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

«___»_____21__ р.

Дипломна робота

на здобуття ступеня бакалавра

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

спеціальності 122 «Комп'ютерні науки»

на тему: «Система розпізнавання монет»

Виконав:
студент IV курсу, групи КА-76
Пивовар Павло Євгенович _____

Керівник:
доцент, к.т.н. Дідковська М. В. _____

Консультант з нормконтролю:
доцент, к.т.н. Коваленко А. Є. _____

Консультант з економічного розділу:
доцент, к.е.н. Рощина Н. В. _____

Рецензент:
доцент, к.т.н Заболотня Т. М. каф. ПЗКС _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент (-ка) _____

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 і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Оксана ТИМОЩУК
«26» травня 2021 р.

ЗАВДАННЯ
на дипломну роботу студенту
Пивовару Павлу Євгеновичу

1. Тема роботи «Система розпізнавання монет», керівник роботи Дідковська Марина Віталівна, доцент кафедри ММСА, затверджені наказом по університету від «26» травня 2021 р. № 1344-с

2. Термін подання студентом роботи 7.06.2021.

3. Вихідні дані до роботи — набір фотографій українських монет.

4. Зміст роботи — 1. Постановка задачі; 2. Аналіз методів розв’язання задачі; 3. Опис процесу побудови системи.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Консультанти розділів роботи

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

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Підготовка даних до роботи	13.02.2021	Виконано
2.	Вивчення літератури за темою роботи	11.03.2021	Виконано
3.	Підготовка першого розділу	26.03.2021	Виконано
4.	Розробка моделей нейронних мереж	02.04.2021	Виконано
5.	Підготовка другого розділу	22.04.2021	Виконано
6.	Інтеграція нейронної мережі з детекторами та дескрипторами	18.05.2021	Виконано
7.	Підготовка третього розділу	24.05.2021	Виконано
8.	Підготовка економічної частини	27.05.2021	Виконано
9.	Підготовка презентації доповіді	29.05.2021	Виконано

Студент Пивовар Павло

Керівник Марина Віталівна

РЕФЕРАТ

Дипломна робота: 97 с., 26 рис., 8 табл., 2 дод., 23 джерела.

КОМП’ЮТЕРНИЙ ЗІР, РОЗПІЗНАВАННЯ ОБРАЗІВ, ДЕТЕКТОРИ ОБ’ЄКТІВ, ДЕТЕКТОРИ КЛЮЧОВИХ ТОЧОК, ДЕКСКРИПТОРИ, НЕЙРОННІ МЕРЕЖІ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, YOLO

Темою роботи є система розпізнавання монет.

Об’єктом дослідження є система розпізнавання монет.

Предметом дослідження є точність й швидкодія розпізнавання системою.

Метою даної работи є побудова системи розпізнавання монет, що з достатньою точністю та швидкістю детектувала б та класифікувала б монети.

Актуальність роботи пов’язана із потребою автоматизації процесу підрахунку монет.

В результаті роботи було побудовано систему розпізнавання монет, що з високою точністю знаходить монети та визначає їх клас.

ABSTRACT

Theme: “Coin recognition system”

Bachelor’s thesis: 97 p., 26 figs, 8 tab., 2 appendix, 23 sources.

The object of research is a coin recognition system.

The subject of the study is the accuracy and speed of the system.

The purpose of this work is to build a coin recognition system that would detect and classify coins with good enough accuracy and speed.

The urgency of the work is based on the need to automate routine process of calculation of the total value of the coins.

The output of the work is a coin recognition system that is capable of detecting and classifying coins with good enough precision and speed.

ЗМІСТ

ВСТУП

В процесі розвитку технологій декількох минулих десятиріч фотокамери стали широко поширені. Сьогодні практично кожен смартфон має вбудовану камеру. До мережі Інтернет щоденно потрапляє велика кількість зображень. Загалом камери стали більш доступними не тільки для звичайних людей, а й для бізнесів, громадських організацій та деражавних структур.

Таким чином, виникає явище великих даних (big data) й зображення є одним із їх елементів. Проте самі по собі дані не мають користі, поки їх не використати для вирішення реальних проблем.

Крім того стрімко розвивається наука, що вивчає те, як можна за допомогою комп'ютерів отримати високорівневі знання з цифрових зображень — комп'ютерне бачення або комп'ютерний зір (computer vision).

Сьогодні за допомогою комп'ютерного бачення люди здатні розв'язати велику кількість задач. Серед найпоширеніших сфер застосування перелічують такі:

— оптичне розпізнавання символів (Optical character recognition, OCR) дозволяє виділяти текст із зображення, наприклад у сервісі Google Translate можна зробити фото сторінки книги й програма спершу виділить текст на зображенні, а потім поверне його переклад;

— каси самообслуговування стають все більш поширенеми серед мереж продуктових магазинів;

— аналіз медичних знімків допомагає робити діагностування хвороб більш точним;

— системи допомоги водію (driver-assistance systems) та автопілот поступово вдосконалюються та допомагають зменшувати кількість дорожньо-транспортних пригод;

— побудова 3D моделей (фотограметрія) дозволяє за допомогою декількох фото підібрати одяг людині;

— камери спостереження дозволяють виявляти порушення громадського порядку.

Список галузей застосування насправді практично не вичерпний й надалі буде тільки збільшуватись.

1 АНАЛІЗ ЗАДАЧІ РОЗПІЗНАВАННЯ МОНЕТ

1.1 Задача розпізнавання об’єктів

Розпізнавання об’єктів (object recognition) — це підгалузь комп’ютерного зору, що вивчає задачу пошуку та ідентифікації об’єктів у фото або відео.

Розпізнавання об’єктів можна поділити на дві підкатегорії: розпізнавання екземплярів об’єктів (instance recognition) та розпізнавання класів об’єктів або просто класифікація (class recognition) [1].

Розпізнавання екземплярів полягає у виділенні певних об’єктів серед фону та інших об’єктів при зміні точки спостереження на ці об’єкти та частковому їх перекриванні іншими. На рисунку 1.1 нижче зображено результат роботи розпізнавання екземплярів із роботи Девіда Лоу [2]. Ліворуч зображені два фото із бази даних. За допомогою алгоритму SIFT, про який піде мова у наступних розділах, об’єкти з бази даних знайдені у сцені наповненій різноманітними об’єктами.

Рисунок 1.1 — Розпізнавання екземплярів об’єктів

Розпізнавання класів є більш складною задачею та полягає у визначенні будь-якого загального класу екземпляру такого як: “кіт”, “собака” чи “велосипед”.

В цьому дослідженні розглядається задача розпізнавання класів й відповідно на виході система має повертати категорію монети. Про визначення множини класів мова піде далі. На рисунку 1.2 зображено результат класифікації системи.

Рисунок 1.2

Ускладненою версією класифікації є детекція або детектування об’єктів (object detection), надалі будемо називати задачею локалізації. На виході отримують не лише клас об’єкту, а й обмежувальну коробку (bounding box), що визначає розміщення об’єкта у фото. В цій роботі також використовують методи детекції об’єктів. На рисунку 1.3 нижче зображено результат роботи детекції.

Рисунок 1.3

Таким чином, система розпізнавання монет побудована в результаті проведення даного дослідження має поєднувати в собі класифікацію та локалізацію об’єктів.

1.2 Аналіз методів розпізнавання об’єктів

Методи розпізнавання об’єктів ділять на дві групи — класичні та засновані на нейронних мережах. Такий поділ зумовлений поворотом, який відбувся у сфері комп’ютерного зору після публікації дослідження Алекса Крижевського, Іллі Суцкевера та Джефрі Гінтона у 2012 році [3]. З того часу глибинне навчання, галузь машинного навчання, що ґрунтується на використанні штучних нейронних мереж, почала бурхливо розвиватись. Проте

як буде показано в даному дослідженні класичні методи для певних сфер застосування все ще здатні показати гарні результати.

1.3 Класичні методи розпізнавання об’єктів

Класичні методи розпізнавання об’єктів також називають методами заснованими на рисах або характеристиках (feature-based). Такі методи працюють у три етапи:

— виділення або детектування ключових точок (feature detection);

— обчислення описів або дескрипцій ключових точок виділених на попередньому кроці (feature description);

— знаходження збігів між дескрипціями (feature matching).

Ключовими точками, рисам або характеристиками (features) у зображенні називаються регіони або точки, які зберігають в собі важливу інформацію про те, що є у даному зображенні, наприклад кути, краї, плями. У випадку фото монети ключовими точками можуть бути очі на портретах відомих особистостей, вуса, букви під портретом тощо. На рисунку 1.4 проілюстровано результат роботи детектування ключових точок для монети. Як можна побачити більшість точок згруповані в околі очей, шиї, вус, бороди та букв. Для виділення ключових точок використовуються спеціальні методи (feature detectors) про які піде мова у наступних розділах.

Рисунок 1.4

Наступний крок виконують алгоритми, що називаються дескрипторами (descriptors). Задача дескрипторів — знайти такі описи ключових точок, які були б інваріантиними до різноманітних перетворень (повороти, зміщення, масштабування, шум тощо). Іншими словами, дескриптори знаходять такий опис ключових рис, що є однаковим для зображень одного й того самого об’єкту зроблених у різних умовах.

На третьому етапі знаходять збіги між дескрипціями двох зображень. Одне зображення виступає в ролі еталону — ідеального представника об’єкту представленого на ньому, сфотографованого в найкращих умовах освітлення, без шуму, без сторонніх предментів. Друге зображення показує об’єкт у довільних умовах. Дескрипції обчислені для одного й другого зображення на попередніх етапах порівнюються. В результаті алгоритм повертає набір точок із одного зображення, яким знайшлась пара із іншого. На рисунку 1.5 наведеному нижче зображені результат роботи пошуку збігів.

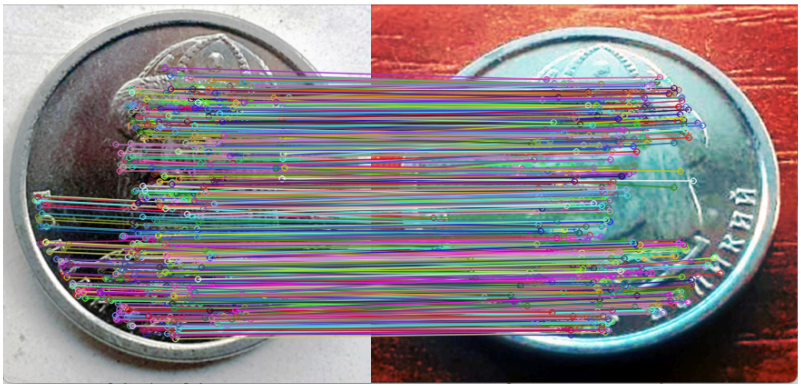


Рисунок 1.5

Так для того, щоб класифікувати зображення за допомогою детекторів та дескрипторів потрібно спершу для кожного класу створити зображення еталон. Вхідне зображення буде порівнюватись із кожним еталоном й клас того еталону, що отримав найбільшу кількість збігів, присвоюється вхідному зображенню. На рисунку 1.6 зображена множина еталонів монет створена для даного дослідження

Рисунок 1.6

Додатковим етапом роботи класичних методів, який може покращити роботу всього класифікатора є відсікання помилок (outlier rejection). З тих рис, що залишились в результаті пошуку збігів як правило певна кількість є помилковими. На рисунку 1.5 можна побачити, що деякі відрізки сполучають точки, що насправді не відповідають одна одній. Тож для зменшення кількісті таких сторонніх точок застосовують методи відсікання.

1.4 Штучні нейронні мережі

Перш за все штучні нейронні мережі — це алгоритми машинного навчання. Машинне навчання (machine learning) — це підгалузь штучного інтелекту, що вивчає алгоритми, які автоматично покращуються через набуття досвіду використовуючи при цьому дані.

Взагалі алгоритми машинного навчання ділять на такі категорії: навчанння із вчителем (supervised learning), навчання без вчителя (unsupervised learning), часткове навчання або напівавтоматичне навчання (semi-supervised learning) та навчання з підкріпленням (reinforcement learning). Вчені застосовували нейронні мережі в кожній із цих парадигм машинного навчання, проте у фокусі даного дослідження буде тільки навчання з вчителем, адже у цьому напрямі було зроблено найбільше досліджень.

Навчання із вчителем — це задача вивчення функції, що відображає вхідні дані до вихідних на основі прикладів вхідних-вихідних пар. У випадку комп’ютерного зору для задачі класифікації вхідними прикладами є пари зображень та ярликів (labels), що позначають клас. Для задачі локалізації

об’єктів окрім зображення й класу ще подається обмежувальна коробка. Нейронна мережа вивчає особливості об’єктів на фото й самостійно виділяє риси, що найкраще характеризують кожен клас. Звідси випливає, що фундаментальною різницею між класичними методами й заснованими на нейронних мережах полягає у тому, що у другому випадку людині більше не потрібно розробляти окремий алгоритм, який би визначав, які риси є корисними, нейронна мережа робить це самостійно шляхом навчання на множині прикладів. В результаті на виході повертається клас або класи, в залежності від кількості предметів у фото. За необхідності можна також повернути ймовірність належності знайденого об’єкта до класу, тобто впевненість передбачення. Для задачі локалізації модель також повертає обмежувальні коробки.

Існує велике різноманіття штучних нейронних мереж, кожен підвид заточений для своєї сфери застосування. В області комп’ютерного зору, зокрема для задач класифікації та локалізації, найбільше значення відіграли згорткові нейронні мережі (convolutional neural networks) вперше запропоновані Яном ЛеКуном [4]. Тому у даному дослідженні будуть розглянуті моделі нейронних мереж похідні від тої, що розробив Ян ЛеКун, так званої LeNet.

1.5 Висновок за розділом 1

Отже, в цьому розділі було визначено, що розпізнавання монет в першу чергу є задачею комп’ютерного зору. Крім того, було пояснено, що система розпізнавання має розв’язувати саме задачі класифікації та локалізації монет. Також було дано визначення загальним підходам до розпізнавання. Пояснено основну різнацю між двома видами методів розпізнавання.

2 МЕТОДИ РОЗПІЗНАВАННЯ ОБ’ЄКТІВ

У попередньому розділі було поставлено задачу розпізнавання монет та наведено, які галузі комп’ютерного зору досліджують її. У цьому ж розділі буде розглянуто конкретні алгоритми розроблені для розв’язання задачі розпізнавання об’єктів.

2.1 Класичні методи розпізнавання об’єктів

2.1.1 SIFT

Девід Лоу розробив алгоритм SIFT (Scale Invariant Feature Transoform) [2] здатний виділяти ключові точки та знаходити такий їх опис, що є інваріантним до масштабування. Це означає, що дескрипції ключових точок із одних регіонів двох зображень отримані в результаті роботи SIFT можуть бути зіставлені незалежно від того наскільки близько розміщена камера до об’єкту в одному й іншому фото.

