesie ze sobą szereg wyzwań technicznych i praktycznych. Jednym z kluczowych wyzwań jest zapewnienie, że system rozpoznawania gestów będzie działał z wysoką dokładnością, nawet w trudnych warunkach oświetleniowych lub w obecności zakłóceń, takich jak tło złożone z wielu elementów. Kolejnym wyzwaniem jest minimalizacja opóźnień między

wykryciem gestu a reakcją systemu sterowania. W przypadku takich zastosowań jak sterowanie dronem, opóźnienia mogą prowadzić do błędnych operacji i potencjalnie niebezpiecznych sytuacji. Systemy muszą być zdolne do szybkiego przetwarzania danych i podejmowania decyzji w czasie rzeczywistym, co wymaga optymalizacji zarówno na poziomie sprzętu, jak i oprogramowania. Różnorodność urządzeń i systemów, które mogą być sterowane za pomocą gestów, stawia wymagania dotyczące interoperacyjności, co oznacza konieczność opracowania uniwersalnych standardów i protokołów komunikacyjnych. Systemy muszą być w stanie rozpoznawać zarówno proste, jak i bardziej złożone gesty, co może wymagać zaawansowanych algorytmów przetwarzania obrazu oraz uczenia maszynowego (Merlin Stampa, Andreas Sutorma, Uwe Jahn, Jörg Thiem, Carsten Wolff, Christof Roehrig, 2021).

Technologia sterowania gestami oferuje różnorodne podejścia, które różnią się pod względem zastosowanych metod i osiąganych wyników. Wśród nich wyróżnia się sterowanie za pomocą gestów w dwuwymiarowej (2D) i trójwymiarowej (3D) przestrzeni. Podejścia 2D, takie jak gesty na ekranach dotykowych, charakteryzują się prostotą i precyzją, ale są ograniczone do płaskich powierzchni. Natomiast gesty 3D, rozpoznawane przy użyciu sensorów głębi, oferują większą swobodę interakcji, choć wymagają bardziej zaawansowanego sprzętu i algorytmów.

Kolejnym aspektem jest rozpoznawanie gestów statycznych i dynamicznych. Gesty statyczne, jak litery języka migowego, są łatwiejsze do rozpoznania, ponieważ opierają się na pojedynczych obrazach. Z kolei dynamiczne gesty wymagają analizy sekwencji ruchów, co zwiększa złożoność procesu rozpoznawania, ale pozwala na bardziej złożone interakcje.

Różne metody sterowania gestami, takie jak tradycyjne algorytmy przetwarzania obrazu czy nowoczesne techniki oparte na głębokim uczeniu (np. CNN), mają swoje zalety i wady. Tradycyjne metody są mniej wymagające obliczeniowo, ale ich skuteczność jest ograniczona w porównaniu do modeli uczenia maszynowego, które oferują wyższą dokładność kosztem większych zasobów obliczeniowych.

Podsumowując, wybór odpowiedniego podejścia do sterowania gestami zależy od specyficznych wymagań aplikacji, dostępnych zasobów oraz oczekiwanej precyzji i wydajności systemu.

2. Rozdział metodyczny

2.1. Opis bibliotek i narzędzi

W projekcie aplikacji sterującej dronem DJI Tello zastosowano kilka kluczowych technologii i narzędzi, które umożliwiły stworzenie efektywnego systemu sterowania dronem za pomocą gestów ręki. Poniżej przedstawiono główne technologie i narzędzia użyte w projekcie.

Python był głównym językiem programowania zastosowanym w projekcie. Wybrano go ze względu na jego prostotę, wszechstronność oraz bogatą bibliotekę narzędzi, które doskonale wspierają zarówno rozwój aplikacji, jak i implementację modeli sztucznej inteligencji. (Mohit Sewak, Md. Rezaul Karim, Pradeep Pujari, 2018).

Biblioteka djitellopy została wykorzystana do komunikacji z dronem DJI Tello. Jest to biblioteka stworzona specjalnie do obsługi tego modelu drona, umożliwiającą łatwe nawiązywanie połączeń, wysyłanie komend sterujących oraz odbieranie obrazu w czasie rzeczywistym. Djitellopy jest w pełni zintegrowana z Pythonem, co umożliwiło płynne połączenie jej funkcjonalności z innymi narzędziami użytymi w projekcie.

Do przetwarzania obrazu z kamery drona wykorzystano bibliotekę opencv-python. OpenCV jest jedną z najpopularniejszych bibliotek do przetwarzania obrazu, oferującą szeroki wachlarz funkcji, takich jak zmiana rozmiaru obrazu, konwersja kolorów i inne podstawowe operacje, które były niezbędne do przygotowania obrazu na wejście do modelu AI. (Alexander Mordvintsev & Abid K, 2017).

PyTorch i Torchvision zostały użyte do ładowania wytrenowanego modelu sztucznej inteligencji oraz jego wykorzystania do analizy klatek obrazu z kamery drona. PyTorch jest jedną z wiodących bibliotek do pracy z modelami głębokiego uczenia, a Torchvision wspiera przetwarzanie obrazu i integrację z modelami. W aplikacji te biblioteki były odpowiedzialne za ładowanie wytrenowanego modelu oraz analizowanie każdej klatki obrazu w celu rozpoznania gestu ręki (Pradeepta Mishra, 2023).

Do trenowania i testowania modeli sztucznej inteligencji użyto narzędzia Jupyter Notebook, które jest interaktywnym środowiskiem programistycznym opartym na języku Python. Jupyter Notebook umożliwił wygodne wykonywanie kodu krok po kroku, co pozwoliło na szybką analizę wyników i testowanie różnych wersji modelu. Jego elastyczność sprawiła, że praca nad iteracjami modelu AI była znacznie bardziej

efektywna, a dzięki graficznej formie wyników łatwiej było monitorować postępy w trenowaniu modeli.

PyTorch, został użyty również podczas trenowania modelu AI, to jedna z najważniejszych bibliotek do budowy i trenowania modeli głębokiego uczenia. Dzięki swojej elastyczności i możliwości pracy na GPU, PyTorch pozwolił na efektywne trenowanie modeli w ramach projektu. Narzędzie to wspiera dynamiczne grafy obliczeniowe, co umożliwiło łatwe eksperymentowanie z różnymi architekturami sieci neuronowych oraz modyfikowanie ich w trakcie procesu trenowania.

Matplotlib był użyty do wizualizacji wyników trenowania modeli. Jest to popularna biblioteka do tworzenia wykresów w Pythonie, która umożliwia generowanie różnych rodzajów wykresów, takich jak funkcje strat czy dokładności modeli podczas trenowania. Wizualizacja wyników była kluczowa w procesie analizy skuteczności modeli i pozwoliła na identyfikowanie momentów, w których proces uczenia się modelu ulegał stagnacji lub zbyt szybkiemu przetrenowaniu.

2.2. Opis zbioru danych

Podstawą skutecznego trenowania modeli sztucznej inteligencji jest odpowiedni dobór zbioru danych, który powinien być dostosowany do specyfiki zadania. W projekcie dotyczącym sterowania dronem DJI Tello przy pomocy gestów ręki zdecydowano się na wykorzystanie ASL Alphabet Dataset (ASL(*American Sign Language*) *Alphabet Dataset*, 2021) – zbioru danych zawierającego obrazy przedstawiające gesty amerykańskiego języka migowego (American Sign Language, ASL).

Zbiór ASL Alphabet jest powszechnie stosowany w zadaniach związanych z rozpoznawaniem gestów ręki, gdyż oferuje szeroki wachlarz gestów reprezentujących litery alfabetu migowego. Zbiór składa się z obrazów dłoni prezentujących różne litery, co umożliwia modelom głębokiego uczenia efektywne naukę na podstawie różnorodnych układów dłoni. W przypadku niniejszego projektu zbiór ten był idealnym wyborem, ponieważ oferował wystarczającą liczbę klas (liter) oraz liczbę obrazów na każdą literę, co umożliwiło właściwe trenowanie modeli neuronowych (Tilottama Goswami, Shashidhar Reddy Javaji, 2020).

Dane te, dostępne na platformie Kaggle, zostały podzielone na zestaw treningowy oraz testowy, co umożliwiło dokładną walidację i ocenę wydajności modeli.

Zbiór ASL Alphabet składa się z obrazów o wysokiej jakości, co pozwala na skuteczne trenowanie modeli konwolucyjnych sieci neuronowych (CNN), takich jak AlexNet, GoogleNet, ResNet oraz autorskiego modelu CNN opracowanego w ramach projektu.

Aby poprawić efektywność nauki i zwiększyć odporność modeli na zmienne warunki testowe, zdecydowano się także na zastosowanie technik augmentacji danych, takich jak obracanie obrazów, zmiana jasności oraz przekształcenie do skali szarości (grayscale). Te zabiegi miały na celu zwiększenie różnorodności danych oraz poprawę generalizacji modeli w warunkach innych niż te, w których były trenowane.

2.3. Przygotowanie danych

Przed przystąpieniem do trenowania modeli sztucznej inteligencji, kluczowym etapem było odpowiednie przygotowanie danych. Proces ten obejmował kilka istotnych kroków, które miały na celu zapewnienie, że dane wejściowe będą odpowiednio przetworzone i dostosowane do wymagań architektur sieci neuronowych.

Pierwszym krokiem było przekształcenie zbioru danych ASL Alphabet do formatu odpowiedniego dla trenowanych modeli. Obrazy w zestawie danych były w formacie RGB, jednak w celu sprawdzenia wpływu różnych reprezentacji kolorystycznych na wydajność modeli, zdecydowano się na dodatkowe przekształcenie ich do skali szarości (grayscale). Umożliwiło to porównanie, jak modele radzą sobie z danymi kolorowymi w porównaniu do przetworzonych obrazów monochromatycznych.

Kolejnym krokiem było dostosowanie rozmiaru obrazów. Wszystkie obrazy zostały przeskalowane do wymiarów 224x224 pikseli, co jest standardowym rozmiarem wykorzystywanym przez wiele pretrenowanych modeli sieci neuronowych, takich jak AlexNet, GoogleNet czy ResNet. Takie przeskalowanie zapewnia jednolity rozmiar danych wejściowych dla sieci i ułatwia trenowanie.

Ostateczny krok przygotowania danych polegał na podziale zbioru na zestaw treningowy i testowy. Wykorzystano standardowy podział 80/20, gdzie 80% obrazów zostało przeznaczonych do trenowania modeli, a pozostałe 20% do walidacji i testowania. Taki podział zapewnił, że modele mogły być skutecznie trenowane, a ich wydajność była oceniana na danych, których nie widziały wcześniej podczas procesu treningu. Przygotowanie danych stanowiło kluczowy etap, który pozwolił na właściwe przetwarzanie obrazów przez modele AI oraz optymalizację procesu trenowania.

