МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Розпізнавання математичних формул з використанням згорткових нейронних мереж

Виконав студент 2 курсу групи МІС-22
спеціальності 122 Комп'ютерні науки
Фумжи Артем Васильович

Керівник к.т.н., доцент
Перелигін Борис Вікторович

Рецензент к.ф.-м.н.,
Ткач Тетяна Борисівна

Одеса 2023

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВИХ ПІЗНАЧЕНЬ І ТЕРМІНІВ

CNN – convolutional neural network – згорткова нейронна мережа.

GUI – графічний інтерфейс користувача.

Mask R-CNN – Mask Region Convolutional Network.

COCO – це великомасштабний набір даних для виявлення, сегментації та субтитрів об'єктів.

HNM – Hard Negative Mining.

RGB – адитивна колірна модель.

R-CNN – це сімейство моделей машинного навчання для комп’ютерного зору та, зокрема, виявлення об’єктів.

RoI – області інтересів.

RPN – мережа пропозицій регіонів.

R-FCN – Region-based Fully Convolutional Network.

R-FCN – регіональна повністю згорткова мережа.

SSD – Single-Shot Detector.

YOLO – You Only Look Once.

ВСТУП

Сьогодні технології дедалі більше проникають у різні сфери життя, починаючи з автоматизації діловодства, закінчуючи постановкою діагнозів рекомендаційними системами.

У багатьох сферах життя нейронні мережі вже давно стали буденністю, а їх застосування в системах вважається запорукою успіху для продажу тих чи інших програм або послуг. Це ж торкнулося й галузі науки.

У цій роботі розглядається система для розпізнавання математичних формул з використанням згорткових нейронних мереж. Подібні системи дозволяють вченим економити час для перенесення своїх розрахунків з паперу або дошки в текстові редактори.

Робота складається з вступу, 5 розділів, висновків, списку літератури. Обсяг роботи складає 71 сторінку, обсяг списку літератури – 35 джерел.

Мабуть, найпопулярніше і найперспективніше завдання нейромереж – технології розпізнавання образів. Вони або окремо, або в інтегрованому вигляді використовуються в таких сферах, як безпека та спостереження, сканування та створення зображень, маркетинг та реклама, доповнена реальність та пошук зображень.

Сьогодні створюються та вже використовуються мережі, в яких машини здатні розпізнавати символи на папері та банківських картках, підписи на офіційних документах, детектувати об'єкти тощо. Ці функції полегшують працю людини і підвищують точність та надійність різних робочих процесів завдяки виключенню із завдання людського фактора. Але навчити комп'ютер розпізнавати об'єкти не так просто. Одна із складнощів полягає в тому, що комп'ютер бачить не так само, як люди. У комп'ютера немає життєвого досвіду та здатності так само, як людський мозок ідентифікувати об'єкти на зображення та відео. Спочатку він не здатний відрізнити будинок від дерева, не маючи якихось вихідних даних. Щоб навчити комп'ютер бачити та розуміти, що знаходиться на зображенні, люди

використовують технології машинного навчання.

І тому збирають великі бази даних, у тому числі формують датасеты. Виділивши ознаки та його комбінації для ідентифікації подібних об'єктів, можна натренувати модель машинного навчання розпізнавати необхідні типи закономірностей. Звичайно, навіть після завантаження кількох датасетів моделі можуть невірно розпізнавати деякі об'єкти. Якщо таке трапляється, моделі «донавчають» на нових наборах даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Згорткові мережі: основні поняття та архітектура

Згорткова нейронна мережа – клас штучних нейронних мереж, який став домінуючим у різних завданнях комп'ютерного зору та викликає інтерес у різних галузях. CNN призначена для автоматичного та адаптивного вивчення просторових ієрархій об'єктів за допомогою зворотного розповсюдження з використанням декількох будівельних блоків, таких як шари згортки, що об'єднують шари та повнозв'язкові шари [1].

Згортковий шар є основним будівельним блоком CNN, і саме тут відбувається більша частина обчислень. Для цього потрібно кілька компонентів: вхідні дані, фільтр та карта об'єктів. Приклад роботи згорткового шару показаний рисунку 1.