Алгоритм складається з чотирьох етапів. Перший полягає у визначенні екстремуму простору шкал. На рисунку 2.1 із документації бібліотеки комп’ютерного зору OpenCV [5] видно, що кут при малому розмірі зображення й малому вікні (фрагменті зображення, що розглядається в поточний момент) стає лінією, якщо достатньо збільшити масштаб. Таким чином, використовувати одне вікно для різних масштабів не можна. Тому у роботі SIFT використовується так зване фільтрування масштабних просторів. Для зображень з різними значеннями параметру масштабування σ знаходять Лапласіан Гаусіану (Laplacian of Gaussian, LoG). LoG виступає як детектор груп пікселів, що мають якісь спільні ознаки (наприклад колір та контрастність) (blob detector) й визначає такі угрупування для різних значень σ. Наприклад на рисунку 2.2 із статті Девіда Лове [2] гауссівське ядро з малим значенням σ надає велике значення малому куту в той час, як гауссівське ядро з великим

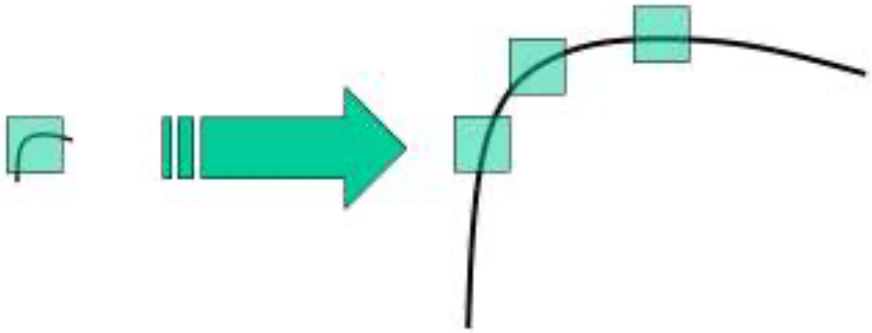


Рисунок 2.1

значенням σ надасть більшого значення великому куту. Таким чином, можна знайти локальний максимум за масштабом й простором, що дає список (x, y, σ). Це означає, що можна знайти потенційну ключову точку (x, y) в масштабі σ. Проте через те, що LoG досить важко обчислити, SIFT використовує його апроксимацію — різницю гаусів (Difference of Gaussians, DoG). DoG обчислюється як різниця Гаусового шуму зображень двох різних масштабів, наприклад σ та $k\sigma$. Цей процес виконується для різних октав зображення у Гаусовій Піраміді.

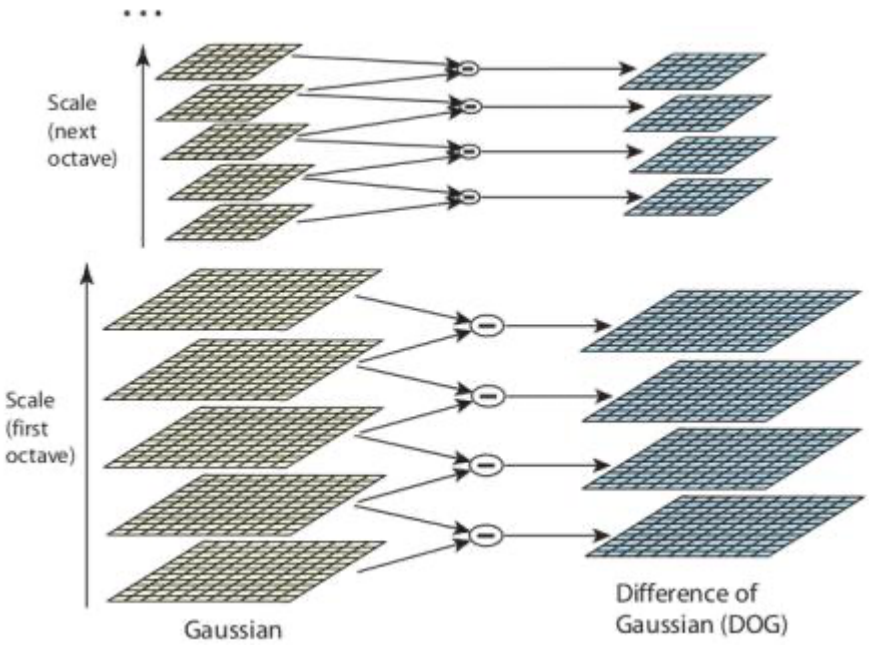


Рисунок 2.2

Коли всі DoG знайдені, знаходяться локальні екстремуми по масштабу й простору. Наприклад, один піксель зображення порівнюється із його вісьмома

сусідами з одного рівня й дев’ятьма пікселями з попереднього масштабу та ще дев’ятьма з наступного. Якщо знайдено локальний екстремум, то це потенційна ключова точка, це означає, що вона найкраще репрезентована у даному масштабі. Описаний процес проілюстровано на рисунку 2.3 із статті Девіда Лове [2].

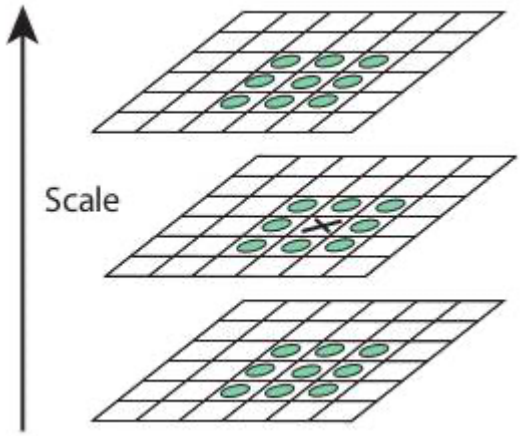


Рисунок 2.3

Другий етап полягає у локалізації ключових точок. Як тільки знайдені потенційні ключові точки, їх потрібно відсіяти, щоб отримати більш точний результат. Для того, щоб знайти більш точне розташування локального екстремуму використовується розклад в ряд Тейлора простору шкал. Якщо інтенсивність у цьому екстремумі менше за значення певного порогу (у статті SIFT використовується 0.03), то він відкидається. Позаяк DoG має сильну чутливість до кутів, вони також мають бути відкинуті. Для цього використовується матриця Гессіана (H) розміром 2x2, щоб обчислити основну кривизну (principal curveture). Відомо, що значення одного власного числа на краях більше за інші. Тож у SIFT використовується наступна функція: якщо поріг названий edgeThreshold (у статті SIFT має значення 10) більший за дане відношення, то ключова точка відкидається. Таким чином, всі ключові точки малого контрасту та на краях відсіюються.

На третьому етапі відбувається присвоєння орієнтації, кожній ключовій точці присвоюється орієнтація для того, що досягнути незалежність (інваріантність) від обертання. Залежно від масштабу обирається окіл ключової точки й у цьому регіоні обчислюється значення та напрям градієнту.

Створюється гістограма орієнтації з 36 стовпцями для охоплення 360 градусів. Обирається найвищий пік гістограми, а також всі інші, що вищі за 80% обираються для обчислення орієнтації. Таким чином, створюються точки з одним розволоженням та в одному масштабі, але з різною орієнтацією, що сприяє більш стабільному пошуку збігів точок на наступних етапах.

На четвертому етапі обчислюються дескрипції ключових точок (keypoint descriptions). Обирається окіл розміром 16x16 навколо ключової точки. Далі виділений регіон розділяється на 16 частин по 4x4. Для кожного підблоку обчислюється гістограма орієнтації з вісьмома стовпцями, тож загалом доступно 128 значень стовпців. Ця структура представляється у вигляді вектора для формування дескриптора ключових точок.

2.1.2 FAST

Існує чимало детекторів, що здатні виділяти ключові точки з достатньо високою якістю, проте не всі вони працюють швидко для застосування в реальному часі. Едвард Ростен та Том Друмонд запропонували детектор FAST (Features from Accelerated Segment Test) [6][7] задля вирішення цієї проблеми.

FAST виділяє ключові точки за наступним алгоритмом:

— вибрати піксель p із інтенсивністю I_p (на наступних кроках перевіряється чи є ця точка ключовою);

— вибрати значення порогу t;

— розглянути регіон із 16 пікселів навколо p, як показано на рисунку 2.4 із статті [6]; p називають кутом, якщо у його околі є n послідовних пікселів, які є яскравішими за $I_p + t$ або темнішими за $I_p - t$ (на картинці вище ці пікселі виділені білою пунктирною лінією), за значення n було обрано 12;

— швидкий тест для відкидання не кутових точок. Цей тест перевіряє тільки чотири пікселі на місцях 1, 9, 5 та 13 (спочатку перевіряють 1 та 9, а потім, якщо ці два проходять тест, перевіряють 5 та 13). Для того, щоб p був кутом хоча б три з чотирьох мають бути яскравішими за $I_p + t$ або темнішими за

I_p - t. Якщо ця умова виконана, можна застосувати тест сегменту й для інших

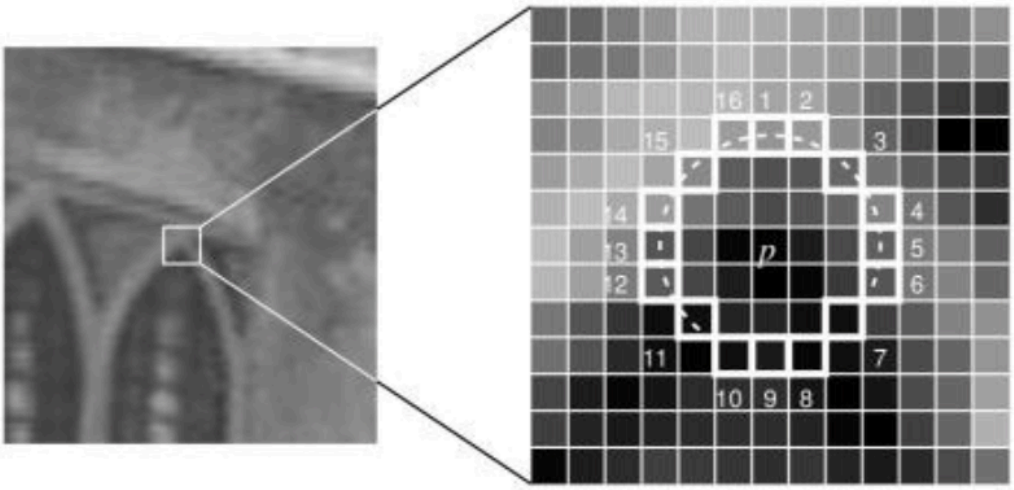


Рисунок 2.4

пікселів із околу.

Алгоритм у тому вигляді, який наведений вище працює ефективно, проте в ньому є недоліки:

— алгоритм не відкидає достатньо пікселів при $n < 12$;

— вибір пікселів є не оптимальним, бо ефективність алгоритму залежить від порядку питань (кут чи не кут) й розподілу вигляду кутів;

— результати швидкого тесту викидаються;

— можуть детектуватись декілька суміжних точок.

Перші три проблеми із списку вище вирішує застосування машинного навчання за наступним алгоритмом:

— збір зображень для навчання (бажано із сфери застосування алгоритму);

— пошук ключових точок методом FAST;

— для кожної знайденої ключової точки зберегти окіл із 16 пікселів як вектор. Зібрати околи всіх ключових точок із всіх зображень в одному векторі характеристик *P;*

— кожен із 16 пікселів (скажімо x) може перебувати в одному з наступних станів, де darker означає, що піксель темніший, similar — близький за яскравістю, a brighter — яскравіший, що підсумовано у наступній системі:

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \le I_p - t & \text{(darker)} \\ s, & I_p - t < I_{p\to x} < I_p + t & \text{(similar)} \\ b, & I_p + t \le I_{p\to x} & \text{(brighter)} \end{cases}$$

— відповідно до вищезгаданого розподілу пікселів по станам вектор характеристик P поділяють на три частини P_d , P_s , P_b ;

— вводимо нову булеву змінну K_p, яка дорівнює True, якщо p є кутом та False у протилежному випадку;

— використовуємо алгоритм ID3 (дерево прийняття рішень) для запиту по кожній підмножині P за допомогою змінної K_p для визначення справжнього класу. Алгоритм обирає такий x, що дає найбільше інформації (ентропії за K_p) про те чи є піксель кутом;

— процедура рекурсивно застосовується для кожної підмножини P допоки ентропія не буде рівна нулю;

— побудоване дерево рішень застосовується для детекції у інших зображеннях.

Детектування декількох суміжних точок із одного регіону вирішується за допомогою пошуку локальних максимумів (non-maximum suppression):

— потрібно обчислити функцію оцінки (score function) V для кожної детектованої ключової точки. Функція оцінки визначається як сума різниць по модулю між яскравостями p та шістнадцятьма пікселями з округу;

— із кожних двух суміжних точок залишається та, що має більше значення функції оцінки.

2.1.3 BRISK

Наступним кроком у розвитку детекторів-дескрипторів після FAST був BRISK (Binary Robust Invariant Scalable Keypoints) [8]. За основу детектора було взято FAST у поєднанні із бінарним дескриптором.

Як і детектор SIFT, детектор BRISK використовує простір масштабів. Пошук максимуму (найкращої ключової точки) відбувається у піраміді

зображень різних масштабів. В якості міри точності використовується оцінка s із детектора FAST.

За методом BRISK піраміда масштабних просторів складається із n окстав c_i та n інтра-октав d_i, для $i = \{0, 1, \ldots, n\text{-}1\}$ й за звичай $n = 4$. Октави формуються послідовною напів-вибіркою (half-sampling) із оригінального зображення, що відповідає c_0. Кожна інтра-октава d_i розташована між октавами c_i та c_{i+1} як зображено на рисунку 2.5 із статті [8].

Першу інтра-октаву d_0 отримують шляхом зменшення вхідного зображення c_0 в 1.5 раз (half sampling), всі наступні інтра-октави отримують послідовного зменшення зображення. Таким чином, якщо t позначає масштаб, то $t(c_i) = 2^i$ та $t(c_i) = 2^i \cdot 1.5$.

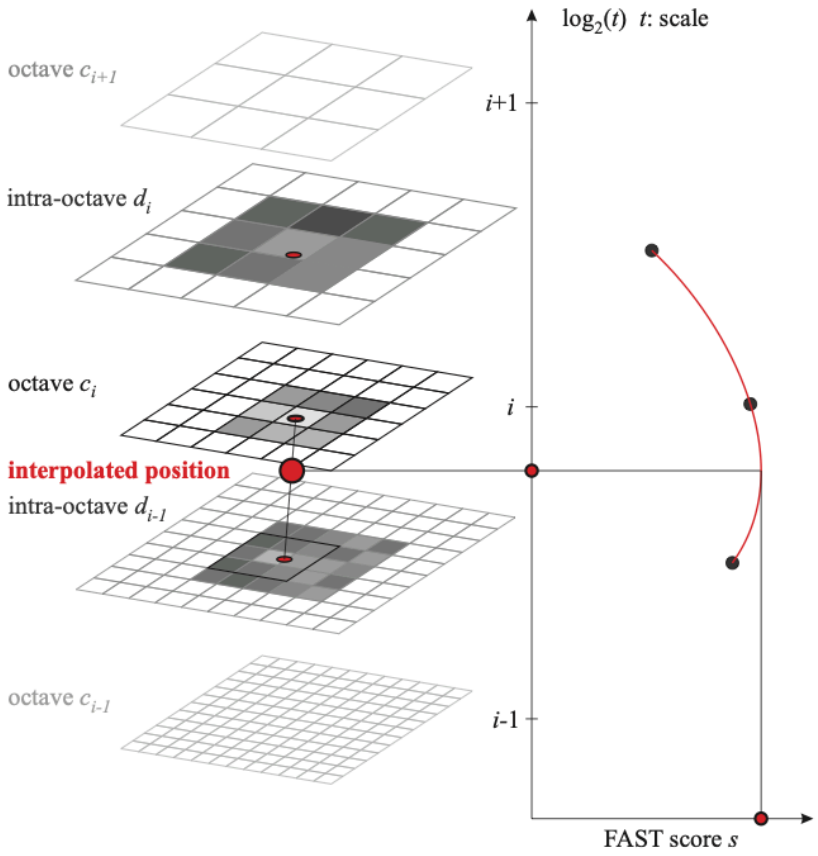


Рисунок 2.5

Спочатку детектор FAST застосовують окремо до кожної октави та інтра-октави із порогом T для того, щоб знайти регіони в яких потенційно можуть бути вдалі ключові точки. Далі до точок із виділених регіонів застосовують

процедуру пошуку локальних максимумів у піраміді зображень. По-перше, точка, яку розглядають має задовільнити умову максимальності (залишається та точка, що має найбільше значення функції оцінки) відносно своїх восьми сусідніх FAST оцінок *s* на своєму рівні. Оцінка *s* визначається як максимальний поріг за якого точка ще вважається кутом. По-друге, оцінки на рівні знизу й згори має бути також меншою. Перевірка відбувається у однакових за розміром квадратах, на рівні де ймовірно знаходиться максимум довжина сторони — 2 пікселі. Оскільки сусідні рівні й відповідно їх оцінки *s* представлені різною дискретизацією, то до країв квадрату застосовується інтерполяція.

