Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАСТОСУНКУ ДЛЯ ДИСТАНЦІЙНОГО УПРАВЛІННЯ НА ОСНОВІ МУЛЬТИМЕДІЙНИХ ЗАСОБІВ
(тема)

Виконав:

студент 4 курсу, групи ІТІНФ-18-1
Філатов О.Є
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Гороховатський В.О.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Філатову Олександру Євгеновичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення програмного застосунку для дистанційного управління на основі мультимедійних засобів

затверджена наказом університету від 16 травня 2022 року No 541Ст

2. Термін подання студентом роботи до екзаменаційної комісії 29 травня 2022 р.

3. Вихідні дані до роботи Науково–методична та науково–технична література, дані інтернет-мережі, Python-бібліотеки OpenCV, MediaPipe для реалізації нейронних мереж комп'ютерного зору, відкрита модель загорткової нейронної мережі BlazePalm, Golang-бібліотека Robotgo для реалізації взаємодії з комп'ютером

4. Перелік питань, що потрібно опрацювати в роботі

1. Аналіз існуючих методів розпізнавання зображень.
2. Огляд технологій для розпізнавання та класифікації жестів людини.
3. Побудова архітектури модуля дистанційного керування.
4. Реалізація системи дистанційного керування.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) візуалізація структур нейронних мереж, візуалізація ключових точок, візуалізація роботи хеш-функції, візуалізація важливих елементів реалізації розроблюваної технології.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Бєлова Н.В.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	18.04.2022	
2	Аналіз завдання, підбір літератури	18.04.22-21.04.22	
3	Аналіз літератури з досліджуваної проблеми	22.04.22-25.04.22	
4	Аналіз технологій детектування об'єктів	26.04.22-30.04.22	
5	Розробка методу дистанційного керування	01.05.22-14.05.22	
6	Реалізація технології дистанційного керування	15.05.22-23.05.22	
7	Оформлення пояснювальної записки	24.05.22-26.05.22	
8	Рецензування	27.05.22	
9	Перевірка на плагіат	28.05.22	
10	Підготовка презентації та доповіді	29.05.22-30.05.22	
11	Занесення роботи в електронний архів	31.05.22	
12	Попередній захист кваліфікаційної роботи	01.06.22	

Дата видачі завдання 18 квітня 2022 р.

Студент ___
(підпис)

Керівник роботи _____ проф. Гороховатський В.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 49 с., 3 табл., 26 рис., 41 джерело.

ДИСТАНЦІЙНЕ КЕРУВАННЯ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, ДЕТЕКТУВАННЯ ЖЕСТІВ.

Об’єктом роботи є дистанційне управління мультимедійними засобами з використанням вебкамери.

Метою роботи є розроблення методу та технології на базі згорткових нейронних мереж, що дають можливість детектувати жести користувача та керувати комп’ютерною системою у залежності від розпізнаного жесту.

Використано методи розпізнавання жестів людини за технологіями, що базуються на згорткових нейронних мережах. Проведено аналіз моделей нейронних мереж задля детектування об’єктів на зображеннях та детектування ключових точок об’єктів. Розроблено алгоритм розпізнавання жестів та дистанційного керування мультимедійною системою.

У результаті роботи здійснена програмна реалізація моделі для дистанційного управління.

CONVOLUTONAL NEURAL NETWORKS, LANDMARK DETECTION, REMOTE OPERATION.

The object of the work is a sequence of digital images obtained by a webcam.

The aim of the work is to develop methods based on convolutional neural networks, which allow to detect user gestures and control the system depending on the recognized gesture.

Methods of human gesture recognition by technologies based on convolutional neural networks are used. Models of neural networks that have the ability to detect objects in images and detect key points of objects were studied. Algorithms for gesture recognition and remote control of the system have been developed.

As a result, the software implementation of the model for remote control of the system was implemented.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CNN – convolutional neural network (згорткова нейронна мережа)

YOLO – You Only Look Once

SSD – Single shot detector

UDP – user datagram protocol

КТ – ключова точка

ВСТУП

У сучасному світі зі швидким покращенням комп’ютерів приходять нові технології, що раніше не могли бути використані через низькі потужності систем. Покращення технологій проводилося поступово, з перфокарт XX століття до сучасних сенсорних екранів та технологій віртуальної реальності, що широко використовуються у сучасному світі[1-5].

Сучасні технології дозволяють отримати новий рівень заглиблення під час керування, та все ще мають занадто велику ціну та потребують сторонніх приладів для використання. Є декілька можливих вирішень даної проблеми – дослідження існуючих систем керування задля покращення працездатності, збільшення можливостей та пошуку більш економічних рішень, чи пошук нових можливих моделей для керування системою, на чому й буде базуватися дана робота.

Зі створенням сучасних швидких нейронних мереж перед розробниками виникає нова можливість для створення засобів керування – розпізнавання об’єктів напряму з відеокамери[6-9]. Дані мережі мають широкий спектр використання, від розпізнавання трафіку самокерованих автомобілів до розумних замків з розпізнаванням обличчя. Таким чином, сучасні технології дозволяють відтворити складні процеси розпізнавання, схожі на технології віртуальної реальності, проте не потребуючи додаткової апаратури, а на базі отриманих даних використовувати як контролер, наприклад, руку людини, що і буде розглянуто у даній роботі.

Задачею даної роботи є створення комп’ютерного програмного забезпечення задля дистанційного керування комп’ютером завдяки вебкамері та розпізнаним нейронною мережею жестам з великими можливостями розширення функціоналу та персоналізації.

1 АНАЛІЗ ЗАСОБІВ ДИСТАНЦІЙНОГО УПРАВЛІННЯ НА ОСНОВІ МУЛЬТИМЕДІЙНИХ ТЕХНОЛОГІЙ

1.1 Розвиток засобів керування комп’ютером

З самого початку історії комп’ютерів розробникам було зрозуміло, що їх використання з чітко заданими програмою та вводом є недоцільним бо потребувало би створення нового комп’ютера кожен раз, коли з’являлася необхідність в зміні одного з цих елементів. Інженери відразу почали шукати шляхи вирішення цієї проблеми, і першим рішенням були перфоратори, що почали використовуватися задовго до комп’ютерів[10-13].

Перфоратор – це машина, що дозволяє зберігати інформацію завдяки пробиванню дірочок на спеціальному папері, що зветься перфокартою. Перфокарта зі стандартним розміром в 80 стовпців та 12 рядків могла вміщати 960 біт інформації. Напряму з перфоратора передати перфокарти до комп’ютера було неможливо, тому їх загружали вручну, а випадкова зміна порядку карт чи зникнення однієї з них могли призвести до помилок у виконанні. Перші обробники перфокарт були створені Германом Голлерітом, який пізніше заснував компанію IBM, для перепису населення Америки 1890 року.

Наступним кроком стали клавіатури, що на початку своєї історії були тими ж самими перфораторами, але з додатковими клавішами для цифр та літер. З часом клавіатури «навчили» передавати дані напряму до комп’ютера, а з появою моніторів процес взаємодії значно облегшився.

Згодом люди почали шукати новий, більш інтуїтивний шлях взаємодії, та у 1963 у Массачусетському технологічному інституті розробили стилуси – електронні ручки, що дозволяли керувати курсором. Цікаво, що використовувалися вони на два десятиліття раніше миші, яка була вперше використана з комп’ютером у 1973 році.

Комп’ютерна миша була створена у 1964 році Дугласом Енгельбартом. У 1967 році було проведене дослідження, за результатами якого миша перевершила стилус за легкістю та зручністю[1], але миша мала широкого використання до 1973 року, коли був створений комп’ютер Xerox Alto, що включав її до своєї комплектації.

Створення сенсорних панелей було наступним кроком за стилусами, але всі вони могли обробляти лише одне натискання. У 1998 році компанія FingerWorks створила перші пристрої з підтримкою декількох дотиків, такі як клавіатура TouchStream та сенсорну панель iGesturePad, що мала замінити мишу. За назвою останньої можна зрозуміти, що далі сталося з компанією – у 2005 році її придбала Apple, та з використанням розроблених технологій був створений перший iPhone.

У 2010 році Microsoft розробила Kinect – технологію, що дозволяє відстежити рухи людини, обробити їх та надалі використати у різних проектах. Ця технологія розроблювалася для використання в іграх консолі Xbox, та мала ряд обмежень, зокрема вважалося, що єдиним об’єктом, що може рухатися у кадрі є людина.

У 2012 році було випущено перші окуляри доповненої реальності від фірми Oculus. Окуляри та контролери мали вбудовані фотосенсори, що перехоплювали світло випромінюване зовнішніми станціями, що обмежували зону взаємодії, що дозволяло виявити позицію користувача у просторі. Можна сказати, що доповнена реальність є продовженням технології Kinect, але має елегантніше рішення, та ще більш заглиблює користувача у обрані ігри (здебільш для цього і використовується дана технологія).

Та технології відстеження рухів не є такими популярними, як розпізнавання мови. Яскравими прикладами даних технологій можуть бути різноманітні мобільні помічники (Siri), чи розумні колонки (Alexa від Amazon)[14-18].

З початку XXI століття широко досліджуються нові технології розпізнавання зображень, а саме нейронні мережі, що дозволяють, базуючись

на заздалегідь визначених дескрипторах зображення[2] доволі точно ідентифікувати окремі об’єкти у сцені з великою варіативністю кольорів, освітлення чи кутів нахилу. Великим плюсом нейронних мереж є їх можливість навчання, що дозволяє маючи каркас системи та велику кількість даних для тренування дозволити мережі самостійно визначити необхідні параметри для повноцінного розпізнавання. Останні розроблені моделі мають доволі велику швидкість, та можуть використовуватися з потоковим відео з частотою кадрів до 60.

