НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Створення СППР розпізнавання дій гравців спортивних команд»

Виконав:
студент IV курсу, групи КА-75
Степанюк Владислав Сергійович _____
Керівник:
Професор, д.т.н., Данилов Валерій Якович _____
Консультант з економічного розділу:
Доцент, к.е.н., Рощина Н.В. _____
Консультант з нормоконтролю:
Доцент, к.т.н. Коваленко А.Є. _____
Рецензент:
Професор, д.т.н., Новіков О. М. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____Степанюк В. С.

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи та методи штучного інтелекту»

ЗАТВЕРДЖУЮ

завідувач кафедри

_____ Оксана ТИМОЩУК

«26» травня 2021 р.

ЗАВДАННЯ

на дипломну роботу студенту

Степанюку Владиславу Сергійовичу

1. Тема роботи «Створення СППР розпізнавання дій гравців спортивних команд», керівник роботи Данилов Валерій Якович, доктор технічних наук, затверджені наказом по університету від «26» травня 2021 р. № 1344-с
2. Термін подання студентом роботи 8.06.2021.

3. Вихідні дані до роботи: коректні результати при взаємодії з системою.
4. Зміст роботи: аналіз різних варіантів побудови систем та реалізація програмного продукту, з використанням обраного варіанту системи.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій

тощо): актуальність задачі, виявлення об’єктів, відстеження об’єктів, yolo-v4.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В.		

7. Дата видачі завдання: 27 березня 2021

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вивчення літератури за темою роботи.	13.04.2021	Виконано
2.	Підготовка першого розділу.	21.04.2021	Виконано
3.	Підготовка другого розділу.	1.05.2021	Виконано
4.	Розробка програмного продукту.	19.05.2021	Виконано
5.	Підготовка третього розділу	25.05.2021	Виконано
6.	Підготовка економічної частини	28.05.2021	Виконано
7.	Оформлення розділів відповідно до нормоконтролю.	29.05.2021	Виконано
8.	Підготовка презентації доповіді.	29.05.2021	Виконано
9.	Оформлення дипломної роботи.	29.05.2021	Виконано

Студент Владислав Сергійович Степанюк

Керівник Валерій Якович ДАНИЛОВ

РЕФЕРАТ

Дипломна робота: 90 с., 6 табл., 31 рис., 2 додатки, 12 джерел.

Об’єкт дослідження – дії гравців спортивних команд. Детальніше в цій роботі буде розглянуто рухи гравців баскетбольних команд.

Предмет дослідження – алгоритми розпізнавання образів, таких як людина(гравець) та баскетбольне поле.

Метою дослідження є створення системи, яка займатиметься розпізнаванням баскетбольних гравців та грального поля, і подальша класифікація їх рухів.

Методи дослідження – методи комп’ютерного зору: семантична сегментація, класифікація, виявлення та відстеження об’єктів.

Актуальність – автоматизація аналізу даних спортивної гри дозволить гравцям набагато швидше навчатись, а тренерам – покращити взаємодію з командою.

Результати роботи – було створено додаток, зі зручним користувацьким інтерфейсом, що допоможе автоматично виявляти на відео, що подане на вхід, гральне поле та гравців на ньому. Рухи гравців при цьому відстежуються та запам’ятовуються впродовж всього відео.

Шляхи подальшого розвитку предмету дослідження – удосконалення методів розпізнавання гравців таким чином, щоб вони змогли виявляти гравців з інших команд, для розпізнавання грального поля – створити модель грального поля, що пришвидшить його виявлення на відео. Також можна додати відстеження грального м’яча та розпізнавання конкретних дій гравців.

ABSTRACT

Thesis: 90 p., 6 tabl., 31 fig., 2 appendces, 12 sources.

The object of research – the actions of sports team players. In this work, the movements of basketball players will be considered in more detail.

The subject of the research – algorithms for pattern recognition, such as a person (player) and a basketball court.

The aim of the study is to create a system that will deal with the recognition of basketball players and the playing field, and further classification of their movements.

Research methods – methods of computer vision: semantic segmentation, classification, detection and tracking of objects.

Relevance – automation of sports game data analysis will allow players to learn much faster, and coaches - to improve interaction with the team.

Results of work – an application was created with a user-friendly interface that will help to automatically detect in the video submitted to the entrance, the playing field and the players on it. Players' movements are tracked and memorized throughout the video.

Ways to further develop the subject of research – to improve methods of player recognition so that they can identify players from other teams, to recognize the playing field - to create a model of the playing field, which will speed up its detection on video. Also, tracking of the game ball and recognition of specific actions of players can be added.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 8

ВСТУП 9

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ 11

1.1 Актуальність задачі, що розглядається 11

1.2 Аналіз існуючих підходів до вирішення задачі 14

1.3 Особливості предметної області 16

1.4 Постановка задачі дослідження 18

1.5 Висновки до розділу 1 18

РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ 19

2.1 Дослідження існуючих методів для вирішення задачі 19

2.2 Критерії якості рішення задачі 28

2.3 Висновок до розділу 2 32

РОЗДІЛ 3 ОПИС ПРОГРАМНОГО ПРОДУКТУ 33

3.1 Обґрунтування вибору платформи та мови програмування 33

3.2 Скріншоти програми 43

3.3 Результати роботи 46

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ 49

4.1 Постановка задачі проектування 50

4.2 Обґрунтування функцій програмного продукту 51

4.3 Обґрунтування системи параметрів ПП 54

4.4 Аналіз експертного оцінювання параметрів 57

4.5 Аналіз рівня якості варіантів реалізації функцій 61

4.6 Економічний аналіз варіантів розробки ПП 62
4.7 Вибір кращого варіанту ПП техніко-економічного рівня 68
4.8 Висновки до розділу 4 69
ВИСНОВКИ 70
ЛІТЕРАТУРА 71
ДОДАТОК А ЛІСТИНГ ПРОГРАМИ 73
ДОДАТОК Б ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ 85

ПЕРЕЛІК СКОРОЧЕНЬ

PDB – Python Debugger.

CNN – Convolutional neural network.

PCA – Principal component analysis.

TLD – Tracking, learning, detection.

FPN – Feature Pyramid Network.

SORT – Simple Onlinre Realtime Tracker.

COCO – Common Objects in Context.

HSV – Hue, Saturation, Value.

ПЗ – програмне забезпечення.

ПП – програмний продукт.

ВСТУП

Проблема. Коли ми декомпозуємо складну проблему, ми часто знаходимо закономірності серед більш дрібних проблем, які ми створюємо. Розпізнавання образів - один з чотирьох наріжних каменів комп'ютерної науки. Воно включає в себе пошук закономірностей або патернів серед невеликих, декомпозованих проблем, що може допомогти нам вирішити більш складні проблеми більш ефективно.

Актуальність. Пошук закономірностей або шаблонів надзвичайно важливий. Шаблони спрощують наше завдання. Проблеми легше вирішити, коли вони мають спільні шаблони, оскільки ми можемо використовувати одне і те ж рішення для вирішення проблем скрізь, де існує шаблон. Чим більше моделей ми зможемо знайти, тим простішим і швидшим буде наше загальне завдання вирішення проблем.

Розпізнавання образів важливо, тому що це необхідність, яка виникає у багатьох практичних проблемах. Крім того, існує безліч інших корисних сценаріїв, коли потрібно, щоб комп'ютер щось розпізнав: об'єкт на зображенні, звук в аудіозаписі і так далі.

Є й інші причини, за якими розпізнавання образів має велике значення, в тому числі:

- виявляє і прогнозує навіть найдрібніші біти прихованих даних;
- допомагає класифікувати невидимі дані;
- робить цінні прогнози, використовуючи методи навчання;
- може розпізнавати та ідентифікувати об'єкт на різних відстанях;
- допомагає створювати прогнози по невидимим даними і вносити практичні пропозиції.

Ступінь вивченості. З приходом інформаційної ери, яка настала наприкінці 1970-х, з’явилися перші спроби створення алгоритмів штучного інтелекту. А значить і розпізнавання образів, яке є галуззю машинного навчання, можна

вважати відносно молодою технологією. Проте з ростом обчислювальних потужностей та еволюцією алгоритмів розпізнавання образів, зростає і ефективність цих алгоритмів, і сучасні можливості комп'ютерного зору вже не порівняти з тими, що були 20 років назад. На даний момент існує дуже багато бібліотек, інструментів та навчених моделей, які допоможуть вивченню цієї технології.

Об'єкт дослідження. Об'єктом досліджень є дії гравців спортивних команд. Детальніше в цій роботі буде розглянуто рухи гравців баскетбольних команд.

Предмет дослідження. Предметом дослідження є алгоритми розпізнавання образів, таких як людина(гравець) та баскетбольне поле.

Мета дослідження. Метою дослідження є створення системи, яка займатиметься розпізнаванням баскетбольних гравців та грального поля, і подальша класифікація їх рухів.

Структура. Дана робота розділена на 4 розділи. Перший розділ буде представляти з себе детальне ознайомлення з предмет та об'єктом дослідження, огляд їх особливостей, пошук та розгляд існуючих алгоритмів та підходів для досягнення мети дослідження. У другому розділі буде досліджено основні методи для вирішення задачі, описані математичні основи та принципи практичної реалізації даних методів, проведений їх аналіз, розглянуто критерії якості того чи іншого підходу та зроблено висновки. У третьому розділу детально розглядаються застосовані методи комп'ютерного зору та алгоритми розпізнавання грального поля, представлене розроблене ПЗ зі скріншотами роботи на різних вхідних даних. У четвертому розділі буде проведено функціонально-вартісний аналіз роботи.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність задачі, що розглядається

Для початку, розглянемо актуальність розпізнавання образів не тільки в галузі спорту, а й для інших областей. По мірі розвитку інтернету в 90-х роках, ставали доступними все більше і більше різноманітних зображень та відео, завдяки чому програми розпізнавання об’єктів процвітали. Ці зростаючі набори даних з відео та зображеннями допомагали машинам розпізнавати не тільки людські обличчя, а й будь-які інші об’єкти. Сьогодні цілий ряд факторів посприяли відродження комп'ютерного зору:

- Мобільні технології з вбудованими камерами наситили світ фотографіями та відео.
- Обчислювальна потужність стала більш дешевою та легкодоступною.
- Обладнання, створене для комп'ютерного зору та аналізу, стало набагато поширенішим.
- Нові алгоритми, такі як згорткові нейронні мережі, можуть скористатися апаратними та програмними можливостями.

Усі ці досягнення вражаючим чином вплинули на область комп'ютерного зору. Показники точності ідентифікації та класифікації об'єктів менше ніж за десятиліття зросли з 50 відсотків до 99 відсотків. І це ще не все - сучасні системи є більш точними, ніж люди, вони здатні швидко виявляти та реагувати на зміни візуальних даних. З огляду на досягнення у галузі штучного інтелекту та інновації в галузі глибокого навчання та нейронних мереж, за останні роки ця галузь мала можливість здійснити неймовірний стрибок, а також перевершити людей у виконанні певних завдань, пов’язаних з виявленням та маркуванням об’єктів. Одним із рушійних елементів розвитку комп'ютерного зору є міра інформації, яка створена сьогодні, а потім використовується для підготовки та вдосконалення комп'ютерного зору. Ця галузь намагається зрозуміти та організувати завдання шляхом обробки, аналізу та розуміння відео, а також цифрових зображень.

Сучасні системи штучного інтелекту можуть крокнути вперед і почати розуміти розпізнаний образ. Існує багато типів комп’ютерного зору, які використовуються по-різному:

- Сегментація зображень: розділяє зображення на кілька областей або фрагментів, які слід дослідити окремо.
- Виявлення об’єкта: визначає конкретний об’єкт на зображенні. Розширене розпізнавання об’єктів розпізнає багато об’єктів на одному зображенні: футбольне поле, гравець, що атакує, гравець оборони, м’яч тощо. Ці моделі використовують координати X, Y, щоб створити обмежену область та визначити все, що знаходиться всередині неї.
- Розпізнавання обличчя: вдосконалений тип виявлення об’єктів, який не тільки розпізнає обличчя людини на зображенні, але і визначає конкретну особу.
- Виявлення країв - це техніка, яка використовується для ідентифікації зовнішнього краю об’єкта чи пейзажу для кращого визначення того, що є на зображенні.
- Виявлення візерунків: процес розпізнавання повторюваних фігур, кольорів та інших візуальних показників на зображеннях.
- Класифікація зображень: групує зображення за різними категоріями.
- Відповідність об’єктів: тип виявлення шаблону, який відповідає схожості зображень, щоб допомогти їх класифікувати.

У спорті штучний інтелект був практично невідомий п'ять років тому, але сьогодні глибинне навчання та комп'ютерний зір проникають у низку програм спортивної індустрії. Незалежно від того, використовуються вони коментаторам для покращення глядацького досвіду того чи іншого виду спорту, або ж самими спортивним клубами, щоб стати більш конкурентоспроможними та досягти успіху. В теперішній реальності для спортивної індустрії набуло поширення впровадження цих сучасних методів.

Більшість основних видів спорту передбачають швидкі та точні рухи, які

іноді можуть стати складними завданнями для тренерів та аналітиків для їх детального відстеження та аналізу. Це особливо важко в тих ситуаціях, коли неможливо використовувати портативний пристрій або датчик для відстеження. На тренувальних заняттях та деяких матчах аналітик продуктивності може отримати лише обмежену кількість ракурсів відеозаписів. Ці кадри лише відображають рухи гравців або м’яча, розмір грального поля, тощо. Дані та уявлення, отримані на кадрах, вимагають щоб аналітик проводив численні години вручну, визначаючи події на полію. Це якраз той сценарій, де застосування методів комп’ютерного зору може пригодитися, пропонуючи нові способи збору даних та отримання цінного аналізу за допомогою автоматизованих систем, які визначають і сегментують кожного гравця, що цікавить та кожну його дію.

1.2 Аналіз існуючих підходів до вирішення задачі

Останні розробки в області нейронних мереж і глибинного навчання значно підвищили продуктивність сучасних систем візуального розпізнавання. Давайте розглянемо, п'ять основних методів комп'ютерного зору.

Класифікація зображень.

Аналіз зображення включає в себе різноманітні проблеми, включаючи зміну точки зору, зміну масштабу, деформацію зображення, оклюзію зображення, умови освітлення та шум у фоні. Дослідники комп’ютерного зору запропонували підхід, на основі даних, для класифікації зображень у різні категорії. Вони надають комп’ютеру кілька прикладів кожного класу зображень. Він дивиться на пікселі та вивчає візуальний вигляд кожного типу. Коротше кажучи, спочатку відбувається створення навчальної вибірки зображень певного типу, а потім комп'ютер займається обробкою цих даних. Згорткові нейронні мережі (CNN) - найвідоміша архітектура, яка використовується для класифікації зображень. Звичний випадок використання CNN - це коли хтось подає на вхід мережі зображення, а мережа класифікує й видає дані на вихід.

Враховуючи цей факт, повний конвеєр класифікації зображень можна формалізувати наступним чином:

- На вхід подається навчальний набір даних, який складається з N зображень, кожне з яких позначено одним із K різних класів.
- Потім ми використовуємо цей навчальний набір для підготовки класифікатора, щоб дізнатися, як виглядає кожен із класів.
- Врешті-решт, ми оцінюємо якість класифікатора, просячи його передбачити мітки для нового набору зображень, яких він раніше ніколи не бачив. Потім ми порівнюємо справжні мітки цих зображень із тими, які передбачив класифікатор.

Виявлення об’єктів. Якщо уявити ситуацію, коли необхідно одночасно виконувати класифікацію та знаходження розташування, то нам потрібне

виявленням об’єктів. У цьому випадку кількість об’єктів, які може містити зображення, невідома, якщо воно взагалі їх містить. Отже, метою виявлення об’єкта є пошук, а потім класифікація змінної кількості об’єктів на зображенні. Завдання ідентифікації об'єктів на зображеннях, як правило, передбачає виведення обмежувальних рамок і позначень для окремих елементів. Він відрізняється від класифікації об’єктів тим, що виділяю в групу багато об’єктів замість одного домінуючого. Проблема була б складною навіть для людини. Деякі об’єкти видно лише частково, деякі перекриваються іншими, а деякі частково знаходяться з кадром. Крім того, розміри подібних предметів сильно варіюються. Основна мета виявлення об’єктів - це підрахунок. Застосування в реальному житті досить різноманітне, від підрахунку різних видів зібраних фруктів до підрахунку людей на таких заходах, як публічні демонстрації чи футбольні матчі.

Відстеження об’єктів. Назва “Відстеження об’єктів” говорить сама за себе – метод займається відстеженням певного об’єкта, що цікавить, або кількох його елементів. Зазвичай відстеження відбувається після первинного виявлення об'єкта за допомогою методу описаного вище. За моделлю спостереження його можна розділити на дві категорії. Однією з них є генеративний метод, який використовує генеративну модель для опису очевидних характеристик і мінімізує помилку реконструкції для пошуку об’єкта, такого як PCA(principal component analysis) – метод головних компонент. Цей алгоритм дозволяє трекеру витягувати із зображень об'єкти різного розміру та форми. Дискримінаційний метод може бути використаний для відокремлення об'єкта від фону. Він є більш надійним в плані продуктивності, і тому стає основним методом відстеження.

Семантична сегментація. Комп’ютерний зір - це процес сегментації, який виділяє цілі зображення на піксельні групи, які можна маркувати та класифікувати. Семантична сегментація намагається зрозуміти роль кожного пікселя в певний момент. Наприклад, якщо ми вибираємо пейзаж, де ми можемо бачити людей, дороги, машини та дерева, нам доведеться окреслити межі кожного об’єкта. Таким чином, на відміну від класифікації, нам потрібні щільні піксельні

передбачення з моделей.

Сегментація екземпляра. Цей метод включає різні моделі класів, такі як маркування п’яти автомобілів п’ятьма різними кольорами. У класифікації зазвичай у фокусі знаходиться зображення з якимось одним об'єктом, і завдання полягає в тому, щоб визначити, що це за об’єкт. Але для сегментації екземплярів нам потрібно виконувати набагато складніші завдання. Це не тільки класифікація об’єктів, що перекриваються, та мають різний фоном, а й визначаємо їх межі, відмінності та відношення один до одного!