2.4. Wybór architektur sieci neuronowych

W projekcie wykorzystano kilka różnych architektur sieci neuronowych, aby zbadać, która z nich najlepiej nadaje się do rozpoznawania gestów na podstawie obrazu dłoni. Architektury te obejmowały zarówno standardowe, pretrenowane sieci, jak i autorski model CNN stworzony specjalnie na potrzeby tego projektu. (Hyun Min Oh, Hyunki Lee, Min Young Kim, 2019)

Pierwszym modelem zastosowanym w projekcie był AlexNet, który jest jedną z pierwszych popularnych architektur CNN stosowanych w rozpoznawaniu obrazów. AlexNet składa się z pięciu warstw konwolucyjnych, połączonych z trzema w pełni połączonymi warstwami klasyfikacyjnymi. Dzięki relatywnie prostej strukturze, AlexNet nadaje się do przetwarzania danych obrazowych, jednak w przypadku bardziej skomplikowanych wzorców, takich jak gesty, jego zdolności mogą być ograniczone.

Drugim modelem był GoogleNet, który pozwala na efektywne przetwarzanie danych dzięki równoległym operacjom konwolucyjnym i agregacji wyników z różnych filtrów w jednej warstwie. Ta wielowymiarowa analiza obrazów sprawia, że GoogleNet jest bardziej elastyczny w rozpoznawaniu skomplikowanych wzorców.

ResNet (Residual Network) to kolejna architektura użyta w projekcie. Zastosowano dwie jej wersje – ResNet18 oraz ResNet50, które różnią się głębokością sieci. ResNet wprowadza tzw. bloki rezydualne, które umożliwiają lepsze trenowanie bardzo głębokich sieci, unikając problemu zanikania gradientu. Dzięki temu, sieci ResNet mogą osiągać wysoką dokładność w zadaniach związanych z rozpoznawaniem obrazów.

W projekcie przetestowano również MobileNetV2 i MobileNetV3, które są zoptymalizowanymi pod kątem urządzeń mobilnych architekturami sieci neuronowych. MobileNet wykorzystuje tzw. warstwy głębokiej separowalnej konwolucji, które pozwalają na zmniejszenie liczby operacji obliczeniowych, co czyni te modele lekkimi i efektywnymi. Mimo że MobileNet zaprojektowano z myślą o zastosowaniach mobilnych, został przetestowany w tym projekcie ze względu na jego dobrą wydajność w zadaniach związanych z przetwarzaniem obrazu.

Ostatnią przetestowaną siecią był autorski model CNN, zaprojektowany specjalnie na potrzeby projektu. Składał się on z trzech warstw konwolucyjnych, każda z połączoną warstwą normalizacyjną i funkcją aktywacji ReLU, oraz w pełni połączoną warstwą klasyfikacyjną. Celem stworzenia tej sieci było uzyskanie modelu o stosunkowo niewielkiej liczbie parametrów, który byłby w stanie rozpoznawać gesty z obrazu dłoni.

Zastosowanie różnych architektur pozwoliło na porównanie ich wydajności w zadaniu rozpoznawania gestów oraz ocenę, która z nich najlepiej radzi sobie z przetwarzaniem obrazów dłoni w dynamicznych warunkach.

W pracy wykorzystano również autorski model konwolucyjnej sieci neuronowej zaprojektowany specjalnie na potrzebę niniejszych badań. Autorski model CNN, został zaprojektowany z myślą o prostocie i efektywności w rozpoznawaniu gestów dłoni. Model ten składał się z trzech głównych warstw konwolucyjnych, z których każda była połączona z warstwami normalizacyjnymi i funkcjami aktywacji ReLU, co miało na celu przyspieszenie procesu uczenia i stabilizację trenowania. Dodatkowo, zastosowanie warstw poolingowych pozwoliło na redukcję wymiarów obrazu, zachowując jednocześnie istotne cechy.

Pierwsza warstwa konwolucyjna przyjmowała obraz wejściowy o trzech kanałach (kolorowych) i przetwarzała go za pomocą 16 filtrów o rozmiarze 3x3. Następnie obraz przechodził przez warstwę normalizacji BatchNorm2d oraz aktywację ReLU. Redukcja rozmiaru obrazu następowała poprzez zastosowanie warstwy MaxPooling, która zmniejszała wymiary obrazu o połowę.

Druga warstwa konwolucyjna rozszerzała liczbę filtrów do 32, również o rozmiarze 3x3, co pozwalało na identyfikację bardziej złożonych cech obrazu. Proces normalizacji i aktywacji pozostał ten sam, a po nim następowała kolejna warstwa MaxPooling, która ponownie zmniejszała wymiary danych.

Trzecia warstwa konwolucyjna wykorzystywała 64 filtry, które były odpowiedzialne za rozpoznanie bardziej szczegółowych cech obrazu, takich jak struktury dłoni czy układ palców. Po tej warstwie również zastosowano normalizację, aktywację oraz warstwę poolingową, która finalnie redukowała wymiary obrazu, umożliwiając jego dalsze przetwarzanie.

Po zakończeniu procesu przetwarzania przez warstwy konwolucyjne, obraz został "spłaszczony" za pomocą funkcji view, co pozwoliło na przekształcenie danych do postaci wektorowej, odpowiedniej do przetwarzania przez w pełni połączone warstwy neuronowe. Pierwsza z w pełni połączonych warstw składała się z 256 neuronów, połączonych z warstwą normalizacji BatchNorm1d oraz funkcją aktywacji ReLU.

Ostatecznie, na końcu sieci znajdowała się warstwa wyjściowa z odpowiednią liczbą neuronów odpowiadającą liczbie klas w zbiorze danych (w tym przypadku klas

odpowiadających gestom dłoni), przekształcona przez funkcję wyjściową Softmax, która zwracała prawdopodobieństwo przynależności obrazu do danej klasy.

Taka konstrukcja modelu, składająca się z trzech warstw konwolucyjnych i dwóch w pełni połączonych, pozwalała na wydajne rozpoznawanie gestów przy stosunkowo niewielkiej liczbie parametrów. Dodatkowo, model ten został zoptymalizowany pod kątem trenowania i wydajności w kontekście ograniczeń obliczeniowych, jakie może narzucać komputer wykorzystywany w projekcie.

Wybór odpowiednich modeli do rozpoznawania gestów ręki w projekcie opierał się na kilku istotnych kryteriach, które miały na celu zapewnienie wysokiej skuteczności, wydajności oraz możliwości implementacji w rzeczywistych warunkach. Przy selekcji architektur sieci neuronowych uwzględniono przede wszystkim skuteczność modeli w zadaniach klasyfikacji obrazów. Modele, które były brane pod uwagę, musiały wykazywać wysoką efektywność w rozpoznawaniu gestów lub pokrewnych zadaniach, takich jak klasyfikacja liter języka migowego (ASL). W tym kontekście szczególnie preferowano te architektury, które uzyskiwały dobre wyniki na dużych, otwartych zbiorach danych.

Kolejnym kluczowym kryterium była szybkość i wydajność modeli. Ze względu na ograniczone zasoby obliczeniowe i wymóg przetwarzania obrazów w czasie rzeczywistym, wybór padł na architektury zoptymalizowane pod kątem pracy w warunkach ograniczonych zasobów, takie jak MobileNet. Szybkie działanie modeli było istotne, by możliwe było efektywne przetwarzanie obrazu z kamery drona, bez opóźnień, które mogłyby wpłynąć negatywnie na wyniki rozpoznawania gestów.

Równie ważna była złożoność architektury. Lżejsze modele, takie jak MobileNetV2 i MobileNetV3, wyróżniały się niską liczbą parametrów. Z kolei bardziej złożone architektury, takie jak ResNet50 czy GoogleNet, były wybierane z myślą o sprawdzeniu, czy większa liczba warstw i głębsze sieci przyniosą znaczące poprawy w skuteczności rozpoznawania gestów.

Wykorzystanie pretrenowanych modeli również odgrywało ważną rolę. Modele takie jak AlexNet, ResNet, GoogleNet, VGGNet oraz MobileNet oferowały możliwość skorzystania z pretrenowanych wag na zbiorze ImageNet, co pozwoliło na przyspieszenie procesu treningu poprzez transfer learning. Pozwalało to również na osiągnięcie lepszych wyników, zwłaszcza przy ograniczonych danych dostępnych w ramach projektu (Dumitru Erhan, Aaron Courville, Yoshua Bengio, Pascal Vincent, 2010).

Istotnym aspektem była także możliwość rozbudowy i modyfikacji architektur. Stworzono autorski model CNN, który miał na celu dostosowanie architektury sieci neuronowej do specyficznych wymagań projektu. Model ten pozwalał na większą elastyczność w kontekście optymalizacji pod kątem rozpoznawania gestów ręki, co było ważnym elementem eksperymentów prowadzonych w projekcie.

Ostatecznym kryterium była zdolność modeli do adaptacji do zmiennych warunków środowiskowych. Ze względu na konieczność testowania systemu w rzeczywistych warunkach, modele musiały być w stanie radzić sobie z różnorodnym oświetleniem, tłem oraz innymi zmiennymi czynnikami. Modele były oceniane pod kątem ich odporności na te zmiany, co miało kluczowe znaczenie w kontekście zastosowania systemu w praktyce (Yann LeCun, Yoshua Bengio, Geoffrey Hinton, 2015).

2.5. Proces trenowania modeli AI

Przygotowanie wybranych architektur sieci neuronowych do procesu trenowania wymagało kilku kroków, które miały na celu dostosowanie modeli do specyficznego zadania rozpoznawania gestów ręki. Pierwszym krokiem było załadowanie pretrenowanych modeli, takich jak GoogleNet, ResNet, MobileNet, AlexNet czy VGGNet, które zostały wstępnie wytrenowane na dużym zbiorze danych ImageNet (Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, 2014). Modele te miały już zdefiniowane wagi i struktury, które zostały zoptymalizowane pod kątem klasyfikacji ogólnych obrazów, co pozwalało na wykorzystanie transfer learningu do zadania rozpoznawania gestów.

Następnie konieczne było zamrożenie wag w początkowych warstwach każdego z modeli. Zamrożenie wag oznaczało, że podczas treningu aktualizowane były jedynie wagi ostatniej warstwy klasyfikacyjnej, co umożliwiało szybsze trenowanie modelu oraz lepsze dostosowanie go do danych specyficznych dla zadania rozpoznawania gestów ręki. W szczególności zamrażanie wag było używane w modelach o złożonej architekturze, takich jak ResNet50, aby zminimalizować czas trenowania i zapobiec nadmiernemu dopasowaniu modelu do stosunkowo niewielkiego zbioru danych.

Kolejnym krokiem było dostosowanie warstwy klasyfikacyjnej do liczby klas w zadaniu rozpoznawania liter języka migowego (ASL). Modele takie jak AlexNet, GoogleNet, czy VGGNet zostały zmodyfikowane, aby ostatnia warstwa w pełni

połączona (fully connected) odpowiadała liczbie klas w zbiorze danych, czyli 29 klasom, odpowiadającym 26 literom alfabetu oraz trzem dodatkowym gestom (np. spacja). Dostosowanie ostatniej warstwy było kluczowe, aby model mógł prawidłowo klasyfikować gesty ręki.