Опис ключової точки отриманої детектором BRISK складається із бінарного рядка шляхом конкатинації результатів тесту порівняння яскравості. Для того, щоб забезпечити інваріантність дескрипцій до поворотів BRISK знаходить характеристичний напрямок кожної ключової точки.

Головною концепцією у дескрипторі BRISK є спосіб вибірки із сусідів ключової точки у розгляді. Для формування вибірки обираються *N* місць рівновіддалених на колах концентричних до ключової точки, що знаходиться по центу (рис. 2.6).

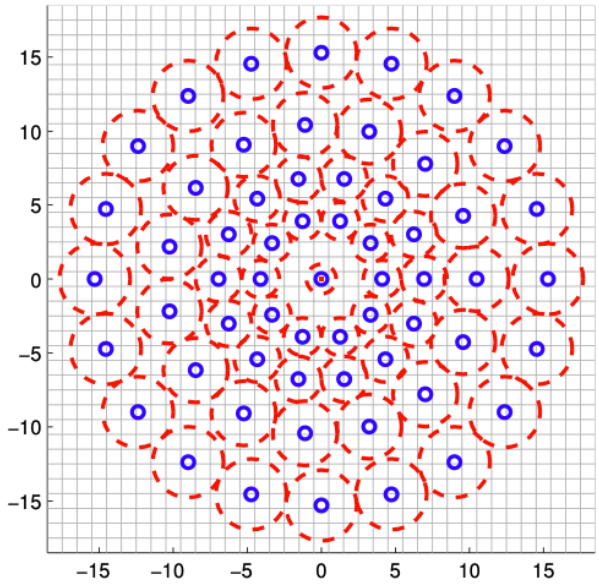


Рисунок 2.6

2.2 Методи розпізнавання об’єктів засновані на нейронних мережах

Дослідники вже багато років працюють над задачею розпізнавання об’єктів. Останнє десятиріччя основну увагу захопили глибинні нейронні мережі. Як показало дослідження Алекса Крижевського, Іллі Суцкевера та Джефрі Гінтона [3], нейронні мережі з великою кількістю прихованих шарів (рис. 2.7) здатні розпізнавати об’єкти з точністю, яку класичні методи комп’ютерного бачення не могли досягнути.

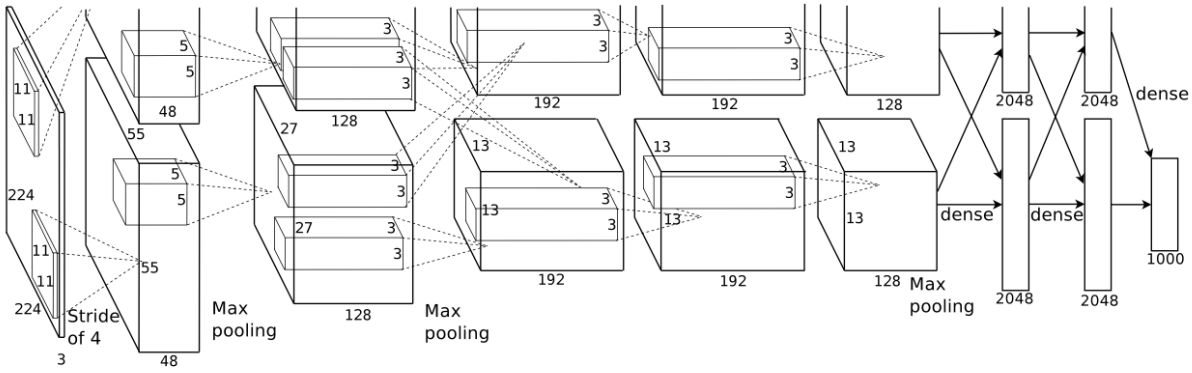


Рисунок 2.7 — Архітектура нейронної мережі розробленої Алексом Крижевським, Іллею Суцкевером та Джефрі Гінтоном

В процесі розвитку нейронних мереж в галузі розпізнавання об’єктів моделі поділилися на дві групи: одноетапні та двоетапні. На рисунку 2.8 із статті YOLOv4 [10] зображена різниця у архітектурі двоетапних моделей та одноетапних.

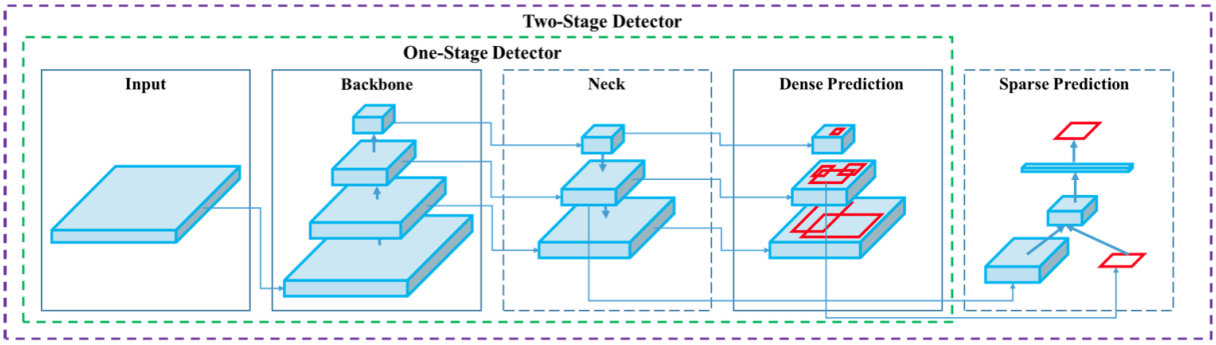


Рисунок 2.8

2.2.1 Двоетапні моделі розпізнавання об’єктів

2.2.1.1 RCNN

Першою моделлю в лінійці двоетапних моделей була RCNN запропонована Росом Гіршіком [11]. Новизною дослідження було (1) те, що згорткові нейронні мережі можна застосовувати до виділених регіонів (region proposals) в яких можуть бути об’єкти та (2) доведення того, що попереднє навчання (pretraining) на допоміжній задачі та наступне підлаштування (fine-tunning) до власної задачі може покращити роботу моделі.

Роботу моделі можна розкласти на три складові:

— виділення зі вхідного зображення методом вибіркового пошуку (selective search) порядку 2000 регіонів, які потенційно можуть містити об’єкти;

— обчислення характеристик (features) регіонів за допомогою згорткової нейронної мережі;

— класифікація регіонів методом опорних векторів з лінійним ядром (linear SVM). На рисунку 2.9 із статті [11] зображена схема роботи R-CNN.

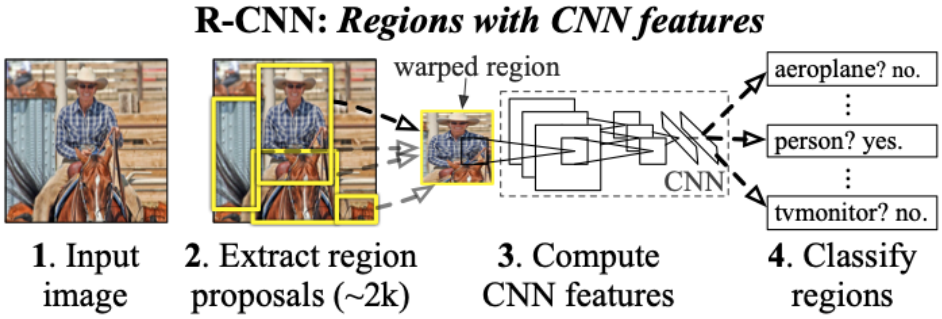


Рисунок 2.9

2.2.1.2 Fast RCNN

Попри свою інноваційність RCNN не могла працювати в реальному часі та її тренування займало багато часу. Вузьким місцем алгоритму був етап виділення регіонів. Для подолання цих недоліків Рос Гіршік запропонував нову версію моделі — Fast RCNN [12]. Замість регіонів на вхід згортковій мережі

подається ціле зображення й в результаті виходить мапа характеристик (feature map). На рисунку 2.10 зображена схема роботи Fast RCNN взята із статті [12].

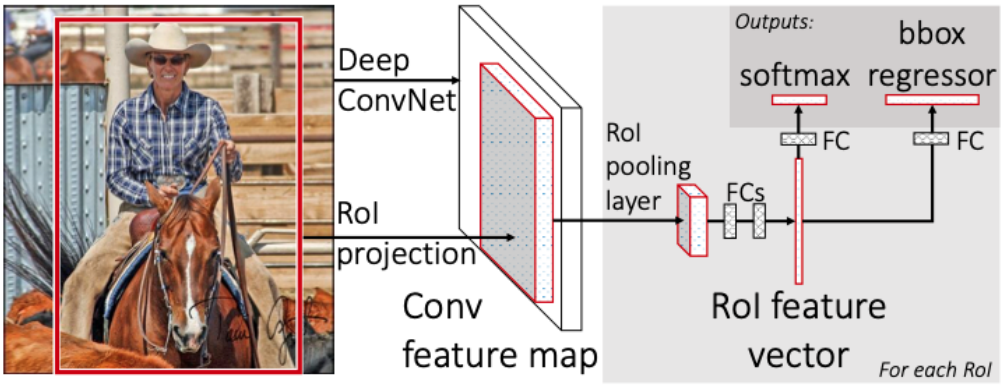


Рисунок 2.10

2.2.1.1 Faster RCNN

Наступна версія моделі — Faster RCNN [13] стала ще швидшою та практично могла досягнути роботи в реальному часі. Головною зміною в новій моделі було застосування окремої згорткової нейронної мережі для пропонування регіонів (Region Proposal Network, RPN) замість вибіркового пошуку. На рисунку 2.11 зображена схема роботи Faster RCNN із статті [13].

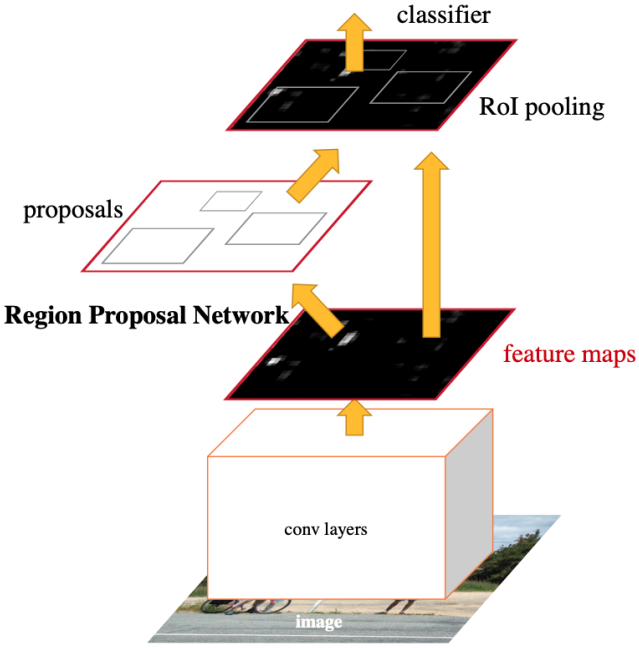


Рисунок 2.11

2.2.2 Одноетапні моделі розпізнавання об’єктів

2.2.2.1 YOLO

Алгоритм YOLO [10, 14, 15, 16] започаткував всю гілку одноетапних детекторів. Замість того, щоб спочатку одним методом виділяти регіон зображення у якому може бути об’єкт, а потім нейронною мережею локалізовувати об’єкт, YOLO визначає обмежувальні коробки та ймовірності класу лише за один крок. На рисунках 2.12 та 2.13 зображені схема роботи алгоритму та архітектура моделі відповідно, взяті із статті [14].

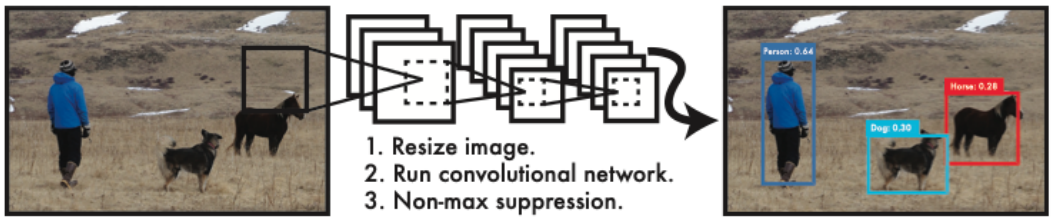


Рисунок 2.12

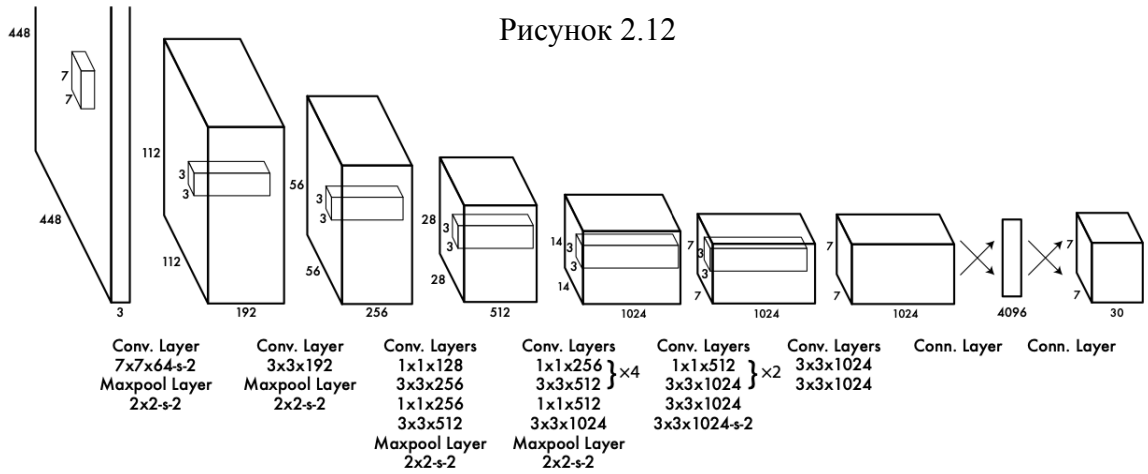


Рисунок 2.13

2.3 Висновки по розділу 2

В даному розділі було наведено методи розпізнавання об’єктів, які можна використати для побудови системи розпізнавання монет. Для того, щоб обрати найкращі алгоритми у наступному розділі буде проведено їх порівняльний аналіз.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ РЕЗУЛЬТАТИ

3.1 Підбір набору даних

Для того, щоб використовувати нейронну мережу для детекції об’єктів її спершу треба навчити, а для цього потрібно мати множину прикладів. Оскільки розпізнавання об’єктів це задача комп’ютерного зору, то даними мають бути зображення із відповідними анотаціями (розмітками). Для детекції об’єктів розміткою є обмежувальні коробки, що окреслюють об’єкти, в нашому випадку монети.