У спортивних змаганнях можна застосувати всі ці методи комп’ютерного зору. Взяти хоча б такі ігри, як теніс, бадмінтон чи крикет чи баскетбол – тут допомагає відстеження об’єктів: для ідентифікації м’ячів проглядається кожне зображення та виконується пошук всіх можливих об'єктів, що нагадують характеристики кулі. Ці знайдені м’ячі можна класифікувати з допомогою методу класифікації зображень: одні м’ячі будуть віднесені до категорії баскетбольних, інші – до тенісних, в залежності від різних особливостей(еліптичності, розміру, кольору і так далі). Після виявлення цих м’ячів з використанням відстеження об’єктів можна побудувати тривимірну траєкторію руху, пов’язуючи кілька кадрів, де м’яч був виявлений і враховуючи різні кути камери. Потім результати цієї системи можуть бути використані для миттєвого визначення того, потрапив м’яч у чи поза межі. А межі ігрового поля можна визначити застосувавши метод Семантичної сегментації, вдосконаливши її таким чином, щоб розрізняти границі поля, лінії штрафних зон або 3-очкові лінії Система забезпечує подальший аналіз, такий як передбачення шляху, по якому пройшов би тенісний м’яч, якби гравець не відбив його або навіть ймовірність попадання у кільце баскетбольного м’ячу.

1.3 Особливості предметної області

Застосування комп’ютерного зору у такій діяльності як спорт має певні особливості. В таких спортивних іграх як баскетбол, футбол або теніс існує

декілька константних умов, для прикладу, гральне поле, лінії границь грального поля або штрафних зон і звісно ж сам гральний м’яч. Для кожного типу гри можна виділити окремий тип м’яча зі своїми характеристиками. Так, у футболі м’яч має форму кулі та здебільшого білий колір. Також такий тип характеризується конкретним діаметром. У тенісі ж м’яч навпроти має набагато менший діаметр та зазвичай жовтий колір. Усі ці відмінності допомагають комп’ютерній програмі визначати, що саме за гра на відеофрагменті або ж на зображенні. Гральне поле в різних областях спорту різниться так само як і м’ячики. Для баскетбольного поля характерна присутність 3-очкової лінії та лінії кіл біля сіток обох команд. У крикеті поле має незвичну для інших видів спорту круглу форму, що допомагає однозначно визначити, що це за вид.

Крім того, кожний спорт має визначену кількість гравців, які, на відміну від звичайних людей рухаються достатньо швидко, що може перешкодити їх виявленню. Характерно також те, що в більшості видів спорту моніторинг за допомогою датчиків або інших пристроїв, прикріплених до гравців, як правило, неможливий.

Унікальність цієї задачі також створює деякі унікальні проблеми и питання. Нижче приведені деякі з них:

Якщо частота кадрів камер низька, то наскільки складно фіксувати швидкі дії з низькою частотою кадрів? Якість зображень може бути низькою, що ускладнює ефективне вирішення завдань комп'ютерного зору. Оскільки гравці у грі достатньо рухливі, то, можливо, що частини одного і того ж руху можна побачити в різних камерах. Задача полягає в тому, щоб об’єднати відповідні кадри з різних камер для ідентифікації цієї дії.

1.4 Постановка задачі дослідження

Першим етапом буде дослідження та аналіз існуючих способів для застосування в галузі комп'ютерного зору. Ці підходи будуть застосовані для розпізнавання та відстеження ігрових поверхонь та людей, а саме – гравців.
Другим етапом вирішення задачі можна звести до списку з дрібніших завдань, які необхідно виконати по черзі:

- Виконати виявлення грального поля, проаналізувавши його границі та всі важливі лінії.
- Здійснити виявлення всіх гравців.
- Виконати відстеження прямування виявлених гравців.
- Здійснити трансформування виявлених поля та гравців в 2-D площину.

1.5 Висновки до розділу 1

У даному розділі було розглянуто актуальність такої задачі як виявлення та розпізнавання рухів гравців. Також проаналізовано різні методи до вирішення цієї задачі.
Розглянуто 5 основних методів для розпізнавання не тільки рухів гравців, а й грального поля.
Розглянуті особливості предметної області допомогли краще зрозуміти, які саме підходи краще застосувати, щоб досягнути найкращого результату.

У результаті сформовано постановку задачі та розбито її на менші підпункти.

РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ

2.1 Дослідження існуючих методів для вирішення задачі

Як уже було описано у попередньому розділі, для вирішення цієї задачі потрібна комбінація декількох технік комп’ютерного зору. Відповідно, для кожної з них існують певні методи. Розглянемо деякі з них, що б допомогли виявити гральне поле.

Одним з рішень є Mask R-CNN. Mask R-CNN розвиває архітектуру Faster R-CNN шляхом додавання ще однієї гілки, яка передбачає положення маски, що покриває знайдений об'єкт, і, таким чином вирішує вже завдання instance segmentation. Маска являє собою просто прямокутну матрицю, в якій 1 на деякій позиції означає приналежність відповідного пікселя об'єкту заданого класу, 0 - що піксель об'єкту не належить.

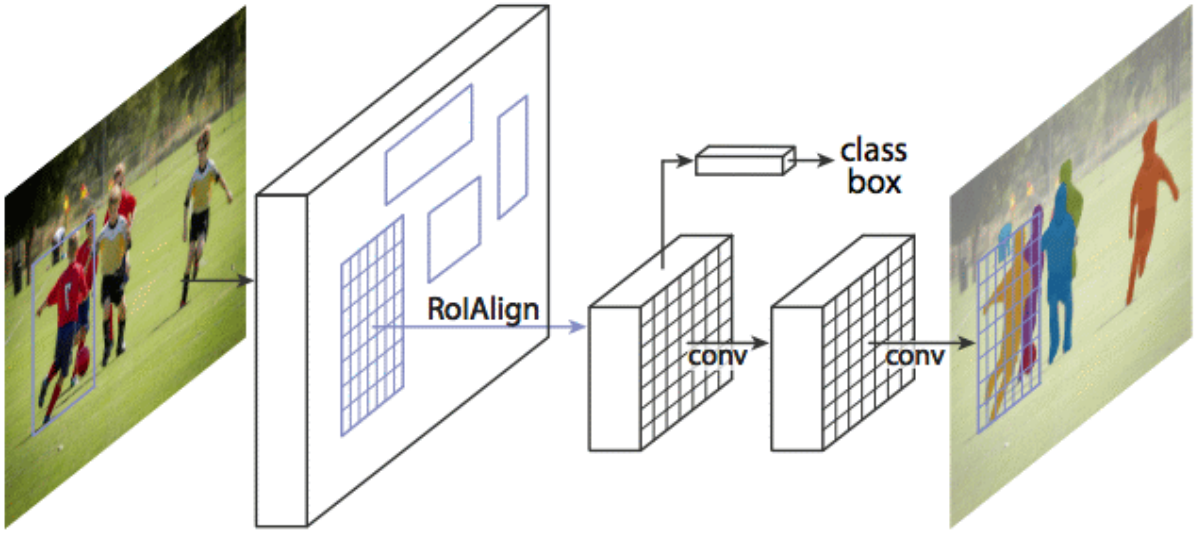


Рисунок 2.1 – Структура Mask R-CNN для сегментації.

Візуалізація різнокольорових масок на вихідних зображеннях може дати кольорові картинки:

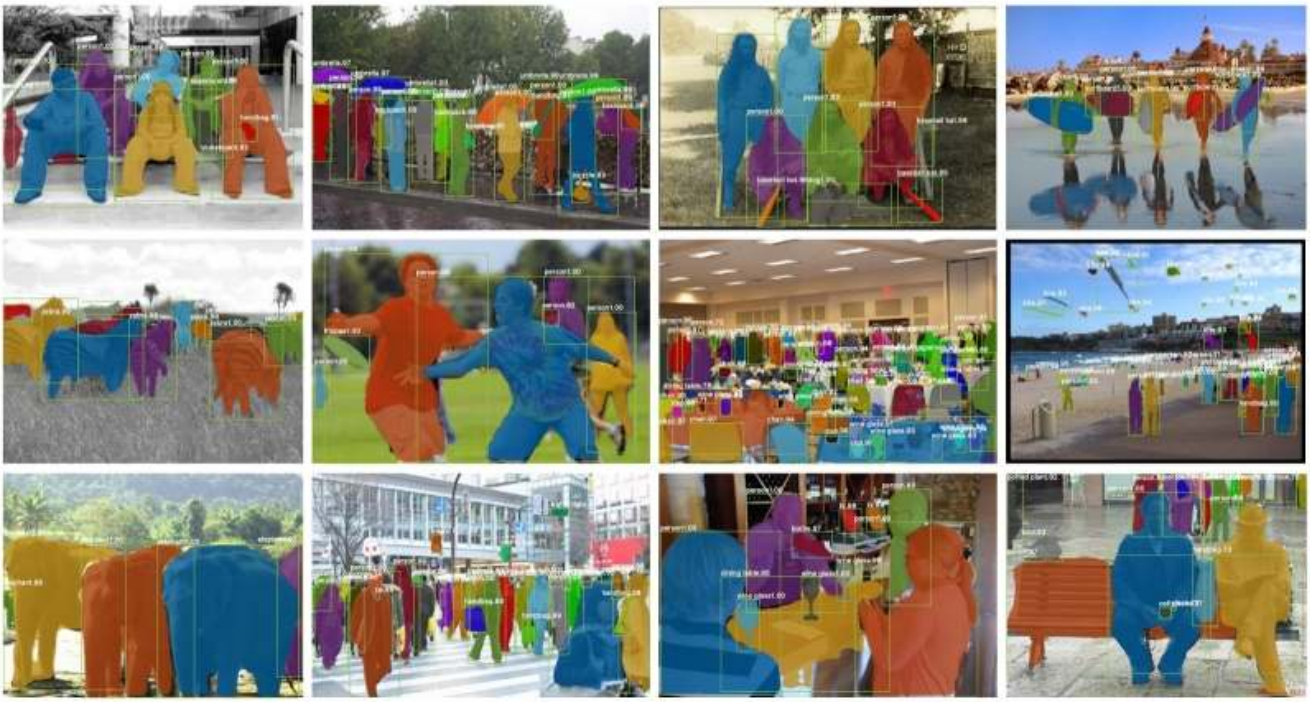


Рисунок 2.2 – Результати Mask R-CNN на наборі COCO.

Існує два етапи маски RCNN. По-перше, він формує пропозиції щодо регіонів, де може бути об’єкт, на основі вхідного зображення. По-друге, він передбачає клас об'єкта, уточнює обмежувальну рамку та генерує маску на рівні пікселів об'єкта на основі пропозиції першого етапу. Обидва етапи пов'язані з структурою backbone (магістралі)

Що таке backbone? Backbone або Магістраль - це глибока нейронна мережа у стилі FPN. Він складається із шляху знизу вгору, шляху зверху знизу та бічних з'єднань. Шляхом знизу вгору може бути будь-який ConvNet, зазвичай ResNet або VGG, який витягує функції з необроблених зображень. Шлях зверху знизу генерує карту пірамід, яка за розміром схожа на шлях знизу вгору. Бічні зв’язки - це згортання та додавання операцій між двома відповідними рівнями двох шляхів. FPN перевершує інші одиночні ConvNets головним чином з тієї причини, що він підтримує сильні семантичні функції в різних масштабах роздільної здатності.

Повністю згорнута мережа (FCN) - це нейронна мережа, яка виконує лише операції згортки (і субодискретизації або передискретизації). Еквівалентно, FCN -

це CNN без повністю зв’язаних шарів. Типова нейронна мережа згортки (CNN) не є повністю згортковою, оскільки вона часто також містить повністю пов'язані шари (які не виконують операцію згортки), багаті на параметри, тобто вони мають багато параметрів (порівняно з еквівалентними ним згортквими шарами). Проте повністю пов'язані шари також можна розглядати як згортки з ядрами, які охоплюють цілі вхідні області, що є основною ідеєю перетворення CNN у FCN. Прикладом повністю згорнутої мережі є U-net (названа таким чином через її U-форму, що ви можете бачити з ілюстрації нижче), яка є відомою мережею, яка використовується для семантичної сегментації, тобто класифікує пікселі зображення таким чином, щоб пікселі, що належать до одного класу (наприклад, людина), були пов’язані з однією і тією ж міткою (тобто особою).

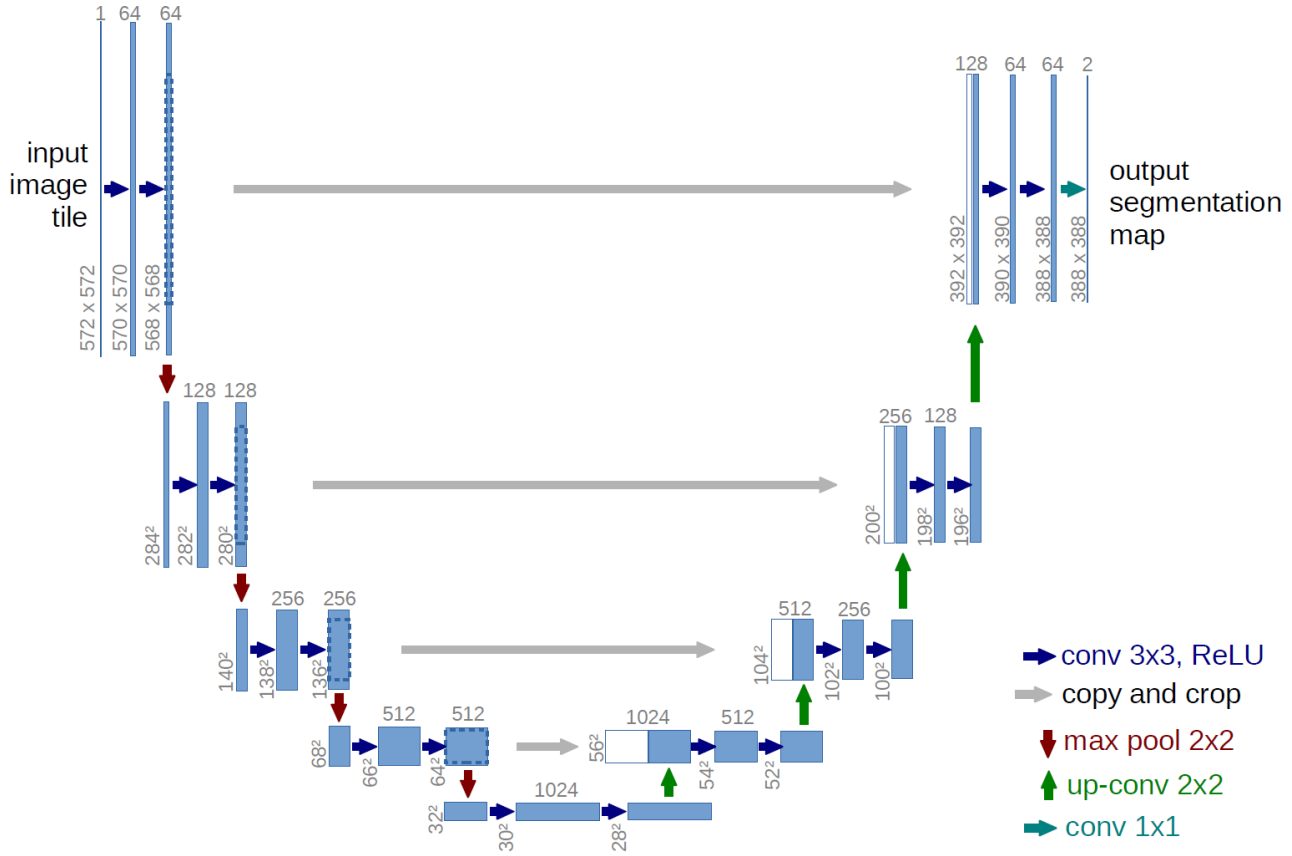


Рисунок 2.3 – Архітектура U-net.

Отже, у семантичній сегментації ви хочете пов’язати мітку з кожним пікселем (або невеликим фрагментом пікселів) вхідного зображення. Ось більш наглядна ілюстрація нейронної мережі, яка виконує семантичну сегментацію.

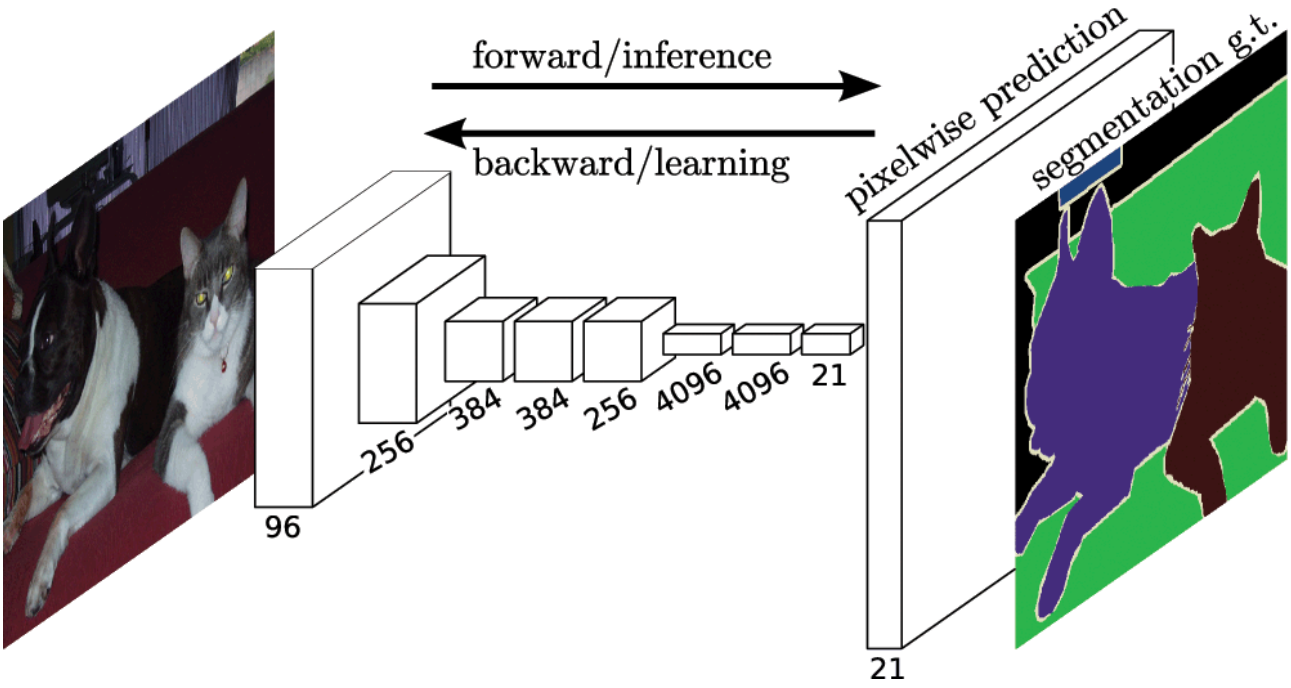


Рисунок 2.4 – Архітектура FCN.