Дана робота, беручи за основу нейронні мережі задля детектування руки людини, має за мету створити програмне забезпечення, що має можливість дистанційного керування комп’ютером завдяки розпізнаній послідовності жестів.

1.2 Нейронні мережі у мультимедійних системах

Основа сучасних нейронних мереж була створена у 1949 році Дональдом Геббом, який дослідив функціонування біологічних нейронних мереж. Він дослідив, як діють та пов’язуються клітини, та висунув свою теорію нейронного ансамблю, що пізніше було використано у комп’ютерній науці задля визначення методології зміни ваг між нейронами моделі.

Однією з перших спроб створення штучної нейронної мережі був концепт неорганізованої машини Алана Тюрінга 1948 року, що мала вигляд бінарної нейронної мережі з випадковим з’єднанням нейронів. Проте, через технологічні обмеження проект не було втілено.

Проте дослідження були дещо призупинені у 1969 році, коли було виявлено ряд проблем нейронних мереж, основною з яких була недостача потужностей системи. Тому основними напрямками досліджень залишались методи лінійної класифікації та більш складні нелінійні класифікатори, такі як метод опорних векторів, хоча нейронні мережі все ще застосовувалися у

окремих областях, зокрема медицині та біології, де вони популярні й понині[3-5].

У 1975 році було розроблено метод зворотного поширення помилки, що дозволив використовувати повноцінне навчання при розробці нейронних мереж завдяки можливості перерахування ваг вузлів. Завдяки цьому методу нейронні мережі знов почали широко досліджуватися, і з малої технологічної революції 1982 року, що змогла значно підвищити обчислювальні потужності комп'ютерів, вже було створено прості моделі нейронних мереж, наприклад, мережа що дозволяла передбачити наступне слово у реченні, створена у 1986 році.

У ті ж часи було створено модель нейронної мережі під назвою нейрокогнітрон, що пропонувала альтернативу старому методу, у якому кожен шар системи був повноз'єднаним, тобто мав вигляд вектору, кожен елемент якого був з'єднаний з кожним елементом минулого шару. Структура неокогнітрону показана на рисунку 1.1.

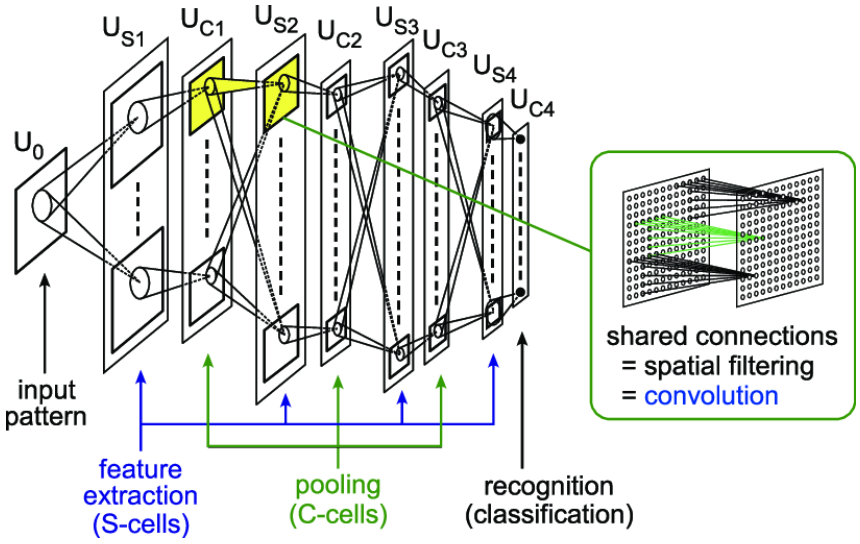

Рисунок 1.1 – Структура неокогнітрону

Можна сказати, що нейрокогнітрон був першим кроком на шляху до створення згорткових нейронних мереж, які згодом стануть основою глибокого навчання – частини машинного навчання, що відповідає за

виконання складних вправ, таких як, наприклад, розрізнення об’єктів на зображеннях[6].

Основою згорткових нейронних мереж є операція згортки, зображена на формулі 1.1:

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x-dx, y-dy), \quad (1.1)$$

де $g(x,y)$ – відфільтроване зображення;

$f(x,y)$ – оригінальне зображення;

ω – ядро згортки.

Простіше передати операцію згортки через матриці: маючи початкове зображення, що передане у вигляді матриці A^0, та ядро згортки – матрицю B, проводиться покрокове накладання матриці B на частину матриці A^0, що показано на рисунку 1.2.

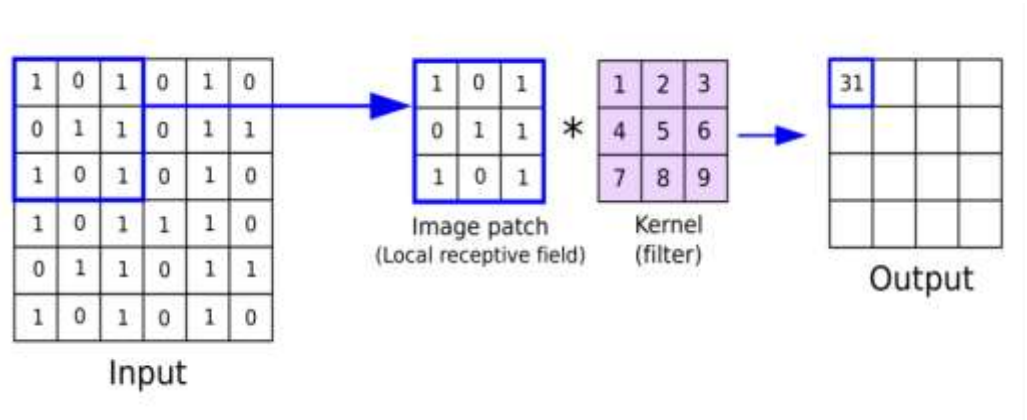

Рисунок 1.2 – Один крок операції згортки

Кожна операція згортки вважається окремим шаром згорткової нейронної мережі. Слід зауважити, що початкове зображення та операції згортки можуть мати декілька каналів, наприклад, зазвичай початкове зображення розподілене на канали моделі передачі кольору RGB, тобто

загальна кількість каналів дорівнює трьом. Відповідно, й ядра згортки можуть мати декілька каналів, але для ядер кожен окремий канал вважається окремим фільтром.

За прикладом, наведеним на рисунку 1.1, можна побачити, що розмір вихідної матриці не відповідає розміру вхідної. Розмір вихідної матриці для простої операції згортки визначається за формулою 1.2:

$$(n - f + 1) \times (n - f + 1), \quad (1.2)$$

де n – розмір матриці оригінального зображення;

f – розмір матриці ядра згортки.

Такої форми матриці достатньо, але іноді виникає потреба у відповідності розмірів вхідної та вихідної матриць. Для цього використовується відступ – поширення розміру вхідної матриці. Відступ розміром 1 зображено на рисунку 1.3.

0	0	0	0	0	0	0	0	0	0
0	10	10	10	10	0	0	0	0	0
0	10	10	10	10	0	0	0	0	0
0	10	10	10	10	0	0	0	0	0
0	10	10	10	10	0	0	0	0	0
0	10	10	10	10	0	0	0	0	0
0	10	10	10	10	0	0	0	0	0
0	10	10	10	10	0	0	0	0	0
0	10	10	10	10	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Рисунок 1.3 – Вхідна матриця з відступом 1, позначеним жовтим кольором.

Нова формула розміру вихідної матриці з врахуванням відступу зображена у формулі 1.3.

$$(n + 2p - f + 1) \times (n + 2p - f + 1), \qquad (1.3)$$

де p – розмір відступу.

Останнім елементом операції згортки є розмір кроку. Не завжди при розрахунку нам потрібно приділяти увагу кожному можливому регіону зображення, бо даних з, наприклад, кожного третього регіону достатньо для висновків. Крок дозволяє «перестрибнути» через непотрібні регіони. Розмір вихідного зображення з урахуванням кроку зображено на формулі 1.4.

$$(\frac{n+2p-f+1}{s}) \times (\frac{n+2p-f+1}{s}), \qquad (1.4)$$

де s – довжина кроку.

Звичайно, однієї операції для створення повноцінної нейронної мережі недостатньо, тому у загорткових нейронних мережах використовуються ще два види шарів, а саме агрегувальні та повноз’єднані шари.

Агрегувальний шар схожий на згортковий, але звичайна операція згортки замість множення та суми зводиться до знаходження середнього (усереднювальне агрегування) чи максимального елементу (максимізаційне агрегування), тобто шар не має окремого ядра згортки. Приклад агрегувального шару з максимізаційним агрегуванням, з розміром фільтру 2 та кроком 2 наведено на рисунку 1.4 [19-24].

Останнім елементом є повноз’єднаний шар, що зазвичай використовується наприкінці роботи задля отримання кінцевого результату. Повноз’єданий шар має вигляд звичайного багатошарового перцептрона та зазвичай задля його використання вхідна матриця «розгортається» до векторного вигляду.