Розмітка зображень є трудомісткою задачею, що вимагає багато часу навіть для випадку, коли об’єкт лише один — монета. Більш того, для сучасних моделей розпізнавання об’єктів, таких як YOLO, кількість навчальних прикладів має бути великою задля досягнення достатньої точності. Проте цю проблему можна обійти, адже в мережі Інтернет є чимало відкритих джерел розмічених даних.

В результаті проведення аналіу доступних наборів даних з фото монет, Open Images Dataset (OID) [17, 18] виявився найбільш придатним. Він містить до 9 мільйонів зображень різних об’єктів та 16 мільйонів обмежувальних коробок для 600 класів. Зміст зображень досить складний, часто на них присутні декілька об’єктів. Анотації містять не лише обмежувальні коробки, а й маски для сегментації, що можна використати для наступних досліджень. Набір даних розділений на тренувальну, валідаційну та тестувальну множини.

Для проведення даного дослідження було завантажено тільки зображення монет, для тренування моделі 1375 зображень, а для тестування 53. Валідаційну множину було об’єднано з навчальною. Оскільки формат анотацій OID не є придатним для тренування YOLO довелося додатково застосувати програмний код для переформатування розмітки.

Для навчання сучасних глибинних нейронних мереж необхідно мати потужні обчислювальні ресурси, зокрема відеокарти, й модель YOLO не виключення. Компанія Google створила сервіс Colab, який дає можливість розробляти програмний код в інтерактивному середовищі з використанням

хмарних обчислювальних ресурсів. У Google Colab є можливість під'єднати відеокарту Tesla K80 з пам’яттю в 12 GB чого вдосталь для навчання YOLO на наборі даних в декілька тисяч зображень.

3.2 Підготовка моделі до тренування

Перш ніж почати навчання YOLO потрібно налаштувати спеціальний конфігураційний файл в якому задані параметри тренування та архітектуру мережі. Конфігурацію необхідно змінювати для того, щоб вона відповідала задачі й даним, які подаються на вхід. Серед важливих параметрів, які довелося змінити були:

— кількість прикладів у одній групі (batch), 64;

— subdivisions, 16;

— max_batches, 6000, бо для дослідження маємо лише один клас.

3.3 Опис процесу тренування

Навчання моделі вимагає чимало часу та обчислювальних ресурсів. Більш того, для досягнення задовільної точності, необхідно мати велику кількість розмічених даних. Цей процес можна спростити за допомогою трансферного навчання, суть якого полягає перенесенні результатів отриманих моделлю на одних даних до інших. Окрім OID у відкритому доступі є такі набори даних як: ImageNet, COCO, Pascal VOC та ін. Наведені набори даних є стандартами у сфері комп’ютерного зору й практично всі дослідження у галузі їх застосовували. Таким чином, існує велика кількість моделей вже навчених на цих даних. Ваги отримані в результаті тренування таких моделей можна використати для навчання власних моделей пристосованих для конкретних задач. Для трансферного навчання необхідно тільки брати виходи не з останніх шарів нейронної мережі для того, щоб передати лише загальні характеристики об’єктів такі як форми, контрастність, кути. Автори останньої версії YOLO (YOLOv4), яка застосовується у цьому дослідженні, у своєму репозиторії

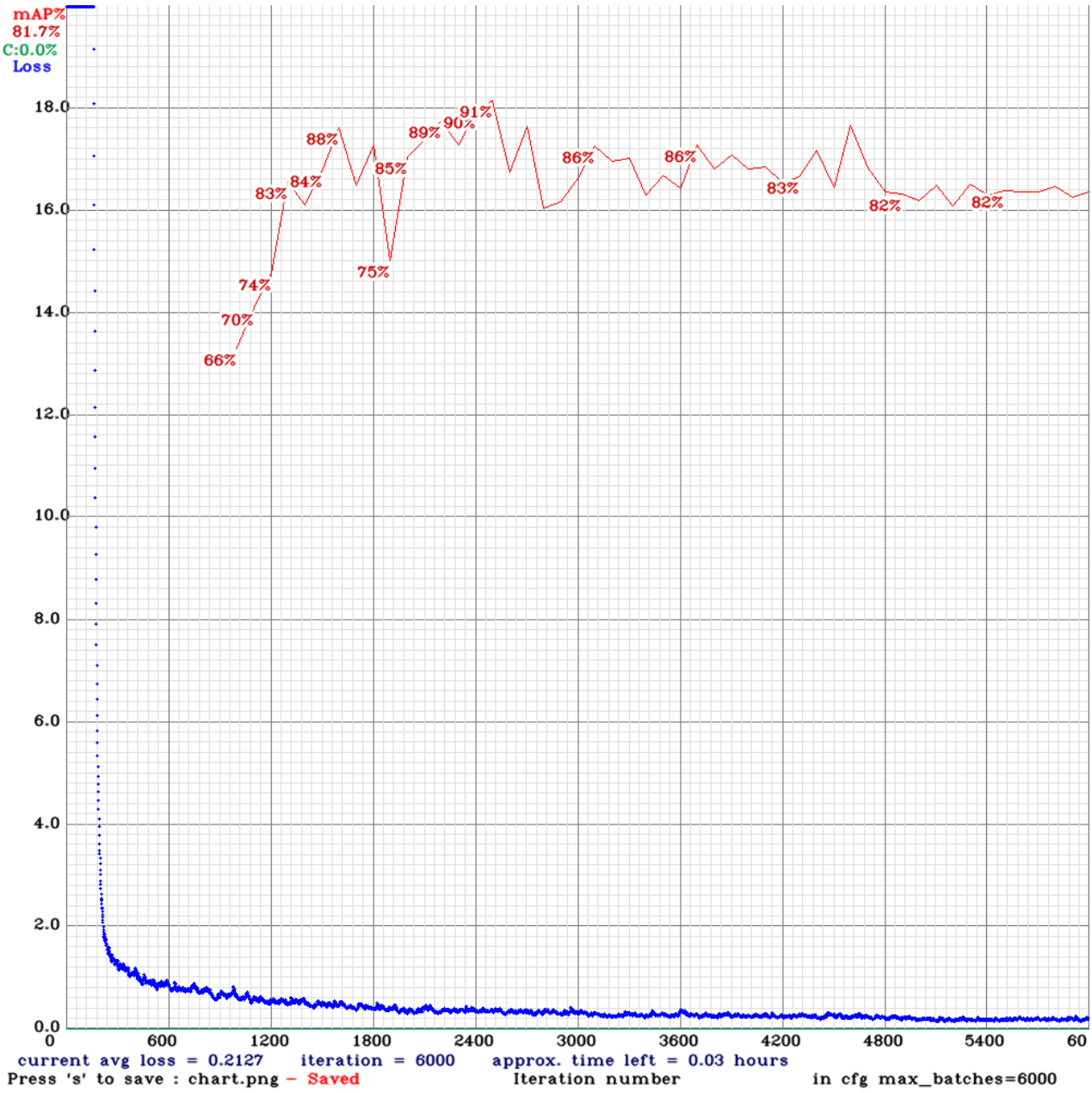


Рисунок 3.1

виклали ваги моделі попередньо натренованої на інших даних. Таким чином, детектор монет вдалося довчити й при цьому уникнути довгого тренування й збору великого набору зображень монет.

Для того, щоб почати тренування YOLOv4 достатньо мати згаданий вище конфігураційний файл, файл з описом розташування даних та ваги з попередньо натренованої моделі. Протягом тренування у консолі зображуються основні метрики: функція втрат, коефіцієнт Жаккара (Intersection Over Union, IoU) та середня точність.

3.4 Результати навчання моделі

На рисунку 3.1 зображено криву навчання та метрики, що оцінюють якість моделі. Після тренування моделі були отримані ваги, які можна застосовувати для детектування монет у фото та відео. Для того, щоб застосовувати ці ваги потрібно використати спеціальні бібліотеки комп'ютерного зору, наприклад PyTorch, Tensorflow, OpenVINO, OpenCV та інші. Для цього дослідження найкраще підходить бібліотека OpenCV тому, що вона містить в собі багато інших корисних функцій для комп'ютерного зору, зокрема готові алгоритми детектори-дескриптори, які необхідні для другого етапу роботи системи розпізнавання монет — класифікації.

3.5 Вибір детектора та дескриптора

Різні детектори-дескриптори заточені під різні задачі й для того, щоб досягнути високих результатів необхідно обрати саме той, що найкраще підходить під задачу розпізнавання монет. Для цього потрібно провести порівняльний аналіз алгоритмів.

У даному дослідженні було розглянуто такі детектори-дескриптори: ORB [19], BRISK, AKAZE [20], SIFT, RootSIFT [21]; детектор FAST у поєднанні з дескриптором ORB, BRISK, SIFT, RootSIFT, FREAK; детектор ORB у поєднанні з дескриптором BRISK, FREAK [22]; детектор BRISK у поєднанні з дескриптором ORB, SIFT, RootSIFT, FREAK; детектор STAR у поєднанні з дескриптором ORB, BRISK, SIFT, RootSIFT, FREAK. Відповідно до статті [23] із сучасних дескрипторів найефективнішими є BRISK та ORB.

Порівняння алгоритмів було виконано на наступними показниками: кількість ключових точок знайдених у еталоні та у зображенні з певним просторовим перетворенням, кількість знайдених збігів точок, кількість відсічених помилкових точок, кількість точок, що залишились після процедури відсікання, час знаходження ключових точок для першого й другого зображень,

час знаходження збігів, час відсікання помилок, загальний час аналізу зображень. При цьому найголовнішими показниками є кількість точок, що залишились після процедури відсікання та загальний час аналізу зображень, адже вони є визначними для розуміння точності та ефективності детектора та дескриптора.

Для того, щоб провести якісний порівняльний аналіз алгоритмів необхідно створити набір зображень монет з різноманітними просторовими перетвореннями. На рисунку 3.2 зображено множину тестових зображень на яких відбулося випробовування алгоритмів. Зліва направо зображені такі просторові перетворення: фото монети без просторових перетворень, нахил камери вниз, нахил камери згори, нахил праворуч, нахил ліворуч.

Кожне тестове зображення попарно порівнювалось із зображеннями еталонами. За результатами порівняння найкращими виявились поєднання на основі ORB, BRISK, FREAK та FAST. Найкращі поєднання детекторів-дескрипторів було окремо проаналізовано, результати зведені у таблиці 3.1.

Рисунок 3.2

Таблиця 3.1 — Оцінка комбінацій детекторів та дескрипторів

Детектор_Дескриптор	Оцінка	Загальний час роботи
ORB	20	0.755
BRISK	19	0.094
FAST_ORB	20	0.127
FAST_BRISK	18	0.208
FAST_FREAK	18	0.211
ORB_BRISK	17	0.654
ORB_FREAK	13	0.172
BRISK_ORB	19	0.145
BRISK_FREAK	17	0.182

Оцінка у таблиці 3.1 обчислювалась наступним чином. Кожне тестове фото порівнювалось із кожним із чотирьох еталонів, якщо алгоритм правильно зіставляв тестове зображення із своїм еталоном, то така комбінація детектора та дескриптора отримувала один бал. Оскільки тестова множина мала всього 20 зображень, миксимальний бал, який міг отримати алгоритм був 20. Відповідно до обраних метрик найкращим поєднанням детектора та дескриптора є BRISK.

3.6 Підбір параметрів для BRISK

Для того, щоб отримати якнайкращі результати роботи алгортиму варто підібрати його параметри. Найважливішими параметрами, які необхідно налаштувати для BRISK є поріг (thresh) та кількість октав (octaves). Підбір параметрів було проведено методом решіткового пошуку. Множиною значень для параметра thresh було обрано [10, 15, 30], для параметра octaves — [2, 3, 4]. Після проведення решіткового пошуку було виявлено, що компромісним набором параметрів для BRISK є thresh=30 та octaves=3, тобто значення, що у функції для BRISK у OpenCV обрані за замовчуванням.

3.7 Побудова програмного коду для застосування детекторів-дескрипторів

Для того, щоб поєднати роботу детектора із дескриптором необхідно було розробити програмний код-обгортку. Логіку роботи дескриптора було об’єднано у класі Classifier який приймає на вхід зображення монети, а на вихід повертає клас до якого ймовірніше за все належить дана монета. Коли на вхід подається зображення з нього виділяються ключові точки та ознаки, що його характеризують. Після цього відбувається послідовне порівняння з еталонними зображеннями, що були обрані за ідеальних представників кожного виду монети. Для того, щоб кожен раз при роботі алгоритму не обчислювати ключові точки й ознаки еталонних зображень, їх було збережено. Таким чином, перед порівнянням опис еталонів зчитується із файлів. Далі за допомогою спеціального методу із бібліотеки OpenCV знаходяться збіги між дескрипціями вхідного зображення та еталонного. Опісля за допомогою алгоритму GMS [24] відхиляються зайві помилкові точки. Клас визначається в залежності від того, який еталон отримав найбільше збігів.

Перед застосуванням дескриптору зображення потрібно обробити: зменшити розмір та застосувати фільтри. Зменшення розміру дозволить пришвидшити роботу дескриптору, бо прибере зайву інформацію із зображення. В якості фільтру було застосовано Unsharp, який підвищує контрастність фото, що в результаті допомагає детектувати більше ключових точок.

Для поєднання роботи детектора та дескриптора було створено окремий скрипт. Загалом послідовність роботи всього алгоритму така:

— на вхід детектору подається фото або кадр із відео;

— детектор обробляє зображення й повертає список обмежувальних коробок;

— з вхідного фото чи кадру вирізаються фрагменти, що відповідають коробкам, ці фрагменти мають містити монети;

— вирізані частини почергово передаються класифікатору й той порівнює їх з еталонами;

— класифікатор повертає назву передбаченого виду монети;

— обмежувальні коробки та назви класів зображуються на фото (відео).

Загальна схема роботи алгоритму зображена на рисунку 3.3 нижче.