Модель COCO Panoptic Segmentation виявляє стелю, стіни та підлогу та відповідно їх забарвлює. Це набагато спрощує роботу для знаходження поля, адже тепер ми можемо обмежити "пошук" у багатокутнику підлоги. Метою паноптичної сегментації є виконання єдиного завдання сегментації. Для цього давайте спочатку зрозуміємо кілька основних понять.

Річ - це обчислюваний об’єкт, такий як люди, машина тощо, отже, це категорія, що має примітки на рівні екземпляра. Матеріал є аморфною областю подібної фактури, такою як дорога, небо тощо, отже, це категорія без приміток на рівні екземпляра. Річ, що вивчається, підпадає під виявлення об’єктів та сегментацію екземплярів, а під час вивчення речей - семантичну.

Кодування міток пікселів при паноптичній сегментації передбачає присвоєння кожному пікселю зображення двох міток - одну для семантичної мітки, а іншу, наприклад, ідентифікатор. Пікселі, що мають однакову мітку, вважаються належними до одного класу, а ідентифікатор екземпляра для речей ігнорується. На відміну від сегментації екземплярів, кожен піксель у паноптичній сегментації має лише одну мітку, що відповідає екземпляру, тобто відсутні перекриваються екземпляри.

У COCO паноптичні анотації зберігаються таким чином:

Кожна структура анотацій є анотацією для кожного зображення, а не анотацією для кожного об'єкта. Кожна анотація для кожного зображення складається з двох частин: PNG, що зберігає сегментацію зображень, що агностикує клас, та структура JSON, яка зберігає семантичну інформацію для кожного сегмента зображення.

Рисунок 2.5 – Модель COCO Panoptic Segmentation.

Ще одним способом є послідовне виконання наступних дій:

- Перетворення зображення у HSV.
- Ізолювання пікселів у межах заданого діапазону відтінків.
- Розробка побітової AND маски.
- Використання розпізнавання країв Canny.
- Використання трансформації Хафа .

Результатом має бути наступне зображення:

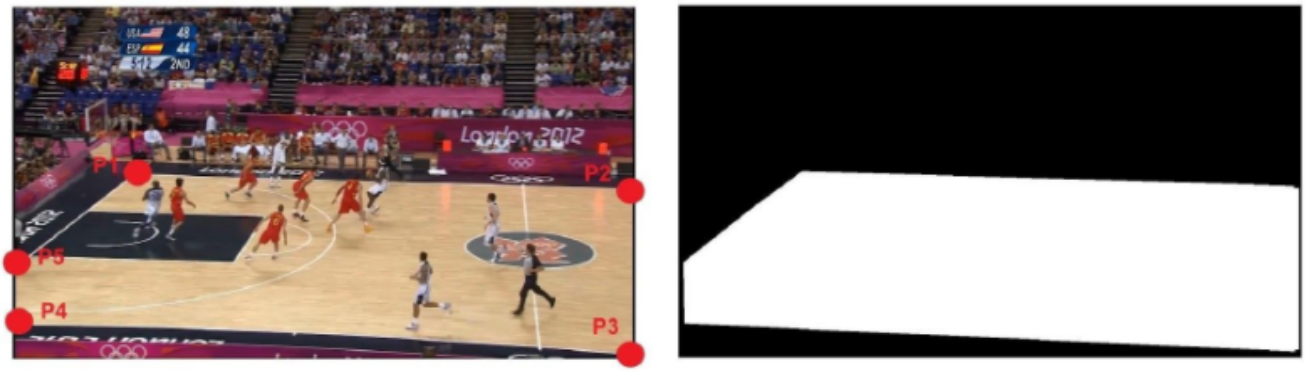


Рисунок 2.6 – Очікуваний результат розпізнавання поля.

Для розпізнавання гравців наявні наступні техніки:

Mask R-CNN. Як і для розпізнавання ігрового поля, для визначення людини на картинці, можна застосувати модель сегментації Mask R-CNN.

Рисунок 2.7 – Розпізнавання людини з використанням Mask R-CNN.

YOLO (you only look once) - є ефективним алгоритмом розпізнавання об’єктів у режимі реального часу, вперше описаним в основній роботі Джозефа Редмона та ін. У вільному доступі є мінімальна реалізація PyTorch YOLOv3 з підтримкою тренування, виведення результатів та оцінок. Ось, приклад роботи на одній із картинок.

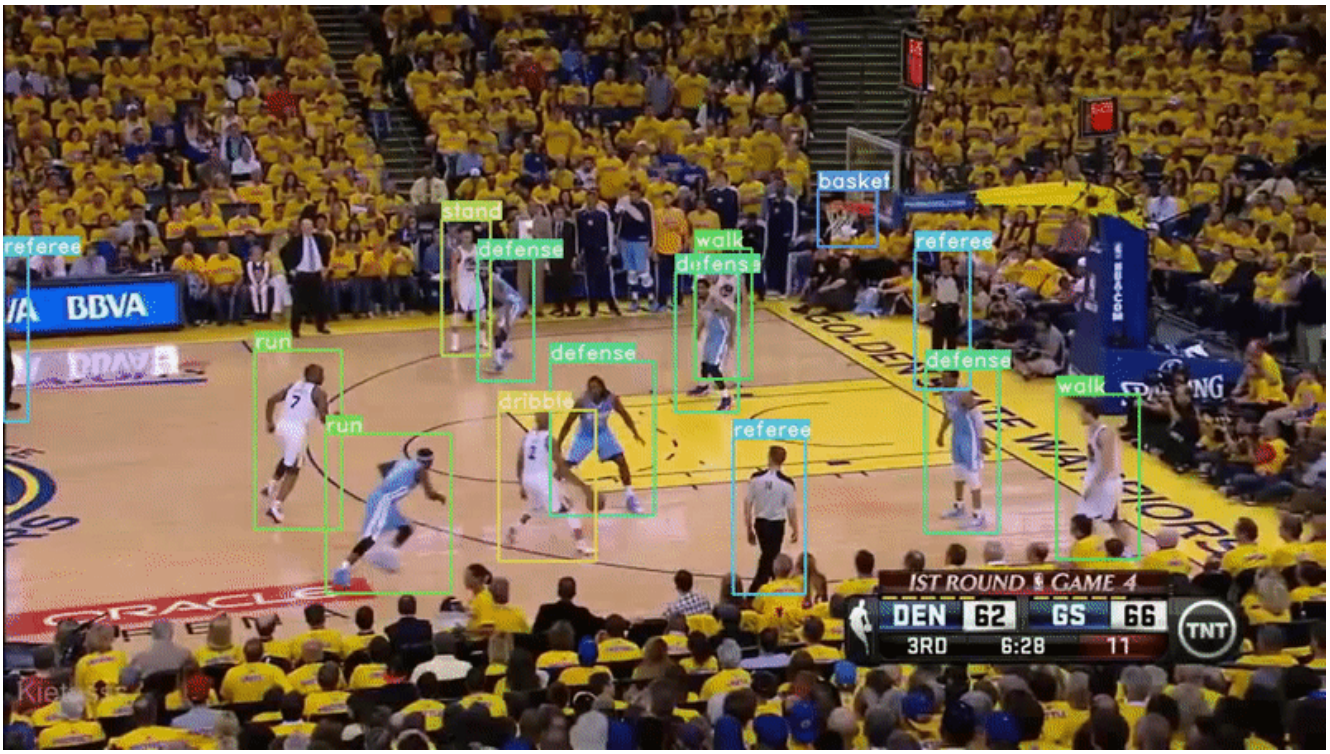


Рисунок 2.8 – Розпізнавання гравців за допомогою YOLO.

Виявлення людини за допомогою API виявлення об’єктів Tensorflow. TensorFlow ™ - це API з відкритим кодом від Google, який широко використовується для вирішення завдань машинного навчання, що включають глибокі нейронні мережі. API виявлення об’єктів Tensorflow - це бібліотека з відкритим кодом, створена на основі Tensorflow для підтримки навчання та оцінки моделей виявлення об’єктів. Сьогодні ми подивимось на «Зоопарк з моделями виявлення Tensorflow», який являє собою набір попередньо навчених моделей, сумісних з API виявлення об’єктів Tensorflow.

На момент написання цього матеріалу, зоопарк Tensorflow Detection Model Zoo складається з 16 моделей виявлення об'єктів, попередньо навчених на наборі даних COCO. Перші 12 із цього списку моделей містять „коробки” як вихідні дані, і вони сумісні з кодом, пов’язаним із цією статтею. Ці моделі здатні виявити 80 типів об'єктів, включаючи людей.

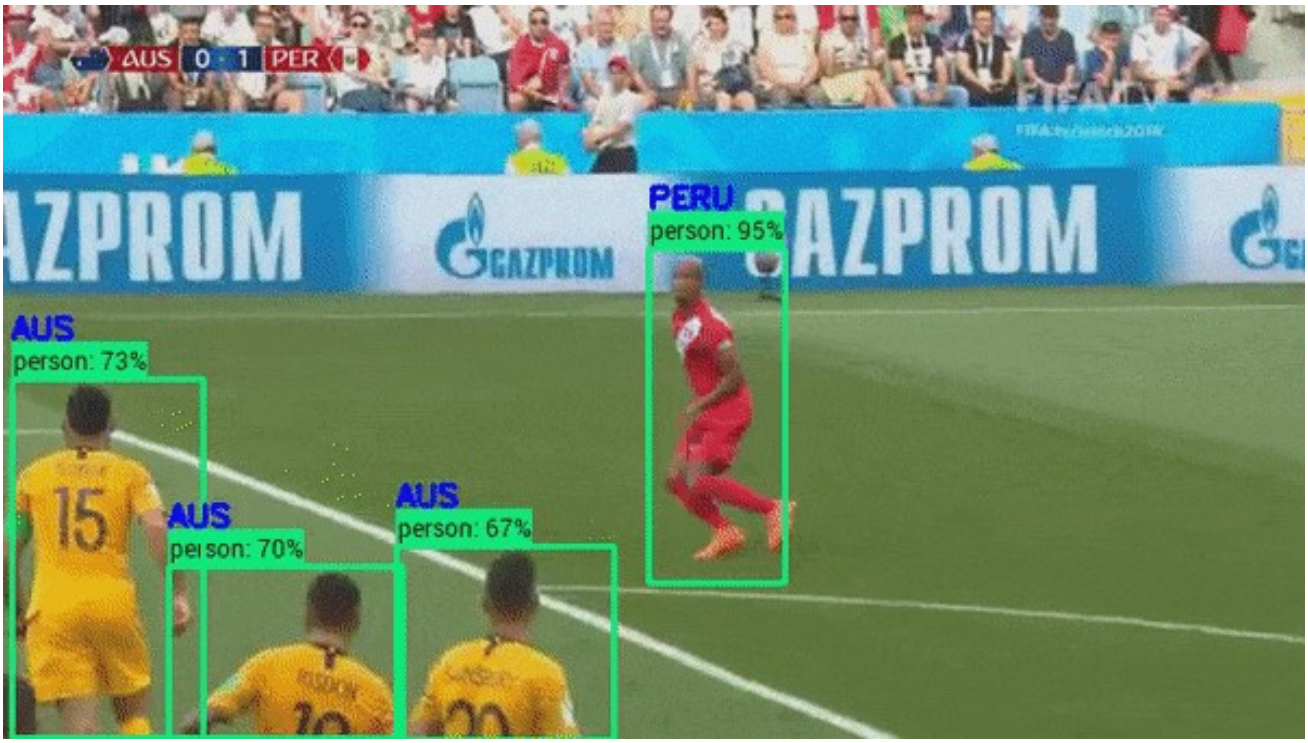


Рисунок 2.9 – Виявлення футболістів за допомогою Tensorflow.

Відстеження рухів гравців. Відстеження об’єктів відноситься до процесу відстеження конкретного об’єкта, що цікавить, або кількох об’єктів у певній сцені. На сьогодні ця техніка використовується у системах автономного водіння, таких як самокеровані транспортні засоби компаній, як Uber та Tesla. Відстеження та ідентифікація гравців у трансляційних видах спорту відео - важке завдання. Відстеження декількох гравців у спортивному відео викликає труднощі з кількох причин:

- Зовнішній вигляд гравців неоднозначний - гравці однієї команди носять форму однакових кольорів;
- Оклюзії серед гравців є частими і іноді довго;
- Гравці мають більш складний шаблон руху, і вони не мають фіксованих місць входу / виходу, як наприклад, пішоходи у відео з камер спостереження.

Відстеження швидше, ніж виявлення: Зазвичай алгоритми відстеження швидші, ніж алгоритми виявлення. Причина проста.

Коли ви відстежуєте об’єкт, виявлений у попередньому кадрі, ви багато

знаєте про зовнішній вигляд об’єкта. Ви також знаєте розташування в попередньому кадрі та напрямок та швидкість його руху. Отже, у наступному кадрі ви можете використовувати всю цю інформацію, щоб передбачити розташування об’єкта в наступному кадрі та зробити невеликий пошук навколо очікуваного розташування об’єкта, щоб точно знайти об’єкт. Хороший алгоритм відстеження використовуватиме всю інформацію про об'єкт до цього моменту, тоді як алгоритм виявлення завжди починається з нуля.

Отже, під час проектування ефективної системи, як правило, виявлення об’єкта виконується на кожному n-му кадрі, тоді як алгоритм відстеження використовується в n-1 кадрах між ними.

Чому б нам просто не виявити об'єкт у першому кадрі та не відстежити його згодом? Це правда, що відстеження виграє через присутність додаткової інформації, яка у нього є, але також можна втратити слід об’єктів, коли вони довгий час перебувають за перешкодою, поза кадром або якщо вони рухаються настільки швидко, що алгоритм відстеження не може наздогнати. Також типовим є те, що алгоритми відстеження накопичують помилки, і обмежувальне поле, що відстежує об’єкт, повільно віддаляється від об’єкта, який він відстежує. Щоб усунути ці проблеми з алгоритмами відстеження, алгоритм виявлення запускається так часто. Алгоритми виявлення тренуються на великій кількості прикладів об’єкта.

Отже, вони мають більше знань про загальний клас об’єкта. З іншого боку, алгоритми відстеження знають більше про конкретний екземпляр класу, який вони відстежують. Відстеження може допомогти, коли виявлення не вдається: якщо ви запускаєте детектор обличчя на відео, а обличчя людини закривається предметом, детектор обличчя, швидше за все, вийде з ладу.

Хороший алгоритм відстеження, навпаки, впорається з певним рівнем оклюзії.

Алгоритми відстеження об’єктів за допомогою OpenCV:

- BOOSTING Tracker.

- MIL Tracker.
- KCF Tracker.
- TLD Tracker.
- MEDIANFLOW Tracker.
- GOTURN tracker.
- MOSSE tracker.
- CSRT tracker.
- Deep SORT.

2.2 Критерії якості рішення задачі

Для кожного з етапів будуть свої плюси та мінуси. Для розпізнавання грального поля краще підійде пошук ігрового поля по пікселям, потім відокремлення знайдених пікселів від інших з використанням розпізнавання країв Canny. В результаті можна отримати масив ліній, що визначають границі поля. Знаючи границі можна замапити зображення на 2-D макет площини поля. Недоліками даної техніки є те, що потрібно попередньо знати колір грального поля. Інші ж техніки гірше будуть шукати поле, так як це фактично є фоном. Такі моделі як Mask R-CNN та FCN краще підходять для виявлення окремих об'єктів такі як гравці або м'яч.

Покращити розпізнавання поля допоможе модель COCO Panoptic Segmentation, що спочатку виокремить підлогу в кадрі, а потім цей сегмент зображення можна використати для подальшого аналізу.

Для виявлення гравців можна порівняти алгоритми відстеження об’єктів за допомогою OpenCV:

BOOSTING Tracker. Цей трекер заснований на онлайн-версії AdaBoost - алгоритму, який використовує каскадний детектор обличчя HAAR всередині себе. Цей класифікатор потрібно навчити під час роботи з позитивними та негативними прикладами об'єкта, тобто з прикладами, де об'єкт присутній і де відсутній відповідно. Початкова обмежувальна коробка, що надана користувачем (або

іншим алгоритмом виявлення об’єкта), приймається як позитивний приклад для об’єкта, і багато кадрів зображень поза обмежувальним полем розглядаються як фон.

Враховуючи новий кадр, класифікатор запускається на кожному пікселі в районі попереднього місця розташування і реєструється оцінка класифікатора. Нове місце розташування об’єкта те, де оцінка максимальна. Отже, маємо ще один позитивний приклад для класифікатора. З появою більшої кількості кадрів класифікатор оновлюється з використанням цих додаткових даних.

Плюси: Жодного. Цьому алгоритму вже десять років і він працює нормально, проте вагомих причин використовувати його немає, особливо коли доступні інші вдосконалені трекери (MIL, KCF), засновані на подібних принципах.

Мінуси: ефективність відстеження посередня. Алгоритм точно не знає, коли відстеження не вдалося.

MIL Tracker. Цей трекер за своїм задумом схожий на трекер BOOSTING, описаний вище. Велика різниця полягає в тому, що замість того, щоб розглядати лише поточне розташування об’єкта як позитивний приклад, він проглядає невеликий район навколо поточного місцезнаходження, щоб згенерувати кілька потенційних позитивних прикладів. Якщо задуматись, то це може бути поганою ідеєю, оскільки в більшості цих «позитивних» прикладів об’єкт не відцентрований.

Тут на допомогу приходить багаторазове навчання (MIL). У MIL ви вказуєте не позитивні і негативні приклади, а позитивні та негативні «мішки». Колекція зображень у позитивному мішку не обов'язково є позитивними прикладами. Натомість, лише одне зображення в позитивній сумці має бути позитивним прикладом!

Навіть якщо поточне місце розташування об'єкта, що відстежується, не є точним, коли зразки з сусіднього місця поточного розташування поміщаються в позитивну сумку, існує велика ймовірність того, що ця сумка містить принаймні

одне зображення, на якому об'єкт добре відцентрований. Сторінка проекту MIL містить більше інформації для людей, які хотіли б глибше дізнатися про внутрішню роботу інструмента MIL.