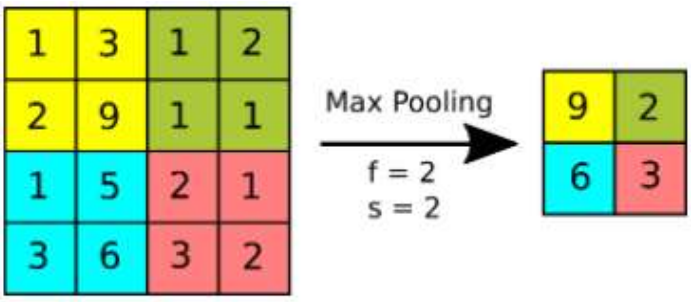

Рисунок 1.4 – Перетворення вхідної матриці за максимізаційним агрегуванням

Однією з перших згорткових нейронних мереж є LeNet-5[7], що була навчена задля розпізнавання набору символів ASCII. Структура даної мережі зображена на рисунку 1.5, а шари розписані у таблиці 1.1.

Таблиця 1.1 – Структура нейронної мережі LeNet

Шар	**Кількість фільтрів**	**Розмір фільтрів**	**Крок**	**Розмір мапи ознак**	**Функція активації**
Вхід	-	-	-	32×32×1	-
Згортка 1	6	5×5	1	28×28×6	tanh
Усереднювальна агрегація	-	2×2	2	14×14×6	-
Згортка 2	16	5×5	1	10×10×6	tanh
Усереднювальна агрегація	-	2×2	2	5×5×16	-
Згортка 3	120	5×5	1	120	tanh
Повноз’єднаний шар 1	-	-	-	84	tanh
Повноз’єднаний шар 2	-	-	-	10	softmax

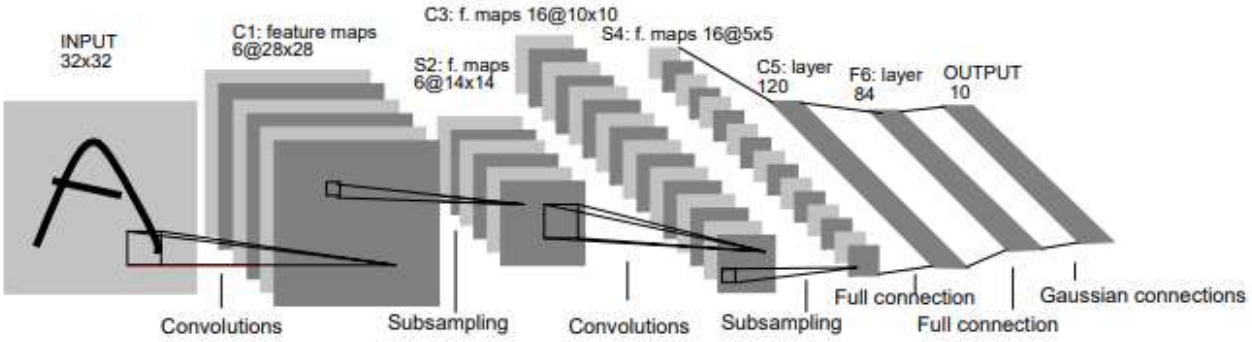

Рисунок 1.5 – Структура нейронної мережі LeNet-5

Створені згорткові нейронні мережі дозволяли розпізнавати зображення[8-10] та отримувати ключові точки[11,12], але вони все ще були недостатньо швидкі[13] для використання у реальному часі. Згодом дослідникам стало зрозуміло, що немає сенсу обробляти повне зображення, а потрібно виділити регіони, де потенційно знаходиться об’єкт, та працювати з ними[3].

Першою реалізацією даної ідеї став метод рухомих вікон[14,15]. Метод на кожному кроці рухає окреме вікно певного розміру, що називається обмежувальною рамкою, обробляючи лише регіон, що потрапляє в його межі. Даний метод мав декілька проблем:

– необхідність створення великої кількості вікон різного розміру для врахування об’єктів на різній відстані;

– довгий час виконання через відносно малий рух вікон.

Для вирішення проблем методу рухомих вікон був розроблений метод селективного відбору регіонів[16], що дозволяє виявити потенційні зони[17], де може знаходитися об’єкт. Метод дозволяє групувати точки зображення за кольорами (чи інтенсивністю для чорно-білих зображень) та враховувати різні розміри ознак[18]. Приклад розподілу на регіони зображено на рисунку 1.6.

Рисунок 1.6 – Визначення регіонів методом селективного відбору з урахуванням різних розмірів об’єктів

З часом дослідники почали задавати нове питання – навіщо нам довго розписувати окремий модуль для визначення регіонів, якщо ми можемо навчити[19,20] нейронну мережу розрізнювати їх самостійно? Так була створена модель мережі пропозиції регіонів, що замість «ручного» алгоритму селективного відбору навчає нейронну мережу задля автоматичного визначення регіонів.

Мережа пропозиції регіонів[21] використовує гістограму напрямлених градієнтів задля отримання ознак зображення; ознаки передаються до опорно-векторної мережі, що повертає набір регіонів, де може лежати об’єкт. Кожен регіон має окреме значення, що показує вірогідність знаходження об’єкту у регіоні[22]. Зазвичай, регіони зі значенням вірогідності нижче за 0,5 відкидаються. У випадку, коли регіони перетинаються, для визначення потреби у відкиданні регіону з’ясовується відношення об’єднання регіонів до їх перетину. Зазвичай, якщо відношення становить 0,5 чи вище, регіон з найменшою вірогідністю відкидається.

1.3 Сучасні мережі для розпізнавання зображень

Нейронна мережа R-CNN[23] є мережею, що використовує метод визначення регіонів для розпізнавання зображень. Перша мережа R-CNN знаходила 2000 можливих регіонів за методом селективного відбору, та викликала нейронну мережу для визначення об’єктів у кожному регіоні окремо. Скорочений алгоритм роботи R-CNN зображено на рисунку 1.7.

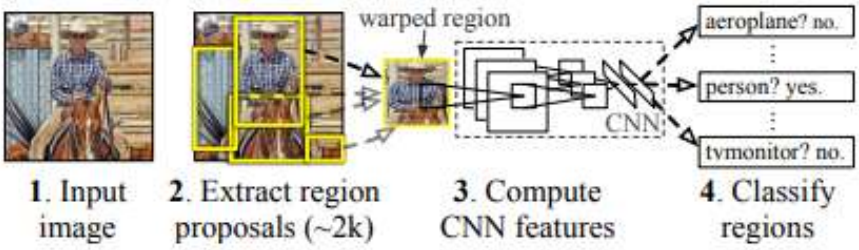

Рисунок 1.7 – Алгоритм роботи R-CNN

Проте R-CNN все ще була недостатньо швидка задля використання у реальному часі та потребувала досить довгого тренування. Тому вже через рік після створення R-CNN ті ж самі автори створили Fast R-CNN[24], основним покращенням якої стало групування отриманих регіонів з подальшою передачею їх до нейронної мережі цільним блоком. Це дозволило пришвидшити роботу мережі приблизно в 25 разів та зменшити час тренування приблизно в 10 разів. Структура Fast R-CNN зображена на рисунку 1.8.

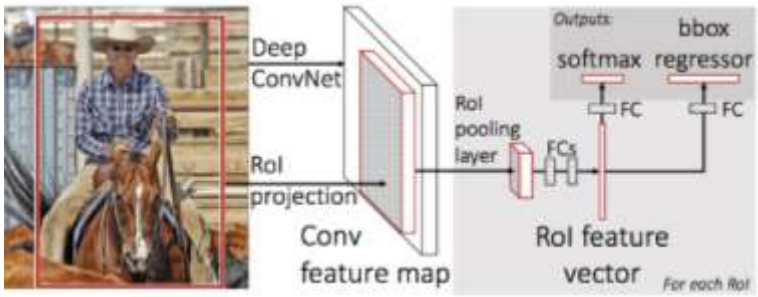

Рисунок 1.8 – Структура Fast R-CNN

Наступним кроком стало створення модель Faster R-CNN[21], у якій було вперше використано мережу пропозиції регіонів замість методу селективного відбору. Це дозволило збільшити швидкість ще в 10 разів порівняно з Fast R-CNN, довівши таким чином швидкість розпізнавання до приблизно 0,2 секунд. Структура Faster R-CNN зображена на рисунку 1.9.

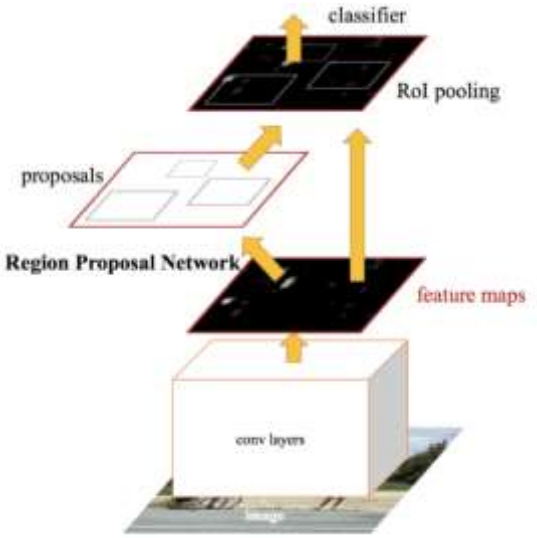

Рисунок 1.9 – Структура мережі Faster R-CNN

Порівняння швидкостей розглянутих методів подано на рисунку 1.10.

Рисунок 1.10 – Порівняння швидкостей оглянутих методів у секундах

Таким чином, Faster R-CNN могла знаходити об’єкти для відео потоку з частотою 5 кадрів в секунду, але це все ще було недостатньо для повноцінного використання.