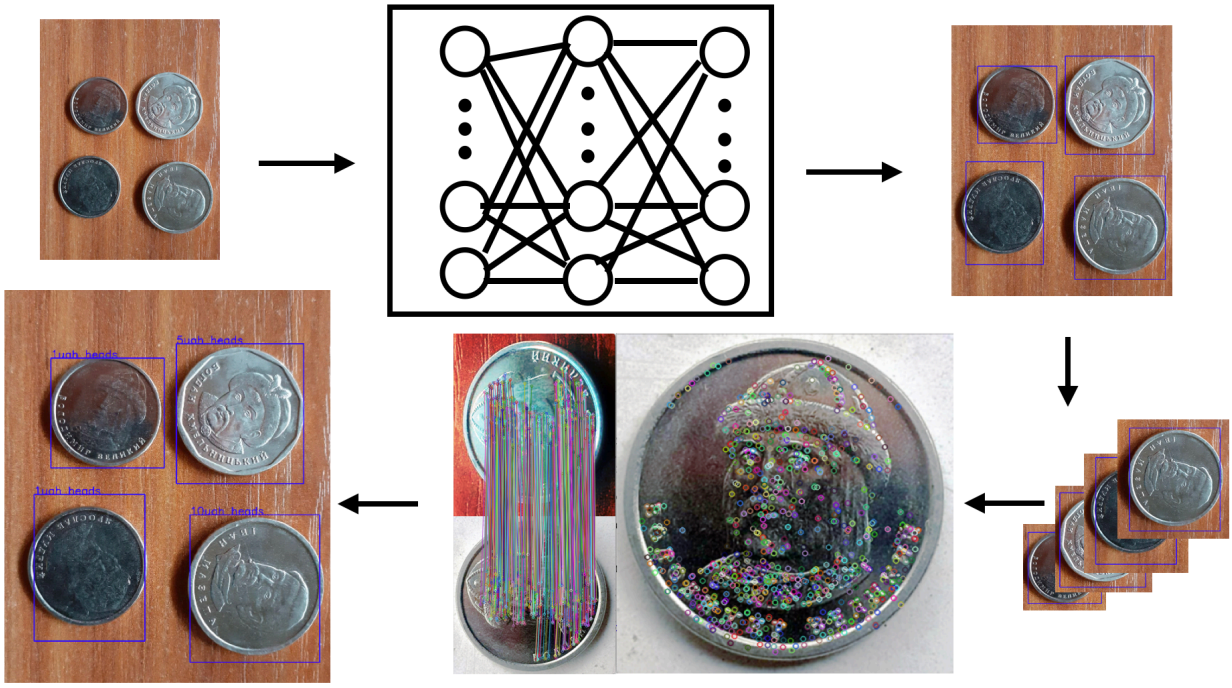


Рисунок 3.3

3.8 Висновки по розділу 3

По-перше, дослідження показало, що YOLOv4 добре підходить для задачі детекції монет. Модель може з достатньо високою точністю локалізувати монети. По-друге, шляхом порівняльного аналізу було визначено, що для другого етапу системи найкращим детектором та дескриптором є BRISK.

4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, розробленого для вирішення задачі розпізнавання монет.

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 проведення техніко-економічного аналізу розробки системи розпізнавання монет. 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 збору та обробки даних, навчання нейронної мережі та інтеграції з додатковими методами.

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 монет. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування;

F_2 – вибір фреймворку комп’ютерного зору;

F_3 – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1: а) Python б) C++

Функція F_2: а) OpenCV; б) Tensorflow

Функція F_3: а) Google Colab; б) Jupiter Notebook.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 5.1).

Морфологічна карта відображає множину всіх можливих варіанти основних функцій.

На основі цієї карти будуємо позитивно-негативну матрицю варіантів основних функцій (Таб.5.2). 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 швидкості вивчення, простоті використання та наявності стандартних бібліотек для обчислення. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F_2:

Обидва варіанти можна використовувати в розробці.

Функція F_3:

Віддаємо перевагу варіанту А оскільки Google Colab надає можлвивість використовувати додаткові хмарні обчислювальні ресурси.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а – F_2а – F_3а

F_1а – F_2*б* – F_3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

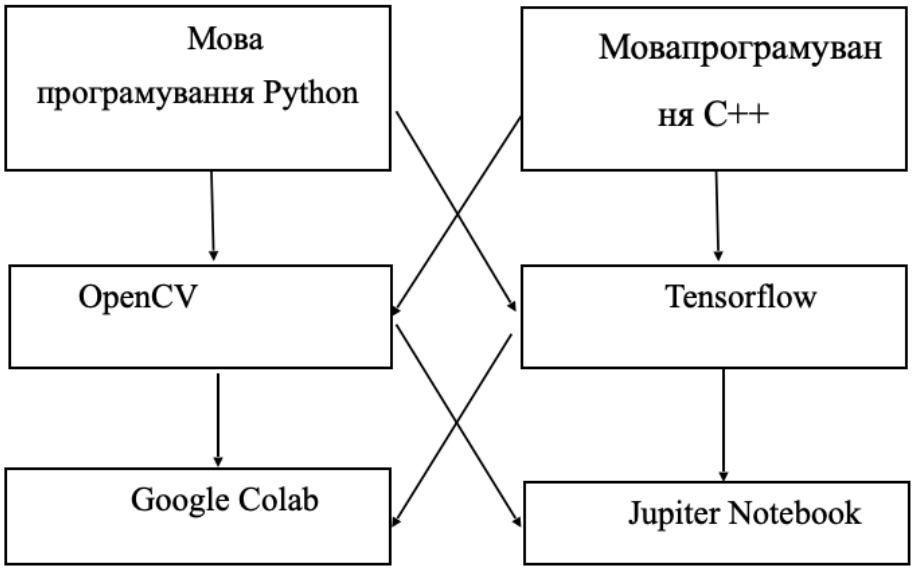


Рисунок 5.1 – Морфологічна карта

Таблиця 5.2 — Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Швидка розробка програми, доступність бібліотек, кросплатформеність	Низька швидкість роботи, особливо, якщо потрібно обробляти велику кількість даних
	Б	Код швидко виконується	Іде багато часу на розробку програми
F_2	*А*	Надійно працює з складними проектами	Не підтримується багатьма мовами
	Б	Надійність	Додатковий час на інсталяцію та вивчення
F_3	*А*	Підтримується багатьма мовами програмування, легко запускається на будь-якому сервері	Відсутня можливість роботи без інтернету
	Б	Багато інструментів, безпечна	Підтримує одночасно лише одну мову програмування

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 для обчислень та збереження даних;

— *X3* – час навчання даних;

— *X4* – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 5.4.

Таблиця 5.4 - Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	10000	14000	19000
Об’єм пам’яті	X2	Мб	420	128	64
Час попередньої обробки даних	X3	мс	4	3	2
Потенційний об’єм програмного коду	X4	кількість рядків коду	4000	2500	1000

За даними таблиці 5.3 будуються графічні характеристики параметрів – рис. 5.2 – рис. 5.5.

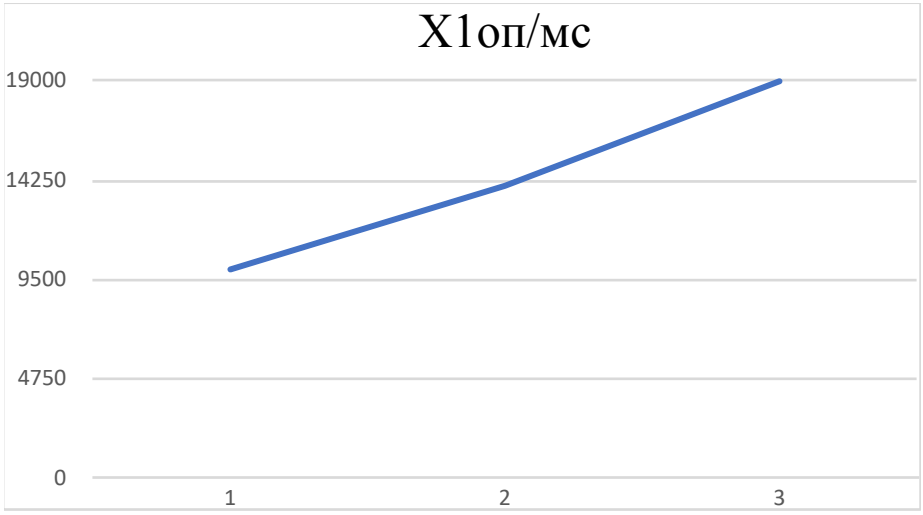


Рисунок 5.2 — X1, швидкодія мови програмування

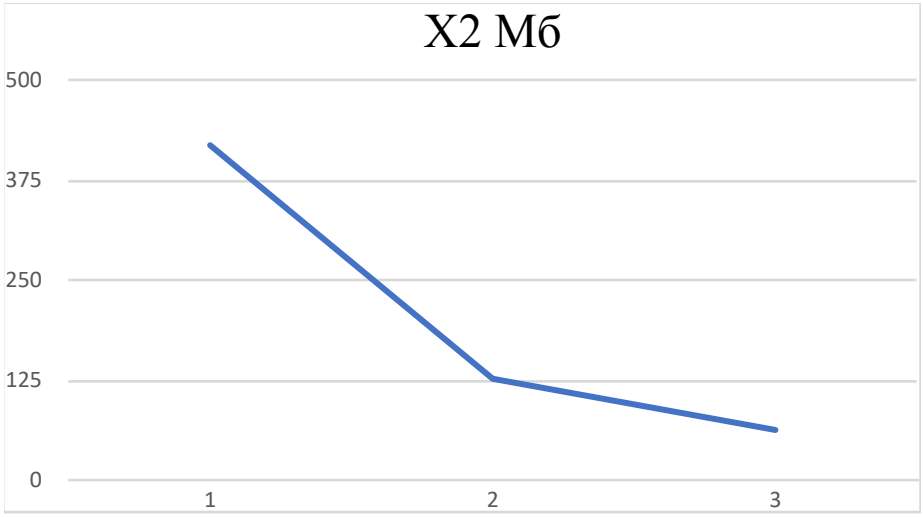


Рисунок 5.3 — X2, об’єм пам’яті

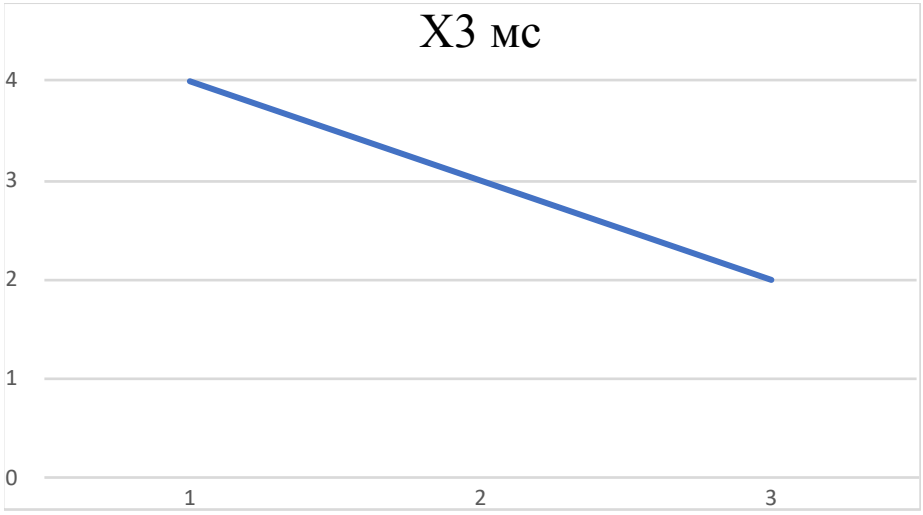


Рисунок 5.4 — X3, час попередньої обробки даних

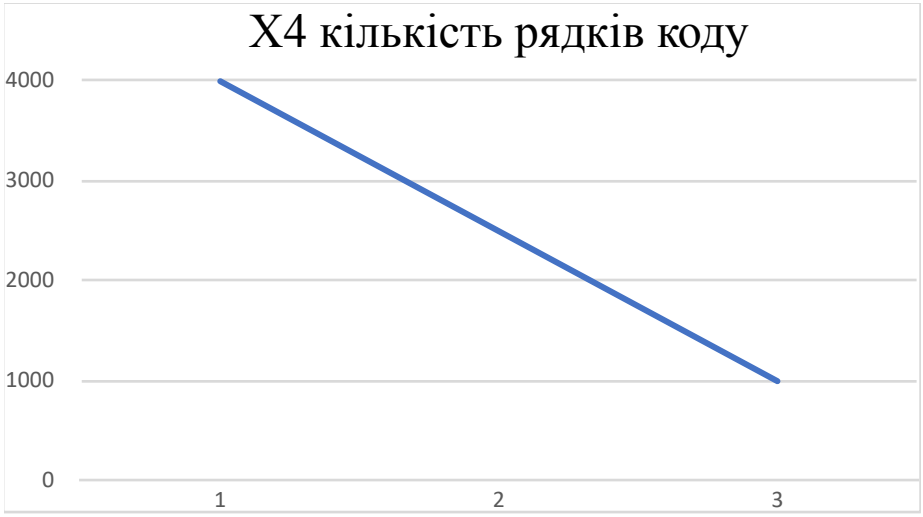


Рисунок 5.5 — X4, потенційний об’єм програмного коду

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 5.4.

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

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

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

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 24{,}5$$

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

$$\Delta_i = R_i - T.$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 211.$$

Таблиця 5.4 — Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення 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

Порахуємо коефіцієнт узгодженості:

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

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 5.5.

Таблиця 5.5 — 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:

$$a_{ij} = \begin{cases} 1.5 \; при \; X_i > X_j \\ 1.0 \; при \; X_i = X_j \\ 0.5 \; при \; X_i < X_j \end{cases}.$$

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

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

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

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

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:

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

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

Як видно з таблиці 5.6, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 5.6 — Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{вi}$	b_i^1	$K_{вi}^1$	b_i^2	$K_{вi}^2$
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 5.7):

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

де n – кількість параметрів;

K_{Bi}– коефіцієнт вагомості *i*–го параметра;

B_i – оцінка *i*–го параметра в балах.

Таблиця 5.7 — Розрахунок показників рівня якості варіантів

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	10000	6	0,25	1,5
F2	А	X2	64	7	0,20	1,4
	Б	X2	128	4	0,11	0,4
F3	А	X3	1000	8	0,2	1,6

За даними з таблиці 5.7 за формулою:

$$K_K = K_{TY}[F_{1k}] + K_{TY}[F_{2k}] + \ldots + K_{TY}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1,5 + 1,4 + 1,6 = 4,5,$$

$$K_{K2} = 1,5 + 0,44 + 1,6 = 3.$$

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

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М},$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

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

K_M – 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 $К_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

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

$$Т_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \text{ людино-днів.}$$

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

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

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

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 25000 грн., один аналітик в області даних з окладом 20000. Визначимо середню зарплату за годину за формулою:

$$СЧ = \frac{М}{T_m \cdot t} грн.,$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$СЧ = \frac{25000 + 25000 + 18000}{3 \cdot 21 \cdot 8} = 134{,}92\ грн\ .$$

Тоді, розрахуємо заробітну плату за формулою:

$$СЗП = C_ч \cdot T_i \cdot КД\ ,$$

де $С_ч$– величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

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

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 134.92 \cdot 1328.64 \cdot 1.2 = 215112.13$ грн.

II. $С_{ЗП} = 134.92 \cdot 1345.52 \cdot 1.2 = 217845.07$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 215112.13 \cdot 0.22 = 47324.67$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 217845.07 \cdot 0.22 = 47925.91$ грн.