Плюси: Продуктивність досить гарна. Він не дрейфує так сильно, як трекер BOOSTING, і розумно реагує при частковій оклюзії. При використанні OpenCV 3.0, це може бути найкращий доступний для трекер. Але якщо ви версія вище, то краще KCF.

Мінуси: Помилка відстеження не завжди правильна і не відновлюється після повної оклюзії.

TLD Tracker. TLD розшифровується як відстеження, навчання та виявлення. Як випливає з назви, цей трекер розкладає довгострокове завдання відстеження на три компоненти - (короткострокове) відстеження, навчання та виявлення. З авторської статті «Трекер стежить за об’єктом від кадру до кадру. Детектор локалізує всі види, які спостерігались дотепер, і коригує трекер, якщо це необхідно.

Навчання оцінює помилки детектора та оновлює його, щоб уникнути цих помилок у майбутньому». Цей результат трекера має тенденцію трохи стрибати. Наприклад, якщо ви стежите за пішоходом, а на місці події є інші пішоходи, цей індикатор може іноді тимчасово відстежувати іншого пішохода, ніж той, кого ви планували відслідковувати. Плюсом є те, що ця доріжка відстежує об’єкт у більшому масштабі, русі та оклюзії. Якщо у вас є відео, де об’єкт захований за іншим об’єктом, цей трекер може бути хорошим вибором.

Плюси: найкраще працює при оклюзії на кількох кадрах. Також добре відстежує при змінах масштабу.

Мінуси: багато помилкових спрацьовувань, що робить його майже непридатним для використання.

Трекер MEDIANFLOW. Під капотом цей трекер відстежує об'єкт у прямому і зворотному напрямках у часі і вимірює розбіжності між цими двома траєкторіями. Мінімізація помилки ForwardBackward дозволяє їм надійно

виявляти помилки відстеження та вибирати надійні траєкторії у відео послідовностях.

У своїх тестах я виявив, що цей трекер працює найкраще, коли рух передбачуваний і невеликий. На відміну від інших трекерів, які продовжують працювати навіть тоді, коли відстеження явно не вдалося, цей трекер знає, коли відстеження не вдалося.

Плюси: Чудово звітує про помилки відстеження. Дуже добре працює, коли рух передбачуваний і немає оклюзії.

Мінуси: Зазнає помилок при швидкому русі.

Трекер GOTURN. З усіх алгоритмів відстеження в класі трекерів, це єдиний, заснований на Згортковій нейронній мережі (CNN). З документації OpenCV можна дізнатися, що вона “надійна для змін точки зору, змін освітлення та деформацій”. Але це не дуже добре справляється з оклюзіями.

Трекер MOSSE. Мінімальна вихідна сума квадратної помилки (MOSSE(Minimum Output Sum of Squared Error)) використовує адаптивну кореляцію для відстеження об’єктів, яка створює стабільні фільтри кореляції при ініціалізації за допомогою одного кадру. Трекер MOSSE надійний до змін освітлення, масштабу, пози та нежорстких деформацій. Він також виявляє оклюзію, роблячи паузу та продовжувати там, де він зупинився, коли об'єкт знову з'явиться. Трекер MOSSE також працює з більшою швидкістю кадрів в секунду (450 кадрів в секунду і навіть більше). З плюсів, його також дуже легко реалізувати, він такий же точний, як і інші складні трекери, і набагато швидший. Але за рівнем продуктивності він відстає від трекерів на основі глибокого навчання.

Deep SORT. Найпопулярнішою та однією з найбільш широко використовуваних, елегантних систем відстеження об'єктів є Deep SORT, розширена версія SORT (простий відстежувач реального часу). Технологія DeepSORT просто поєднує два поняття – фільтр Калмана і відстань Махалонобіса між собою для того, щоб передати інформацію від одного кадра до іншого, і додає нову метрику під назвою appearance. Ця метрика і відрізняє його від звичайного

SORT. Цей “зовнішній вигляд” був натренований окремою нейронною мережею, яка була створена авторами Deep SORT.

Плюси: має відкритий код джерела, постійно підтримуєтья та має гарну документацію. Тестувався з детектором YOLOv4 та показав доволі непогані результати.

Мінуси: якщо обмежувальні рамки занадто великі, то занадто велика кількість фонового зображення “захоплюється” у функціях, що знижує ефективність алгоритму. Якщо люди одягнені так само, як це відбувається у спорті, це може призвести до того, що їх ідентифікатори можуть змінитися.

2.3 Висновок до розділу 2

У результаті, було детально розглянуто доступні способи розпізнавання грального поля такі як Mask R-CNN, FCN та визначення та виділення пікселів, які мають колір грального поля. Для визначення положення гравців та відстеження їх руху розглянуті моделі YOLO та TensorFlow для виявлення гравців, а також декілька методів бібліотеки OpenCV. Вищезазначені алгоритми до вирішення було порівняно та визначено основні плюси та мінуси. В результаті, для визначення ігрового поля у нашому випадку краще підійде виявлення та відокремлення пікселів майданчика, а для аналізу рухів гравців детектор YOLOv4 та трекер DeepSORT.

РОЗДІЛ 3 ОПИС ПРОГРАМНОГО ПРОДУКТУ

3.1 Обґрунтування вибору платформи та мови програмування

Існує кілька варіантів мов програмування для використання в цілях комп'ютерного зору – OpenCV за допомогою C ++, OpenCV за допомогою Python або MATLAB. Однак у більшості інженерів є особистий фаворит, залежно від завдання, яке вони виконують. Новачки часто вибирають OpenCV з Python для його гнучкості. Це мова, знайома більшості програмістів, і завдяки своїй універсальності дуже популярна серед розробників.

Фахівці з комп'ютерного зору рекомендують мову програмування Python з наступних причин:

- Простота у використанні: Python одна з найлегших у вивченні мов програмування, особливо для початківців. Це одна з перших мов програмування, яку вивчає більшість користувачів. Крім того, я маю особистий досвід роботи з цією мовою, так як я мав предмет з вивчення Python в університеті. Ця мова також легко адаптується до будь-яких потреб програмування.
- Найбільш вживана обчислювальна мова: Python пропонує широке навчальне середовище для людей, які хочуть використовувати його для різних видів викликів Computer Vision та Machine Learning. Такі бібліотеки numpy, scikit-learn, matplotlib та OpenCV, які він пропонує надають вичерпний ресурс для будь-яких програм комп'ютерного зору.
- Налагодження та візуалізація: Python має вбудований налагоджувач „PDB”, який робить коди налагодження на цій мові програмування більш доступними. Подібним чином Matplotlib – це зручна бібліотека на мові програмування Python для візуалізації даних двовимірною 2D графікою (3D графіка також підтримується).
- Розробка бекендів для веб: такі фреймворки, як Django, Flask та Web2py є чудовими інструментами для побудови веб-сторінок. Python сумісний з

цими фреймворками, і його можна легко налаштувати відповідно до бажаних вимог.

MATLAB - це інша мова програмування, популярна серед комп'ютерних експертів. Давайте розглянемо переваги використання MATLAB:

- Набори інструментів: MATLAB має один із найповніших наборів інструментів; незалежно від того, чи це набір інструментів статистичного та машинного навчання, чи набір інструментів для обробки зображень, MATLAB має будь-який набір для всіх потреб. Зрозумілі інтерфейси кожного з цих наборів інструментів дозволяють реалізувати цілий ряд алгоритмів. MATLAB також має набір інструментів для оптимізації, який забезпечує найкращу роботу всіх алгоритмів.
- Потужна бібліотека матриць: Зображення та інший візуальний вміст містить багатовимірні матриці разом з лінійною алгеброю в різних алгоритмах, а це полегшить роботу з ними в MATLAB. Процедури лінійної алгебри, включені до MATLAB, працюють швидко та ефективно.
- Налагодження та візуалізація: Оскільки в MATLAB існує єдина інтегрована платформа для кодування, написання, це набагато спрощує візуалізація та налагодження програмного продукту.
- Відмінна документація: MATLAB дозволяє адекватно задокументувати свою роботу, щоб вона була доступна пізніше. Документація необхідна не тільки для подальшого використання, а й для того, щоб допомогти інженерам програмного забезпечення працювати швидше. Документація MATLAB дозволяє користувачам працювати вдвічі швидше, ніж з використанням OpenCV.

Експерти з комп’ютерного зору також тяжіють до OpenCV з наступних причин:

- Бібліотека з відкритим кодом: OpenCV є безкоштовною бібліотекою з відкритим кодом, на відміну від MATLAB, де комерційної версії без інструментів близько 2000 дол. США(100 дол. США – ціна для навчальних

закладів з мінімальним набором інструментів), а це якраз те, що найкраще підходить у моєму випадку. Його можна використовувати для комерційних додатків, та навіть за необхідності внести зміни в існуючий код, а також поширити виправлення з іншими розробниками. Найважливішою перевагою використання OpenCV є те, що вам не потрібно робити свій проект відкритим.

- Платформа та пристрої: Деякі вбудовані програми для зору та мобільні додатки віддають перевагу OpenCV, як основній бібліотеці комп'ютерного бачення, орієнтованої на продуктивність. Завдяки великій гнучкості її можна використовувати на всіх платформах та пристроях.
- Велика спільнота: OpenCV використовують понад 9 мільйонів людей, які постійно оновлюють її та допомагають одне одному через блоги та форуми. Важливою перевагою використання OpenCV є те, що на всі проблеми з використанням можна знайти підтримку спільноти. Оскільки такі компанії, як Google, Intel і AMD фінансують OpenCV розвиток, вона швидко розвивається.

Існує два типи моделей виявлення об'єктів, одноступеневі та двоступеневі моделі. Одноступенева модель здатна виявляти об'єкти без необхідності попереднього кроку. Навпаки, двоступеневий детектор використовує попередній етап, під час якого виконується семантична сегментація важливих регіонів на всьому зображенні, а потім ці регіони класифікуються, для того щоб зрозуміти, чи не було виявлено об'єкт у цих областях. Перевагою одноступеневого детектора є швидкість прогнозування, що дозволяє використовувати його в реальному часі.

З одного боку, у нас є двоступеневі детектори, такі як Faster R-CNN (регіональні згорткові нейронні мережі) або Mask R-CNN. Такі моделі досягають найвищих показників точності, але, як правило, повільніші. З іншого боку, у нас є одноступінчасті детектори, такі як YOLO (You Only Look Once) і SSD (Singe Shot MultiBox Detector), які розглядають виявлення об'єктів як просту проблему регресії. Вони беруть вхідне зображення та аналізують ймовірність належності

певного об’єкту на картинці до відповідного класу та визначають координати обмежувального вікна. Такі моделі досягають нижчих показників точності, але набагато швидші, ніж двоступеневі детектори об’єктів.

Ось деякі найбільші переваги YOLO порівняно з іншими алгоритмами виявлення об’єктів:

- Дуже швидко (45 кадрів в секунду - краще, ніж у реальному часі)
- Легка і швидша версія: YOLO має меншу версію архітектури під назвою Tiny-YOLO, яка може працювати з більшою частотою кадрів (155 кадрів на секунду), проте з меншою точністю порівняно з фактичною моделлю.
- Мережа розуміє узагальнене представлення об’єктів, що означає, що прогнозування зображення в реальному світі та твори мистецтва досить точні.
- Детектор YOLO є безкоштовним та має відкрите джерело коду

Так як апаратне забезпечення, використане для цієї роботи має відносно низькі показники для використання в галузі комп’ютерного зору, то краще застосувати одноступеневі детектори.

Детектор YOLO (You only look once) був запропонований у 2016 р. і орієнтований на обробку в реальному часі. YOLO був натхненний нейромережею для розпізнавання зображень GoogleNet, а основна його ідея полягала у застосуванні унікальної нейронної мережі до всього зображення, і таким чином мережа буде ділити зображення на частини і одночасно передбачати обмежувальні рамки та ймовірності для кожного регіону. Ці обмежувальні рамки зважуються за прогнозованими ймовірностями.

YOLO розбиває зображення на сітку N × N, де кожна клітинка передбачає лише один об’єкт. Це передбачення – як фіксована кількість межових ящиків, де кожна шкала має свою оцінку достовірності. Він виявляє по одному об'єкту на комірку сітки, незалежно від кількості вікон, застосовуючи алгоритм Non Maximum Suppression. YOLO зазвичай використовує ImageNet для тренування

параметрів, а потім використовує набори даних виявлення цілей для навчання розпізнавання цілей.

Було запропоновано кілька вдосконалень в архітектурі YOLO (тобто версії YOLOv2, YOLOv3, YOLOv4, а також YOLOv5, яка була представлена недавно), які підвищили точність виявлення, зберігаючи при цьому дуже високу швидкість виявлення.

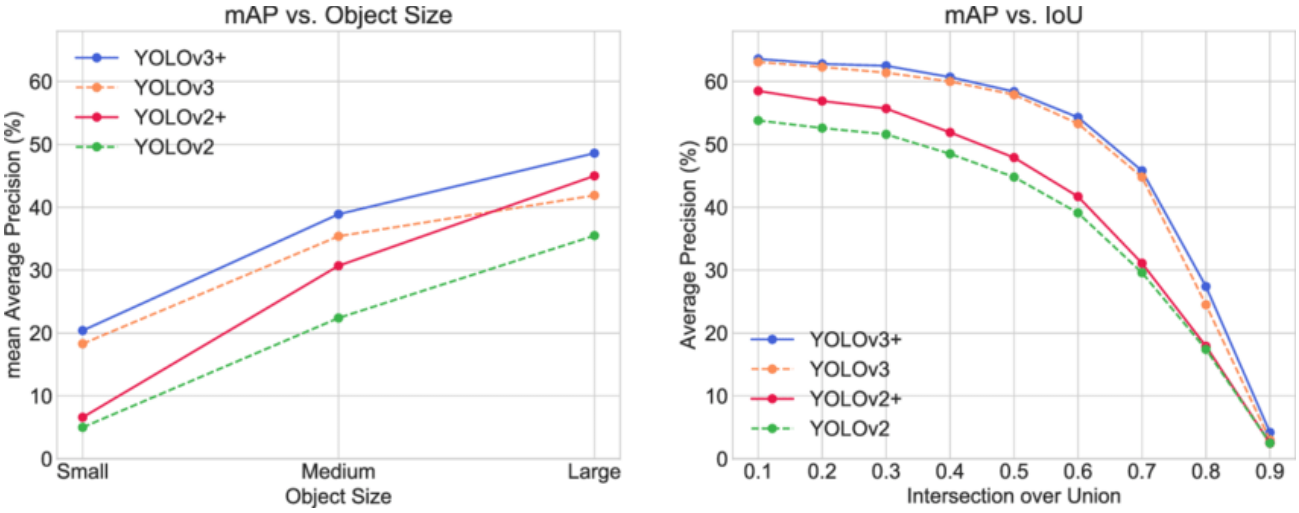


Рисунок 3.1 – Порівняння різних версій детектора YOLO.

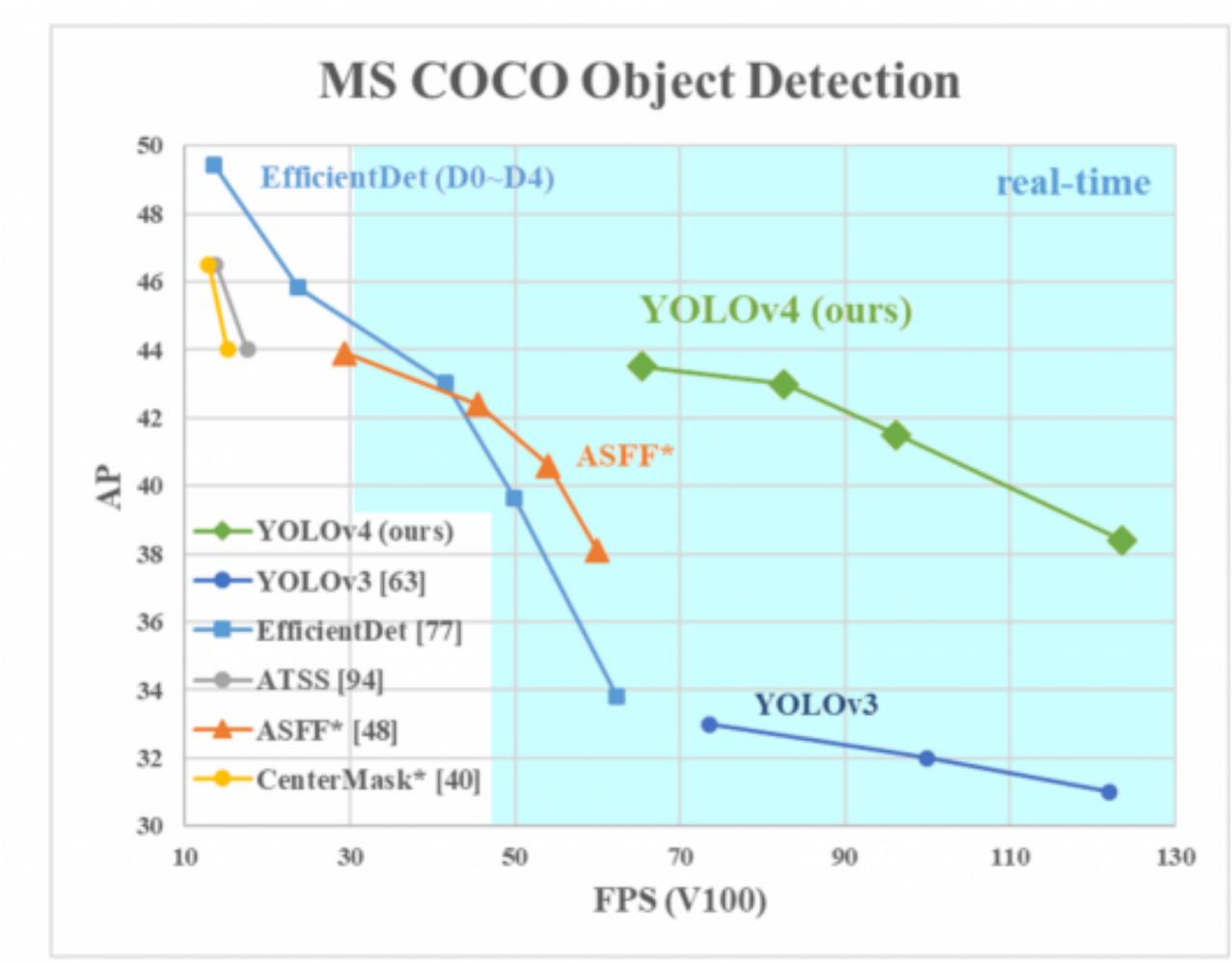


Рисунок 3.2 – Порівняння архітектур YOLOv3 та YOLOv4.