W przypadku autorskiego modelu CNN, przygotowanie obejmowało zdefiniowanie architektury sieci od podstaw. Model składał się z trzech warstw konwolucyjnych, połączonych z warstwami normalizacyjnymi oraz funkcjami aktywacji ReLU, a następnie zakończony warstwą klasyfikacyjną. Ten proces wymagał odpowiedniego skonfigurowania liczby filtrów w każdej warstwie oraz parametrów takich jak rozmiar filtrów, kroki (strides) i padding, co miało zapewnić efektywne przetwarzanie obrazów gestów.

Ostatnim etapem było przeniesienie modeli na GPU, aby przyspieszyć proces trenowania. Wykorzystanie karty graficznej umożliwiło szybsze przetwarzanie danych oraz równoczesne wykonywanie operacji na większych partiach danych (batchach), co znacząco skróciło czas potrzebny na wytrenowanie modeli. Dzięki temu można było zoptymalizować proces trenowania i osiągnąć lepsze wyniki w krótszym czasie.

W procesie trenowania modeli sztucznej inteligencji jednym z kluczowych elementów jest dobór odpowiedniej funkcji strat oraz optymalizatora, które mają bezpośredni wpływ na skuteczność nauki modelu. W przypadku projektu rozpoznawania gestów ręki wykorzystano funkcję strat typu Cross-Entropy Loss, która jest standardem w zadaniach klasyfikacyjnych. Funkcja ta mierzy różnicę między przewidywanymi a rzeczywistymi etykietami klas, co pozwala na ocenę, jak dobrze model uczy się na danych treningowych.

Funkcja Cross-Entropy Loss jest szczególnie użyteczna w problemach wieloklasowych, takich jak rozpoznawanie gestów, ponieważ uwzględnia prawdopodobieństwo przypisania przykładu do każdej z dostępnych klas. Podczas trenowania modelu dąży się do minimalizacji tej funkcji, co oznacza, że przewidywania modelu coraz bardziej zbliżają się do rzeczywistych etykiet klas.

Równie istotnym elementem procesu trenowania był dobór optymalizatora. W projekcie wykorzystano optymalizator Adam (Adaptive Moment Estimation), który jest szeroko stosowany w zadaniach związanych z głębokim uczeniem. Optymalizator ten jest ulepszoną wersją klasycznego algorytmu gradientu prostego (Stochastic Gradient Descent) i charakteryzuje się tym, że automatycznie dostosowuje tempo nauki dla każdego parametru modelu, co pozwala na szybszą i bardziej stabilną konwergencję.

Optymalizator Adam korzysta z dwóch mechanizmów: średniej ruchomej gradientów oraz kwadratów gradientów, co pozwala na lepsze radzenie sobie z lokalnymi minimami i złożonymi funkcjami strat. W trakcie trenowania modeli AI w tym projekcie, optymalizator ten pozwalał na szybkie aktualizowanie wag modelu, co przekładało się na szybszą poprawę wyników w porównaniu do bardziej klasycznych optymalizatorów.

Podczas eksperymentów z trenowaniem modeli, dobór hiperparametrów, takich jak współczynnik uczenia (learning rate), również odgrywał istotną rolę. Początkowy współczynnik uczenia został ustawiony na wartość 0.001, co pozwoliło na stabilną i efektywną naukę modeli. W niektórych przypadkach, np. w późniejszych etapach trenowania, współczynnik ten był modyfikowany, aby dostosować tempo nauki w zależności od osiągniętych wyników.

Wnioski z procesu doboru funkcji strat i optymalizatora sugerują, że Cross-Entropy Loss oraz optymalizator Adam stanowią skuteczne narzędzia w problemach klasyfikacyjnych związanych z rozpoznawaniem gestów ręki, zapewniając stabilną konwergencję oraz wysoką efektywność trenowania modeli.

2.6. Metody oceny skuteczności modeli

Ocena skuteczności modeli sztucznej inteligencji w zadaniach klasyfikacyjnych, takich jak rozpoznawanie gestów ręki, wymaga zastosowania odpowiednich metryk, które pozwalają na dokładną analizę wyników uzyskanych przez modele. W projekcie tym zastosowano kilka standardowych metryk, które umożliwiają obiektywną ocenę wydajności modeli.

Podstawową metryką stosowaną do oceny modeli była dokładność (accuracy), która jest najczęściej używaną miarą w zadaniach klasyfikacyjnych. Dokładność mierzy stosunek liczby poprawnie sklasyfikowanych przykładów do całkowitej liczby przykładów w zbiorze testowym. Wyrażana jest w procentach i informuje, jak dobrze model radzi sobie z klasyfikacją na poziomie ogólnym. Choć dokładność jest intuicyjna i prosta do zrozumienia, nie zawsze jest wystarczająca w przypadku nie zrównoważonych zbiorów danych, gdzie jedna klasa może dominować nad innymi.

Kolejną metryką używaną w projekcie było strata (loss), która jest wartością bezpośrednio wyliczaną przez funkcję strat (w tym przypadku Cross-Entropy Loss). Strata jest używana do monitorowania procesu trenowania modeli i informuje o tym,

jak dobrze model uczy się w kolejnych epokach. Niska wartość funkcji strat sugeruje, że model dobrze dopasowuje się do danych treningowych.

Po zakończeniu etapu trenowania modeli sztucznej inteligencji, kluczowym krokiem w procesie oceny ich wydajności było testowanie na odpowiednio przygotowanych zbiorach testowych. Zbiory testowe, w przeciwieństwie do zbiorów treningowych, zawierają dane, których model nie widział wcześniej. Dzięki temu możliwe jest uzyskanie obiektywnej miary skuteczności modelu oraz jego zdolności do generalizacji.

Testowanie modeli w projekcie opierało się na zastosowaniu wybranego zbioru danych ASL, który zawierał obrazy gestów ręki reprezentujące litery alfabetu języka migowego. Każdy model został oceniony na podstawie swojej zdolności do prawidłowego rozpoznania gestów w zbiorze testowym, w którym znalazły się obrazy różniące się od tych użytych w fazie trenowania. Zbiór testowy zapewniał różnorodność pod względem ujęć dłoni oraz warunków oświetleniowych, co umożliwiło wszechstronną ocenę modeli.

W ramach testowania, dla każdego z modeli – takich jak AlexNet, GoogleNet, ResNet czy autorski model CNN – zostały przeprowadzone pomiary na tych samych danych testowych. Wyniki testów były zapisywane i analizowane w kontekście przewidywań modeli na poszczególnych obrazach (Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, 2016).

Pomimo wysokiej skuteczności niektórych modeli w fazie trenowania, testowanie na rzeczywistych, nowych danych testowych ujawniło pewne trudności, zwłaszcza w rozpoznawaniu bardziej skomplikowanych gestów lub w warunkach odbiegających od idealnych. Na przykład, niektóre modele miały problemy z rozpoznaniem litery "G", szczególnie w warunkach zmiennego oświetlenia i różnorodnych ujęć dłoni.

Testowanie modeli w rzeczywistych warunkach pozwoliło także na weryfikację ich stabilności oraz odporności na szумы i zakłócenia w danych, co jest kluczowe w praktycznym zastosowaniu systemów rozpoznawania gestów. Ostateczne wyniki testów umożliwiły ocenę, które z zastosowanych modeli są najbardziej efektywne w rozpoznawaniu gestów ręki i jakie usprawnienia mogą być wprowadzone w przyszłych iteracjach projektu.

3. Rozdział utylitarny

3.1. Opis aplikacji sterującej dronem

Proces tworzenia aplikacji sterującej dronem DJI Tello za pomocą gestów ręki wymagał kilku kluczowych etapów, od konfiguracji środowiska programistycznego, przez implementację, aż po testowanie i optymalizację. W trakcie realizacji projektu napotkano na różne wyzwania, które skłoniły do eksperymentowania z różnymi podejściami i technologiami.

Pierwszym krokiem było przygotowanie odpowiedniego środowiska programistycznego. W celu izolacji środowiska oraz ułatwienia zarządzania zależnościami, wykorzystano narzędzie virtualenv, które umożliwia tworzenie odizolowanych środowisk wirtualnych dla projektów Python.

Aby utworzyć nowe środowisko wirtualne, wykonano następujące kroki (Listing 1):

```
# Instalacja virtualenv, jeśli nie jest jeszcze zainstalowany
pip install virtualenv

# Utworzenie nowego środowiska wirtualnego
virtualenv venv

# Aktywacja środowiska wirtualnego
source venv/bin/activate # Na systemach Linux/macOS
venv\Scripts\activate   # Na systemach Windows
```

Listing 1. Tworzenie i aktywacja środowiska wirtualnego.

Wszystkie zależności projektu zostały zainstalowane w ramach tego środowiska wirtualnego. Zainstalowano je za pomocą pliku requirements.txt, który zawiera listę wszystkich niezbędnych pakietów. Aby zainstalować zależności, wykonano (Listing 2):

```
pip install -r requirements.txt
```

Listing 2. Instalacja zależności z pliku requirements.txt.

Plik requirements.txt wyglądał następująco (Listing 3):

```
djitellopy
opencv-python
torch
torchvision
```

Listing 3. Zawartość pliku requirements.txt.

Po skonfigurowaniu środowiska przystąpiono do tworzenia aplikacji. Podczas implementacji aplikacji podjęto próby realizacji projektu w dwóch głównych technologiach: Node.js oraz Pythonie, zarówno z wykorzystaniem gotowych bibliotek, jak i za pomocą rozwiązań niskopoziomowych, opartych na bezpośrednich połączeniach UDP (H.H. Khairi, Sharifah H. S. Ariffin, N. M. Abdul Latiff, Kamaludin Mohamad Yusof, M. K. Hassan, Mohammad Rava., 2020).

W Node.js, początkowo skorzystano z dostępnych bibliotek, które miały ułatwić komunikację z dronem DJI Tello oraz odbieranie strumienia wideo. Mimo że biblioteki te znacząco uprościły implementację podstawowych funkcji, pojawiły się problemy z niezawodnością odbioru obrazu, co skutkowało tym, że obraz z kamery drona nie był wyświetlany. W odpowiedzi na te trudności, zdecydowano się na napisanie aplikacji bez użycia bibliotek, polegając wyłącznie na bezpośrednim połączeniu UDP. Chociaż udało się nawiązać połączenie i wysyłać komendy do drona, problemy z odbiorem i wyświetlaniem obrazu nadal występowały.

Podobne podejście zastosowano w Pythonie, gdzie początkowo wykorzystano dedykowaną bibliotekę djitellopy do zarządzania dronem. Choć biblioteka ta umożliwiła szybkie uruchomienie podstawowych funkcji drona, problemy z odbiorem i wyświetlaniem obrazu z kamery były zbliżone do tych napotkanych w Node.js. W dalszej kolejności, w Pythonie podjęto próbę implementacji aplikacji bez użycia gotowych bibliotek, opierając się na bezpośrednim połączeniu UDP. Mimo poprawnego nawiązania połączenia, obraz z kamery drona także nadal nie był wyświetlany.

Ostatecznie, po dłuższej analizie, okazało się, że problemem był zaporę sieciową na komputerze, która blokowała ruch UDP niezbędny do przesyłania strumienia wideo. Aby odblokować odpowiednie porty na firewallu w systemie Linux z użyciem narzędzia UFW, wykonano następujące komendy (Listing 4).

```
sudo ufw allow 8889/udp
sudo ufw allow 11111/udp
sudo ufw reload
```

Listing 4. Komendy do odblokowania portów na firewallu za pomocą UFW.