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

Так як одна ЕОМ обслуговує одного програміста з окладом 25000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 25000 \cdot 0{,}2 = 60000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 60000 \cdot (1 + 0.2) = 72000 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 72000 \cdot 0{,}22 = 15840 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 30000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 30000 = 8625 \text{ грн.},$$

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:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 30000 \cdot 0.05 = 1725 \text{ грн.},$$

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:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1677.6 \cdot 0,2 \cdot 0.4 \cdot 3.51 = 417.07 \text{ грн.},$$

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:

$$С_Н = Ц_{ПР} \cdot 0.67 = 20000 \cdot 0,67 = 13400 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н,$$

$$С_{ЕКС} = 28800 + 6336 + 10589,76 + 5750 + 1150 + 417.07 + 13400 = 55853,07 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 55853,07 / 1677.6 = 33,29 \text{ грн/год.}$$

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:

$$С_М = С_{М-Г} \cdot Т,$$

I. $С_М = 33,29 \cdot 1328,64 = 44230,42 \text{ грн.}$

II. $C_М = 33,29 \cdot 1345.52 = 44792,36$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_Н = C_{ЗП} \cdot 0,67,$$

I. $C_Н = 215112,13 \cdot 0,67 = 144125,13$ грн.

II. $C_Н = 217845,07 \cdot 0,67 = 145956,2$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н,$$

I. $C_{ПП} = 215112,13 + 47324,67 + 44230,42 + 144125,13 = 450792,35$ грн.

II. $C_{ПП} = 217845,07 + 47925,91 + 44792,36 + 145956,2 = 456519.54$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{Кj} / C_{Фj},$$

$$К_{ТЕР1} = 4,5/ 450792,35 = 9.98 \cdot 10^{-6},$$

$$К_{ТЕР2} = 3/ 456519,54 = 6,57 \cdot 10^{-6}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 9.98 \cdot 10^{-6}$.

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 $К_{ТЕР} = 9.98 \cdot 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

— мова програмування – Python;

— використання моделей з великою ємністю;

— використання стандартного інтерфейсу візуалізації, швидкість розробки.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

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

Отже, в ході виконання даної дипломної роботи було побудовано систему здатну класифікувати та локалізувати монети у фото та відео. Роботи в реальному часі не вдалося досягнути, проте це не означає, що поєднанням класичних методів із нейронними мережами

даній дипломній роботі було розглянуто методи комп'ютерного зору для розпізнавання об'єктів. Розглянуто як класичні методи, так і більш сучасні засновані на нейронних мережах. Поставлено задачу розпізнавання монет та наведено шляхи до її вирішення.

У роботі було описано принципи роботи базових алгоритмів розпізнавання образів, які можна використати для даного дослідження. Методи було порівняно та обрано найкращі з них.

В ході проведення дослідження було побудовано програмний продукт. Описано процес збору даних, попередньої обробки даних, навчання моделі нейронної мережі, проведення порівняльного аналізу детекторів-дескрипторів та інтеграція компонентів системи.

Окрім того, проведено фінансово-економічний аналіз проєкту. Порівняно дві стратегії побудови програмного продукту та обрано найкращу з них.

Подальшими кроками для розвитку системи розпізнавання монет та цього дослідження можуть бути:

— порівняльний аналіз архітектур нейронних мереж для розпізнавання об'єктів;

— збір більшої кількості даних;

— очищення навчальної вибірки від помилок;

— включення до порівняльного аналізу детектрів-дескрипторів методів заснованих на нейронних мережах;

— оптимізація програмного коду;

— порівняльний аналіз методів відсікання помилок детекторами-дескрипторами.

ПЕРЕЛІК ПОСИЛАНЬ

1. Computer Vision: Algorithms and Applications. URL: http://szeliski.org/Book/ (дата звернення: 15.05.2021)
2. David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. January 5, 2004. URL: https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf
3. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. NIPS 2012. ImageNet Classification with Deep Convolutional Neural Networks. URL: https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
4. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner. Gradient-Based Learning Applied to Document Recognition. November, 1998. URL: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf
5. OpenCV. Introduction to SIFT (Scale-Invariant Feature Transform). URL: https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html (дата звернення: 15.05.2021)
6. Edward Rosten and Tom Drummond. Machine learning for high speed corner detection. 2006. URL: https://www.edwardrosten.com/work/rosten_2006_machine.pdf
7. Edward Rosten, Reid Porter, Tom Drummond. Faster and better: a machine learning approach to corner detection. 14 October, 2008. URL: https://arxiv.org/abs/0810.2434
8. S. Leutenegger, M. Chli and R. Y. Siegwart. BRISK: Binary Robust invariant scalable keypoints. November, 2011. URL: https://www.researchgate.net/publication/221110715_BRISK_Binary_Robust_invariant_scalable_keypoints
9. Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. 23 April 2020. URL: https://arxiv.org/abs/2004.10934

10. Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. 22 October 2014. URL: https://arxiv.org/pdf/1311.2524.pdf
11. Ross Girshick. Fast R-CNN. 27 September 2017. URL: https://arxiv.org/pdf/1504.08083.pdf
12. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 6 January, 2016. URL: https://arxiv.org/pdf/1506.01497.pdf
13. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. 8 June, 2015. URL: https://arxiv.org/abs/1506.02640
14. Joseph Redmon, Ali Farhadi. YOLO9000: Better, Faster, Stronger. 25 December 2016. URL: https://arxiv.org/abs/1612.08242
15. YOLOv3: An Incremental Improvement. URL: https://pjreddie.com/media/files/papers/YOLOv3.pdf (дата звернення: 15.05.2021)
16. Krasin I., Duerig T., Alldrin N., Ferrari V., Abu-El-Haija S., Kuznetsova A., Rom H., Uijlings J., Popov S., Kamali S., Malloci M., Pont-Tuset J., Veit A., Belongie S., Gomes V., Gupta A., Sun C., Chechik G., Cai D., Feng Z., Narayanan D., Murphy K. OpenImages: A public dataset for large-scale multi-label and multi-class image classification, 2017. URL: https://storage.googleapis.com/openimages/web/index.html
17. A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari. The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale. IJCV, 2020.
18. E. Rublee, V. Rabaud, K. Konolige and G. Bradski. ORB: An efficient alternative to SIFT or SURF. November, 2011. URL: https://www.researchgate.net/publication/221111151_ORB_an_efficient_alternative_to_SIFT_or_SURF

19. Pablo F Alcantarilla, Jesús Nuevo, and Adrien Bartoli. Fast explicit diffusion for accelerated features in nonlinear scale spaces. 2011. URL: http://www.bmva.org/bmvc/2013/Papers/paper0013/paper0013.pdf
20. R. Arandjelović and A. Zisserman. Three things everyone should know to improve object retrieval. 2012. URL: https://ieeexplore.ieee.org/document/6248018
21. A. Alahi, R. Ortiz and P. Vandergheynst, "FREAK: Fast Retina Keypoint," 2012 IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 510-517, doi: 10.1109/CVPR.2012.6247715.
22. S. A. K. Tareen and Z. Saleem, "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK," 2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), 2018, pp. 1-10, doi: 10.1109/ICOMET.2018.8346440.
23. J. Bian, W. Lin, Y. Matsushita, S. Yeung, T. Nguyen and M. Cheng, "GMS: Grid-Based Motion Statistics for Fast, Ultra-Robust Feature Correspondence," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 2828-2837, doi: 10.1109/CVPR.2017.302.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

Модуль inference.py

```
import argparse
from time import time
import cv2 as cv
import os
from os.path import join, isfile
from src.classifier import Classifier
from src.utils import (DATA_PATH, TEST_IMG_PATH, OUTPUT_PATH,
                       DESCRIPTOR_NAMES, YOLO_CONFIDENCE,
NMS_THRESHOLD)
from yolo import Yolo
from src.preprocessing import prepare_image, extract_detection,
IMG_WIDTH_SIZE


def _parse_args():
    arg_parse = argparse.ArgumentParser()
    arg_parse.add_argument("-i", "--image", required=False,
                           help="Image to perform inference on.")
    arg_parse.add_argument("-v", "--video", required=False,
                           help="Video to perform inference on.")
    arg_parse.add_argument("-o", "--output", required=False,
                           help="The name of the output video.")
    # TODO: There should be two arguments: detector and descriptor.
    arg_parse.add_argument("-d", "--descriptor", default="ORB",#,
choices=DESCRIPTOR_NAMES,
```

```
                    help="The name of the descriptor used for classification.")
    arg_parse.add_argument("-r", "--reject-outliers", default="yes", choices=["yes",
"no"],
                    help="Whether to apply outlier rejection to matches.")
    # TODO: add debug option for more detailed visualization
    return arg_parse



if __name__ == '__main__':
    parser = _parse_args()
    args = parser.parse_args()


    if args.image is None and args.video is None:
        raise ValueError(
            "Neither image nor video arguments are provided! Please,"
            " specify either the path to an image or video or both."
        )
    yolo = Yolo(confidence=YOLO_CONFIDENCE,
nms_threshold=NMS_THRESHOLD)
    yolo.load_model()
    if args.image:
        if not isfile(args.image):
            raise ValueError("The provided path to an image does not exist! "
                             "Please, provide a path relative to the project:) "
                             "The directory with test images is test_images/")
        image = cv.imread(args.image)
        prepared_image = prepare_image(image.copy(), resize=False)
```

```
        img_for_viz = image.copy()
        yolo.load_data(prepared_image)
        layer_outputs = yolo.detect()
        bboxes = yolo.process_outputs(layer_outputs)
        # TODO: try crops = extract_detection(prepared_image, bboxes)
        crops = extract_detection(image, bboxes)
        clf = Classifier(args.descriptor, True if args.reject_outliers == "yes" else False)
        for bbox, crop in zip(bboxes, crops):
            if crop.shape[0] < IMG_WIDTH_SIZE:
                print("Some coins are too small to classify"
                      "Please, try to zoom closer to the coins")
            predicted_class = clf.classify(crop)
            x, y, w, h = bbox
            cv.rectangle(img_for_viz, (x, y), (x + w, y + h), (255, 0, 0), 2)
            cv.putText(img_for_viz, predicted_class, (x, y),
                       fontScale=1, fontFace=cv.FONT_HERSHEY_SIMPLEX, color=(255,
0, 0), thickness=2)
        cv.imshow("Detections", img_for_viz)
        cv.waitKey(0)

    if args.video:
        if args.video == "real-time":
            cap = cv.VideoCapture(0)
        else:
            cap = cv.VideoCapture(join(DATA_PATH, args.video))

        width, height = None, None
```

```
writer = None
prev_time = 0

if args.video != "real-time":
    fourcc = cv.VideoWriter_fourcc(*"MJPG")
    frame_width = int(cap.get(3))
    frame_height = int(cap.get(4))
    writer = cv.VideoWriter(join(DATA_PATH, args.output), fourcc, 20.,
                            (frame_width, frame_height), True)

while cap.isOpened():
    grabbed, frame = cap.read()
    if not grabbed:
        break
    if width is None or height is None:
        height, width = frame.shape[:2]
    # prepared_frame = prepare_image(frame)
    frame_for_viz = frame.copy()
    yolo.load_data(frame)
    layer_outputs = yolo.detect()
    bboxes = yolo.process_outputs(layer_outputs)
    for bbox in bboxes:
        x, y, w, h = bbox
        cv.rectangle(frame_for_viz, (x, y), (x+w, y+h), (255, 0, 0), 2)
    if args.video != "real-time":
        writer.write(frame)
    cur_time = time()
```

```
            sec = cur_time - prev_time
            prevTime = cur_time
            fps = 1 / sec
            cv.imshow('frame', frame_for_viz)
            if cv.waitKey(1) == ord('q'):
                break
        if args.video != "real-time":
            writer.release()
        cap.release()
cv.destroyAllWindows()
```

Модуль classifier.py

```
import cv2 as cv
from cv2.xfeatures2d import matchGMS
import numpy as np
import os
from os.path import join, exists
import pickle

from tqdm import tqdm

from src.utils import FEATURES_PATH, CLASS_NAMES, ORB, SIFT,
DETECTOR_DESCRIPTOR_MAP

GMS_THRESHOLD = 4
LOWE_THRESHOLD = 0.75
MIN_MATCHES = 10



class Classifier:
    def __init__(self, descriptor_name=ORB, reject_outliers=True):
        self.descriptor = DETECTOR_DESCRIPTOR_MAP[descriptor_name]
        self.descriptor_name = descriptor_name
        self.features_path = join(FEATURES_PATH, descriptor_name)
        self.matcher = cv.BFMatcher_create() if descriptor_name in [SIFT] \
            else cv.BFMatcher_create(cv.NORM_HAMMING, True)
        self.reject_outliers = reject_outliers
```

```
    def classify(self, image):
        input_points, input_descriptions = self.descriptor.detectAndCompute(image,
None)
        scores = []
        for cls_name, (ref_points, ref_descriptions, ref_img_shape) in
tqdm(self.load_features().items()):
            matches = self.matcher.match(input_descriptions, ref_descriptions)
            print(f"Class {cls_name}")
            print(f"Matches: {len(matches)}")
            print(f"Input image shape {image.shape[:2]}, reference image shape:
{ref_img_shape}")
            # Outlier rejection
            if self.reject_outliers:
                matches = matchGMS(image.shape[:2], ref_img_shape, input_points,
ref_points, matches,
                                   withScale=True, withRotation=True,
thresholdFactor=GMS_THRESHOLD)
            scores.append(len(matches))
        predicted_class = CLASS_NAMES[np.argmax(scores)]
        return predicted_class
    def load_features(self):
        class_features = {}
        for class_name_pickle in os.listdir(self.features_path):
            if class_name_pickle.rstrip(".pickle") not in CLASS_NAMES:
                continue
            with open(join(self.features_path, class_name_pickle), "rb") as pickle_file:
                keypoint_components, descriptions, shape = pickle.load(pickle_file)
```

```
            class_features.update({
                class_name_pickle.strip(".pickle"): (
                    [cv.KeyPoint(*args) for args in keypoint_components],
                    descriptions, shape
                )
            })
        return class_features
    def dump_features(self, class_name, image):
        if not exists(self.features_path):
            os.makedirs(self.features_path)

        keypoints, descriptions = self.descriptor.detectAndCompute(image, None)

        def parse_keypoint(cv_keypoint):
            args = (
                *cv_keypoint.pt, cv_keypoint.size, cv_keypoint.angle,
                cv_keypoint.response, cv_keypoint.octave, cv_keypoint.class_id
            )
            return args

        keypoint_components = [parse_keypoint(point) for point in keypoints]

        with open(join(self.features_path, f"{class_name}.pickle"), 'wb') as pickle_file:
            pickle.dump((keypoint_components, descriptions, image.shape[:2]),
pickle_file)
```