В даній роботі буде використано YOLOv4 як детектор гравців, нижче буде детально розглянуто його архітектуру.

YoloV4 - це важливе вдосконалення YoloV3. Впровадження нової архітектури в Backbone та модифікації в Neck покращили mAP на 10%, а кількість FPS на 12%. Крім того, стало легше навчати цю нейронну мережу на одному графічному процесорі. Автори розглянули такі основи для детектора об'єктів YOLOv4:

- CSPResNext50
- CSPDarknet53
- EfficientNet-B3

CSPResNext50 і CSPDarknet53 засновані на DenseNet. DenseNet був розроблений для з'єднання шарів у згорткових нейронних мережах. Це необхідно

з наступних причин: зменшити проблему зникаючого градієнта (важко повернути сигнали втрат через дуже глибоку мережу), посилити поширення особливостей між різнмим шарами , “заохотити” мережу до повторного використання знайдених особливостей та зменшити кількість параметрів мережі. У CSPResNext50 та CSPDarknet53 DenseNet було відредаговано таким чином, щоб відокремити мапу ознак базового шару, скопіювавши її, а іншу направити на наступний етап.

Ідея CSPResNext50 та CSPDarknet53 полягає у тому, щоб усунути вузькі місця в DenseNet та покращити навчання, передавши версію мапи ознак, яка не редагувалася. EfficientNet був розроблений Google Brain для вивчення в першу чергу проблеми масштабування згорткових нейронних мереж. Існує багато рішень, які можна прийняти при масштабуванні, включаючи розмір вхідних даних, масштабування в ширину, масштабування в глибину або взагалі масштабування всього вищезазначеного. EfficientNet перевершує інші мережі в своїй категорії у класифікації зображень. Однак автори YOLOv4 стверджують, що інші мережі можуть працювати краще в задачах виявлення об'єктів і пропонують експериментувати з усіма ними.

Глибока нейронна мережа Backbone, що складається в основному із згорткових шарів застовується в архітектурі YOLOv4. Головною метою основної Backbone є вилучення важливих особливостей, вибір Backbone є ключовим кроком, який покращить ефективність виявлення об'єктів. Часто для підготовки Backbone використовують попередньо навчені нейронні мережі. Архітектура Backbone YoloV4 складається з трьох частин:

- Bag of freebies
- Bag of specials
- CSPDarknet53

Bag of freebies – це сукупність методів, які лише збільшують вартість навчання або змінюють стратегію навчання, залишаючи низьку ціну висновку.

Розглянемо кілька простих методів, які зазвичай використовуються в комп'ютерному зорі.

Data augmentation. Основною метою методів Data augmentation (укрупнення даних) є збільшення варіацій одного і того ж об’єкту на різних зображеннях, щоб поліпшити тренування моделі. Найбільш часто використовувані методи - це Photometric Distortion, Geometric Distortion, MixUp, CutMix та GAN.

Photometric Distortion або Фотометричне спотворення як зрозуміло з назви створює нові зображення, регулюючи яскравість, відтінок, контраст, насиченість та шум, щоб відображати більше різновидів одного і того ж зображення.

Рисунок 3.3 – Фотометричне спотворення.

У наведеному вище прикладі було скориговано відтінок, щоб створити нові зразки зображення для урізноманітнення даних для тренування.

Методи Geometric distorsion або Геометричного спотворення – це такі методи, що використовуються для обертання зображення, масштабування або обрізання.

Рисунок 3.4 – Геометричне спотворення.

Як працює YOLO? Ось основні поняття того, як YOLO може виявити об’єкт. Детектор YOLO може передбачити клас об'єкта, його обмежувальну рамку та ймовірність класу об'єкта в обмежувальній рамці. Кожен обмежувальний блок має такі параметри:

- Центральне положення обмежувального вікна на зображенні (bx, by)
- Ширина коробки (bw)
- Висота коробки (bh)
- Клас об'єкта (c)

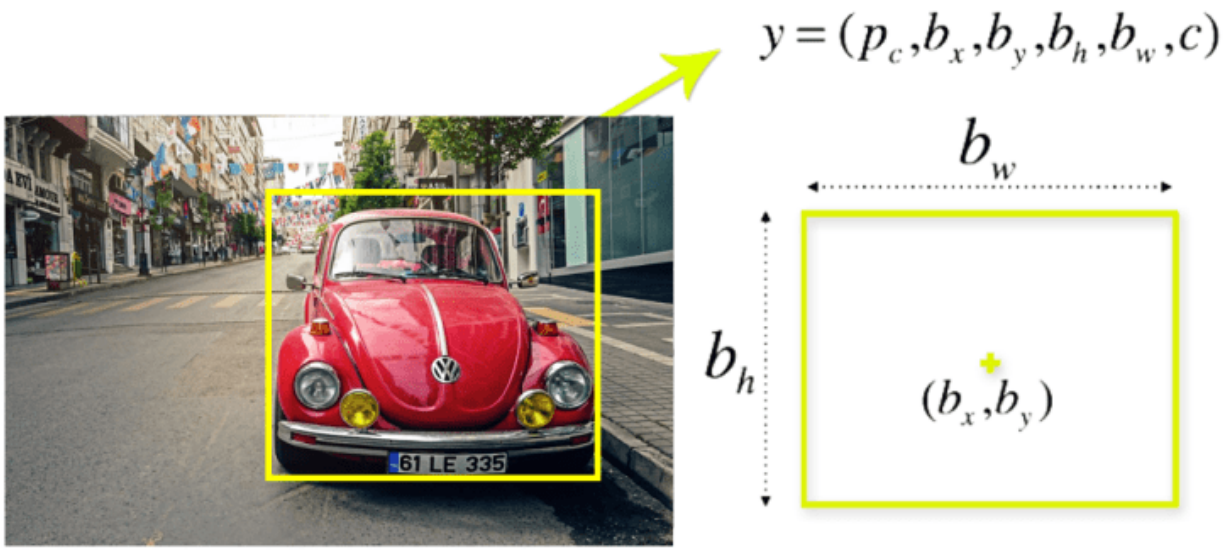


Рисунок 3.5 – Обмежувальна коробка та її характеристики.

Кожне обмежувальне поле пов'язане зі значенням ймовірності (pc), це імовірність знаходження класу об'єкта в цьому обмежувальному полі.

YOLO не шукає семантично цікавих регіонів, замість цього він розбиває зображення на кілька комірок, як правило, використовуючи сітку 19 на 19. Кожна комірка відповідає за передбачення 5 обмежувальних полів (у комірці може бути один або кілька об’єктів). В результаті, отримаємо 1805 обмежувальних коробок для одного зображення.

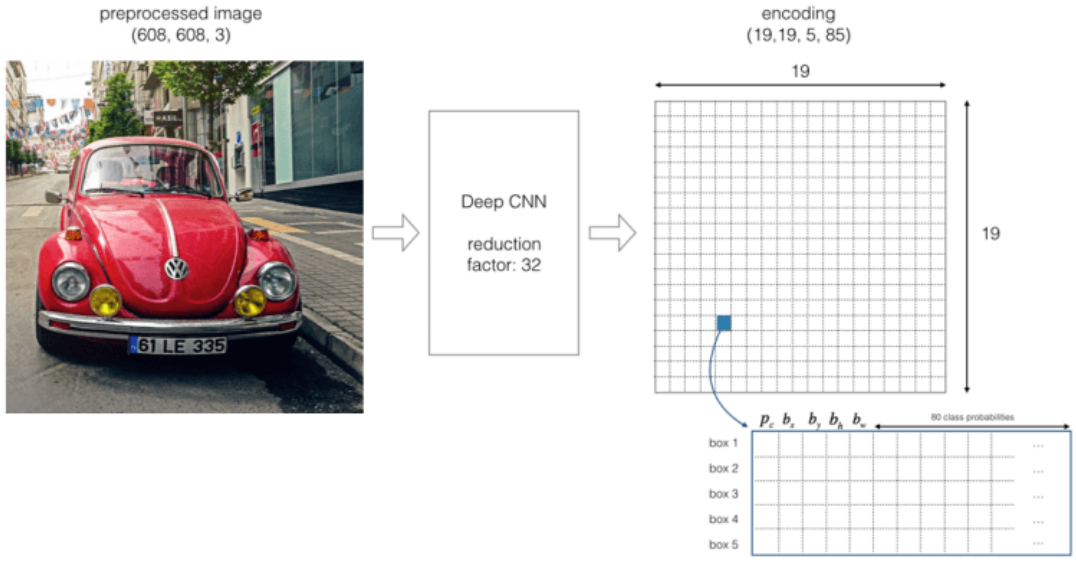


Рисунок 3.6 – Розбиття зображення на обмежувальні коробки.

Більшість обмежувальних коробок у комірці можуть і не мати об’єкта. Фільтрація цих обмежувальних коробок виконується на основі ймовірності класу об'єктів pc. Non-max suppression усуває
небажані обмежувальні рамки, і залишаться лише обмежувальні рамки з найбільшою ймовірністю.

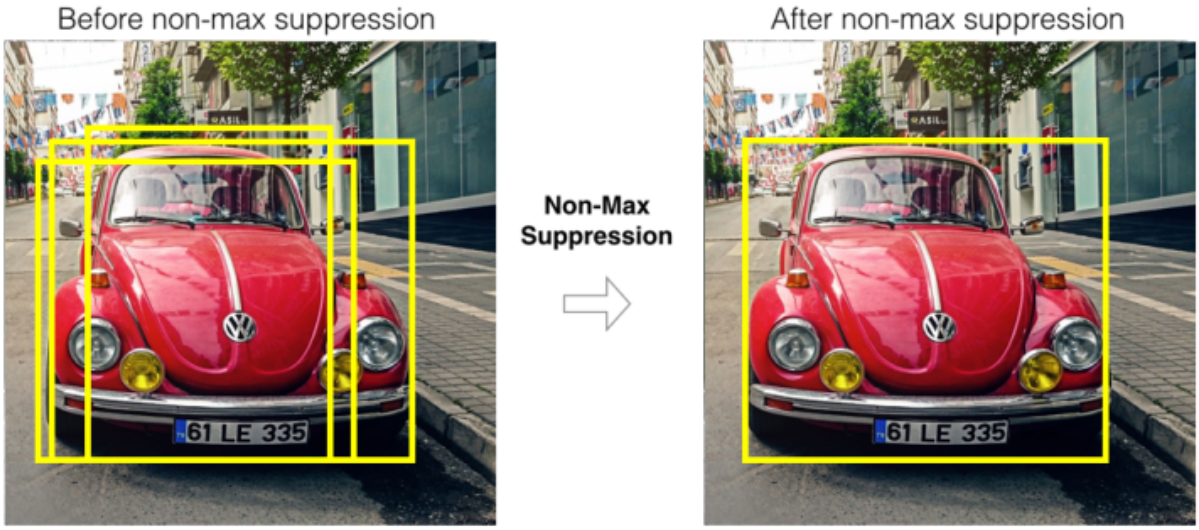


Рисунок 3.7 – Процес Non-max suppression.

3.2 Скріншоти програми

Для початку розглянемо покроково результати визначення грального поля використовуючи обраний алгоритм.
Візьмемо, наприклад, наступне зображення та застосуємо на ньому алгоритм.

Рисунок 3.8 – Вхідне зображення.

Конвертуємо його у колірний простір HSV.

Рисунок 3.9 – Зображення після трансформації в HSV.

Ізолюємо необхідні пікселі конвертованого зображення підібравши підходящі параметри.

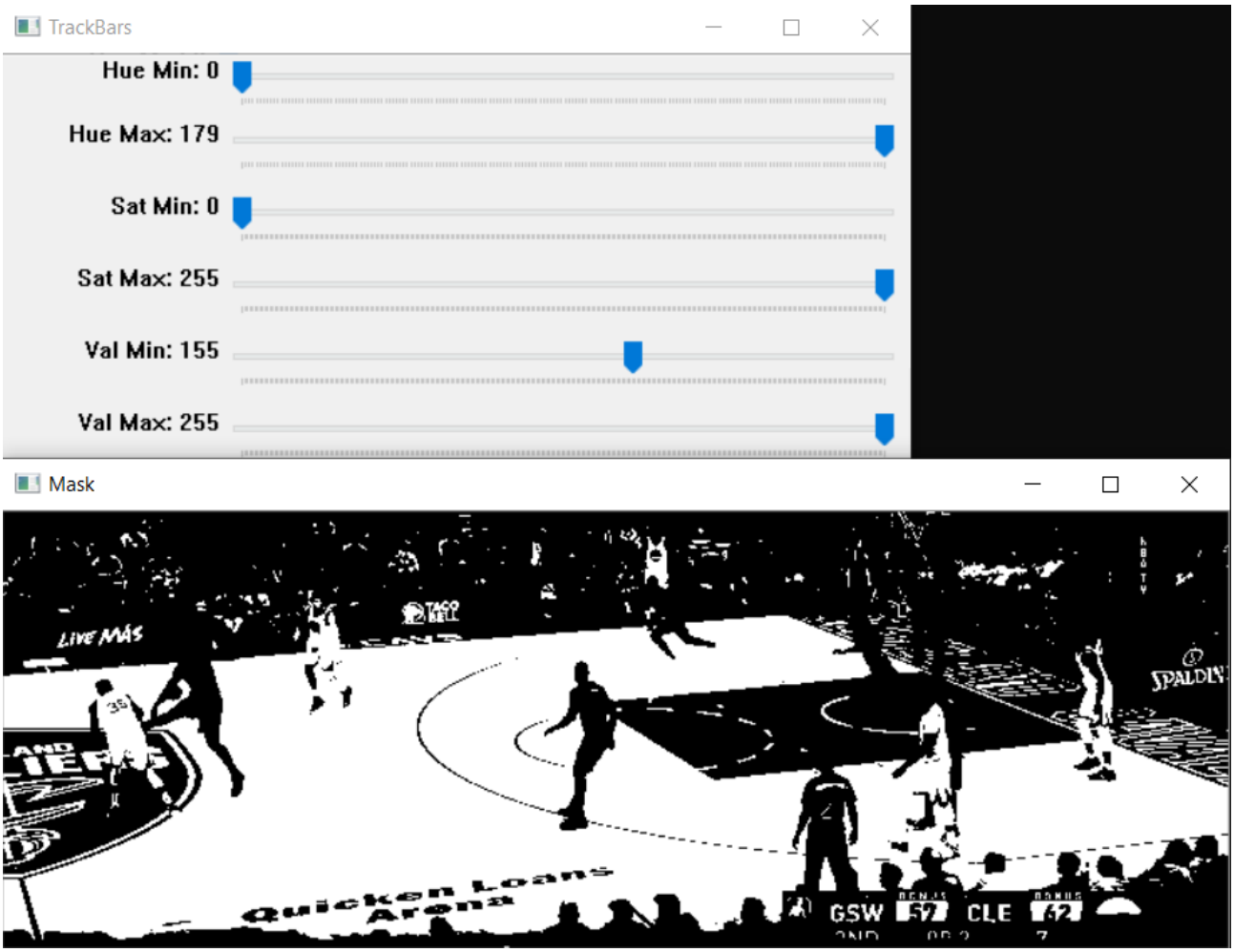


Рисунок 3.10 – Підбір параметрів для виявлення грального поля.

Накладемо маску побітового І на оригінальне зображення, та на зображення з виділеними пікселями.

Рисунок 3.11 – Частина зображення з виділеним баскетбольним полем.

В результаті отримали зображення, яке містить в основному пікселі грального поля.

Застосуємо алгоритм Canny для визначення границь зображення.

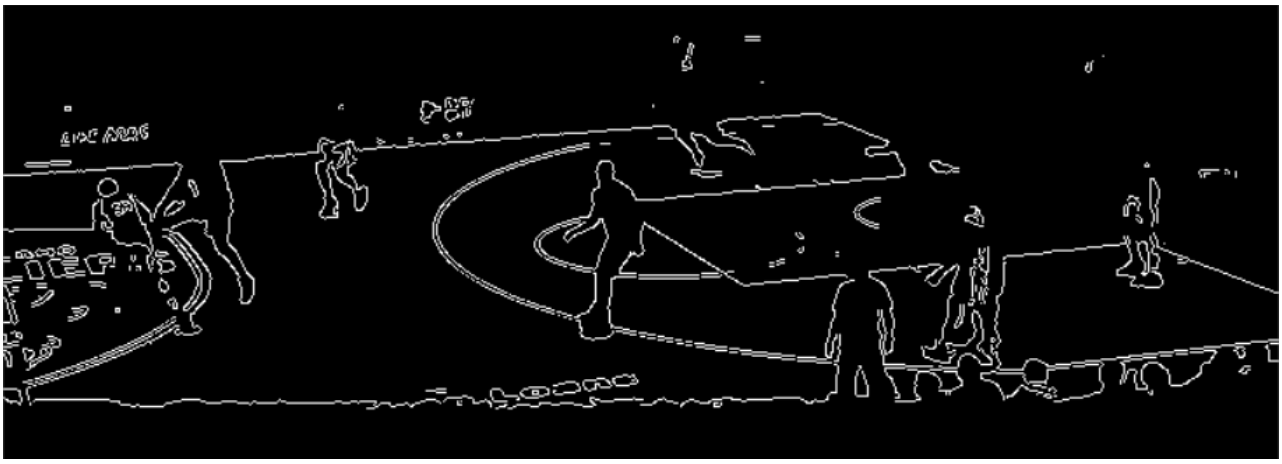


Рисунок 3.12 – Визначення границь грального поля.

Розпізнавання та відстеження рухів гравців. Нижче скрішоти результатів розпізнавання гравців за допомогою детектора об’єктів YOLOv4.

Рисунок 3.13 – Розпізнавання гравців на баскетбольному полі.

Взагалі, детектор YOLOv4 здатний працювати в режимі реального часу видаючи при цьому в середньому 50-70 кадрів в секунду, проте мені довелося чекати поки він обробить відео, щоб отримати результат, так як моє апаратне забезпечення не зможе задовільнити мінімальних вимог цього детектора (це насправді не так багато, необхідно як мінімум 3Гб відеопам’яті для коректної роботи).