Першою моделлю, що дозволила використання у реальному часі, була YOLO (You Only Look Once)[25]. Алгоритм роботи методу досить нескладний: модель розбиває зображення за сіткою певного розміру, і для кожної клітини сітки знаходить вірогідність знаходження у ній об’єкту та його обмежувальну рамку. Після отримання результатів об’єкти з вірогідністю менше за 0,5 відкидаються. Інноваційним у даній моделі було те, що розробники звели усі процеси методів лінії R-CNN до однієї нейронної мережі, що значно пришвидшило процес виконання. За результатами досліджень, мережа могла обробляти відео зі швидкістю від 45 до 120 кадрів в секунду залежно від обладнання. Структура мережі YOLO зображена на рисунку 1.11.

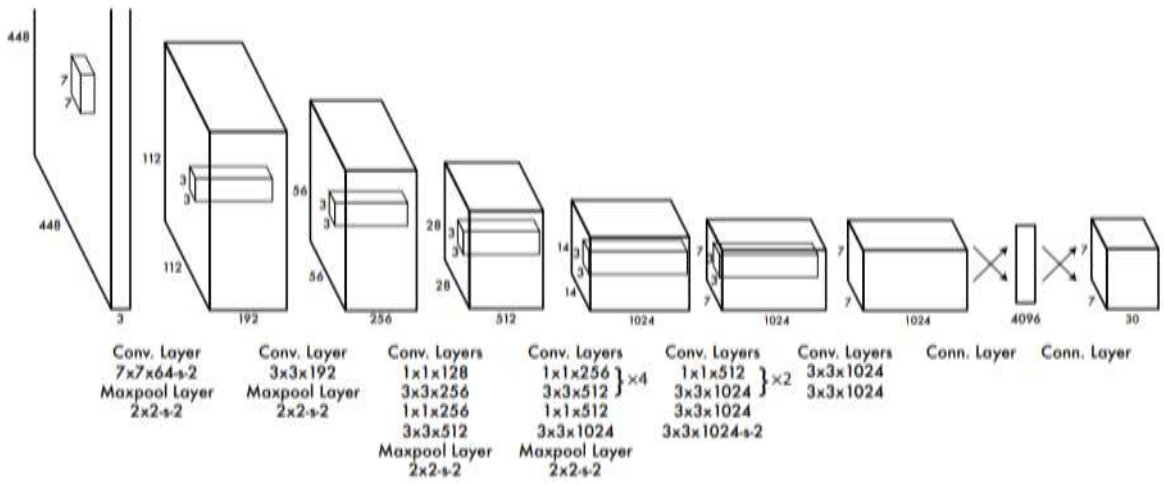

Рисунок 1.11 – Структура мережі YOLO

Наступним покращенням була технологія SSD (Single Shot Detector)[26], основним аспектом якої було виключення важких для розрахунку повноз’єднаних шарів та заміна їх на декілька згорткових, що покращило швидкість з 45 кадрів в секунду до 60, та збільшило точність на приблизно 10 відсотків. Порівняння структур SSD та YOLO показано на рисунку 1.12.

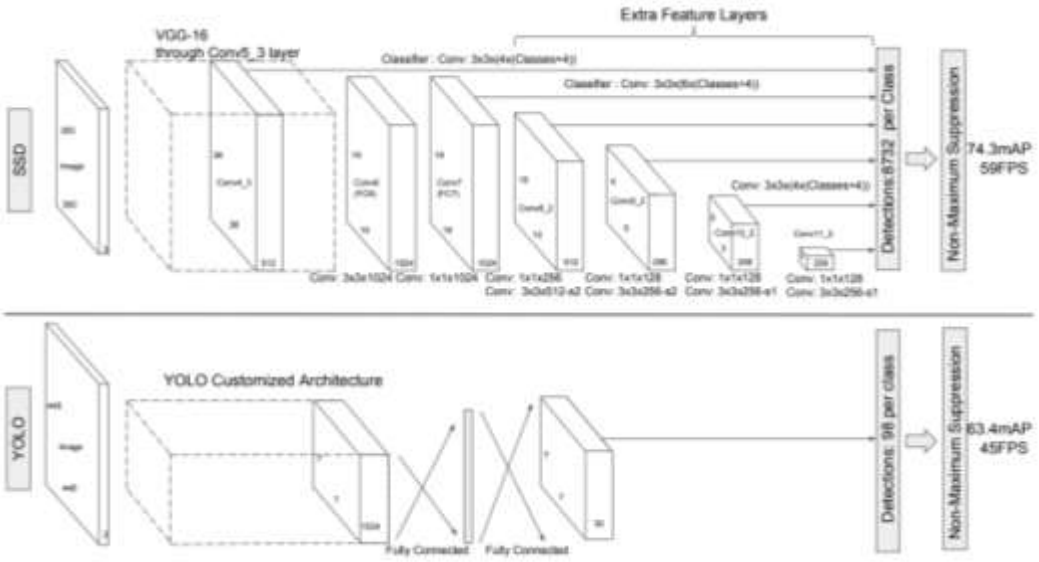

Рисунок 1.12 – Порівняння структур SSD та YOLO

У 2017, стало можливе використання моделей на телефонах завдяки проекту MobileNet[27]. Автори з компанії Google знайшли шлях пришвидшення проходження шару згортки завдяки заміні об’ємних ядер на декілька шарів згортки з простішими двовимірними та одновимірними ядрами. За результатом заміни можна було розрахувати зниження кількості розрахунків, що показано на формулі 1.5.

$$\frac{f_x \cdot f_y \cdot n_c^{l-1} \cdot n_x \cdot n_y + n_c^l \cdot n_c^{l-1} \cdot n_x \cdot n_y}{f_x \cdot f_y \cdot n_c^{l-1} \cdot n_c^l \cdot n_x \cdot n_y} = \frac{1}{n_c^l} + \frac{1}{f_x \cdot f_y}, \quad (1.5)$$

де n_c^{l-1} – кількість вхідних каналів,

n_c^l – кількість вихідних каналів,

f_x, f_y – розміри ядра згортки,

n_x, n_y – розміри вхідної матриці.

Трошки покращена технологія MobileNet V2[28] швидше за модель YOLOv2 у 20 разів з приблизно рівною точністю.

У 2019 році компанія Google розробила велику бібліотеку навчених нейронних мереж з використанням технологій SSD та MoblieNet під назвою MediaPipe[29], що дозволяла виконувати розпізнавання різноманітних об’єктів на великому спектрі пристроїв від домашніх комп’ютерів до мобільних телефонів. Зокрема нас цікавить можливість розпізнавання жестів, що надається моделлю BlazePalm[30]. Основним блоком моделі є BlazeBlock[31], та подвійний BlazeBlock, що показані на рисунку 1.13.

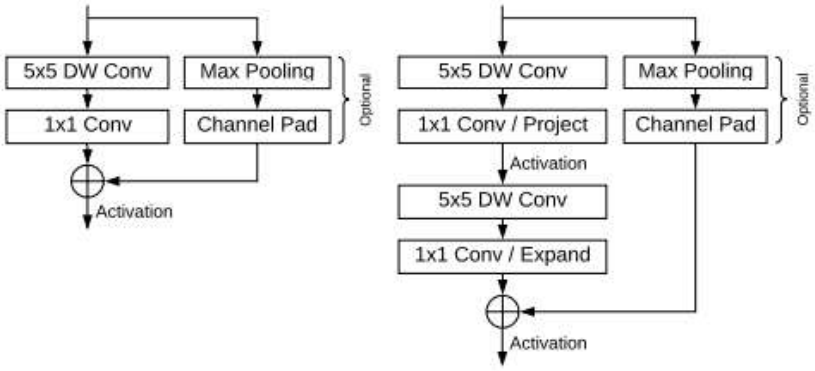

Рисунок 1.13 – BlazeBlock (зліва) та подвійний BlazeBlock (справа)

Структура моделі розпізнавання долоні з використанням даних блоків зображена на рисунку 1.14.

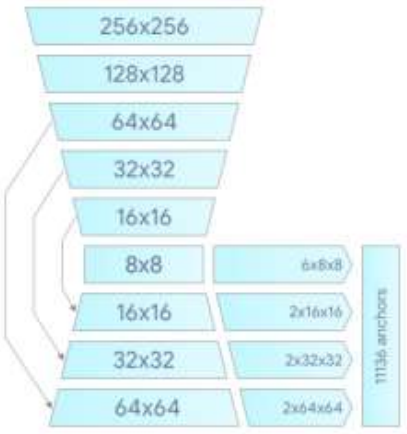

Рисунок 1.14 – Структура моделі BlazePalm

За результатом роботи мережі отримуємо набір розпізнаних рук, кожна з яких має 21 ключову точку, що зображені на рисунку 1.15.

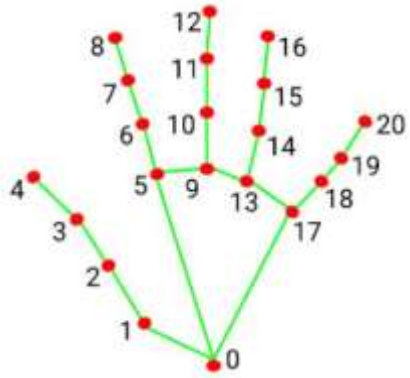

Рисунок 1.15 – Ключові точки долоні

Тестування моделі показало, що на телефоні Google Pixel 3 один кадр обробляється від 7 мілісекунд для моделі Lite до 39 для моделі Heavy. На iPhone 11 швидкість відповідних моделей становить 1,1 та 7 мілісекунд.

1.4 Постановка задачі

Таким чином, з сучасними нейронними мережами розробка дистанційного керування комп’ютером через розпізнавання жестів стає актуальною задачею. Тому ставиться завдання розробки алгоритму задля дистанційного керування комп’ютером з використанням згорткових нейронних мереж.

Об’єктом роботи є дистанційне управління мультимедійними засобами з використанням вебкамери.