Aby zweryfikować działanie strumienia wideo, podjęto także próby wyświetlenia obrazu z drona przy użyciu zewnętrznych narzędzi, takich jak mpv i FFmpeg. Narzędzia te pozwoliły na testowanie odbioru wideo bez konieczności integracji z aplikacją Python. Wyniki tych testów wskazały na istnienie znaczących opóźnień,

sięgających 10-20 sekund, które były niezależne od samej aplikacji, co wskazywało na problemy związane z przesyłem danych.

Podstawowym elementem aplikacji było nawiązanie połączenia z dronem DJI Tello. W tym celu wykorzystano bibliotekę `djitellopy`, która umożliwiła nawiązanie połączenia z dronem poprzez WiFi. Kluczowym krokiem było skonfigurowanie aplikacji tak, aby mogła wysyłać komendy do drona oraz odbierać strumień wideo z kamery. Przykład kodu do nawiązania połączenia z dronem przedstawiono w listingu 5.

```
from djitellopy import Tello

tello = Tello()
tello.connect()
tello.streamon()
```

Listing 5. Kod do nawiązania połączenia z dronem DJI Tello.

Początkowo napotkano problemy z uzyskaniem płynnego obrazu z kamery drona. Po kilku próbach okazało się, że przyczyną były ustawienia zapory sieciowej na komputerze, która blokowała ruch UDP niezbędny do przesyłania strumienia wideo. Aby odblokować odpowiednie porty na firewallu w systemie Linux z użyciem narzędzia UFW, wykonano następujące komendy (Listing 6).

```
sudo ufw allow 8889/udp
sudo ufw allow 11111/udp
sudo ufw reload
```

Listing 6. Komendy do odblokowania portów na firewallu za pomocą UFW.

Po uzyskaniu strumienia wideo z kamery drona, przystąpiono do przetwarzania obrazu w czasie rzeczywistym. W tym celu wykorzystano bibliotekę OpenCV. Każda klatka była skalowana do odpowiedniego rozmiaru i konwertowana na format akceptowalny przez model sztucznej inteligencji. Przykład przetwarzania klatki obrazu przedstawiono w listingu 7.

```
import cv2

frame = tello.get_frame_read().frame
frame = cv2.resize(frame, (224, 224))
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

Listing 7. Przetwarzanie obrazu z kamery drona za pomocą OpenCV.

Kolejnym krokiem była integracja modelu sztucznej inteligencji, który miał za zadanie rozpoznawać gesty ręki na podstawie przetworzonych klatek wideo. Model AI, wytrenowany wcześniej przy użyciu PyTorch, był ładowany do aplikacji i używany do analizy każdej klatki obrazu. Przykład kodu do ładowania modelu i przetwarzania obrazu przedstawiono w listingu 8.

```
import torch

model = torch.load("path_to_model.pth")
frame_tensor = torch.tensor(frame, dtype=torch.float32).permute(2, 0, 1).unsqueeze(0)

with torch.no_grad():
    output = model(frame_tensor)
```

Listing 8. Ładowanie modelu AI i przetwarzanie obrazu w aplikacji.

Model analizował każdą klatkę wideo i zwracał przewidywany gest, który był następnie przetwarzany przez aplikację. Początkowo wyniki były wyświetlane w formie tekstowej (Tabela 9):

```
print("Output: ", output)
```

Listing 9. Wyświetlanie wyników rozpoznawania gestów.

Po rozwiązaniu problemów z zaporą sieciową oraz poprawnym odbieraniem strumienia wideo, aplikacja została doprowadzona do stanu funkcjonalnego. Ostateczna wersja umożliwia nawiązywanie połączenia z dronem DJI Tello, odbiór obrazu z jego kamery oraz przetwarzanie każdej klatki w czasie rzeczywistym. Mimo pewnych wyzwań, takich jak opóźnienia w przesyłaniu obrazu, aplikacja realizuje podstawowe funkcje związane z obsługą drona.

Głównym problemem było znaczne opóźnienie w przesyłaniu obrazu z kamery drona, co utrudniało realizację zadania rozpoznawania gestów w czasie rzeczywistym. Początkowo podejrzewano problemy z zaporą sieciową, jednak po ich wyeliminowaniu okazało się, że opóźnienia wynikały z ograniczeń protokołu UDP, który odpowiada za transmisję danych. Opóźnienia rzędu 10-20 sekund sprawiały, że analiza gestów w czasie rzeczywistym była niemożliwa.

Integracja modelu sztucznej inteligencji (AI) również napotkała trudności. Mimo że model został poprawnie zaimplementowany w aplikacji, to nie był w stanie skutecznie rozpoznawać gestów na podstawie przesyłanych klatek wideo. Problemy z opóźnieniami oraz brak odpowiedniej jakości danych do trenowania modelu AI sprawiły, że model działał nieprawidłowo, co znacząco ograniczyło możliwości aplikacji.

Po zakończeniu implementacji, aplikację poddano testom w rzeczywistych warunkach. Testy miały na celu ocenę stabilności połączenia z dronem, responsywności interfejsu oraz ogólnej funkcjonalności aplikacji. Podczas testów sprawdzono, czy dron prawidłowo reaguje na podstawowe komendy, takie jak start, lądowanie oraz obrót, a także monitorowano stabilność strumienia wideo i płynność wyświetlania kolejnych klatek.

Testy ujawniły, że opóźnienia w transmisji obrazu wynoszą od 10 do 20 sekund, co znacznie ograniczało możliwość efektywnego rozpoznawania gestów. Chociaż aplikacja stabilnie nawiązywała połączenie z dronem i poprawnie wykonywała podstawowe polecenia, problemy z opóźnieniami uniemożliwiały prawidłowe działanie modelu AI. Wyniki testów wskazują, że aplikacja działa poprawnie w zakresie zarządzania dronem, jednakże problemy z transmisją wideo i działaniem AI stanowią istotne wyzwanie dla dalszego rozwoju.

Podsumowując, aplikacja jest w stanie nawiązać połączenie z dronem, odbierać strumień wideo i przetwarzać obrazy, jednak problemy z działaniem modelu AI oraz duże opóźnienia w transmisji wideo ograniczają jej praktyczne zastosowanie. Aby aplikacja mogła być używana w rzeczywistych warunkach, takich jak inspekcje techniczne lub zdalne sterowanie, konieczna jest dalsza optymalizacja, w szczególności w zakresie stabilności połączenia i działania modelu sztucznej inteligencji.

3.2. Opis modelu AI

W ramach prac nad modelem rozpoznawania gestów ręki podjęto decyzję o wykorzystaniu zbioru danych American Sign Language (ASL) Alphabet Dataset. Zbiór ten jest jednym z najpopularniejszych zestawów danych przeznaczonych do rozpoznawania liter alfabetu amerykańskiego języka migowego. Jego wybór był

podyktowany dużym rozmiarem oraz różnorodnością obrazów, co stanowiło solidną podstawę do efektywnego trenowania modeli głębokiego uczenia.

Dane zostały pobrane z platformy Kaggle, która jest jednym z największych repozytoriów zbiorów danych dla celów naukowych i badawczych. ASL Alphabet Dataset składa się z około 87 000 obrazów, przy czym każda z 29 klas (reprezentujących 26 liter alfabetu, spacji oraz braku znaku) zawiera po około 3000 zdjęć. Obrazy przedstawiają dłonie wykonujące gesty odpowiadające literom alfabetu ASL, co czyni ten zbiór idealnym do zastosowań związanych z rozpoznawaniem gestów ręki. Każde zdjęcie ma rozmiar 200x200 pikseli, co umożliwia bezpośrednie wykorzystanie ich w modelach sieci neuronowych konwolucyjnych (CNN).

Przygotowanie danych do trenowania modeli obejmowało kilka kluczowych etapów. Po pierwsze, obrazy zostały przeskalowane do rozmiaru 200x200 pikseli i znormalizowane, wykorzystując standardowe wartości średniej i odchylenia standardowego dla danych RGB. Kolejnym krokiem był podział danych na zbiór treningowy i testowy w proporcjach 80:20, co pozwoliło na skuteczne trenowanie modeli, jednocześnie zachowując odpowiedni zestaw danych do walidacji (Christina Bornberg 2020).

W kolejnych próbach, w celu uproszczenia danych wejściowych oraz zbadania potencjalnych korzyści z tego płynących, obrazy zostały przetworzone do skali szarości. Chociaż przekształcenie obrazów w odcienie szarości miało na celu zmniejszenie złożoności modelu i przyspieszenie procesu treningu, okazało się, że taka modyfikacja nie przyniosła oczekiwanej poprawy w dokładności rozpoznawania gestów. Niemniej jednak, wszystkie te eksperymenty przyczyniły się do lepszego zrozumienia wpływu przetwarzania danych na wyniki modeli sztucznej inteligencji.

W projekcie kluczowym elementem było dobranie odpowiednich architektur sieci neuronowych do zadania rozpoznawania gestów ręki na podstawie obrazów z kamery drona. Aby wybrać najbardziej efektywne rozwiązanie, zdecydowano się na porównanie kilku sprawdzonych architektur głębokiego uczenia, które są szeroko stosowane w dziedzinie przetwarzania obrazów. Wybrane modele to AlexNet, GoogleNet, ResNet, VGG, MobileNet oraz autorski model typu Convolutional Neural Network (CNN). Wybór tych modeli był podyktowany ich różnorodnymi właściwościami i potencjalną przydatnością w kontekście ograniczeń sprzętowych oraz wymagań dotyczących szybkości i dokładności rozpoznawania gestów (Angga Prima Syahputra, Alda Cendekia Siregar, Rachmat Wahid Saleh Insani, 2023).

AlexNet to jedna z pierwszych sieci neuronowych, która osiągnęła sukces w klasyfikacji obrazów, charakteryzując się relatywnie prostą architekturą. Sieć ta składa się z kilku warstw konwolucyjnych, po których następują warstwy w pełni połączone. Dzięki tej architekturze AlexNet był w stanie osiągnąć znaczące wyniki w konkursie ImageNet w 2012 roku, co sprawiło, że stał się punktem odniesienia dla przyszłych badań nad sieciami neuronowymi.

GoogleNet jest bardziej zaawansowaną siecią, która wprowadza koncepcję modułów inception. Te moduły pozwalają na równoczesne wykonywanie konwolucji z różnymi rozmiarami filtrów, co umożliwia lepsze przetwarzanie informacji o różnych skalach w jednym kroku sieci. Dzięki swojej zaawansowanej architekturze, GoogleNet zdobył pierwsze miejsce w konkursie ImageNet w 2014 roku.

MobileNet V2 i MobileNet V3 zostały wybrane ze względu na ich wydajność oraz optymalizację pod kątem przetwarzania obrazów w czasie rzeczywistym. Sieci te charakteryzują się mniejszą liczbą parametrów oraz zmniejszonym zapotrzebowaniem na zasoby obliczeniowe, co sprawia, że są dobrze dopasowane do zadań wymagających szybkiego przetwarzania i analizy obrazów.

ResNet18 i ResNet50 to sieci o głębokiej architekturze, które wprowadzają tzw. skip connections, czyli dodatkowe połączenia między warstwami, pozwalające na skuteczniejsze trenowanie bardzo głębokich sieci. Architektury te stały się popularne dzięki możliwości trenowania bardzo głębokich sieci bez efektu zanikania gradientu, co jest częstym problemem w sieciach o dużej liczbie warstw.

VGG16 i VGG19 to sieci o prostszej, ale głębokiej strukturze, które składają się z wielu warstw konwolucyjnych o małych filtrach (Syed Rehan Shah, Salman Qadri, Hadia Bibi, Syed Muhammad Waqas Shah, Muhammad Imran Sharif, Francesco Marinello, 2023). Dzięki temu sieci te są w stanie uchwycić szczegóły na różnych poziomach abstrakcji, co sprawia, że są one bardzo skuteczne w klasyfikacji obrazów. Ich architektura jest jednak bardziej zasobochłonna w porównaniu do sieci takich jak MobileNet.

Dodatkowo, stworzono autorski model CNN, który miał na celu dostosowanie architektury sieci neuronowej do specyficznych wymagań projektu. Model ten składa się z trzech warstw konwolucyjnych, z których każda jest połączona z warstwami normalizacyjnymi i funkcjami aktywacji ReLU. Warstwy te są następnie łączone z w pełni połączoną warstwą klasyfikacyjną. Taka konstrukcja miała na celu stworzenie sieci, która byłaby zdolna do efektywnego rozpoznawania gestów ręki przy

zachowaniu stosunkowo niewielkiej liczby parametrów. W listingu 10 przedstawiono implementację autorskiego modelu w języku Python, z użyciem biblioteki PyTorch:

```
import torch
import torch.nn as nn

class CustomCNN(torch.nn.Module):
    def __init__(self, num_classes):
        super(CustomCNN, self).__init__()

        # Warstwa konwolucyjna 1
        self.conv1 = torch.nn.Conv2d(in_channels=3, out_channels=16,
kernel_size=3, stride=1, padding=1)
        self.bn1 = torch.nn.BatchNorm2d(16)
        self.relu1 = torch.nn.ReLU()
        self.pool1 = torch.nn.MaxPool2d(kernel_size=2, stride=2,
padding=0)

        # Warstwa konwolucyjna 2
        self.conv2 = torch.nn.Conv2d(in_channels=16, out_channels=32,
kernel_size=3, stride=1, padding=1)
        self.bn2 = torch.nn.BatchNorm2d(32)
        self.relu2 = torch.nn.ReLU()
        self.pool2 = torch.nn.MaxPool2d(kernel_size=2, stride=2,
padding=0)

        # Warstwa konwolucyjna 3
        self.conv3 = torch.nn.Conv2d(in_channels=32, out_channels=64,
kernel_size=3, stride=1, padding=1)
        self.bn3 = torch.nn.BatchNorm2d(64)
        self.relu3 = torch.nn.ReLU()
        self.pool3 = torch.nn.MaxPool2d(kernel_size=2, stride=2,
padding=0)

        # W pełni połączona warstwa
        self.fc1 = torch.nn.Linear(64 * 25 * 25, 256) # Rozmiar po
pooling: 64 * (200/8) * (200/8)
        self.bn4 = torch.nn.BatchNorm1d(256)
        self.relu4 = torch.nn.ReLU()

        # Wyjściowa warstwa klasyfikacji
        self.fc2 = torch.nn.Linear(256, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu1(x)
        x = self.pool1(x)

        x = self.conv2(x)
        x = self.bn2(x)
        x = self.relu2(x)
        x = self.pool2(x)

        x = self.conv3(x)
        x = self.bn3(x)
        x = self.relu3(x)
        x = self.pool3(x)

        x = x.view(x.size(0), -1) # Flattenowanie
```

```

x = self.fc1(x)
x = self.bn4(x)
x = self.relu4(x)

x = self.fc2(x)
return x

```

Listing 10. Implementacja autorskiego modelu CNN.

3.3. Proces treningu modelu

Proces treningu modelu sztucznej inteligencji był kluczowym etapem, który polegał na dostosowaniu wybranych architektur sieci neuronowych do specyficznego zadania rozpoznawania gestów ręki (Daniel Mas MontserratQian LinJan AllebachEdward J. Delp 2017). W projekcie zastosowano podejście transfer learningu, które umożliwiło wykorzystanie wcześniej wytrenowanych modeli, takich jak AlexNet, GoogleNet, ResNet18, ResNet50, MobileNet V2, MobileNet V3, VGG16, VGG19 oraz autorskiego modelu CNN.

Na początku procesu treningu, większość wag w modelach została zamrożona, co oznacza, że nie były one aktualizowane podczas treningu. Dzięki temu możliwe było skupienie się na dostosowaniu ostatnich warstw sieci do specyficznego zadania rozpoznawania gestów ręki. W listingu 11 przedstawiono kod, który demonstruje zamrażanie wag w modelu:

```

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

```

Listing 11. Kod zamrażający wagi w modelu.

Następnie, ostatnia w pełni połączona warstwa w modelach została dostosowana do liczby klas odpowiadających literom alfabetu ASL. W zależności od architektury sieci neuronowej, modyfikacja ta wyglądała nieco inaczej dla każdego modelu.

- AlexNet - przedstawione w listingu 12:

```

# Modify the last fully connected layer to fit your number of classes
model.classifier[6] = nn.Linear(model.classifier[6].in_features,
num_classes)

```

Listing 12 Kod dostosowujący ostatnią warstwę w modelu AlexNet do liczby klas.

- GoogleNet - przedstawione w listingu 13:

```

# Modify the last fully connected layer to fit your number of classes
model.fc = nn.Linear(model.fc.in_features, num_classes)

```

Listing 13 Kod dostosowujący ostatnią warstwę w modelu GoogleNet do liczby klas.

- ResNet18 / ResNet50 - przedstawione w listingu 14:

```
# Modify the last fully connected layer to fit your number of classes
model.fc = nn.Linear(model.fc.in_features, num_classes)
```

Listing 14 Kod dostosowujący ostatnią warstwę w modelu ResNet18 / ResNet50 do liczby klas

- MobileNet V2 - przedstawione w listingu 15:

```
# Modify the last fully connected layer to fit your number of classes
model.classifier[1] = nn.Linear(model.classifier[1].in_features,
num_classes)
```

Listing 15 Kod dostosowujący ostatnią warstwę w modelu MobileNet V2 do liczby klas

- MobileNet V3 - przedstawione w listingu 16:

```
# Modify the last fully connected layer to fit your number of classes
model.classifier[3] = nn.Linear(model.classifier[3].in_features,
num_classes)
```

Listing 16 Kod dostosowujący ostatnią warstwę w modelu MobileNet V3 do liczby klas

- VGG16 / VGG19 - przedstawione w listingu 17:

```
# Modify the last fully connected layer to fit your number of classes
model.classifier[6] = nn.Linear(model.classifier[6].in_features,
num_classes)
```

Listing 17 Kod dostosowujący ostatnią warstwę w modelu VGG16 / VGG19 do liczby klas.

Następnie model został przeniesiony na GPU (jeśli było dostępne), aby przyspieszyć proces treningu. W listingu 18 przedstawiono kod odpowiedzialny za przeniesienie modelu na GPU:

```
# Move the model to the device (GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

Listing 18. Kod przenoszący model na GPU.

Kolejnym krokiem było zdefiniowanie funkcji straty oraz optymalizatora, które są niezbędne do procesu treningu modelu. W projekcie zastosowano funkcję straty CrossEntropyLoss, a optymalizacja była prowadzona za pomocą optymalizatora Adam z początkową wartością współczynnika uczenia ustawioną na 0.001. W listingu 19 przedstawiono kod definiujący te elementy:

```
import torch
import torch.nn as nn
import torch.optim as optim
```



```
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Listing 19. Definicja funkcji straty i optymalizatora.

Do przeprowadzenia treningu wykorzystano specjalnie zaprojektowaną funkcję “train_model”, która zautomatyzowała proces treningu i umożliwiła łatwe śledzenie postępów modelu w każdej epoce. Funkcja ta przyjmowała model, zestaw danych treningowych, funkcję straty oraz optymalizator jako argumenty, a następnie przeprowadzała trening przez zdefiniowaną liczbę epok. W listingu 20 przedstawiono implementację tej funkcji:

```
def train_model(model, train_loader, criterion, optimizer, device,
num_epochs=10):
    epochs_losses = []
    for epoch in range(num_epochs):
        model.train()
        epoch_loss = 0.0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item() * inputs.size(0)
        epoch_loss /= len(train_loader.dataset)
        epochs_losses.append(epoch_loss)
        print(f"Epoch {epoch+1}/{num_epochs}, Loss: {epoch_loss:.4f}")
    return epochs_losses
```

Listing 20. Funkcja treningu modelu “train_model”.

Na początku procesu treningu, modele i ich lossy nie były zapisywane do plików wynikowych. Powodowało to konieczność ponownego trenowania wszystkich modeli po każdej modyfikacji w kodzie lub w przypadku, gdy trening musiał zostać przerwany (na przykład z powodu konieczności wyłączenia komputera). W odpowiedzi na te wyzwania, wprowadzono mechanizm zapisywania wytrenowanych modeli oraz odpowiadających im strat do plików. Dzięki temu możliwe było pomijanie modeli, które już zostały wytrenowane, co znacząco przyspieszyło iteracyjny proces doskonalenia sieci. Jeśli zachodziła potrzeba ponownego treningu konkretnego modelu, wystarczyło usunąć istniejący plik wynikowy, aby model został przetrenowany od nowa. Przykładowo, wytrenowany model AlexNet był zapisywany w sposób przedstawiony w listingu 21:

```
# Save the model and losses
torch.save(AlexNet_model, 'tmp/models/ASL_AlexNet_model_V1.0.0.pth')
```