3.3 Результати роботи

В результаті отримуємо зображення грального поля та відео з розпізнаними гравцями. Нижче представлені кадри з іншого баскетбольного відео. Для початку розпізнається гральне поле та відбувається його трансформація в двовимірну площину, таким чином, ніби поле видно зверху. Потім для кожного кадру алгоритм розпізнає гравців, за допомогою матриці трансформації отримує координати для двовимірної площини та показує на діаграмі, де знаходяться гравці. Трекер, в свою чергу, запам’ятовує положення гравців та дає кожному окремий ідентифікатор, що також відображається на діаграмі.

Рисунок 3.14 – Початкове зображення.

Рисунок 3.15 Трансформація грального поля на двовимірну площину.

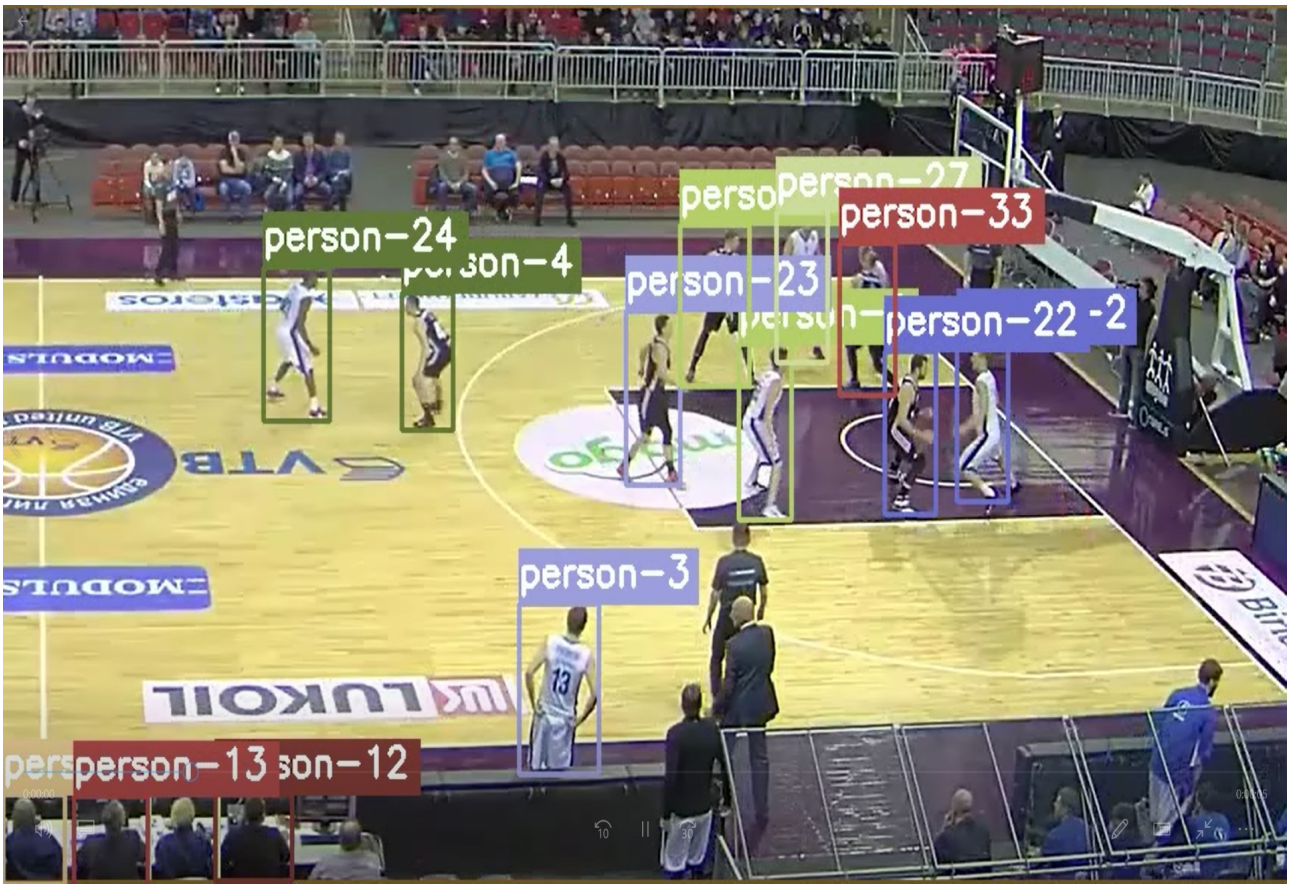


Рисунок 3.16 – Розпізнавання гравців на полі.

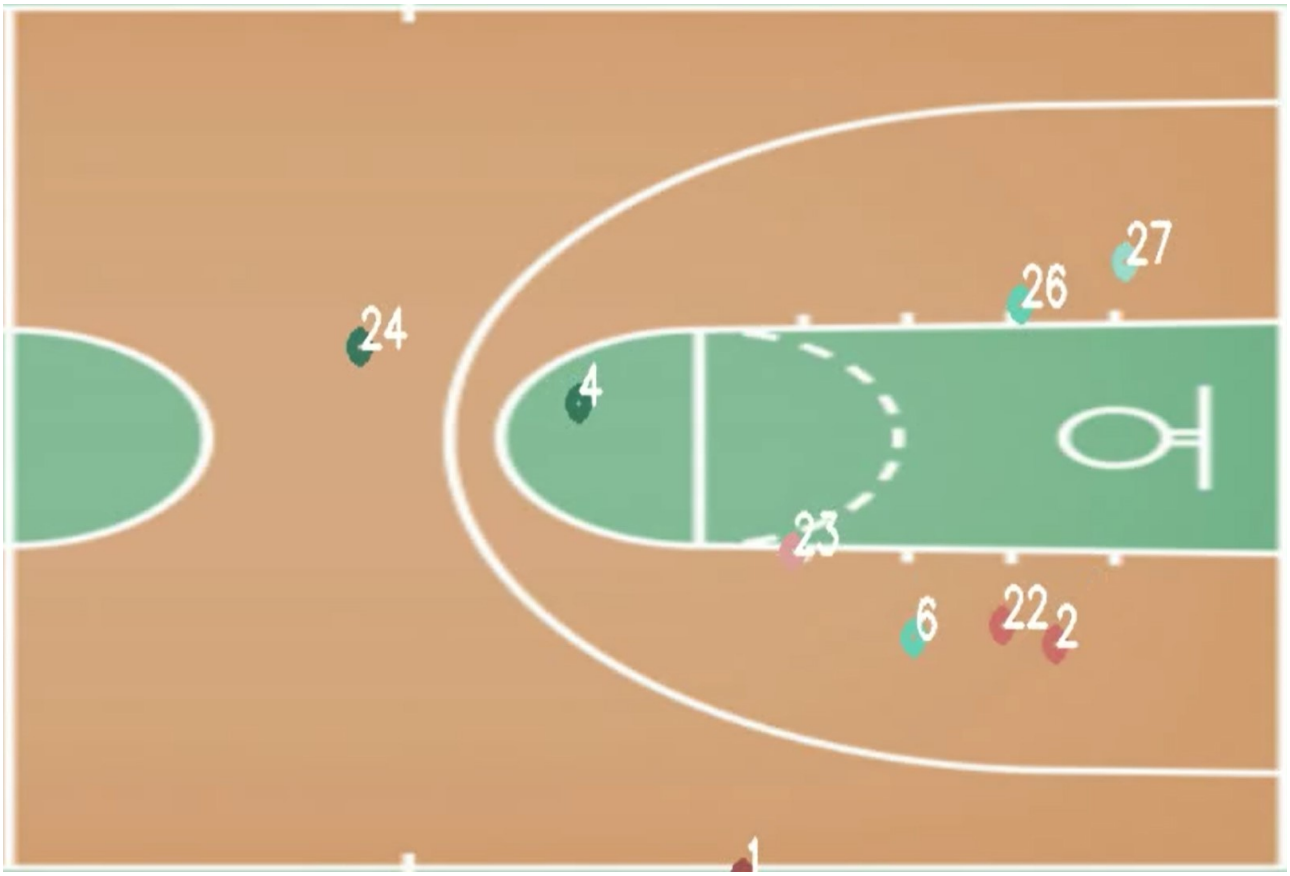


Рисунок 3.17 – Діаграма грального поля з розпізнаними гравцями.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, розробленого для вирішення задач розпізнавання рухів гравців та контурів грального поля.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для створення СППР розпізнавання дій гравців спортивних команд. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для виявлення грального поля та гравців.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп’ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує задачу розпізнавання та відстеження гравців та спортивного поля. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування;

F_2 – вибір бібліотеки для взаємодії з зображеннями;

F_3 – вибір моделей для розпізнавання об’єктів.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Python;

б) C++.

Функція F_2:

а) OpenCV;

б) MATLAB.

Функція F_3:

а) YOLOv4;

б) Mask-RCNN.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

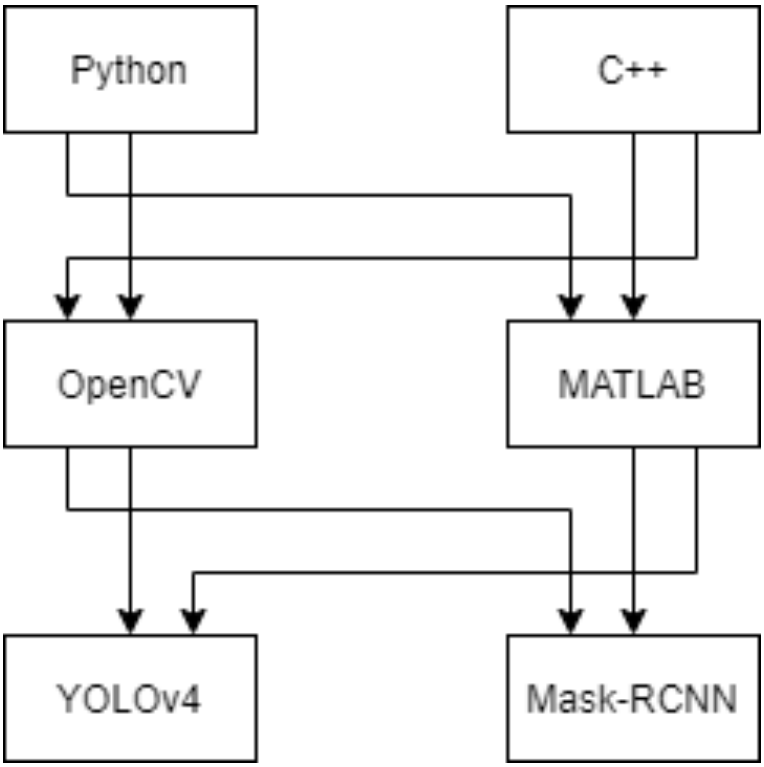


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіанти основних функцій.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Швидка розробка програми, доступність бібліотек,	Порівняно низька швидкість роботи, особливо, якщо медіафайли високої роздільної здатності
	Б	Код швидко виконується	Іде багато часу на розробку програми

F_2	*А*	Безкоштовна бібліотека з відкритим джерелом	Не такий широкий набір функцій
	Б	Широка функціональність	Не є безкоштовною
F_3	*А*	Швидка робота, навіть на середніх пристроях	Низька точність
	Б	Висока точність розпізнавання	Повільна робота, потребує потужного апаратного забезпечення

На основі цієї карти будуємо позитивно-негативну матрицю варіантів основних функцій (Таб.4.1). Робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1*:*

Перевагу віддаємо швидкій та простій розробці, наявності сучасної документації та великої спільноти розробників, що можуть допомогти. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F_2*:*

Так як перший варіант є безкоштовним, він буде у пріоритеті. Варіант Б буде відкинутий.

Функція F_3*:*

Обидва варіанти підходять.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а – F_2а – F_3а

F_1а – F_2а – F_3б

Для оцінювання якості розглянутих функцій обрана система параметрів,

описана нижче.

4.3 Обґрунтування системи параметрів ПП

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – швидкодія мови програмування;
- *X2* – об’єм пам’яті;
- *X3* – кількість кадрів в секунду під час розпізнавання;
- *X4* – вартість інструментів.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	5000	15000	25000
Об’єм пам’яті	X2	Мб	2010	1250	500
Кількість кадрів в секунду під час розпізнавання	X3	Кадрів/сек.	10	40	70
Вартість інструментів	X4	Долларів США	2000	1000	0