Метою роботи є розроблення методу та технології на базі згорткових нейронних мереж, що дають можливість детектувати жести користувача та керувати комп’ютерною системою у залежності від розпізнаного жесту. Для досягнення мети необхідно вирішити такі завдання:

– розробити алгоритм детектування ключових точок руки;

– розробити алгоритм дистанційного керування на основі отриманих ключових точок;

– реалізувати комп’ютерну модель для дистанційного керування комп’ютером.

2 ПРОГРАМНИЙ ЗАСТОСУНОК ДЛЯ ДИСТАНЦІЙНОГО УПРАВЛІННЯ

2.1 Обґрунтування вибору середовища програмної реалізації

Модульне рішення дозволяє не зупинятися на окремому середовищі під час розробки. Так, для розробки модуля розпізнавання використовувалася мова програмування Python.

Python – мова програмування, створена у 1990 році. Вона здобула популярність в першу чергу завдяки простому синтаксису та динамічній типізації. Python є однією з найпопулярніших мов програмування, там можна сміливо казати, що вона є лідером за кількістю сторонніх бібліотек, доступ до яких можна легко отримати завдяки вбудованому командному інтерфейсу.

Проте дана мова має й ряд недоліків – через інтерпретованість, тобто не пряме перетворення програми до машинного коду, а покрокове зчитування коду інтерпретатором, мова є досить повільною порівняно з, наприклад, C чи C++. Проблемою може бути й динамічна типізація, що разом з інтерпретованістю може призвести до неочікуваних результатів[31-35].

З широким розповсюдженням нейронних мереж саме Python став основним майданчиком для дослідження даної технології, завдяки чому була створена велика кількість бібліотек, що дозволяють легко будувати власні нейронні мережі чи використовувати готові рішення для окремих популярних задач.

Для модуля дистанційного керування була обрана мова Golang, що була створена Google для вирішення питань, з якими компанія зіткнулася під час розробки власних систем.

На відміну від Python, Golang є статично типізованою компільованою мовою програмування, що дозволяє підвищити швидкість виконання програм та уникнути проблем, що виникають з динамічною типізацією.

Хоч вона й не має такої великої спільноти, як Python, мова досить швидко розвивається. Зі свого створення у 2009 році на Golang перейшли такі міжнародні фірми як Netflix, Twitch, Uber, Soundcloud та багато інших. Цією мовою були створені сервіси, без знання яких зараз майже неможливо влаштуватися працювати програмістом – Kubernetes та Docker. Мова перейняла простий синтаксис від Javascript, швидкість від C++ – хоча й не повністю, але вона все ж швидше за Python у 10-20 разів. Мабуть найприємнішим для розвитку є можливість вбудовування коду мовою C++, що може значно полегшити створення нових бібліотек. Так, бібліотека що використовується у цьому проекті (robotgo) містить переважну кількість коду C++ для взаємодії з командами низького рівня.

Задля взаємодії двох модулів було використано мережевий протокол UDP, що зазвичай використовується у відеоіграх чи сервісах по типу відео конференцій, де можна не зосереджуватися на перехваті кожного окремого пакету, та саме це є основною відмінністю протоколу від його альтернативи – TCP. UDP дозволяє передавати дані без очікування підтвердження отримання з серверу, що дозволяє уникнути проблем з випадковою втратою пакетів.

2.2 Побудова алгоритму дистанційного керування

У розділі 1 було розглянуто модель для визначення КТ руки проекту MediaPipe. Використовуючи отримані КТ, розробимо алгоритм їх обробки для подальшого використання[32-35].

Задля простоти будемо розрізняти жести за тим, зігнуті чи розігнуті окремі пальці. Для цього визначимо відстань від низу долоні (точка 0 рисунку 1.14) до кінчика пальця (точка 4 для великого пальця) та до перетину середньої та нігтьової фаланг (точка 3 для великого пальця). Якщо відстань до кінчика більше, ніж до перетину фаланг, палець вважається розігнутим. Узагальнена формула статусу пальця має вигляд:

$$s_i = d\big((i+1)\cdot 4; 0\big) - d((i+1)\cdot 4 - 1; 0), \qquad (2.1)$$

де $d(a, b)$ – відстань від ключової точки a до точки b,

i – номер пальця (0 – великий, 4 – мізинець)

Окремим випадком є великий палець, що згинається у іншій площині. Тому статус великого пальця розраховується окремо, та залежить від відстані кінчика пальця та перетину його фаланг до початку мізинця (точка 17), що відображено у формулі 2.2.

$$s_0 = d(4; 17) - d(3; 17). \qquad (2.2)$$

Так як палець має лише два статуси, його можна передати окремим бітом, значення 0 якого вважається за зігнутий палець, а значення 1 – розігнутий. Перевести результат формули 2.1 до бітового вигляду можна за наступною формулою:

$$b_i = \frac{|s_i| + s_i}{2s_i}. \qquad (2.3)$$

Враховуючи усі п’ять пальців, ми маємо $2^5 = 32$ можливі жести, що є замалим, тому додамо додатковий критерій для жесту: кут нахилу руки. Задля простоти поділимо простір, у якому лежить кут нахилу, на вісім секторів по 45°, та будемо вважати нахилом номер сектору, у який потрапляє кут між лінією, що об’єднує точки 0 (зап’ястя) та 9 (початок середнього пальця), та лінією під кутом у 67.5°. Сектори та відповідний нахил показані на рисунку 2.1.

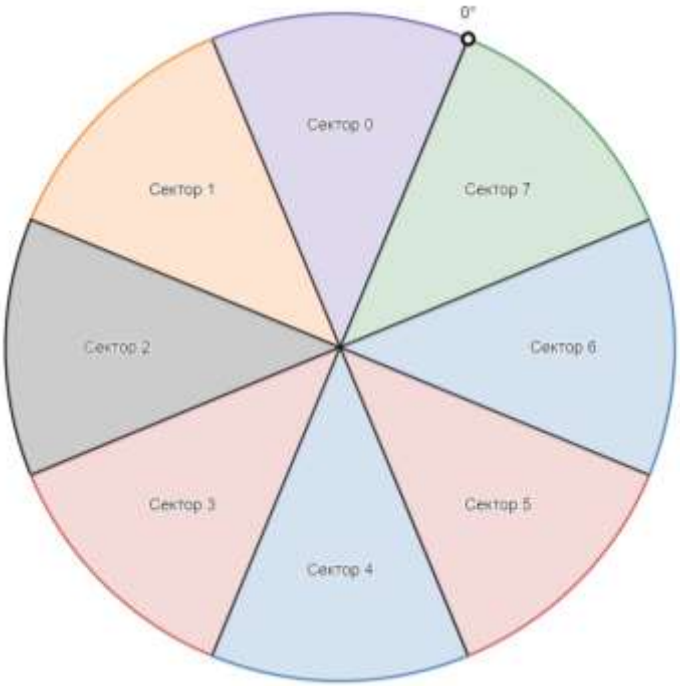

Рисунок 2.1 – Розподіл простору кута на сектори

Таким чином, склавши 5 біт коду пальців та 3 біти коду нахилу, ми можемо повністю вмістити код жесту в одному байті, тобто відтворити хешування даних[36,37]. Структура отриманого байту показана на рисунку 2.2.

128	64	32	16	8	4	2	1
Великий палець	Вказівний палець	Середній палець	Безіменний палець	Мізинець	Нахил		

Рисунок 2.2 – Структура коду жесту

Створена модель дозволяє використовувати до 256 різних команд, але слід врахувати, що не всі команди є рівноцінними. Деякі жести (наприклад, підняті безіменний палець та мізинець під кутом -45° – код 0011101) досить складні, та їх виконання тільки погіршить досвід користувача.

Маємо й іншу проблему – не кожна команда має бути закріплена за лише одним кодом. Наприклад, візьмемо ситуацію, у якій за кодом 01000000 (піднятий вказівний палець, кут 90°) закріплена команда руху миші. З даною

конфігурацією виникає ще одна незручність – як тільки нахил руки перейде до іншого сектору, команда припинить виконання. Враховуючи, що діапазон рухів руки при переміщенні миші може бути досить немалий, сталий кут у цій конкретній ситуації є ще одним недоліком, від якого слід позбавитися.

Друга проблема вирішується досить просто – слід надати можливість відключати врахування кута нахилу руки для деяких жестів, що робиться через кон'юнкцію початкового коду з бітовою маскою, що виключає непотрібні біти. Математична функція, що дозволяє отримати відфільтрований код жесту, має вигляд:

$$C(x) = x \wedge m, \tag{2.4}$$

де x – початковий код жесту;

m – маска жесту.

Для видалення даних кута маска повинна дорівнювати 7 (00000111) – це дозволить їй видалити останні три біти коду, що відповідають саме за нахил руки. Але вона може мати й куди більш широке використання в залежності від її значення – так, маска з кодом 001 дозволяє відокремити діагональний нахил руки від горизонтального та вертикального, які стають рівноцінні, а маска з кодом 011 робить рівноцінними протилежні кути нахилу. Накладання ж маски на перші 5 бітів дозволяє ігнорувати окремі пальці. Результати дій різних масок наведені у таблиці 2.1.

Проте маска має й недоліки: вона суттєво зменшує кількість можливих жестів. Повертаючись до ситуації з рухом миші, наведеній вище, можна сказати: так, проблема вирішена, але кількість «порожніх» жестів зросла на 8.

Вирішенням проблеми є створення ієрархії команд[38-40], де кожна команда може мати нащадків. Таким чином, словник кодів на кожному кроці буде залежати від того, яка саме команда є наразі керівною.