Модуль preprocessing.py

```
import cv2 as cv
import numpy as np

# Image shape to use for preprocessing (resizing)
IMG_WIDTH_SIZE = 416
IMG_HEIGHT_SIZE = 320
RESIZE_INTERPOLATION_METHOD = cv.INTER_AREA


def extract_detection(image, bboxes):
    object_crops = []
    padding_ratio = 0.1
    for bbox in bboxes:
        x, y, w, h = bbox
        x_padding = padding_ratio * w
        y_padding = padding_ratio * y
        padded_image = image[
            int(y-y_padding):int(y+h+y_padding), int(x-x_padding):int(x+w+x_padding)]
        object_crops.append(padded_image)
    return object_crops


def _image_resize(image, width=None, height=None, inter=cv.INTER_AREA):
    # TODO: use IMG_WIDTH_SIZE constant from above
    # initialize the dimensions of the image to be resized and
```

```
    # grab the image size
    dim = None
    (h, w) = image.shape[:2]
    # if both the width and height are None, then return the
    # original image
    if width is None and height is None:
        return image
    # check to see if the width is None
    if width is None:
        # calculate the ratio of the height and construct the
        # dimensions
        r = height / float(h)
        dim = (int(w * r), height)
    else:
        r = width / float(w)
        dim = (width, int(h * r))
    # resize the image
    resized = cv.resize(image, dim, interpolation=inter)
    return resized


def _resize(image):
    # TODO: redo
    # If new image size is not specified, return original image.
    if IMG_WIDTH_SIZE is None and IMG_HEIGHT_SIZE is None:
        return image


    # If input image already has necessary dimensions, return it.
```

```
    (current_height, current_width) = image.shape[:2]
    if current_height == IMG_WIDTH_SIZE and current_width ==
IMG_HEIGHT_SIZE:
        return image

    # Define new shape of the image.
    if current_width > current_height:
        ratio = IMG_WIDTH_SIZE / float(current_width)
        new_shape = (int(ratio*current_height), IMG_WIDTH_SIZE)
    else:
        ratio = IMG_HEIGHT_SIZE / float(current_height)
        new_shape = (IMG_HEIGHT_SIZE, int(ratio*current_width))

    # Resize image
    resized = cv.resize(image, tuple(reversed(new_shape)),
                        interpolation=RESIZE_INTERPOLATION_METHOD)
    # result = np.zeros((IMG_HEIGHT_SIZE, IMG_WIDTH_SIZE, 3),
dtype=np.uint8)
    # result[:resized.shape[0], :resized.shape[1]] += resized
    # return result
    return resized



def _bgr_equalization(bgr_image):
    blue, green, red = cv.split(bgr_image)
    blue = cv.equalizeHist(blue)
    green = cv.equalizeHist(green)
```

```
    red = cv.equalizeHist(red)
    equalized_bgr_image = cv.merge((blue, green, red))
    return equalized_bgr_image


def _unsharp(img):
    blurred_img = cv.GaussianBlur(img, (3, 3), 10)
    unshapred = cv.addWeighted(img.copy(), 2, blurred_img, -1, 0, img.copy())
    return unshapred


def prepare_image(image, resize=True):
    if resize:
        # image = _resize(image)
        image = _image_resize(image, height=416)
    image = _bgr_equalization(image)
    image = _unsharp(image)
    image = cv.GaussianBlur(image, (3, 3), 0)
    # img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    return image
```

Модуль yolo.py

```
import cv2 as cv
from cv2.dnn import blobFromImage, blobFromImages
import numpy as np
import time

from src.utils import WEIGHTS_PATH, CONFIG_PATH

IMG_SIDE_SIZE = 416
SCALE_FACTOR = 1/255.0



class Yolo:
    """Custom class for using YOLO with OpenCV"""

    def __init__(self, confidence=0.5, nms_threshold=0.6):
        self.confidence = confidence
        self.nms_threshold = nms_threshold
        self.model = None
        self.output_layer_names = None
        self.height = None
        self.width = None

    def load_model(self):
        self.model = cv.dnn.readNetFromDarknet(CONFIG_PATH, WEIGHTS_PATH)
```

```
    def load_data(self, data):
        self.output_layer_names = self.model.getLayerNames()
        self.output_layer_names = [self.output_layer_names[i[0] - 1] for i in
self.model.getUnconnectedOutLayers()]
        if len(data.shape) == 4:
            self.height, self.width = data.shape[1:3]
            # TODO: image shape should not be hard coded.
            blob = blobFromImages(data, SCALE_FACTOR, (IMG_SIDE_SIZE,
IMG_SIDE_SIZE),
                                  swapRB=True, crop=False)
            self.model.setInput(blob)
        else:
            self.height, self.width = data.shape[:2]
            blob = blobFromImage(data, SCALE_FACTOR, (IMG_SIDE_SIZE,
IMG_SIDE_SIZE),
                                 swapRB=True, crop=False)
            self.model.setInput(blob)


    def detect(self):
        if self.output_layer_names:
            layer_outputs = self.model.forward(self.output_layer_names)
        else:
            raise ValueError("Invalid output layer names! Please, run Yolo.load_data
first.")
        return layer_outputs


    def process_outputs(self, layer_outputs):
        bboxes, confidences, class_ids = [], [], []
```

```
        for output in layer_outputs:
            for detection in output:
                scores = detection[5:]
                class_id = np.argmax(scores)
                confidence = scores[class_id]

                if confidence > self.confidence:
                    bbox = detection[:4] * np.array([self.width, self.height, self.width,
self.height])
                    center_x, center_y, w, h = bbox.astype("int")

                    x = int(center_x - w / 2)
                    y = int(center_y - h / 2)

                    bboxes.append([x, y, int(w), int(h)])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)

        indexes = cv.dnn.NMSBoxes(bboxes, confidences, self.confidence,
self.nms_threshold)
        if len(indexes) > 0:
            best_bboxes = [bboxes[i] for i in indexes.flatten() if len(indexes) > 0]
        else:
            best_bboxes = []
        return best_bboxes
        # return bboxes
```

Модуль compare_descriptors.py

```
"""Script for building a comparison tables of descriptors"""
import os
from os.path import join
from time import time

import cv2 as cv
from cv2.xfeatures2d import matchGMS
import pandas as pd
from tqdm import tqdm

from src.preprocessing import prepare_image
from src.utils import (
    DESCRIPTOR_NAMES, DETECTOR_NAMES, CLASS_NAMES,
TEST_IMG_PATH,
    SIFT, DATA_PATH, DETECTOR_DESCRIPTOR_MAP, OUTPUT_PATH,
BRISK,
    ORB, FAST, FastFreak, ROOT_SIFT, FAST_ROOT_SIFT, BRISK_FREAK)
from zoo.brisk import BriskFreak
from zoo.fast import FastRootSift
from zoo.sift import RootSift

GMS_THRESHOLD = 4
TABLE_COLUMNS = [
    "Features Detected in the 1st Image", "Features Detected in the 2nd Image",
    "Features Matched", "Outliers Rejected", "Features Left After GMS",
```

```
    "1st Image Detection & Description Time", "2nd Image Detection & Description
Time",
    "Feature Matching Time", "Outlier Rejection Time", "Total Image Matching Time"
]

MAX_FEATURES_GRID = [5000, 10_000, 15_000]
# For ORB for max_features > 500 it should be 0.
FAST_THRESHOLD = 0
BRISK_THRESHOLD_GRID = [15, 30, 45]
BRISK_OCTAVES_GRID = [3, 4, 5]



DETECTOR_DESCRIPTOR_FUNC_MAP = {
    ORB: cv.ORB_create,
    BRISK: cv.BRISK_create,
    ROOT_SIFT: RootSift,
    FAST_ROOT_SIFT: FastRootSift,
    BRISK_FREAK: BriskFreak
}



def create_param_grid():

    param_grid = {}

    for detector_descriptor_name in [ROOT_SIFT]:
        for threshold in MAX_FEATURES_GRID:
```

```
        detector_descriptor = DETECTOR_DESCRIPTOR_FUNC_MAP[detector_descriptor_name](threshold)
        if detector_descriptor_name == ORB:
            detector_descriptor.setFastThreshold(0)
        param_grid.update({
            f"{detector_descriptor_name}(max_features={threshold})": detector_descriptor
        })



    # for threshold in brisk_threshold_grid:
    #     for octaves in brisk_octaves_grid:
    #         param_grid.update(
    #             {f"{BRISK}(threshold={threshold}, octaves={octaves})": cv.BRISK_create(threshold, octaves)}
    #         )

    return param_grid



if __name__ == '__main__':
    # TODO: Bring this out of the main!!!
    for cls_name in tqdm(CLASS_NAMES, desc=f"Iterating over image classes"):
        template_img = cv.imread(join(DATA_PATH, f"{cls_name}.jpg"))
        prepared_templ_img = prepare_image(template_img.copy())
        cls_test_dir = join(TEST_IMG_PATH, cls_name)
        for test_img_name in tqdm(os.listdir(cls_test_dir), desc="Iterating over test cases"):
```

```
        test_img = cv.imread(join(cls_test_dir, test_img_name))
        prepared_test_img = prepare_image(test_img.copy())
        descriptor_results = []

        # for detector_descriptor_name, detector_descriptor in tqdm(
        #         DETECTOR_DESCRIPTOR_MAP.items(), desc="Iterating over detectors and descriptros"):
        for detector_descriptor_name, detector_descriptor in tqdm(
                create_param_grid().items(), desc="Performing grid search"):

            description_time_img1_begin = time()
            templ_kps, templ_decs = detector_descriptor.detectAndCompute(prepared_templ_img, None)
            description_time_img1_end = time()
            first_img_detection_time = description_time_img1_end - description_time_img1_begin

            description_time_img2_begin = time()
            test_kps, test_decs = detector_descriptor.detectAndCompute(prepared_test_img, None)
            description_time_img2_end = time()
            second_img_detection_time = description_time_img2_end - description_time_img2_begin

            matcher = cv.BFMatcher_create() if f"_{SIFT}" in detector_descriptor_name \
                else cv.BFMatcher_create(cv.NORM_HAMMING, True)
```

```
            # if f"_{SIFT}" in detector_descriptor_name or f"-{SIFT}" in
detector_descriptor_name:
            #


            feature_matching_time_begin = time()
            matches = matcher.match(templ_decs, test_decs)
            feature_matching_time_end = time()
            feature_matching_time = feature_matching_time_end -
feature_matching_time_begin


            outlier_rejection_time_begin = time()
            best_matches = matchGMS(
                template_img.shape[:2], test_img.shape[:2], templ_kps, test_kps,
matches,
                withRotation=True, withScale=True,
thresholdFactor=GMS_THRESHOLD
            )
            outlier_rejection_time_end = time()
            outlier_rejection_time = outlier_rejection_time_end -
outlier_rejection_time_begin


            descriptor_results.append([
                detector_descriptor_name,
                len(templ_kps), len(test_kps), len(matches), len(matches) -
len(best_matches),
                len(best_matches), first_img_detection_time,
second_img_detection_time,
                feature_matching_time, outlier_rejection_time,
                # Total matching time
```

```
                first_img_detection_time + second_img_detection_time +
feature_matching_time
                + outlier_rejection_time
            ])
        # Dump results
        with open(join(OUTPUT_PATH, "all_dfs.csv"), 'a') as f:
            columns = [f"{cls_name}-{test_img_name.split('.')[0]}"] +
TABLE_COLUMNS
            pd.DataFrame(descriptor_results, columns=columns).round(4).to_csv(f)
            f.write("\n")
```

Модуль score_descriptors.py

```
"""Script for building a comparison tables of descriptors"""
import os
from os.path import join
from time import time

import cv2 as cv
import numpy as np
from cv2.xfeatures2d import matchGMS
import pandas as pd
from tqdm import tqdm

from src.preprocessing import prepare_image
from src.utils import (
    DESCRIPTOR_NAMES, DETECTOR_NAMES, CLASS_NAMES,
TEST_IMG_PATH,
    SIFT, DATA_PATH, DETECTOR_DESCRIPTOR_MAP, OUTPUT_PATH,
BRISK,
    ORB, FAST, FastFreak, ROOT_SIFT, FAST_ROOT_SIFT, BRISK_FREAK,
TEST_IMG_NAMES, sift_is_descriptor)
from zoo.brisk import BriskFreak
from zoo.fast import FastRootSift
from zoo.sift import RootSift

GMS_THRESHOLD = 4
TABLE_COLUMNS = [
    "Features Detected in the 1st Image", "Features Detected in the 2nd Image",
```

```
    "Features Matched", "Outliers Rejected", "Features Left After GMS",
    "1st Image Detection & Description Time", "2nd Image Detection & Description Time",
    "Feature Matching Time", "Outlier Rejection Time", "Total Image Matching Time"
]

MAX_FEATURES_GRID = [5000, 10_000, 15_000]
# For ORB for max_features > 500 it should be 0.
FAST_THRESHOLD = 0
BRISK_THRESHOLD_GRID = [15, 30, 45]
BRISK_OCTAVES_GRID = [3, 4, 5]


DETECTOR_DESCRIPTOR_FUNC_MAP = {
    ORB: cv.ORB_create,
    BRISK: cv.BRISK_create,
    ROOT_SIFT: RootSift,
    FAST_ROOT_SIFT: FastRootSift,
    BRISK_FREAK: BriskFreak
}


if __name__ == '__main__':
    # TODO: Bring this out of the main!!!
    descriptor_results = []
    for detector_descriptor_name, detector_descriptor in tqdm(
```

```
        DETECTOR_DESCRIPTOR_MAP.items(), desc="Iterating over detectors-descriptors"):
    # TODO: make scoring relative and not absolute.
    detector_descriptor_score = 0
    total_matching_time = []
    for cls_index, cls_name in enumerate(tqdm(CLASS_NAMES)):
        for test_img_name in TEST_IMG_NAMES:
            test_img = cv.imread(join(TEST_IMG_PATH, cls_name, f"{test_img_name}.jpeg"))
            prepared_test_img = prepare_image(test_img.copy())

            test_img_description_time_begin = time()
            test_keypoints, test_descriptions = detector_descriptor.detectAndCompute(
                prepared_test_img, None)
            test_img_description_time_end = time()
            test_img_description_time = test_img_description_time_end - test_img_description_time_begin

            matches_counts = []
            for template_img_name in tqdm(CLASS_NAMES, desc="Matching images"):
                # TODO: Iterate over images and not image names so that you don't have to read them each time
                template_img = cv.imread(join(DATA_PATH, f"{template_img_name}.jpg"))
                prepared_template_img = prepare_image(template_img.copy())

                template_img_description_time_begin = time()
```

```
            template_keypoints, template_descriptions =
detector_descriptor.detectAndCompute(
                prepared_template_img, None)
            template_img_description_time_end = time()
            template_img_description_time = template_img_description_time_end -
template_img_description_time_begin

            # TODO: Is it necessary to create matcher every time?
            matcher = cv.BFMatcher_create() if
sift_is_descriptor(detector_descriptor_name) \
                else cv.BFMatcher_create(cv.NORM_HAMMING, True)

            feature_matching_time_begin = time()
            matches = matcher.match(template_descriptions, test_descriptions)
            feature_matching_time_end = time()
            feature_matching_time = feature_matching_time_end -
feature_matching_time_begin

            outlier_rejection_time_begin = time()
            best_matches = matchGMS(
                template_img.shape[:2], test_img.shape[:2], template_keypoints,
test_keypoints, matches,
                withRotation=True, withScale=True,
thresholdFactor=GMS_THRESHOLD
            )
            outlier_rejection_time_end = time()
            outlier_rejection_time = outlier_rejection_time_end -
outlier_rejection_time_begin
```

```
                matches_counts.append(len(best_matches))

            if np.argmax(matches_counts) == cls_index:
                detector_descriptor_score += 1

            total_matching_time.append(
                test_img_description_time + template_img_description_time
                + feature_matching_time + outlier_rejection_time
            )
        descriptor_results.append([
            detector_descriptor_name, detector_descriptor_score,
            np.mean(total_matching_time)
        ])

    pd.DataFrame(descriptor_results).to_csv(join(OUTPUT_PATH, "all_dfs.csv"))
                # descriptor_results.append([
                #     detector_descriptor_name,
                #     len(templ_kps), len(test_kps), len(matches), len(matches) -
len(best_matches),
                #     len(best_matches), first_img_detection_time,
test_img_detection_time,
                #     feature_matching_time, outlier_rejection_time,
                #     # Total matching time
                #     first_img_detection_time + test_img_detection_time +
feature_matching_time
                #     + outlier_rejection_time
                # ])
            # Dump results
```

```
            # with open(join(OUTPUT_PATH, "all_dfs.csv"), 'a') as f:
            #     columns = [f"{cls_name}-{test_img_name.split('.')[0]}"] +
TABLE_COLUMNS
            #     pd.DataFrame(descriptor_results,
columns=columns).round(4).to_csv(f)
            #     f.write("\n")
```