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

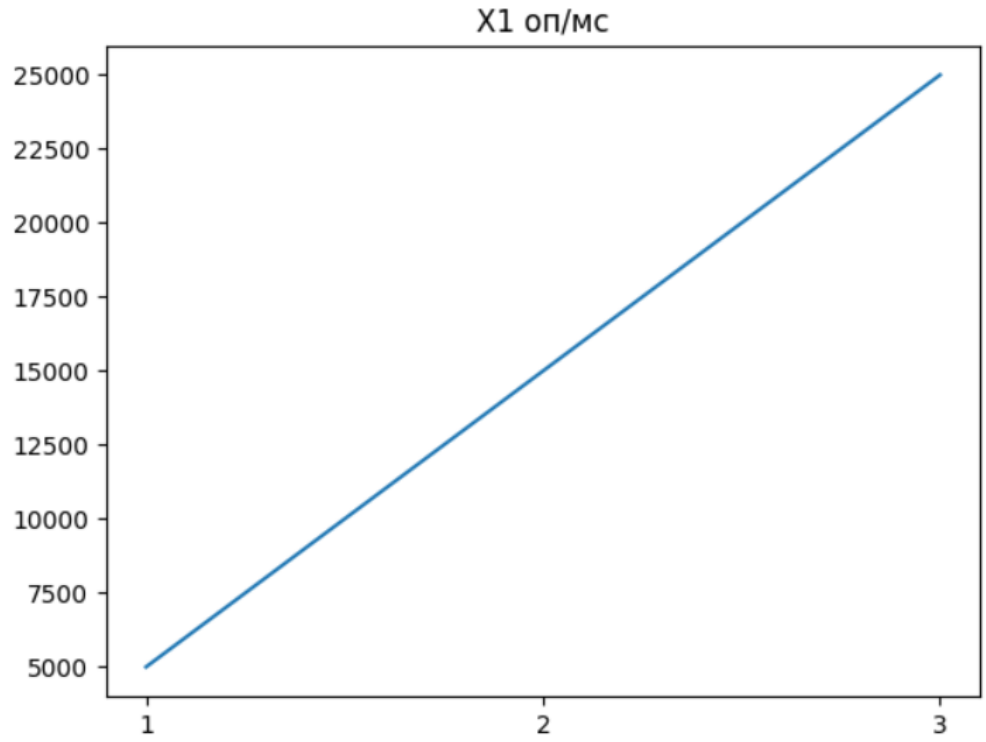


Рисунок 4.2 – X1, швидкодія мови програмування

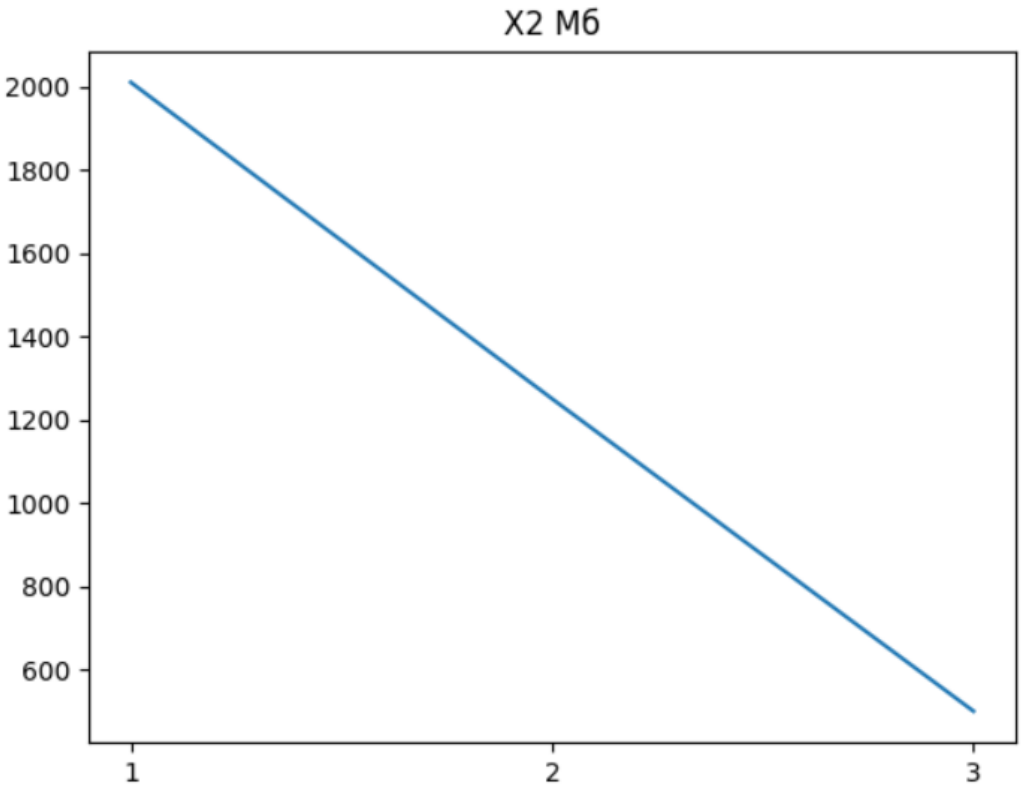


Рисунок 4.3 – X2, об’єм пам’яті

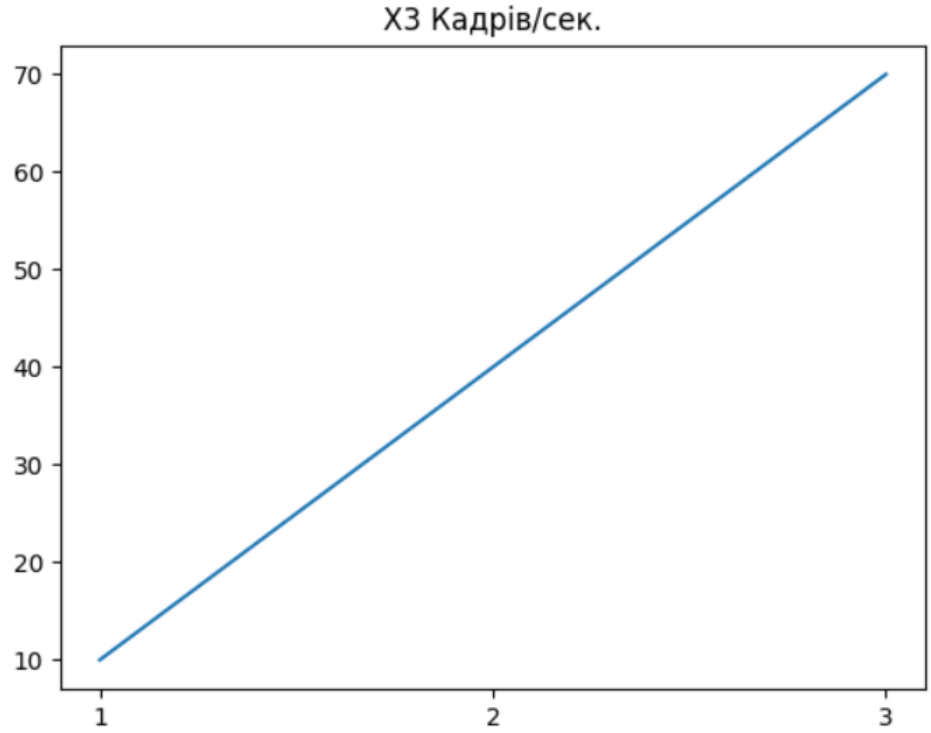


Рисунок 4.4 – X3, кількість кадрів в секунду під час розпізнавання

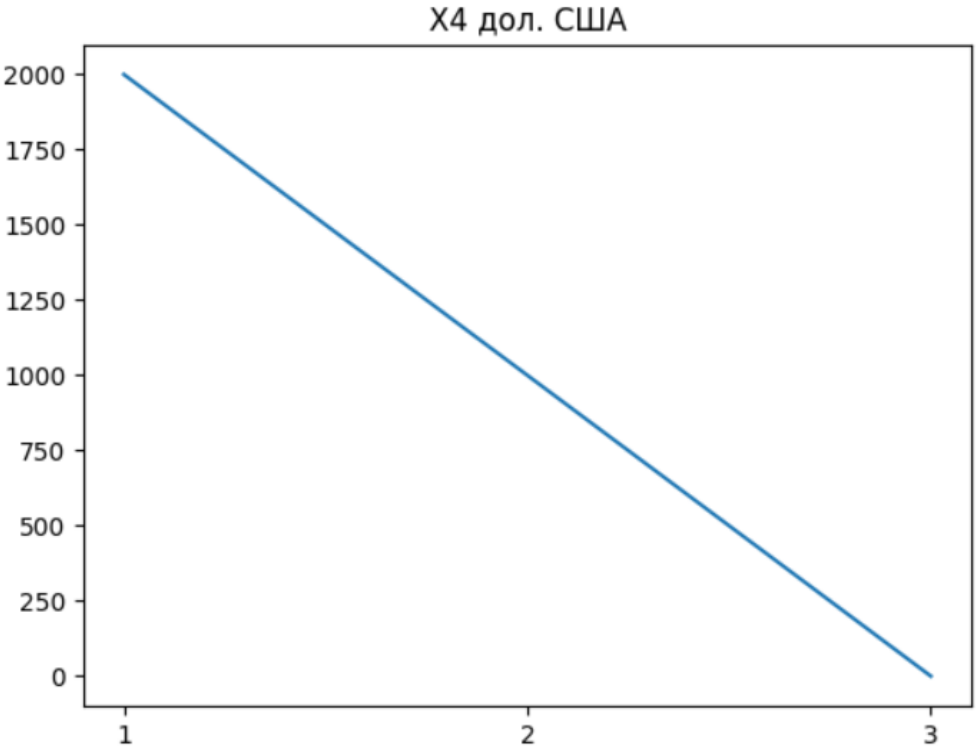


Рисунок 4.5 – X4, вартість інструментів

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхи-лення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	4	5	2	5	3	4	5	28	3,5	12,25
X2	Об’єм пам’яті	Mb	2	1	3	1	2	1	2	12	-12,5	156,25
X3	Час попередньої обробки даних	мс	5	3	5	5	4	5	3	30	5,5	30,25

X4	Потенційний об'єм програмного коду	Кількість рядків коду	3	5	4	3	5	4	4	28	3,5	12,25
	Разом		14	14	14	14	14	14	14	98	0	211

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 98, \quad (4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 24{,}5 \quad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \quad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 211. \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 211}{7^2(4^3 - 4)} = 0{,}86 > W_k = 0{,}67. \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	<	<	>	>	>	>	1,5
X1 і X3	<	>	<	=	<	<	>	<	0,5
X1 і X4	>	>	<	=	<	=	>	>	1,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	>	<	>	>	<	>	<	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю A=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \tag{4.7}$$

$$b_i = \sum_{i=1}^{N} a_{ij} \tag{4.8}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{4.10}$$

.Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметриx_i	Параметриx_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1,0	1,5	0,5	1,5	4,5	0,36	17,75	0,25	73,38	0,25
X2	0,5	1,0	1,5	0,5	3,5	0,28	15,75	0,22	65,63	0,23
X3	1,5	1,5	1,0	1,5	5,5	0,44	22,75	0,33	93,6	0,32
X4	0,5	1,5	0,5	1,0	3,5	0,28	13,75	0,2	57,63	0,2
Всього:					12,5	1	70	1	290,25	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об'єм пам'яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.7):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (4.11)$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	10000	7	0,25	1,25
F2	А	X2	64	6	0,23	1,16
F3	А	X3	128	8	0,32	1,7
	Б	X3	1000	2	0,2	0.7

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1{,}25 + 1{,}16 + 1{,}7 = 4{,}11,$$

$$K_{K2} = 1{,}25 + 1{,}16 + 0{,}7 = 3{,}11.$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як

$$\text{Т}_\text{О} = \text{Т}_\text{Р} \cdot \text{К}_\text{П} \cdot \text{К}_\text{СК} \cdot \text{К}_\text{М} \cdot \text{К}_\text{СТ} \cdot \text{К}_\text{СТ.М}, \quad (4.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P = 90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто T_P = 27 людино-днів, $K_П$ =0.9,$K_{СК}$ = 1,$K_{СТ}$ =0.8:

$$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (122.4 + 19.44 + 4.8 + 19.44) \cdot 8 = 1328{,}64 \text{ людино-годин.}$$

$$T_{II} = (122.4 + 19.44 + 6.91 + 19.44) \cdot 8 = 1345.52 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 20000 грн., один аналітик в області даних з окладом 35000. Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{грн.,} \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_Ч = \frac{20000 + 20000 + 35000}{3 \cdot 21 \cdot 8} = 148{,}89 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д\,, \quad (4.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 148{,}89 \cdot 1328.64 \cdot 1.15 = 227\,356{,}88$ грн.

II. $С_{ЗП} = 148{,}89 \cdot 1345.52 \cdot 1.15 = 230\,245{,}38$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 102811,43 \cdot 0.22 = 50\ 018,89$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 104117,62 \cdot 0.22 = 50\ 653,99$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 20000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 20000 \cdot 0,2 = 48000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 48000 \cdot (1 + 0.2) = 57600 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 57600 \cdot 0,22 = 13305 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 40000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 40000 = 11500 \text{ грн.},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 40000 \cdot 0.05 = 2300 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.9 =$$

$$= 1677.6 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1677.6 \cdot 0,3 \cdot 0.5 \cdot 3,51 = 883,55 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 40000 \cdot 0,67 = 26800 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \quad (4.17)$$

$$С_{ЕКС} = 57600 + 13305 + 11500 + 2300 + 883,55 + 26800 =$$

$$112\ 388,55 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 112\ 388,55/1677.6 = 66,99 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т, \quad (4.18)$$

I. $С_М = 66,99 \cdot 1328,64 = 89\ 005,59$ грн.

II. $С_М = 66,99 \cdot 1345.52 = 90\ 136,38$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67, \quad (4.19)$$

I. $С_Н = 227\ 356,88 \cdot 0,67 = 152\ 329,11$ грн.

II. $С_Н = 230\ 245,38 \cdot 0,67 = 154\ 264,40$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \quad (4.20)$$

I. $С_{ПП} = 227\ 356,88 + 50\ 018,89 + 89\ 005,59 + 152\ 329,11 = 518\ 710,47$ грн.

II. $С_{ПП} = 230\ 245,38 + 50\ 653,99 + 90\ 136,38 + 154\ 264,40 = 525\ 300,15$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{Кj} / С_{Фj}, \quad (4.21)$$

$$К_{ТЕР1} = 4,11/ 518\ 710,47 = 7,92 \cdot 10^{-6},$$

$$К_{ТЕР2} = 3,11/ 525\ 300,15 = 5,92 \cdot 10^{-6}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 7,92 \cdot 10^{-6}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились

після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості
$К_{ТЕР} = 7,92 \cdot 10^{-6}$.

4.8 Висновки до розділу 4

Проведено повний функціонально-вартісний аналіз програмного продукту. Визначено та проведено оцінку основних функцій програмного продукту. Визначено параметри, які характеризують програмний продукт. Проведено експертне оцінювання параметрів та аналіз якості варіантів реалізації функцій.

Проведено економічний аналіз варіантів розробки – трудомісткість, витрати на заробітну плату та інші витрати.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

В даній бакалаврській роботі було сформульовано та розв’язано задачу розпізнавання рухів гравців спортивних команд.

Було проведено пошук найпопулярніших підходів до вирішення цієї проблеми та методів її вирішення. Основним критерієм вибору підходящих методів комп’ютерного зору були особливості предметної області, а саме гравці на полі рухаються досить швидко та інколи перекривають один одного. Також вирішальним фактором стали фізичні обмеження пристроїв, на яких планувалося робити дослідження.

Знайдені методи було проаналізовано, проведено їх порівняльну характеристику та обрано кращий набір алгоритмів, що, в цілому, дозволить досягти результату.

В результаті було розроблено програмний продукт, що дозволяє маючи фрагмент відео баскетбольного змагання здійснити окреме розпізнавання грального поля покадрове розпізнавання гравців та їх рухів на цьому баскетбольному полі. Програма пропонує зручний користувацький інтерфейс, при взаємодії з яким можна обрати різні опції розпізнавання поля або відстеження гравців.

У майбутньому планується вдосконалення алгоритмів розпізнавання грального поля для більш універсальної роботи з різними типами відео(рухоме, половина поля). Також плюсом буде вдосконалити класифікацію гравців по командам. Для розпізнавання рухів було б непогано додати виявлення та класифікацію дій гравців, а саме оборона, напад.

ЛІТЕРАТУРА

1. Bolme, David S.; Beveridge, J. Ross; Draper, Bruce A.; Lui, Yui Man. Visual Object Tracking using Adaptive Correlation Filters. In CVPR, 2010.
2. Wei-Lwun Lu, Jo-Anne Ting, James J. Little, Kevin P. Murphy. Learning to Track and Identify Players from Broadcast Sports Videos.
   URL: https://www.cs.ubc.ca/~murphyk/Papers/weilwun-pami12.pdf
3. Naiyan Wang, Dit-Yan Yeung. Learning a Deep Compact Image Representation for Visual Tracking.URL:
   https://papers.nips.cc/paper/2013/file/dc6a6489640ca02b0d42dabeb8e46bb7-Paper.pdf
4. Pei-Chih Wen, Wei-Chih Cheng, Yu-Shuen Wang, Hung-Kuo Chu, Nick C. Tang, and Hong-Yuan Mark Liao, Fellow, IEEE. Court Reconstruction for Camera Calibration in Broadcast Basketball Videos.
   URL: https://people.cs.nctu.edu.tw/~yushuen/data/BasketballVideo15.pdf
5. Alexey Bochkovskiy, Chien-Yao Wang∗ Institute of Information Science Academia Sinica, Taiwan, Hong-Yuan Mark Liao Institute of Information Science Academia Sinica, Taiwan. YOLOv4: Optimal Speed and Accuracy of Object Detection. URL: https://arxiv.org/pdf/2004.10934.pdf
6. Simone Francia, Classificazione di Azioni Cestistiche mediante Tecniche di Deep Learning.
   URL:
   https://www.researchgate.net/publication/330534530_Classificazione_di_Azioni_Cestistiche_mediante_Tecniche_di_Deep_Learning
7. James Le, The 5 Computer Vision Techniques That Will Change How You See The World. URL: https://heartbeat.fritz.ai/the-5-computer-vision-techniques-that-will-change-how-you-see-the-world-1ee19334354b
8. Pierrick RUGERY, Explanation of YOLO V4 a one stage detector. URL: https://becominghuman.ai/explaining-yolov4-a-one-stage-detector-cdac0826cbd7

9. Mingxing Tan Ruoming Pang Quoc V. Le Google Research, Brain Team. EfficientDet: Scalable and Efficient Object Detection.
URL: https://arxiv.org/pdf/1911.09070.pdf
10. Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. CBAM: Convolutional Block Attention Module.
URL: https://arxiv.org/pdf/1807.06521.pdf
11. Deep Sort tracker and YOLO-v4 detector.
URL: https://github.com/theAIGuysCode/yolov4-deepsort
12. David Held, Sebastian Thrun, Silvio Savarese. Learning to Track at 100 FPS with Deep Regression Networks.
URL: https://davheld.github.io/GOTURN/GOTURN.pdf

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

- court_detector.py

```
import cv2
import numpy as np

img = cv2.imread('Picture1.png')
font = cv2.COLOR_BGR2BGR555

# show image
img = cv2.resize(img, (1080, 720))
cv2.imshow('image', img)

def four_point_transform(image, pts, line_pts):
        # obtain a consistent order of the points and unpack them
        # individually
  (tl, tr, br, bl) = pts
        # compute the width of the new image, which will be the
        # maximum distance between bottom-right and bottom-left
        # x-coordiates or the top-right and top-left x-coordinates
  widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
  widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
  maxWidth = max(int(widthA), int(widthB))
  # compute the height of the new image, which will be the
  # maximum distance between the top-right and bottom-right
  # y-coordinates or the top-left and bottom-left y-coordinates
  heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
  heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
  maxHeight = max(int(heightA), int(heightB))
  # now that we have the dimensions of the new image, construct
  # the set of destination points to obtain a "birds eye view",
  # (i.e. top-down view) of the image, again specifying points
  # in the top-left, top-right, bottom-right, and bottom-left
  # order
  dst = np.array([
    [0, 0],
```

```
        [maxWidth - 1, 0],
        [maxWidth - 1, maxHeight - 1],
        [0, maxHeight - 1]], dtype = "float32")
    # compute the perspective transform matrix and then apply it
    M = cv2.getPerspectiveTransform(pts, dst)
    warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))
    line_pts = np.float32(line_pts).reshape(-1,1,2)
    transformed_points = cv2.perspectiveTransform(line_pts,  M)
    print(transformed_points)
    for point in transformed_points:
        print((point[0][0], point[0][1]))
        cv2.circle(warped, (int(point[0][0]), int(point[0][1])), 4, font, 1)
    # return the warped image
    return warped

square_points = []
line_points = []

def mouse_click(event, x, y, flags, params):
    if event == cv2.EVENT_LBUTTONDOWN:
        print(x,y)
        cv2.circle(img, (x, y), 4, font, 1)
        square_points.append((x,y))

        cv2.imshow('image', img)
    if event == cv2.EVENT_RBUTTONDOWN:
        line_points.append((x,y))
    if event == cv2.EVENT_MBUTTONDOWN:
        print(line_points)
        warped = four_point_transform(img, np.array(square_points, dtype = "float32"),  np.array(line_points, dtype =
"float32"))
        cv2.imshow("Warped", warped)
cv2.setMouseCallback('image', mouse_click)
cv2.waitKey(0)
# close all the opened windows.
cv2.destroyAllWindows()
```