Таблиця 2.1 – Результати дії різних масок на різні кути нахилу руки

Маска	**Код нахилу**								**Опис**
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	
001	0	1	0	1	0	1	0	1	Відділення діагональних (0) кутів від вертикальних та горизонтальних (1)
010	0	0	2	2	0	0	2	2	Визначення напрямків верх-низ (0) та право-ліво (1)
100	0	0	0	0	4	4	4	4	Визначення напрямків ліво-верх (0) та право-низ (4)
011	0	1	2	3	0	1	2	3	Групування протилежних напрямків
110	0	0	2	2	4	4	6	6	Об’єднання сусідніх секторів

Поділивши команди на проміжні, які мають нащадків, та фінальні, які їх не мають, можна сформулювати правила вибору керівної команди:

- Команда стає керівною під час її виклику;
- У випадку, коли керівна команда є проміжною, вона залишається керівною доки не буде викликано одного з її нащадків, чи не буде передано коду повернення до минулої команди;
- У випадку, коли керівна команда є фінальною, вона залишається керівною, доки не буде передано код, що суперечить коду команди. Після цього керівною стає минула проміжна команда.

Таким чином, виконання команд залежить не від самих жестів, а їх послідовностей. Ще одним важливим аспектом такого підходу є можливість виконувати послідовність команд за одним кодом завдяки додаванню до проміжної команди команду з тим самим кодом.

Сучасні комп'ютери мають велику кількість додатків, та найважливіші функції у кожному з них різні; так, у музичному програвачі важливими є зміна гучності та перемикання пісень, а у браузері – перехід по вкладках. Використання для таких дій незручних жестів, чи жестів що знаходяться занадто глибоко у ієрархії погано позначиться на досвіді користувача. Тому слід передбачити можливість виконання різних команд для одного жесту в залежності від відкритого додатку.

2.3 Програмна реалізація

Для реалізації модуля виділення ключових точок долоні була використана мова програмування Python з використанням бібліотеки Mediapipe.

Програмна реалізація отримання ключових точок руки зображена на рисунку 2.3.

```
with mp_hands.Hands(
    model_complexity=0,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as hands:
  while cap.isOpened():

    success, image = cap.read()
    if not success:
      continue

    image.flags.writeable = False
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = hands.process(image)

    if results.multi_hand_landmarks:
      for hand_landmarks in results.multi_hand_landmarks:
        x = hand_landmarks.landmark[0].x*10/6-1/3
        y = hand_landmarks.landmark[0].y*10/6-1/3
        sock.sendto(str.encode("{:.4f} {:.4f} {}".format(1-x,y, getByteCode(hand_landmarks))),serverAddr)
    cv2.imshow('MediaPipe Hands', cv2.flip(image, 1))
    if cv2.waitKey(5) & 0xFF == 27:
      break
cap.release()
```

Рисунок 2.3 – Програмна реалізація отримання ключових точок руки

Після отримання ключових точок потрібно конвертувати їх до байтового коду жесту. Для цього потрібно для кожного пальця виявити, зігнутий він чи ні, а також знайти кут нахилу руки. Реалізація конвертування до байтового коду зображена на рисунку 2.4.

```
def toByte(x):
  if x == 0:
    return 1
  return int((abs(x)+x)/(2*x))

def getDistance(hand,f1,f2):
  return math.sqrt((hand.landmark[f1].x-hand.landmark[f2].x)**2
    +(hand.landmark[f1].y-hand.landmark[f2].y)**2
    +(hand.landmark[f1].z-hand.landmark[f2].z)**2)

def getAngle(hand,id1,id2):
  return math.atan2(hand.landmark[id1].y - hand.landmark[id2].y,
            hand.landmark[id1].x-hand.landmark[id2].x)*180/math.pi

def getByteCode(hand):
  angle = getAngle(hand,0,9)
  angleVal = int(math.floor((angle - 67.5)/45+8))%8
  thumb = toByte(getDistance(hand,17,4)-getDistance(hand,17,3))
  index = toByte(getDistance(hand,0,8) - getDistance(hand,0,7))
  middle = toByte(getDistance(hand,0,12) - getDistance(hand,0,11))
  ring = toByte(getDistance(hand,0,16) - getDistance(hand,0,15))
  pinky = toByte(getDistance(hand,0,20) - getDistance(hand,0,19))
  return angleVal+thumb*128+index*64+middle*32+ring*16+pinky*8
```

Рисунок 2.4 – Конвертування ключових точок до байтового коду

Після конвертування код жесту та координати вказівного пальця по протоколу UDP через порт 5032 передаються до програми дистанційного керування, що написана мовою Golang. Реалізація отримання даних програмою наведена на рисунку 2.5.

Отримані дані передаються до керуючого класу, що намагається виконати певні команди в залежності від коду жесту. Команди скомпоновані у окремі дії – кожна дія містить окремі команди для початку виконання жесту, його закінчення та тривалого утримання. Таким чином можна реалізувати і натискання клавіші миші, що потребує одноразового виконання команди, і рух миші, що викликає команду поки жест не зміниться.

```
func SetupConn() {
    connection, err := net.ListenUDP("udp", &net.UDPAddr{
        Port:5032,
        IP:net.ParseIP("127.0.0.1")})
    if err != nil {
        fmt.Println(err)
        return
    }

    defer connection.Close()
    buffer := make([]byte, 1024)
    for {
        n, _, err := connection.ReadFromUDP(buffer)
        if err!=nil {
            panic(err)
        }
        hand.Loop(string(buffer[0:n]))
    }
}
```

Рисунок 2.5 – Отримання даних програмою дистанційного керування

Дія може містити нащадків, що згруповані за маскою коду. Якщо код жесту з накладеною маскою дорівнює коду нащадку, керуючою дією стає цей нащадок.

Останнім компонентом дії є специфічні команди, що викликаються в залежності від активного вікна.

Програмна структура класу дії зображена на рисунку 2.6.

```
type Action struct {
    Code uint8 `json:"-"`
    Mask uint8 `json:"-"`
    Final bool `json:"-"`
    Name string `json:"name"`
    Active bool `json:"-"`
    Parallel bool `json:"parallel",omitempty`
    ChildrenCnt int `json:"-"`
    Parent *Action  `json:"-"`
    Start *Gesture  `json:"start,omitempty"`
    Loop *Gesture   `json:"loop,omitempty"`
    End *Gesture    `json:"end,omitempty"`
    Apps map[string]*Action `json:"apps,omitempty"`
    Children map[uint8]map[uint8]*Action `json:"children,omitempty"`
}
```

Рисунок 2.6 – Структура класу дії

Мова програмування Golang дозволяє після типу змінної у структурі вказувати структурні теги, що дозволяють легко керувати відображенням класів у файлах опису об’єктів різних форматів, таких як json чи xml.

Основною метою під час створення додатку була можливість розширення, яка була реалізована через створення окремого файлу налаштування у форматі json. Після запуску додаток зчитує файл налаштування та створює структуру жестів, вказану користувачем. Приклад зовнішнього вигляду файлу налаштування наведено на рисунку 2.7.

```
{
    "backCode": "0",
    "inputModule": [
        "python",
        "gestures/webcamModule.py"
    ],
    "action": {
        "name": "Start menu",
        "children": {
            "255": {
                "128": {
                    "name": "Mouse menu",
                    "children": {
                        "248": {
                            "96": {
                                "name": "Move Mouse",
                                "loop": {
                                    "mouse": {
                                        "action": "move"
                                    }
                                }
                            },
                            "64": {
                                "name": "Mouse Click Left",
                                "start": {
                                    "mouse": {
                                        "action": "click",
                                        "key": "left"
                                    }
                                }
                            },
```


Рисунок 2.7 – Структура файлу налаштування

В даній структурі також можна побачити такі поля, як backCode та inputModule. Перше поле відповідає за узагальнений код повернення до минулої дії, який у даній структурі відповідає піднятому догори кулаку. Друге поле відповідає за консольну команду, що запускає модуль розпізнавання жестів. Команда, наведена у структурі викликає написаний

мовою python додаток, що називається webcamModule. За бажанням користувач зможе власноруч написати модуль розпізнавання, та при правильному форматі вихідних даних програма керування зможе їх оброблювати.

2.4 Інструкція користувача

Для використання програми потрібно встановити Python, що можна зробити з офіційного сайту мови програмування. Перед запуском потрібно встановити декілька бібліотек, зокрема OpenCV, Tensorflow, та Mediapipe. Встановити їх можна відповідними консольними командами:

- pip install opencv-python;
- pip install tensorflow;
- pip install mediapipe.

Перед запуском основного додатку рекомендується спробувати запустити модуль розпізнавання командою python webcamModule.py для перевірки бібліотек. Якщо були встановлені не всі бібліотеки, програма видасть помилку з назвою потрібних бібліотек, які можна встановити командою pip install назва_бібліотеки.

Застосунок запускається через файл controller.exe. Після запуску користувач отримає можливість перевірити роботу програми зі стандартною конфігурацією жестів:

- Кулак з великим пальцем назовні, рука на північ – меню миші;
    1) вказівний та середній палець догори – рух миші;
    2) вказівний палець догори – ліва кнопка миші;
    3) мізинець догори – права кнопка миші;
    4) Вказівний та мізинець догори («коза»), рука на північ – прокрутка миші вверх;

5) «Коза», рука на південь – прокрутка миші вниз;

– Кулак з великим пальцем назовні, рука на північний захід – меню клавіатури;

1) Великий палець догори – написати «Hello World!»;

– Кулак з великим пальцем назовні, рука на захід – меню програм;

1) Вказівний палець догори – минула вкладка для Google Chrome;

2) Вказівний та середній пальці догори – наступна вкладка Google Chrome;

– Кулак з великим пальцем всередину, рука на північ – повернутися до минулої дії.