Модуль utils.py

```
"""File containing constants and utilities for scripts."""
from os.path import join, dirname, normpath

import cv2 as cv
from cv2 import SIFT_create

from zoo.brisk import BriskOrb, BriskSift, BriskRootSift, BriskFreak
from zoo.orb import OrbBrisk, OrbSift, OrbRootSift, OrbFreak
from zoo.sift import RootSift
from zoo.fast import FastFreak, FastOrb, FastSift, FastRootSift
from zoo.star import StarOrb, StarBrisk, StarSift, StarRootSift, StarFreak



# Path constants
REPO_ROOT = normpath(join(dirname(__file__), ".."))
DATA_PATH = join(REPO_ROOT, "data")
TEST_IMG_PATH = join(REPO_ROOT, "test_images")
FEATURES_PATH = join(REPO_ROOT, "features")
OUTPUT_PATH = join(REPO_ROOT, "output")
LABELS_PATH = join(REPO_ROOT, "config", "coins.names")
WEIGHTS_PATH = join(REPO_ROOT, "config", "yolov4_coins.weights")
CONFIG_PATH = join(REPO_ROOT, "config", "yolov4_coins.cfg")



CLASS_NAMES = ["1uah_heads", "2uah_heads", "5uah_heads", "10uah_heads"]
```

```
TEST_IMG_NAMES = ["front", "top", "bottom", "left", "right"]

# Detectors / Detectors + Descriptors
ORB = "ORB"
BRISK = "BRISK"
SIFT = "SIFT"
FAST = "FAST"
AKAZE = "AKAZE"
FAST = "FAST"
ROOT_SIFT = "Root-SIFT"
STAR = "STAR"

DETECTOR_NAMES = [ORB, BRISK, FAST, AKAZE, SIFT, ROOT_SIFT, STAR]

# Descriptors
FREAK = "FREAK"
# TODO: add BRIEF to the zoo.
BRIEF = "BRIEF"
DESCRIPTOR_NAMES = [ORB, BRISK, AKAZE, SIFT, ROOT_SIFT, FREAK]

# Hybrids
FAST_ROOT_SIFT = f"{FAST}_{ROOT_SIFT}"
BRISK_FREAK = f"{BRISK}_{FREAK}"

# Detector's / descriptor's parameters
DEFAULT_MAX_FEATURES = 10_000
```

```
# TODO: create a method that would generate DETECTOR_DESCRIPTOR_MAP.
DETECTOR_DESCRIPTOR_MAP = {
    ORB: cv.ORB_create(DEFAULT_MAX_FEATURES, fastThreshold=0),
    BRISK: cv.BRISK_create(),
    AKAZE: cv.AKAZE_create(),
    # SIFT: SIFT_create(DEFAULT_MAX_FEATURES),
    # ROOT_SIFT: RootSift(DEFAULT_MAX_FEATURES),
    # Hybrids
    # FAST detector combined with other descriptor
    f"{FAST}_{ORB}": FastOrb(),
    f"{FAST}_{BRISK}": FastFreak(),
    # f"{FAST}_{AKAZE}": FastFreak,
    # f"{FAST}_{SIFT}": FastSift(),
    # FAST_ROOT_SIFT: FastRootSift(),
    f"{FAST}_{FREAK}": FastFreak(),
    # ORB detector combined with other descriptor
    f"{ORB}_{BRISK}": OrbBrisk(),
    # f"{ORB}_{AKAZE}": FastFreak,
    # f"{ORB}_{SIFT}": OrbSift(),
    # f"{ORB}_{ROOT_SIFT}": OrbRootSift(),
    f"{ORB}_{FREAK}": OrbFreak(),
    # # BRISK detector combined with other descriptor
    f"{BRISK}_{ORB}": BriskOrb(),
    # f"{BRISK}_{AKAZE}": FastFreak,
    # f"{BRISK}_{SIFT}": BriskSift(),
    # f"{BRISK}_{ROOT_SIFT}": BriskRootSift(),
    f"{BRISK}_{FREAK}": BriskFreak(),
```

```
    # # STAR detector combined with other descriptor.
    f"{STAR}_{ORB}": StarOrb(),
    f"{STAR}_{BRISK}": StarBrisk(),
    # f"{STAR}_{AKAZE}": FastFreak,
    # f"{STAR}_{SIFT}": StarSift(),
    # f"{STAR}_{ROOT_SIFT}": StarRootSift(),
    f"{STAR}_{FREAK}": StarFreak(),
}

# Object detector's params
YOLO_CONFIDENCE = 0.5
NMS_THRESHOLD = 0.3

def sift_is_descriptor(detector_descriptor_name):
    # TODO: try to make it smarter with regexp.
    if f"_{SIFT}" in detector_descriptor_name or f"-{SIFT}" in
detector_descriptor_name:
        return True
    else:
        return False
```

Модуль generate_features.py

```
"""Generate keypoints and descriptions for template images and dump them."""
import cv2 as cv
import os
from os.path import join

from tqdm import tqdm

from src.classifier import Classifier
from src.preprocessing import prepare_image
from src.utils import DATA_PATH, CLASS_NAMES, DESCRIPTOR_NAMES,
STAR, ORB, FAST

if __name__ == '__main__':
    # Change the following list to chose descriptors.
    descriptors = [f"{FAST}_{ORB}"]
    for name in descriptors:
        # reject_outliers = False if name == FAST_FREAK else True
        clf = Classifier(descriptor_name=name)
        for image_name in tqdm(os.listdir(DATA_PATH)):
            if image_name.split('.')[0] in CLASS_NAMES:
                image = cv.imread(join(DATA_PATH, image_name))
                clf.dump_features(image_name.rstrip(".jpg"),
                                  prepare_image(image, resize=True))
```

ДОДАТОК Б ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

Слайд 1

Система розпізнавання монет

Дипломний керівник - Доцент, к.т.н. Дідковська Марина Віталівна
Студент - Пивовар Павло
КА-76

2021

Слайд 2

Актуальність

- Монети все ще використовують у багатьох державах
- Підрахунок великої кількості монет є досить складною задачею для людини
- Система може бути корисна для нумізматів та істориків

Слайд 3

Існуючі підходи

- Класичні методи є недостатньо швидкими
- Нейронні мережі погано масштабуються

Слайд 4

Постановка задачі

Побудувати систему, що поєднувала б швидкість та точність нейронних мереж, але при цьому могла бути легко масштабованою

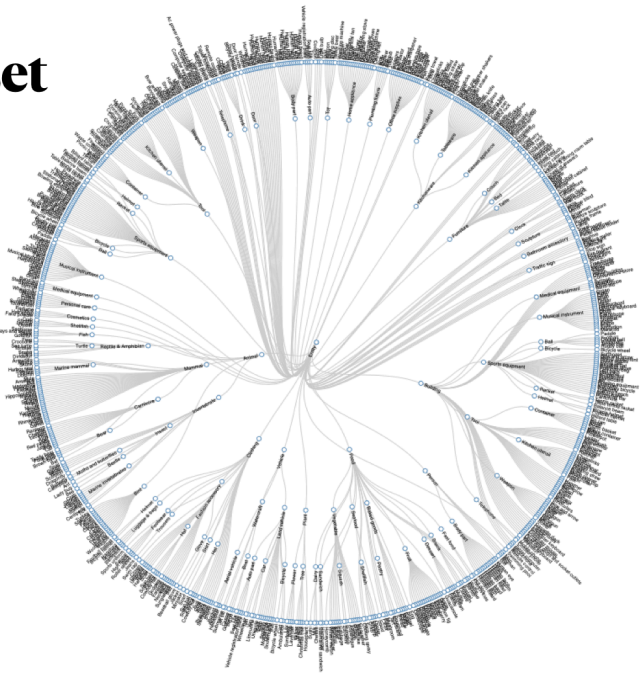
Слайд 5

- **Мета** — побудувати систему, що могла б розпізнавати монети у реальному часі
- **Об’єкт** — система розпізнавання монет
- **Предмет** — точність й швидкодія розпізнавання системою

Слайд 6

Open Images Dataset

- 15,851,536 обмежувальних коробок для 600 категорій
- Більше 1000 зображень монет

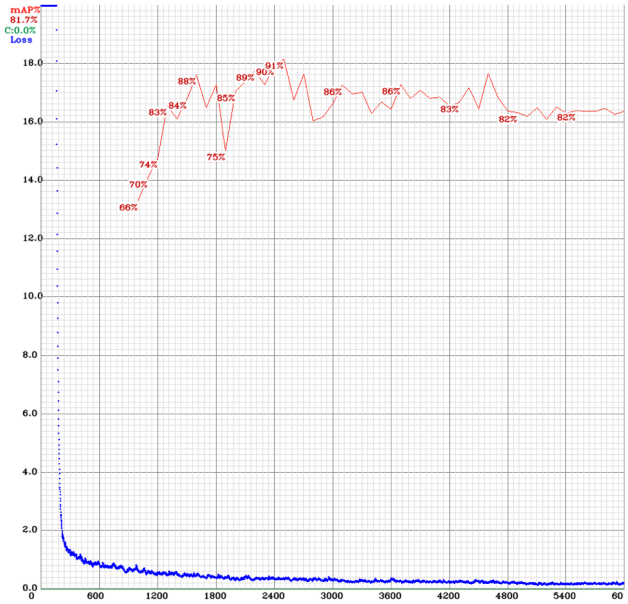


Слайд 7

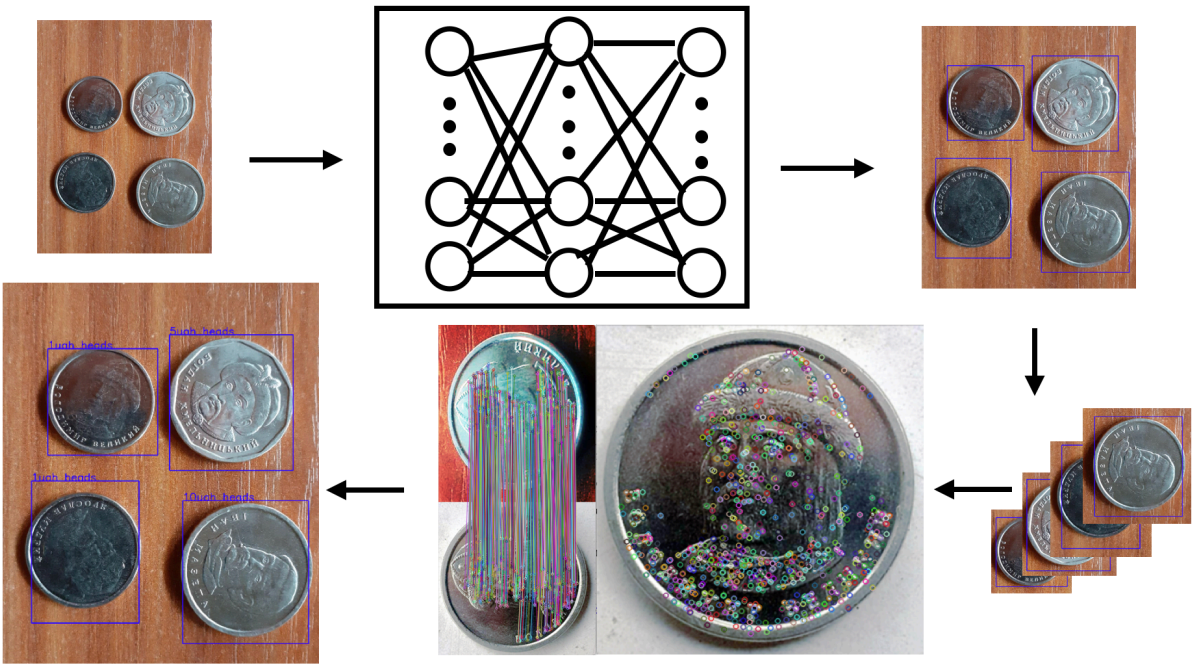
Архітектура

YOLO

- Тренування за допомогою Google Colab (посилання)
- Близько 5 годин тренування



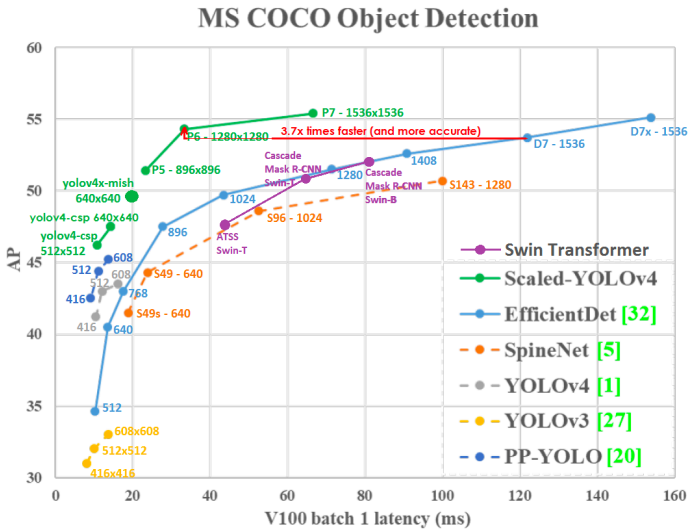
Слайд 8



Слайд 9

Критерії порівняння

- Для детектору (object detector) монет перевага надана швидкодії
- Для детекторів-дескрипторів (feature detectors)
  - кількість та якість знайдених ключових точок
  - швидкодія



Слайд 10

Порівняльний аналіз

Детектор_Дескриптор	Оцінка (max 20)	Загальний час роботи (с)
ORB	20	0.755
BRISK	19	0.094
FAST_ORB	20	0.127
FAST_BRISK	18	0.208
FAST_FREAK	18	0.211
ORB_BRISK	17	0.654
ORB_FREAK	13	0.172
BRISK_ORB	19	0.145
BRISK_FREAK	17	0.182

Слайд 11

Висновки

- Потрібно масштабувати систему для розпізнання більшої кількості класів монет
- Потрібно зробити порівняльний аналіз детекторів об’єктів
- Почистити датасет