```
with open('tmp/model_losses/ASL_AlexNet_model_losses_V1.0.0.json', 'w')
as filehandle:
    json.dump(AlexNet_model_losses, filehandle)
```

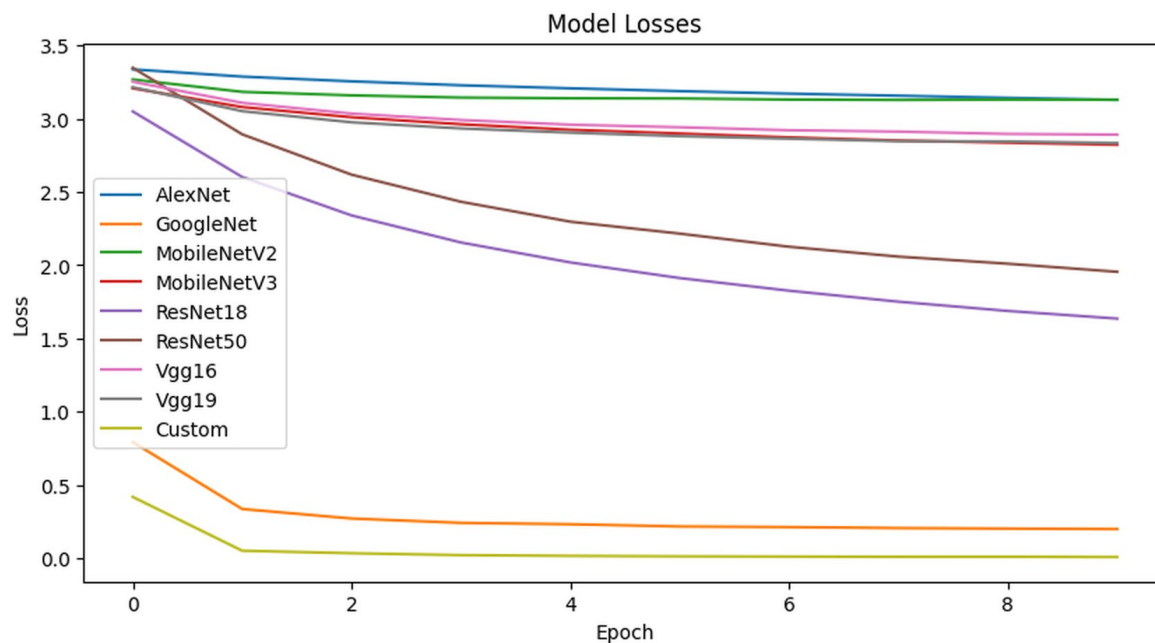
Listing 21. Kod zapisu wytrenowanego modelu i wyników do plików.

Po zakończeniu procesu treningu, dokonano analizy przebiegu treningu dla każdego z modeli, opierając się na wartościach funkcji straty (loss) zarejestrowanych w trakcie kolejnych epok treningu. Wykresy przedstawiające zmiany wartości funkcji straty pozwalają na ocenę, jak efektywnie każdy model uczył się rozpoznawania gestów ręki.

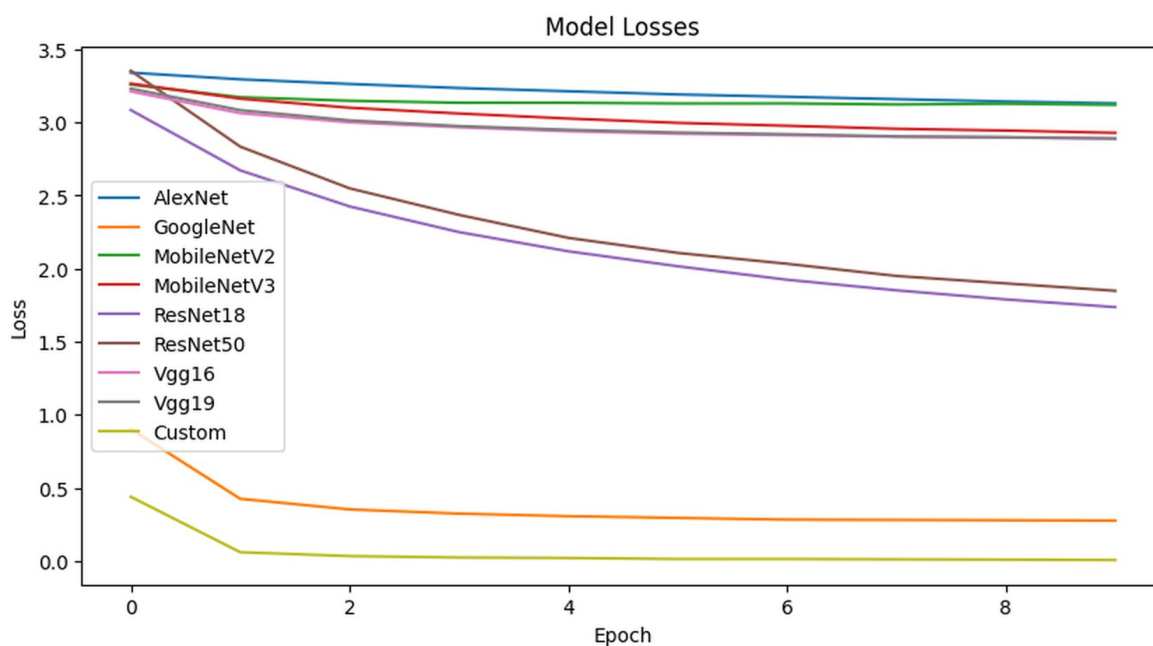
Wykresy te stanowią cenną informację, umożliwiającą identyfikację momentów, w których model uczył się najintensywniej, oraz momentów, w których napotykał na problemy z konwergencją.

Poniżej przedstawiono wykresy funkcji straty dla modeli trenowanych w ramach dwóch prób:

- Pierwszy wykres (Rysunek 3) przedstawia przebieg treningu dla danych w oryginalnym formacie kolorowym. Na tym wykresie widać, że model Custom oraz GoogleNet osiągnęły najszybszy spadek wartości straty (loss) już na początkowych etapach treningu. Inne modele, takie jak AlexNet, ResNet18, oraz MobileNet V2, wykazują bardziej stopniowy spadek, co może świadczyć o ich większej stabilności, ale też wolniejszym procesie adaptacji do danych.
- Drugi wykres (Rysunek 4) przedstawia przebieg treningu dla danych przekształconych do skali szarości (grayscale). Podobnie jak w przypadku danych kolorowych, model Custom oraz GoogleNet również tutaj osiągnęły szybki spadek strat. Co istotne, w porównaniu do danych kolorowych, modele trenowane na danych grayscale wykazują bardziej stabilne zachowanie, z mniejszymi różnicami pomiędzy poszczególnymi modelami.



Rysunek 3. Przebieg treningu dla danych kolorowych.



Rysunek 4. Przebieg treningu dla danych przekształconych do skali szarości.

Z perspektywy efektywności, można zauważyć, że dane w skali szarości nie wpłynęły negatywnie na wydajność modeli, a wręcz przeciwnie, mogły przyczynić się do ich większej stabilności w trakcie treningu. Analiza wyników pokazuje, że zastosowanie danych w skali szarości może być korzystnym podejściem, szczególnie

dla modeli, które wykazują większą zależność od cech strukturalnych obrazów, a nie kolorów.

Wyniki pokazują, że model Custom oraz GoogleNet wykazały się największą skutecznością w obu próbach, co sugeruje ich wysoką zdolność do adaptacji i efektywnego uczenia się w różnych warunkach.

3.4. Testowanie modelu i wyniki

Po zakończeniu treningu modele zostały poddane testom na zestawie danych testowych, które nie były używane w procesie treningu. Celem testów było określenie zdolności modeli do prawidłowego rozpoznawania gestów ręki. Testowanie każdego modelu polegało na przepuszczeniu przez niego obrazów z zestawu testowego oraz porównaniu przewidywań modelu z rzeczywistymi etykietami gestów. Oceny dokonano na podstawie wskaźnika dokładności (accuracy) oraz funkcji straty (loss).

W procesie testowania modeli wykorzystano kod z listingu 22, który ilustruje sposób przeprowadzania testów:

```
# test model function
def test_model(model, test_loader, criterion, device):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = correct / total * 100
    print(f"Test Accuracy: {accuracy:.2f}%")
    return accuracy
```

Listing 22: Kod do testowania modeli.

Wyniki testów wskazują na różnice w wydajności modeli w zależności od rodzaju danych, na których były trenowane i testowane. Wśród modeli trenowanych na danych kolorowych, model GoogleNet wykazał się wysoką skutecznością, osiągając dokładność na poziomie 97.48%. Co ciekawe, model autorski (CustomCNN) osiągnął najwyższą dokładność, sięgając niemal 100%, choć jego wyniki mogą sugerować możliwość nadmiernego dopasowania (overfitting). Wyniki można zaobserwować w listingu 23:

Model	Loss	Accuracy (%)
-------	------	--------------

AlexNet	3,130349	8,81
GoogleNet	0,275972	97,02
MobileNet V2	3,120085	16,16
MobileNet V3	2,928593	24,75
ResNet18	1,736209	55,94
ResNet50	1,954710	47,29
Vgg16	2,889976	25,97
Vgg19	2,889561	25,45
CustomCNN	0,005587	99,69

Tabela 1: Wyniki testów na danych przekształconych do skali szarości (grayscale).

W przypadku danych przekształconych do skali szarości, wyniki testów również wskazują na wysoką skuteczność modelu GoogleNet, który uzyskał dokładność na poziomie 97.02%. Model autorski ponownie osiągnął bardzo wysoką dokładność (99.69%), co sugeruje, że uproszczenie danych poprzez usunięcie informacji kolorystycznej może wspomagać modele w lepszym rozpoznawaniu gestów. Wyniki można zaobserwować w listingu 24:

Model	Loss	Accuracy (%)
AlexNet	3,130349	8,81
GoogleNet	0,275972	97,02
MobileNet V2	3,120085	16,16
MobileNet V3	2,928593	24,75
ResNet18	1,736209	55,94
ResNet50	1,847950	48,18
Vgg16	2,889976	25,97
Vgg19	2,889561	25,45
CustomCNN	0,005587	99,69

Tabela 2: Wyniki testów na danych przekształconych do skali szarości (grayscale).

Porównując oba zestawy wyników, można zauważyć, że przekształcenie danych do skali szarości przyniosło niewielkie, ale zauważalne różnice w wydajności modeli.

W niektórych przypadkach, jak w modelu AlexNet, poprawa dokładności była wyraźna, co może sugerować, że niektóre modele lepiej radzą sobie z uproszczonymi danymi.

Przeprowadzono także praktyczne testy na obrazach litery "G" z zestawu testowego oraz na prywatnie zrobionym zdjęciu dłoni wykonującej ten gest.

W pierwszej kolejności przeprowadzono testy na danych w formacie kolorowym, bez żadnych przekształceń obrazu wejściowego. Wyniki przedstawiają się następująco:

- AlexNet: nie rozpoznał litery "G".
- GoogleNet: poprawnie rozpoznał literę "G".
- MobileNet V2: nie rozpoznał litery "G".
- MobileNet V3: nie rozpoznał litery "G".
- ResNet 18: nie rozpoznał litery "G".
- ResNet 50: nie rozpoznał litery "G".
- Vgg16: nie rozpoznał litery "G".
- Vgg19: nie rozpoznał litery "G".
- Autorski model CNN: nie rozpoznał litery "G".

Następnie, przeprowadzono testy na danych przekształconych do skali szarości (grayscale) oraz na obrazach odwróconych (inverted). Dla każdego z tych przekształceń sprawdzono, jak modele radzą sobie z rozpoznawaniem litery "G". Wyniki były następujące:

- AlexNet:
 - Inverted: nie rozpoznał litery "G".
 - Gray: nie rozpoznał litery "G".
- GoogleNet:
 - Inverted: poprawnie rozpoznał literę "G".
 - Gray: nie rozpoznał litery "G".
- MobileNet V2:
 - Inverted: nie rozpoznał litery "G".
 - Gray: nie rozpoznał litery "G".
- MobileNet V3:
 - Inverted: nie rozpoznał litery "G".
 - Gray: nie rozpoznał litery "G".
- ResNet 18:
 - Inverted: nie rozpoznał litery "G".

- Gray: nie rozpoznał litery "G".
- ResNet 50:
 - Inverted: poprawnie rozpoznał literę "G".
 - Gray: poprawnie rozpoznał literę "G".
- Vgg16:
 - Inverted: nie rozpoznał litery "G".
 - Gray: nie rozpoznał litery "G".
- Vgg19:
 - Inverted: nie rozpoznał litery "G".
 - Gray: nie rozpoznał litery "G".
- Autorski model CNN:
 - Inverted: nie rozpoznał litery "G".
 - Gray: nie rozpoznał litery "G".

Podsumowując, GoogleNet i ResNet 50 okazały się najskuteczniejsze w rozpoznawaniu litery "G" w różnych warunkach przetwarzania obrazu. W szczególności, ResNet 50 zdołał poprawnie rozpoznać literę "G" zarówno na obrazach w skali szarości, jak i na obrazach odwróconych, co świadczy o jego większej zdolności do generalizacji w porównaniu do innych modeli. Pozostałe modele miały trudności z prawidłowym rozpoznaniem tego gestu, co sugeruje, że mogły być mniej odpowiednie dla tego specyficznego zadania lub że wymagają dalszej optymalizacji.

3.3. Testy i wyniki

Po zakończeniu implementacji aplikacji oraz wytrenowaniu modeli sztucznej inteligencji, przystąpiono do testów w rzeczywistych warunkach. Głównym celem testów było zweryfikowanie poprawności działania aplikacji w zakresie komunikacji z dronem, odbioru obrazu oraz przetwarzania klatek wideo w czasie rzeczywistym. Testy odbyły się zarówno w pomieszczeniach zamkniętych, jak i na otwartej przestrzeni, co pozwoliło na ocenę wpływu zmiennych warunków oświetleniowych na funkcjonowanie systemu.

Strumień wideo z kamery drona DJI Tello był przesyłany do komputera, gdzie aplikacja analizowała obraz i logowała przewidywania modeli sztucznej inteligencji za pomocą konsoli. W trakcie testów przetestowano kilka modeli sztucznej inteligencji,

aby ocenić ich skuteczność, porównując wyniki uzyskane w testach terenowych z wynikami osiągniętymi w środowisku laboratoryjnym.

Podczas testów w rzeczywistych warunkach aplikacja napotkała szereg problemów, które znacząco wpłynęły na skuteczność działania systemu rozpoznawania gestów. Najbardziej istotnym wyzwaniem było duże opóźnienie w transmisji obrazu z kamery drona DJI Tello, wynoszące od 10 do 20 sekund. Opóźnienie to uniemożliwiało przetwarzanie obrazu w czasie rzeczywistym, co było kluczowe dla poprawnego rozpoznawania gestów ręki. Mimo prób optymalizacji ustawień sieciowych i konfiguracyjnych problem ten nie został rozwiązany.

Kolejnym problemem była nieskuteczność modeli sztucznej inteligencji w rozpoznawaniu gestów w dynamicznych, terenowych warunkach. Żaden z modeli, w tym GoogleNet i ResNet50, nie zdołał poprawnie rozpoznać gestu "G", ani na danych kolorowych, ani po przekształceniu obrazu do skali szarości (grayscale) lub po jego inwersji (inverted). Wyniki te wskazują na ograniczenia modeli w adaptacji do rzeczywistych, zmiennych warunków, takich jak różnorodność tła, zmienne oświetlenie oraz zakłócenia w strumieniu wideo.

Przeprowadzone testy ujawniły istotne trudności w rozpoznawaniu gestów. Mimo że modele sztucznej inteligencji podczas treningu na statycznych obrazach osiągały wysokie wyniki, w praktyce żaden z nich nie był w stanie poprawnie zidentyfikować gestu "G". Zarówno dla danych kolorowych, jak i przekształconych do skali szarości oraz inwersji, modele nie były w stanie poradzić sobie z dynamicznymi warunkami otoczenia.

Czynniki takie jak zmienne oświetlenie, różnorodne tło oraz zakłócenia obrazu miały kluczowy wpływ na wyniki testów. Wysokie opóźnienie w transmisji obrazu dodatkowo utrudniało przetwarzanie klatek w czasie rzeczywistym, co w praktyce uniemożliwiło poprawne działanie aplikacji. W związku z tym, wyniki testów wskazują na potrzebę dalszej optymalizacji zarówno modeli sztucznej inteligencji, jak i procesów przetwarzania obrazu oraz transmisji danych.

W trakcie testów napotkano szereg wyzwań oraz ograniczeń, które znacząco wpłynęły na efektywność działania systemu. Najważniejszym wyzwaniem było opóźnienie w transmisji obrazu z kamery drona DJI Tello, które wynosiło od 10 do 20 sekund. Problem ten nie został rozwiązany mimo prób optymalizacji połączeń sieciowych, co ograniczyło możliwość natychmiastowej reakcji na rozpoznane gesty i wpłynęło na ogólną funkcjonalność aplikacji.

Kolejnym wyzwaniem były zmienne warunki oświetleniowe oraz różnorodne tło, które wpływały na skuteczność działania modeli sztucznej inteligencji. Modele trenowane na statycznych obrazach miały trudności z adaptacją do dynamicznego środowiska, co powodowało błędne rozpoznawanie gestów lub brak ich identyfikacji.

Ograniczenia związane z jakością obrazu, takie jak niższa rozdzielczość i zakłócenia wynikające z warunków testowych, również wpłynęły negatywnie na skuteczność modeli. Nawet w przypadku modeli, które wykazywały wysoką skuteczność w środowisku laboratoryjnym, rzeczywiste warunki okazały się bardziej wymagające, co podkreśla konieczność dalszych badań nad optymalizacją algorytmów.

Wszystkie te wyzwania i ograniczenia wskazują na potrzebę dalszych badań oraz potencjalnych usprawnień, zarówno w zakresie przetwarzania danych, jak i w samych architekturach modeli sztucznej inteligencji. Wprowadzenie bardziej zaawansowanych technik poprawy jakości obrazu oraz dynamiczne przystosowanie modeli do zmiennych warunków środowiskowych może przyczynić się do poprawy skuteczności systemu rozpoznawania gestów.

Podsumowanie i wnioski

Praca miała na celu stworzenie systemu sterowania dronem DJI Tello za pomocą gestów ręki, opierając się na konwolucyjnych sieciach neuronowych (CNN) oraz technikach sztucznej inteligencji. W toku badań opracowano aplikację, która łączyła się z dronem, umożliwiała jego kontrolowanie oraz komunikację w czasie rzeczywistym. Jednakże, mimo poprawnej implementacji połączenia z dronem i wyświetlania liter w konsoli, model sztucznej inteligencji nie rozpoznawał gestów w sposób prawidłowy. Dodatkowo, wystąpiło znaczące opóźnienie w obrazie z kamery drona, wynoszące 10–20 sekund, co dodatkowo wpłynęło na ograniczenia w praktycznym zastosowaniu systemu. Wyniki te wskazują na potrzebę dalszych optymalizacji oraz dokładnej analizy przyczyn niepowodzenia w rozpoznawaniu gestów i opóźnienia w przesyłaniu obrazu.

Główne wyzwania, jakie napotkano podczas realizacji projektu, związane były z niską skutecznością modelu w klasyfikacji gestów ręki oraz z dużym opóźnieniem w obrazie z kamery drona. Mimo że aplikacja działała poprawnie pod względem technicznym – łączyła się z dronem, reagowała na wprowadzone komendy i wyświetlała wyniki w konsoli – sam model uczenia maszynowego nie był w stanie prawidłowo rozpoznać żadnego z gestów. Oznacza to, że problem nie leżał w samej aplikacji, lecz w modelach odpowiadających za przetwarzanie i klasyfikację gestów.

Jednym z kluczowych wniosków wynikających z przeprowadzonych testów jest to, że wybrane modele sieci neuronowych nie były odpowiednio skalibrowane do specyficznych warunków użycia, co mogło wynikać z niewystarczającego zbioru danych treningowych lub błędów w procesie trenowania modelu. Ponadto, problematyczne okazały się także warunki środowiskowe, takie jak zmienne oświetlenie i różnorodność tła, co znacząco wpłynęło na skuteczność rozpoznawania gestów. Dodatkowo, opóźnienia w przesyłaniu obrazu uniemożliwiły efektywne reagowanie systemu na gesty w czasie rzeczywistym, co stanowiło istotne ograniczenie w praktycznym zastosowaniu.

Pomimo tych trudności, system stanowi solidną podstawę do dalszego rozwoju i optymalizacji. W pierwszej kolejności zaleca się przeanalizowanie danych treningowych, które mogły nie odpowiadać realnym warunkom testowym. Możliwe, że zastosowanie bardziej zróżnicowanego i obszerniejszego zbioru danych, uwzględniającego różne warunki oświetleniowe oraz tła, pozwoli na poprawę