Дана конфігурація має ознайомити користувача з основними можливостями програми.

Для налаштування конфігурації відкрийте файл config.json. Найпростіша структура файлу налаштування зображена на рисунку 2.8.

```
{
    "backCode": "0",
    "inputModule": [
        "python",
        "gestures/webcamModule.py"
    ],
    "action": {}
}
```

Рисунок 2.8 – Найпростіша структура файлу налаштування

Розберемо окремі елементи файлу:

– backCode – код жесту повернення до минулого елементу;

– inputModule – консольна команда, що запускає модуль розпізнавання жестів. Кожна команда та аргумент мають бути розділені у окремі рядки;

– action – основна дія, з якої починається робота програми. Зазвичай містить елемент меню, у якому відсутні команди та присутні лише дії-нащадки.

Основними елементами файлу налаштування є дії. Структура дії наведена на рисунку 2.9.

```
"action":{
    "name":"Назва дії",
    "parallel":true|false,
    "start":{
        Початкова команда
    },
    "loop":{
        Тривала команда
    },
    "end":{
        Кінцева команда
    },
    "children":{
        "код_маски_1":{
            "код_жесту_1":{
                Дія-нащадок
            },
            "код_жесту_2":{
                ...
            }
        },
        "код_маски_2":{
            ...
        }
    },
    "apps":{
        "назва_додатку_1":{
            Специфічна дія для додатку
        },
        "назва_додатку_2":{
            ...
        }
    }
}
```

Рисунок 2.9 – Структура дії.

Єдиним обов’язковим полем дії є name. Дія може виконувати три різні команди на початку виклику, у процесі та після зміни жесту (поля start, loop, end). Нащадки дії розташовані у полі children, вони згруповані за кодом маски. Найкориснішими масками є 255 (повне урахування руки) та 248 (відкидання нахилу). Всередині коду маски знаходяться нащадки дії, розподілені за кодом жесту.

Альтернативні дії, що залежать від активного додатку, знаходяться у полі apps та розподілені за назвою додатку. Назва додатку може бути

часткова та не чутлива до регістру; наприклад, назва «google» зможе активуватися при відкритому браузері Google Chrome. У випадку, коли назви перетинаються, випадково викликається одна з цих дій. Тому рекомендується виділяти унікальні імена, наприклад для Google Chrome та Google Meet використовувати назви «chrome» та «meet», а не «google».

Тепер розберемо структуру команди, позначену на рисунку 2.10.

```
"mouse":{
    "action":"move"|"click"|"toggleUp"|"toggleDown"|"scrollUp"|"scrollDown",
    "key":"left"|"right"|"middle",
    "doubleClick":true|false
},
"keyboard":{
    "action":"tap"|"up"|"down"|"print",
    "key":"",
    "args":["","",...],
    "typeStr":""
}
```

Рисунок 2.10 – Структура команди

Наразі існує два типи команд: миші та клавіатури. Розберемо структуру команди миші. В залежності від обраної дії (action) окремі поля можна проігнорувати. Так, при діях move, scrollUp, scrollDown інші поля непотрібні; поле doubleClick використовується лише з дією click.

Команда клавіатури має три можливі дії: tap, toggle, print. Перелік можливих клавіш можна знайти на сторінці бібліотеки robotgo[41]. Керуючими клавішами є клавіші типу Alt, Ctrl та ін.. Для дії toggle замість першої керуючої клавіші слід використовувати up, якщо клавішу слід віджати та down, якщо її слід зажати. Поле typeStr використовується лише для дії print.

У випадку коли потрібна лише одна команда, наприклад, для миші, команду для клавіатури можна проігнорувати. При використанні обох команд першою виконується команда миші.

Якщо ж потрібно виконати послідовно декілька команд, слід використовувати дії-нащадки з тим же кодом жесту, що й в основної дії. У

такому випадку по зміні жесту керуюча дія буде переходити до останньої дії-меню системи.

Під час налаштування може виникнути ситуація, коли паралельно мають виконуватися декілька дій. В такому випадку можна використати прапор parallel, що дозволяє виконувати декілька команд одночасно. Розглянемо приклад, у якому ми створюємо команди для гоночної гри, за такими правилами:

– при розігнутої долоні машина їде вперед, при зігнутій – назад;

– при повороті долоні машина має повертати вліво чи вправо;

– при жесті «коза» рівень має перезапускатися.

При повороті все ще має бути натиснута клавіша вперед чи назад, залежачи від жесту. Тобто у будь який момент часу можуть спрацьовувати декілька команд, чого при минулих конфігураціях досягнути досить складно. Прапор дії parallel дозволяє легко це зробити. Можлива структура такої дії позначена на рисунку 2.11.

За масками дій-нащадків видно, що одна з цих дій розглядає лише пальці (маска 248), а інша – лише нахил (маска 7). У нащадку з маскою 248 та тим же кодом можна побачити структуру команд: початкова команда натискає клавішу «вперед», а кінцева відтискає. Команди інших дій працюють аналогічно, та змінюють лише кнопку, що має бути натиснута.

Слід зазначити, що при паралельному виконанні не передбачена можливість використання команд у процесі утримання жесту. Паралельні дії мають використовувати лише початкову та кінцеву дії, де початкова як правило вносить зміну до керування, а кінцева цю дію відміняє. Використання команди, що має виконуватися у процесі утримання жесту буде проігноровано.

Рисунок 2.11 – Структура дії з паралельним виконанням команд

2.5 Тестування розробленої моделі

Задля тестування моделі було розроблено окрему програму, що дозволяє аналізувати потік даних жестів. Програма розраховує, як довго жест був незмінний, що дозволяє виявити шуми у потоці даних. На основі аналізу

було виявлено, що жести, що продовжуються чотири та більше циклів програми становлять 92% усіх жестів, а жести довжиною один цикл (шуми) – близько 7%. Частка жестів довжиною 2 та 3 цикли ж менше відсотка.

Таким чином, проведене тестування дозволило виявити головний недолік системи – через відсутність фільтрування даних можлива випадкова зміна команди після появи невірно розпізнаного жесту. За інформацією методу MediaPipe, точність їхнього методу розпізнавання становить близько 95%, що відповідає отриманим результатам. Слід зазначити, що основою помилки є переважно зовнішнє середовище та його сприйняття через вебкамеру. Розмір помилки може змінюватися в залежності від середовища та апаратного забезпечення[3].

Задля вирішення проблеми шумів було вирішено розробити метод фільтрування, що дозволив би уникати випадків надходження жестів довжиною в один цикл до системи. Для цього було обрано метод буферизації, за яким минулі коди команд зберігаються у буфері довжиною в три жести, серед яких фільтр обирає найбільш вірогідний варіант. Методика вибору зображена у таблиці 2.2.

Таблиця 2.2 – Вибір найбільш вірогідного значення жесту

Значення окремого поля буферу			**Результат**
Поле 1	**Поле 2**	**Поле 3**	
x	x	y	x
y	x	y	y
x	y	y	y
x	y	z	x

Єдиним випадком коли можлива поява жесту з довжиною виконання в один цикл, є випадок у якому усі три останні жести мають довжину 1.

Вірогідність цього становить $0{,}07^3 = 0{,}03\%$, але впродовж довгого користування така вірогідність не може вважатися малою та відкидатися. Так, з даною вірогідністю, один жест-шум буде виникати приблизно один раз на 100 секунд.

Щоб позбавитися від цієї проблеми було вирішено зберігати останній результат фільтрації, та використовувати його замість результату для випадку, вказаного вище.

Але, хоч обраний метод буферизації і має значні переваги, в нього є серйозний недолік – відповідно зі зростанням розміру буферу зростає й затримка опрацювання жесту. Так, вважаючи, що за вірний жест вважається той, що має більше половини появ у буфері, різниця в часі між першою появою коду жесту в буфері та його опрацюванням буде становити половину довжини буферу. З обраним розміром буфера рівним 3, затримка перед виконанням дії становить один цикл виконання.

Подальше тестування показало, що з використанням фільтрувального методу буферизації проблема шумів при використанні програми була усунута.

2.6 Перспективи подальшої роботи

Основним напрямком подальшого розширення має бути розширення можливих команд. Можна сказати, що з використанням лише клавіатури та миші користувач зможе виконувати більшість необхідних дій, але все ще є шляхи для покращення, такі як додавання можливості виконання великої послідовності команд окремою дією. Наступним важливим кроком може бути розробка команд для керування системою, таких як виклик консольних команд, керування вікнами тощо.

Також завдяки модульності стає можливою реалізація нових модулів розпізнавання для іншої периферії (тактильних рукавичок, стерео камер). Що

важливіше, модульність дозволить легко перейти з досить простого розпізнавання, що лише з’ясовує піднятий палець чи ні, до складного розпізнавання жестів нейронними мережами. Модуль розпізнавання дає розробникам повну свободу в усьому окрім одного аспекту – вихідних даних. За бажанням він навіть може генерувати статичні дані, що може бути використано для автоматизації виконання команд.

ВИСНОВКИ

У рамках даної роботи було розроблено і реалізовано метод дистанційного керування комп’ютером з використанням мультимедійних засобів, зокрема – вебкамери комп’ютера.

Задля отримання команд користувача було використано технологію BlazePalm, що дозволяє за відео потоком з камери визначати ключові точки долоні. Дана технологія використовує навчені згорткові нейронні мережі, що базуються на технологіях SSD та MobileNetV2. Вона дозволяє виконувати обчислення на швидкості до 60 кадрів на секунду та має точність близько 95%.