- object_tracker.py

```
import os
# comment out below line to enable tensorflow logging outputs
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import time
import tensorflow as tf
physical_devices = tf.config.experimental.list_physical_devices('GPU')
if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)
from absl import app, flags, logging
from absl.flags import FLAGS
import core.utils as utils
from core.yolov4 import filter_boxes
from tensorflow.python.saved_model import tag_constants
from core.config import cfg
from PIL import Image
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
# deep sort imports
from deep_sort import preprocessing, nn_matching
from deep_sort.detection import Detection
from deep_sort.tracker import Tracker
from tools import generate_detections as gdet
flags.DEFINE_string('framework', 'tf', '(tf, tflite, trt')
flags.DEFINE_string('weights', './checkpoints/yolov4-416',
                    'path to weights file')
flags.DEFINE_integer('size', 416, 'resize images to')
flags.DEFINE_boolean('tiny', False, 'yolo or yolo-tiny')
flags.DEFINE_string('model', 'yolov4', 'yolov3 or yolov4')
flags.DEFINE_string('video', './data/video/test.mp4', 'path to input video or set to 0 for webcam')
flags.DEFINE_string('output', None, 'path to output video')
flags.DEFINE_string('output_format', 'XVID', 'codec used in VideoWriter when saving video to file')
flags.DEFINE_float('iou', 0.45, 'iou threshold')
flags.DEFINE_float('score', 0.50, 'score threshold')
flags.DEFINE_boolean('dont_show', False, 'dont show video output')
flags.DEFINE_boolean('info', False, 'show detailed info of tracked objects')
flags.DEFINE_boolean('count', False, 'count objects being tracked on screen')
```

```
font = cv2.COLOR_BGR2BGR555

square_points = []
line_points = []
transformed_points = []

def main(transf_martix, court_width, court_height):
    # Definition of the parameters
    max_cosine_distance = 0.4
    nn_budget = None
    nms_max_overlap = 1.0
    results = []

    # initialize deep sort
    model_filename = 'model_data/mars-small128.pb'
    encoder = gdet.create_box_encoder(model_filename, batch_size=1)
    # calculate cosine distance metric
    metric = nn_matching.NearestNeighborDistanceMetric("cosine", max_cosine_distance, nn_budget)
    # initialize tracker
    tracker = Tracker(metric)

    # load configuration for object detector
    config = ConfigProto()
    config.gpu_options.allow_growth = True
    session = InteractiveSession(config=config)
    STRIDES, ANCHORS, NUM_CLASS, XYSCALE = utils.load_config(FLAGS)
    input_size = FLAGS.size
    video_path = FLAGS.video

    # load tflite model if flag is set
    if FLAGS.framework == 'tflite':
        interpreter = tf.lite.Interpreter(model_path=FLAGS.weights)
        interpreter.allocate_tensors()
        input_details = interpreter.get_input_details()
        output_details = interpreter.get_output_details()
        print(input_details)
        print(output_details)
    # otherwise load standard tensorflow saved model
```

```
else:
    saved_model_loaded = tf.saved_model.load(FLAGS.weights, tags=[tag_constants.SERVING])
    infer = saved_model_loaded.signatures['serving_default']

# begin video capture
try:
    vid = cv2.VideoCapture(int(video_path))
except:
    vid = cv2.VideoCapture(video_path)

out = None

print(transf_martix)

# get video ready to save locally if flag is set
if FLAGS.output:
    # by default VideoCapture returns float instead of int
    width = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(vid.get(cv2.CAP_PROP_FPS))
    codec = cv2.VideoWriter_fourcc(*FLAGS.output_format)
    out = cv2.VideoWriter(FLAGS.output, codec, fps, (width, height))
    court_out = cv2.VideoWriter('./outputs/court.avi', codec, fps, (720, 480))

frame_num = 0
# while video is running

while True:
    return_value, frame = vid.read()
    if return_value:
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image = Image.fromarray(frame)
    else:
        print('Video has ended or failed, try a different video format!')
        break
    frame_num +=1
    print('Frame #: ', frame_num)
    frame_size = frame.shape[:2]
    image_data = cv2.resize(frame, (input_size, input_size))
```

```
image_data = image_data / 255.
image_data = image_data[np.newaxis, ...].astype(np.float32)
start_time = time.time()

# run detections on tflite if flag is set
if FLAGS.framework == 'tflite':
    interpreter.set_tensor(input_details[0]['index'], image_data)
    interpreter.invoke()
    pred = [interpreter.get_tensor(output_details[i]['index']) for i in range(len(output_details))]
    # run detections using yolov3 if flag is set
    if FLAGS.model == 'yolov3' and FLAGS.tiny == True:
        boxes, pred_conf = filter_boxes(pred[1], pred[0], score_threshold=0.25,
                                        input_shape=tf.constant([input_size, input_size]))
    else:
        boxes, pred_conf = filter_boxes(pred[0], pred[1], score_threshold=0.25,
                                        input_shape=tf.constant([input_size, input_size]))
else:
    batch_data = tf.constant(image_data)
    pred_bbox = infer(batch_data)
    for key, value in pred_bbox.items():
        boxes = value[:, :, 0:4]
        pred_conf = value[:, :, 4:]

boxes, scores, classes, valid_detections = tf.image.combined_non_max_suppression(
    boxes=tf.reshape(boxes, (tf.shape(boxes)[0], -1, 1, 4)),
    scores=tf.reshape(
        pred_conf, (tf.shape(pred_conf)[0], -1, tf.shape(pred_conf)[-1])),
    max_output_size_per_class=50,
    max_total_size=50,
    iou_threshold=FLAGS.iou,
    score_threshold=FLAGS.score
)

# convert data to numpy arrays and slice out unused elements
num_objects = valid_detections.numpy()[0]
bboxes = boxes.numpy()[0]
bboxes = bboxes[0:int(num_objects)]
scores = scores.numpy()[0]
scores = scores[0:int(num_objects)]
```

```
        classes = classes.numpy()[0]
        classes = classes[0:int(num_objects)]

        # format bounding boxes from normalized ymin, xmin, ymax, xmax ---> xmin, ymin, width, height
        original_h, original_w, _ = frame.shape
        bboxes = utils.format_boxes(bboxes, original_h, original_w)

        # store all predictions in one parameter for simplicity when calling functions
        pred_bbox = [bboxes, scores, classes, num_objects]

        # read in all class names from config
        class_names = utils.read_class_names(cfg.YOLO.CLASSES)

        # by default allow all classes in .names file
        allowed_classes = list(class_names.values())

        # custom allowed classes (uncomment line below to customize tracker for only people)
        #allowed_classes = ['person']

        # loop through objects and use class index to get class name, allow only classes in allowed_classes list
        names = []
        deleted_indx = []
        for i in range(num_objects):
            class_indx = int(classes[i])
            class_name = class_names[class_indx]
            if class_name not in allowed_classes:
                deleted_indx.append(i)
            else:
                names.append(class_name)
        names = np.array(names)
        count = len(names)
        if FLAGS.count:
            cv2.putText(frame, "Objects being tracked: {}".format(count), (5, 35),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 2, (0, 255, 0), 2)
            print("Objects being tracked: {}".format(count))
        # delete detections that are not in allowed_classes
        bboxes = np.delete(bboxes, deleted_indx, axis=0)
        scores = np.delete(scores, deleted_indx, axis=0)
```

```
    # encode yolo detections and feed to tracker
    features = encoder(frame, bboxes)
    detections = [Detection(bbox, score, class_name, feature) for bbox, score, class_name, feature in zip(bboxes, scores, names, features)]

    #initialize color map
    cmap = plt.get_cmap('tab20b')
    colors = [cmap(i)[:3] for i in np.linspace(0, 1, 20)]

    # run non-maxima supression
    boxs = np.array([d.tlwh for d in detections])
    scores = np.array([d.confidence for d in detections])
    classes = np.array([d.class_name for d in detections])
    indices = preprocessing.non_max_suppression(boxs, classes, nms_max_overlap, scores)
    detections = [detections[i] for i in indices]

    # Call the tracker
    tracker.predict()
    tracker.update(detections)

    court = cv2.imread("court.png")
    court = cv2.resize(court, (court_width, court_height))

    # update tracks
    for track in tracker.tracks:
      if not track.is_confirmed() or track.time_since_update > 1:
        continue
      bbox = track.to_tlbr()
      class_name = track.get_class()
    # draw bbox on screen
      color = colors[int(track.track_id) % len(colors)]
      color = [i * 255 for i in color]
      # if(len(results) == 0):
      #     results.append([])
      #     results[0] = [track.track_id, [(int(bbox[2]), int(bbox[3]))]]
      # else:
      #     t = 0
      #     for i, res in enumerate(results):
      #         if len(res[1]) > 1:
```

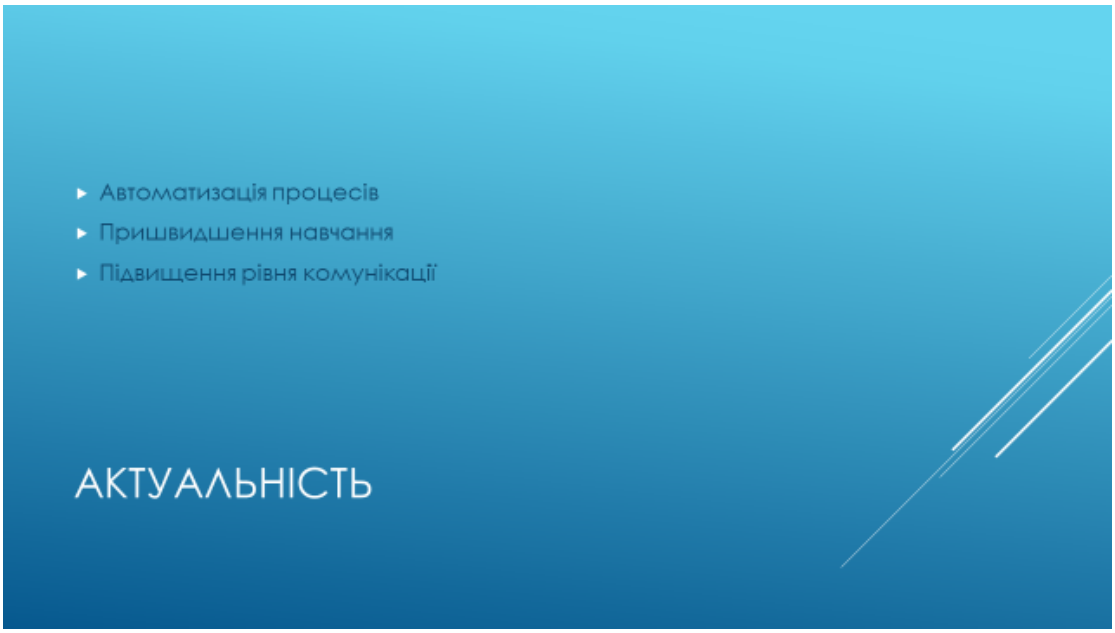
```
            #             a = np.array(res[1])
            #             cv2.polylines(frame, [a], False, color, 2)
            #         if res[0] == track.track_id:
            #             results[i][1].append((int(bbox[2]), int(bbox[3])))
            #             t = 1
            #             break
            #     if t == 0:
            #         results.append([])
            #         results[len(results)-1] = [track.track_id, [(int(bbox[2]), int(bbox[3]))]]

            cv2.rectangle(frame, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])), color, 2)
            cv2.rectangle(frame,  (int(bbox[0]),  int(bbox[1]-30)),  (int(bbox[0])+(len(class_name)+len(str(track.track_id)))*17, int(bbox[1])), color, -1)
            cv2.putText(frame, class_name + "-" + str(track.track_id),(int(bbox[0]), int(bbox[1]-10)),0, 0.75, (255,255,255),2)

            line_pts = np.array([(int(bbox[2]), int(bbox[3]))], dtype = "float32")
            line_pts = np.float32(line_pts).reshape(-1,1,2)
            transformed_points = cv2.perspectiveTransform(line_pts,  transf_martix)
            # print('transformed_points')
            # print(transformed_points)
            for point in transformed_points:
                cv2.circle(court, (point[0][0], point[0][1]), 5, color, 5)
                cv2.putText(court, str(track.track_id),(point[0][0], point[0][1]),0, 0.75, (255,255,255),2)
                cv2.imshow("Court", court)

        # if enable info flag then print details about each track
            if FLAGS.info:
                print("Tracker ID: {}, Class: {},  BBox Coords (xmin, ymin, xmax, ymax): {}".format(str(track.track_id), class_name, (int(bbox[0]), int(bbox[1]), int(bbox[2]), int(bbox[3]))))

        court = cv2.resize(court, (720, 480))
        cv2.imshow("Court", court)
        # calculate frames per second of running detections
        fps = 1.0 / (time.time() - start_time)
        print("FPS: %.2f" % fps)
        result = np.asarray(frame)
        result = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)

        if not FLAGS.dont_show:
```

```
            cv2.imshow("Output Video", result)

        # if output flag is set, save video file
        if FLAGS.output:
            out.write(result)
            court_out.write(court)
        if cv2.waitKey(1) & 0xFF == ord('q'): break
    cv2.destroyAllWindows()


def prepare_court(args):
    print(args)
    video_path = FLAGS.video
    vid = cv2.VideoCapture(video_path)
    return_value, first_frame = vid.read()
    cv2.imshow('image', first_frame)
    transf_matrix = None

    def mouse_click(event, x, y, flags, params):
        if event == cv2.EVENT_LBUTTONDOWN:
            print(x,y)
            # cv2.circle(img, (x, y), 4, font, 1)
            square_points.append((x,y))
            # cv2.imshow('image', img)

        if event == cv2.EVENT_RBUTTONDOWN:
            # print(square_points)
            # a = np.array(square_points)
            # print(a)
            # cv2.drawContours(img, [a], 0, (255,255,255), 2)
            # cv2.imshow('image', img)
            line_points.append((x,y))
        if event == cv2.EVENT_MBUTTONDOWN:
            # print(square_points)
            # a = np.array(square_points)
            # print(a)
            # cv2.drawContours(img, [a], 0, (255,255,255), 2)
            # cv2.imshow('image', img)
            print(line_points)
```

```
        warped = four_point_transform(first_frame, np.array(square_points, dtype = "float32"),  np.array(line_points, dtype = "float32"))
        cv2.imshow("Warped", warped)

    def four_point_transform(image, pts, line_pts):

        # obtain a consistent order of the points and unpack them
        # individually
        (tl, tr, br, bl) = pts
        # compute the width of the new image, which will be the
        # maximum distance between bottom-right and bottom-left
        # x-coordiates or the top-right and top-left x-coordinates
        widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
        widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
        maxWidth = max(int(widthA), int(widthB))
        # compute the height of the new image, which will be the
        # maximum distance between the top-right and bottom-right
        # y-coordinates or the top-left and bottom-left y-coordinates
        heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
        heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
        maxHeight = max(int(heightA), int(heightB))
        # now that we have the dimensions of the new image, construct
        # the set of destination points to obtain a "birds eye view",
        # (i.e. top-down view) of the image, again specifying points
        # in the top-left, top-right, bottom-right, and bottom-left
        # order
        dst = np.array([
            [0, 0],
            [maxWidth - 1, 0],
            [maxWidth - 1, maxHeight - 1],
            [0, maxHeight - 1]], dtype = "float32")
        # compute the perspective transform matrix and then apply it
        transf_matrix = cv2.getPerspectiveTransform(pts, dst)
        cv2.destroyAllWindows()
        main(transf_matrix, maxWidth, maxHeight)
        # warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))
        # line_pts = np.float32(line_pts).reshape(-1,1,2)
        # transformed_points = cv2.perspectiveTransform(line_pts,  M)
        # print(transformed_points)
```

```
        # for point in transformed_points:
        #     print((point[0][0], point[0][1]))
        #     cv2.circle(warped, (int(point[0][0]), int(point[0][1])), 4, font, 1)
        # return warped

    cv2.setMouseCallback('image', mouse_click)
    cv2.waitKey(0)

if __name__ == '__main__':
    try:
        app.run(prepare_court)
    except SystemExit:
        pass
```

ДОДАТОК Б ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ РОБОТИ

- Об'єкт дослідження – дії гравців спортивних команд. Детальніше в цій роботі буде розглянуто рухи гравців баскетбольних команд.
- Предмет дослідження – алгоритми розпізнавання образів, таких як людина(гравець) та баскетбольне поле.
- Мета дослідження – створення системи, яка займатиметься розпізнаванням баскетбольних гравців та грального поля, і подальша класифікація їх рухів.

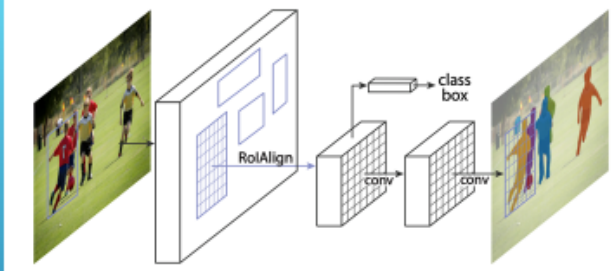
ПОСТАНОВКА ЗАДАЧІ

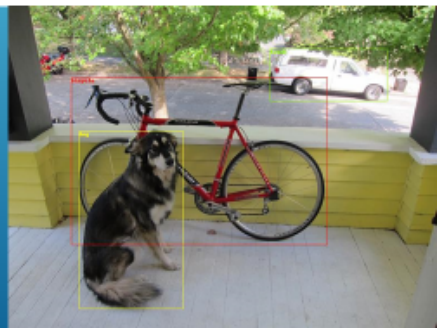
- Дослідження наявних методів комп'ютерного зору
- Якісний аналіз та порівняння знайдених методів
- Вибір інструментів для реалізації задачі
- Розробка ПЗ

- Класифікація зображень
- Виявлення об'єктів
- Відстеження об'єктів
- Семантична сегментація

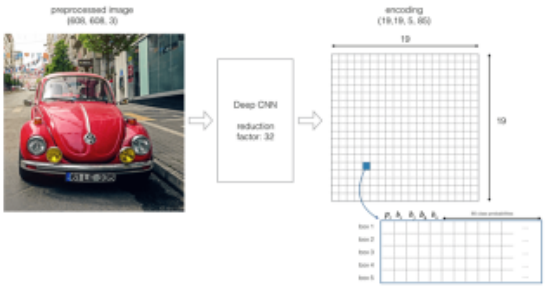
ДОСЛІДЖЕННЯ ІСНУЮЧИХ МЕТОДІВ

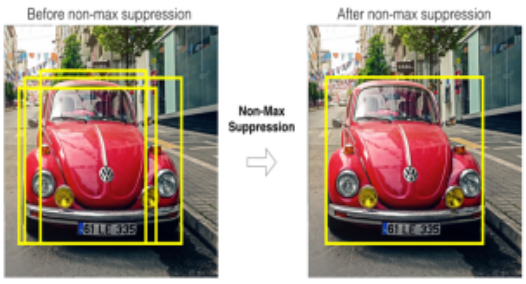
- Mask R-CNN
- YOLO





ВИЯВЛЕННЯ ОБ'ЄКТІВ



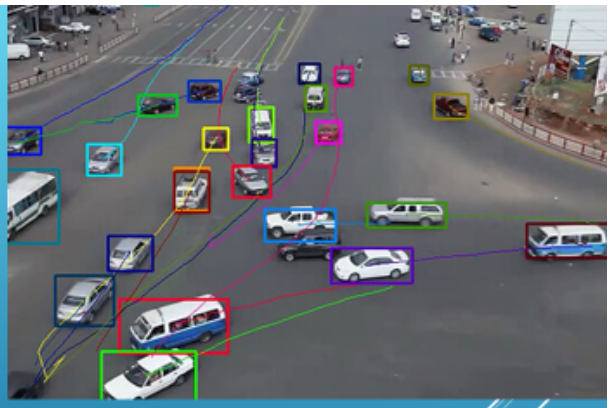


Розбиття на комірки

Процес Non-Max Suppression

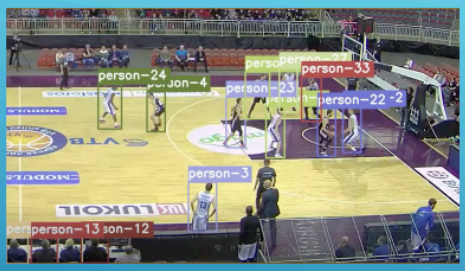
YOLOV4

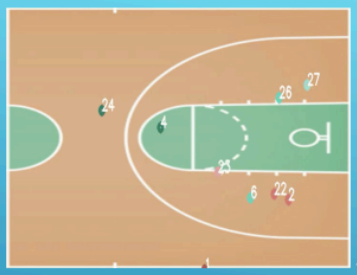
- Deep-SORT



ВІДСТЕЖЕННЯ ОБ'ЄКТІВ

РЕЗУЛЬТАТИ РОБОТИ





РЕЗУЛЬТАТИ РОБОТИ

ВИСНОВКИ

- Розглянуто основні методи комп'ютерного зору
- Проаналізовано та обрано інструменти для виконання задачі
- Розроблено програму для виявлення та відстеження грального поля та гравців на ньому

ДЯКУЮ ЗА УВАГУ!