skuteczności modelu. Kolejnym krokiem powinna być dokładna analiza architektury zastosowanych sieci neuronowych oraz rozważenie innych algorytmów, które mogą lepiej radzić sobie z dynamicznym rozpoznawaniem gestów, a także z redukcją opóźnień w przetwarzaniu obrazu.

W dalszych badaniach warto również skupić się na testowaniu różnych technik augmentacji danych, takich jak zmiany jasności, rotacje obrazu czy przekształcenia kolorystyczne, które mogłyby pomóc w lepszym przygotowaniu modeli do pracy w zmiennych warunkach. Dodatkowo, rozważenie użycia sensorów głębi mogłoby poprawić precyzję systemu, zwłaszcza w rozpoznawaniu bardziej złożonych gestów w trójwymiarowej przestrzeni, a jednocześnie mogłoby wpłynąć na przyspieszenie procesu analizy obrazu.

Podsumowując, choć aplikacja działała poprawnie pod względem technicznym, a komunikacja z dronem przebiegała bez problemów, model sztucznej inteligencji nie spełnił założonych oczekiwań w zakresie rozpoznawania gestów ręki, a opóźnienia w obrazie znacząco ograniczyły funkcjonalność systemu w czasie rzeczywistym. Przeprowadzone badania wykazały, że konieczne są dalsze prace nad optymalizacją modelu oraz jego treningiem, a także wdrożenie bardziej zaawansowanych technik przetwarzania obrazu. Wskazane jest także uwzględnienie większej liczby danych treningowych oraz rozważenie innych algorytmów i metod, które mogą przyczynić się do zwiększenia skuteczności i szybkości działania systemu. Te wnioski stanowią cenną podstawę do dalszych badań i rozwoju tego typu interfejsów, które w przyszłości mogą znaleźć szerokie zastosowanie w różnorodnych dziedzinach życia, od ratownictwa po przemysł i rozrywkę.

Literatura

1. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton (2014). *ImageNet Classification with Deep Convolutional Neural Networks*
2. Alexander Mordvintsev & Abid K (2017) *OpenCV-Python Tutorials Documentation*
3. Aldhahri, E., Aljuhani, R., Alfaidi, A., Alshehri, B., Alwadei, H., Aljojo, N., Alshutayri, A., & Almazroi, A. (2023). Arabic Sign Language Recognition Using Convolutional Neural Network and MobileNet. *Arabian Journal for Science and Engineering*, 48(2), 2147–2154. doi: 10.1007/S13369-022-07144-2/TABLES/2
4. Amazon. (2022). *Amazon Prime Air prepares for drone deliveries*. Amazon. Retrieved from <https://www.aboutamazon.com/news/transportation/amazon-prime-air-prepares-for-drone-deliveries>
5. Angga Prima Syahputra, Alda Cendekia Siregar, Rachmat Wahid Saleh Insani (2023). *Comparison of CNN Models With Transfer Learning in the Classification of Insect Pests*
6. *ASL(American Sign Language) Alphabet Dataset*. (2021). Retrieved from <https://www.kaggle.com/datasets/debashishsau/aslamerican-sign-language-alphabet-dataset>
7. Baheti, P. (2021). *The Essential Guide to Neural Network Architectures*. Retrieved from <https://www.v7labs.com/blog/neural-network-architectures-guide#h5>
8. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning* (M. Jordan, J. Kleinberg, & B. Scholkopf, Eds.). Singapore: Springer.
9. Bukowski, P., & Szala, G. (2018). Postępy w inżynierii mechanicznej Developments in mechanical engineering. *Scientific-Technical Journal*, 11(6), 5–19.
10. Chavan, S., Yu, X., & Saniie, J. (2021). Convolutional Neural Network Hand Gesture Recognition for American Sign Language. *IEEE International Conference on Electro Information Technology*, 2021-May, 188–192. doi: 10.1109/EIT51626.2021.9491897
11. Chen, R. C., Dewi, C., Zhang, W. W., Liu, J. M., & Huang, S. W. (2020). Integrating gesture control board and image recognition for gesture recognition

- based on deep learning. *International Journal of Applied Science and Engineering*, 17(3), 237–248. doi: 10.6703/IJASE.202009_17(3).237
12. Christina Bornberg (2020). *Training data preparation with computer vision algorithms for a Mask R-CNN*
 13. Daniel Mas MontserratQian LinJan AllebachEdward J. Delp (2017) *Training Object Detection And Recognition CNN Models Using Data Augmentation*
 14. Dumitru Erhan, Aaron Courville, Yoshua Bengio, Pascal Vincent (2010). Why Does Unsupervised Pre-training Help Deep Learning?
 15. Dyndal, G. L., Berntsen, T. A., & Redse-Johansen, S. (2017). *Autonomiczne drony wojskowe: już nie science fiction*. NATO. Retrieved from <https://www.nato.int/docu/review/pl/articles/2017/07/28/autonomiczne-drony-wojskowe-juz-nie-science-fiction/index.html>
 16. European Union Aviation Safety Agency. (2024). *Open Category - Low Risk - Civil Drones*. Retrieved from <https://www.easa.europa.eu/en/domains/drones-air-mobility/operating-drone/open-category-low-risk-civil-drones>
 17. Felix Zhan (2019). Hand Gesture Recognition with Convolution Neural Networks
 18. Gio, N., Brisco, R., & Vuletic, T. (2021). CONTROL OF A DRONE WITH BODY GESTURES. *Proceedings of the Design Society*, 1, 761–770. doi: 10.1017/PDS.2021.76
 19. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. Retrieved from www.deeplearningbook.org
 20. H.H. Khairi, Sharifah H. S. Ariffin, N. M. Abdul Latiff, Kamaludin Mohamad Yusof, M. K. Hassan, Mohammad Rava. (2020) *The impact of firewall on TCP and UDP throughput in an openflow software defined network*.
 21. Hyun Min Oh, Hyunki Lee, Min Young Kim (2019). *Comparing Convolutional Neural Network(CNN) models for machine learning-based drone and bird classification of anti-drone system*.
 22. K. P. Valavanis (2008). *Advances in Unmanned Aerial Vehicles*
 23. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2016). *Deep Residual Learning for Image Recognition*
 24. Kim, T.-K. (2007). *Cambridge Hand Gesture Data set*. Retrieved from https://labicvl.github.io/ges_db.htm

25. Kossowski, M. (2022). Zastosowania dronów w badaniach rzeźby terenu, struktury i tekstury osadów na przykładach z Polski Środkowej. *Acta Universitatis Lodzensis. Folia Geographica Physica*, 21(21), 35–47.
26. Li, W. (2023). *MSR Gesture3D*. Retrieved from <https://sites.google.com/view/wanqingli/data-sets/msr-gesture3d>
27. Merlin Stampa, Andreas Sutorma, Uwe Jahn, Jörg Thiem, Carsten Wolff, Christof Roehrig (2021). Maturity Levels of Public Safety Applications using Unmanned Aerial Systems: a Review
28. Mohit Sewak, Md. Rezaul Karim, Pradeep Pujari (2018). *Practical Convolutional Neural Networks: Implement advanced deep learning models using Python*
29. Murali, R. S. L., Ramayya, L. D., & Anil Santosh, V. (2022). Sign Language Recognition System Using Convolutional Neural Network And Computer Vision. *International Journal of Engineering Innovations in Advanced Technology*, 4(4), 137–142.
30. *Najlepsze drony do zdjęć – co wybrać?* (n.d.). DJI. Retrieved from <https://dji-ars.pl/Najlepsze-drony-do-zdjec-co-wybrac-news-pol-1720770653.html>
31. Naumienko, M. (2023). *Nowoczesne ratownictwo z wykorzystaniem dronów*. Polska Izba Systemów Bezzałogowych. Retrieved from <https://pisb.pl/nowoczesne-ratownictwo-z-wykorzystaniem-dronow/>
32. Pradeepta Mishra (2023). *CNN and RNN Using PyTorch*
33. Robin R. Murphy, Carlos Xavier Soto (2012). *Hand gesture recognition with depth images: A review*
34. Syed Rehan Shah, Salman Qadri, Hadia Bibi, Syed Muhammad Waqas Shah, Muhammad Imran Sharif, Francesco Marinello (2023). *Comparing Inception V3, VGG 16, VGG 19, CNN, and ResNet 50: A Case Study on Early Detection of a Rice Disease*
35. Szóstek, M., & Nowobilski, T. (2022). Bezzałogowe statki powietrzne w budownictwie – protokół bezpiecznego przygotowania i lotu dronem. *Przegląd Budowlany*, 9–10, 117–120. Retrieved from www.przegladbudowlany.pl
36. Tilottama Goswami, Shashidhar Reddy Javaji (2020). *CNN Model for American Sign Language Recognition*
37. Venugopalan, A., & Reghunadhan, R. (2023). Applying Hybrid Deep Neural Network for the Recognition of Sign Language Words Used by the Deaf COVID-

- 19 Patients. *Arabian Journal for Science and Engineering*, 48(2), 1349–1362.
doi: 10.1007/S13369-022-06843-0/TABLES/3
38. Xianghan Wang, Jie Jiangb, Yingmei Wei, Lai Kang and Yingying Gao (2018),
Research on Gesture Recognition Method Based on Computer Vision
39. Yann LeCun, Yoshua Bengio, Geoffrey Hinton (2015). *Deep learning*
40. Zhang, Y., Cao, C., Cheng, J., & Lu, H. (2018). EgoGesture: A New Dataset
and Benchmark for Egocentric Hand Gesture Recognition. *IEEE Transactions
on Multimedia (T-MM)*, 20(5), 1038–1050. Retrieved from
<https://nlpr.ia.ac.cn/iva/yfzhang/datasets/egogesture.html>

Spis skrótów zastosowanych w pracy

1. AI - ang. Unmanned Aerial Vehicle
2. ANN - ang. Artificial Neural Network
3. AR - ang. Augmented reality
4. ASL - ang. American Sign Language
5. CNN - ang. Convolutional Neural Network
6. DL - ang. Deep Learning
7. GPS - ang. Global Positioning System
8. HOG - Histogramy orientacji gradientów
9. KNN - ang. K-Nearest Neighbors
10. LSTM - ang. Long Short-Term Memory
11. R-CNN - ang. Region-based Convolutional Neural Network
12. RNN - ang. Recurrent Neural Network
13. ReLU - ang. Rectified Linear Unit
14. ResNet - ang. Residual Network
15. SGD - ang. Stochastic Gradient Descent
16. SVM - ang. Support Vector Machines
17. UAV - ang. Unmanned Aerial Vehicle
18. VR - ang. Virtual Reality
19. WiFi - ang. Wireless Fidelity
20. YOLO - ang. You Only Look Once

Spis rysunków

Rysunek 1. Schemat przydziału klas do kategorii dronów (źródło: European Union Aviation Safety Agency, 2024).....

Rysunek 2. Przedstawienie warstw sieci neuronowej (Źródło: Bishop, 2006 <https://controlengineering.pl/sieci-neuronowe-w-sterowaniu-procesami-technologicznymi/>).....

Rysunek 3. Przebieg treningu dla danych kolorowych.....

Rysunek 4. Przebieg treningu dla danych przekształconych do skali szarości.....

Spis tabel i listingów

Listing 1. Tworzenie i aktywacja środowiska wirtualnego	31
Listing 2. Instalacja zależności z pliku requirements.txt	31
Listing 3. Zawartość pliku requirements.txt	31
Listing 4. Komendy do odblokowania portów na firewallu za pomocą UFW	32
Listing 5. Kod do nawiązania połączenia z dronem DJI Tello	33
Listing 6. Komendy do odblokowania portów na firewallu za pomocą UFW	33
Listing 7. Przetwarzanie obrazu z kamery drona za pomocą OpenCV	33
Listing 8. Ładowanie modelu AI i przetwarzanie obrazu w aplikacji	34
Listing 9. Wyświetlanie wyników rozpoznawania gestów	34
Listing 10. Implementacja autorskiego modelu CNN	39
Listing 11. Kod zamrażający wagi w modelu	40
Listing 11. Kod zamrażający wagi w modelu	40
Listing 12. Kod dostosowujący ostatnią warstwę w modelu AlexNet do liczby klas .	40
Listing 13. Kod dostosowujący ostatnią warstwę w modelu GoogleNet do liczby klas	40
Listing 14. Kod dostosowujący ostatnią warstwę w modelu ResNet18 / ResNet50 do liczby klas	40
Listing 15. Kod dostosowujący ostatnią warstwę w modelu MobileNet V2 do liczby klas	40
Listing 16. Kod dostosowujący ostatnią warstwę w modelu MobileNet V3 do liczby klas	40
Listing 17. Kod dostosowujący ostatnią warstwę w modelu VGG16 / VGG19 do liczby klas	40
Listing 18. Kod przenoszący model na GPU	41
Listing 19. Definicja funkcji straty i optymalizatora	41
Listing 20. Funkcja treningu modelu "train_model"	42
Listing 21. Kod zapisu wytrenowanego modelu i wyników do plików	42
Listing 22. Kod do testowania modeli	45
Tabela 1. Wyniki testów na danych przekształconych do skali szarości (grayscale)	45
Tabela 2. Wyniki testów na danych przekształconych do skali szarości (grayscale)	46

Spis załączników

1. Archiwum zawierające aplikację sterującą dronem oraz kod trenujący modele do rozpoznawania gestów ręki