Задля покращення точності програми було використано фільтраційний метод буферизації, що дозволяє уникнути випадків виникнення шумів у потоці даних, отриманих з аналізу ключових точок, за рахунок невеликої затримки у виконанні команд користувача.

Розроблена модель має великий спектр розширення функціоналу завдяки двом її ознакам: модульності та можливості налаштування команд. Модульність дозволяє відокремити програму керування та програму розпізнавання, завдяки чому є можливість переносу розпізнавальної функції на інші технічні засоби, такі як стерео камери чи тактильні рукавички. Можливість налаштування команд дозволяє структурувати та доповнювати список можливих дій системи через окремий файл налаштування формату JSON.

Окремим напрямком для подальших досліджень є навчання нейронних мереж на основі апарату дескрипторів ключових точок зображень, дослідження у цій сфері могли б дати більш якісну класифікацію за менший час завдяки гнучкості підходу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. English, W. K., Engelbart, D. C., & Berman, M. L. (1967). Display-selection techniques for text manipulation. IEEE Transactions on Human Factors in Electronics, (1), 5-15.

2. Gorokhovatsky, V.O. and Gadetska, S.V., (2019) Determination of Relevance of Visual Object Images by Application of Statistical Analysis of Regarding Fragment Representation of their Descriptions, Telecommunications and Radio Engineering, 78 (3), pp. 211–220.

3. Gorokhovatskyi, V., Rusakova, N., Tvoroshenko, I. (2020) The application of image analysis methods and predicate logic in applied problems of magnetic monitoring. Telecommunications and Radio Engineering, 79 (20), pp. 1801-1811.

4. M. A. Ahmad, V. Gorokhovatskyi, I. Tvoroshenko, N. Vlasenko, S. Kh. Mustafa (2021) The Research of Image Classification Methods Based on the Introducing Cluster Representation Parameters for the Structural Description, International Journal of Engineering Trends and Technology, 69(10), pp. 186-192.

5. Гороховатський В.О., Пупченко Д.В., Солодченко К.Г. (2018) Аналіз властивостей, характеристик та результатів застосування новітніх детекторів для визначення особливих точок зображення. Системи управління, навігації та зв'язку. С. 93–98.

6. Гороховатський В.О., Творошенко І.С. (2022) Аналіз багатовимірних даних за описом у формі множини компонент: монографія. Харків: ХНУРЕ. 124 с.

7. M. A. Ahmad, V. Gorokhovatskyi, I. Tvoroshenko, N. Vlasenko, S. Kh. Mustafa (2021) The Research of Image Classification Methods Based on the Introducing Cluster Representation Parameters for the Structural Description, International Journal of Engineering Trends and Technology, 69(10), pp. 186-192.

8. Путятин Е.П., Аверин С.И. (1990) Обработка изображений в робототехнике. Москва: Машиностроение, 320 с.

9. Gorokhovatskyi V., Putyatin Y., Gorokhovatskyi O, Peredrii O. (2018) Quantization of the Space of Structural Image Features as a Way to Increase Recognition Performance. The Second IEEE International Conference on DataStream Mining & Processing 21-25 August 2018, Lviv, Ukraine. – pp. 464 – 467.

10. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., Al-Dhaifallah M., (2022) Classification of Images Based on a System of Hierarchical Features, Computers, Materials & Continua, 72(1), pp. 1785-1797.

11. Gorokhovatskyi O., Gorokhovatskyi V., Peredrii O. (2018) Analysis of Application of Cluster Descriptions in Space of Characteristic Image Features. p. 52.

12. Gorokhovatskyi V.A. (2018) Image Classification Methods in the Space of Descriptions in the Form of a Set of the Key Point Descriptors. Telecommunications and Radio Engineering, 77 (9), pp. 787-797.

13. Гороховатский В.А., Путятин Е.П., Столяров В.С. (2017) Исследование результативности структурных методов классификации изображений с применением кластерной модели даннях. Радиоэлектроника, информатика, управление, №3 (42). С. 78–85.

14. Путятін Є.П., Гороховатський В.О., Матат О.О. (2006) Методи та алгоритми комп’ютерного зору: навч. посібник.

15. Gorokhovatskyi V., Gadetska S., Ponomarenko R. (2020) Recognition of Visual Objects Based on Statistical Distributions for Blocks of Structural Description of Image. Lecture Notes in Computational Intelligence and Decision Making. Proc. of the XV International Scientific Conference “Intellectual Systems of Decision Making and Problems of Computational Intelligence” (ISDMCI'2019), Ukraine, May 21–25, 2019, pp. 501-512.

16. Van de Sande, K. E., Uijlings, J. R., Gevers, T., & Smeulders, A. W. (2011, November). Segmentation as selective search for object recognition. In 2011 international conference on computer vision (pp. 1879-1886). IEEE.

17. Гороховатский В.А., Передрий Е.О. (2009) Корреляционные методы распознавания изображений путем голосования систем фрагментов. Радіоелектроніка. Інформатика. Управління, №1(20), с.74–81.

18. Gorokhovatsky, V.O. and Gadetska, S.V., (2019) Determination of Relevance of Visual Object Images by Application of Statistical Analysis of Regarding Fragment Representation of their Descriptions, Telecommunications and Radio Engineering, 78 (3), pp. 211–220.

19. Гадецька С.В., Гороховатський В.О., Стяглик Н.І. (2020) Вивчення критеріїв інформативності даних при впровадженні апарату дерев рішень у методах структурної класифікації зображень. Радіоелектроніка, інформатика, управління, №3 , с. 78–87.

20. Scherer R. (2018) Computer Vision Methods for Fast Image Classification and retrieval, Częstochowa, Springer, p 137.

21. Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28.

22. Гороховатський В.О., Гадецька С.В., Стяглик Н.І. (2019) Вивчення статистичних властивостей моделі блочного подання для множини дескрипторів ключових точок зображень. Радіоелектроніка, інформатика, управління, №2, с. 100–107.

23. Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).

24. Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

25. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

26. Liu, W. et al. (2016). SSD: Single Shot MultiBox Detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science(), vol 9905. Springer, Cham.

27. Howard, Andrew & Zhu, Menglong & Chen, Bo & Kalenichenko, Dmitry & Wang, Weijun & Weyand, Tobias & Andreetto, Marco & Adam, Hartwig. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.

28. Sandler, Mark & Howard, Andrew & Zhu, Menglong & Zhmoginov, Andrey & Chen, Liang-Chieh. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. 4510-4520. 10.1109/CVPR.2018.00474.

29. Lugaresi, Camillo & Tang, Jiuqiang & Nash, Hadon & McClanahan, Chris & Uboweja, Esha & Hays, Michael & Zhang, Fan & Chang, Chuo-Ling & Yong, Ming & Lee, Juhyun & Chang, Wan-Teh & Hua, Wei & Georg, Manfred & Grundmann, Matthias. (2019). MediaPipe: A Framework for Building Perception Pipelines.

30. Zhang, Fan & Bazarevsky, Valentin & Vakunov, Andrey & Tkachenka, Andrei & Sung, George & Chang, Chuo-Ling & Grundmann, Matthias. (2020). MediaPipe Hands: On-device Real-time Hand Tracking.

31. Bazarevsky, Valentin & Kartynnik, Yury & Vakunov, Andrey & Raveendran, Karthik & Grundmann, Matthias. (2019). BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs.

32. Gorokhovatsky, V.A., (2014), Structural analysis and intellectual data processing in computer vision, SMIT, Kharkiv, p 316.

33. Gadetska, S.V., Gorokhovatskyi, V. O., Stiahlyk, N. I., Vlasenko, N.V. (2021) Statistical data analysis tools in image classification methods based on the description as a set of binary descriptors of key points. Radio Electronics, Computer Science, Control, №4, pp. 58-68.

34. Гороховатський В.О., Гадецька С.В., Стяглик Н.І., Власенко Н.В. (2020) Класифікація зображень на підставі ансамблю статистичних розподілів за класами еталонів для компонентів структурного опису. Радіоелектроніка, інформатика, управління, №4 , с. 85–94.

35. Gorokhovatskyi V.A., Zamula A.A. (2016) Employment of Intelligent Technologies in Multiparametric Control Systems. Telecommunications and Radio Engineering. Vol. 75, No 19, p. 1775–1785.

36. Гороховатський, В.О., Власенко, Н.В., Рибалка, М.О. (2021) Застосування засобів хешування даних для прискорення класифікаційних рішень у структурних методах розпізнавання зображень. Сучасні інформаційні системи, т. 5, №2, с. 13–20.

37. Gorokhovatskiy V.A., Gorokhovatskiy A.V., Peredrii Ye.O. (2018) Hashing of Structural Descriptions at Building of the Class Image Descriptor, Computing of Relevance and Classification of the Visual Objects. pp. 1159–1168.

38. Гороховатський, В.О., Творошенко, І.С. (2021) Методи інтелектуального аналізу та оброблення даних: навч. посібник. Харків: ХНУРЕ.

39. Tvoroshenko I., and Gorokhovatskyi V. (2022) The Application of Hybrid Intelligence Systems for Dynamic Data Analysis, International Journal of Engineering and Information Systems, 6(2), pp. 40-48.

40. Tvoroshenko, I.S., Gorokhovatsky, V.O. (2019) Modification of the branch and bound method to determine the extremes of membership functions in fuzzy intelligent systems. Telecommunications and Radio Engineering, 78 (20), pp. 1857-1868.

41. robotgo/keys at master go-vgo/robotgo. URL: https://github.com/go-vgo/robotgo/blob/master/docs/keys.md (дата звернення 16.05.2022).