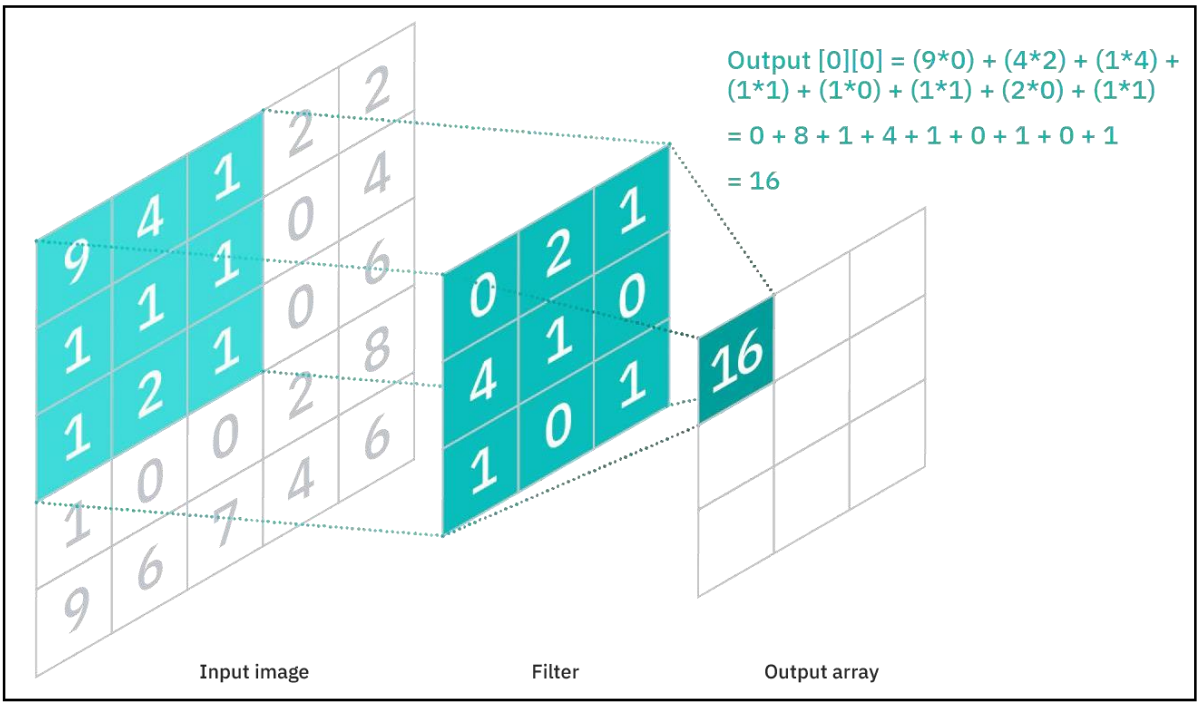


Рисунок 1 – Приклад роботи згорткового шару

Об'єднуючий шар, також відомий як субдискретизаційний шар, знижує

розмірність, зменшуючи кількість параметрів на вході. Подібно до згортального шару, операція об'єднання фільтрує дані по всьому входу, але відмінність полягає в тому, що цей фільтр не має ваг. Натомість ядро застосовує функцію агрегування до значень у приймаючому полі, заповнюючи вихідний масив.

Назва повнозв'язкового шару точно описує себе. Значення пікселів вхідного зображення не пов'язані безпосередньо з вихідним шаром у частково пов'язаних шарах. Однак на повнозв'язковому рівні кожен вузол вихідного шару безпосередньо з'єднується з вузлом попереднього рівня.

Цей шар виконує завдання класифікації на основі ознак, вилучених за допомогою попередніх шарів та їх різних фільтрів. У той час як згорткові шари та шари об'єднання зазвичай використовують функції ReLu, повнозв'язкові шари зазвичай використовують функцію активації для відповідної класифікації вхідних даних, створюючи ймовірність від 0 до 1.

Розглянемо докладніше архітектуру згорткової нейронної мережі.

Базова архітектура має 3 складові: вхідний шар; прихований шар; вихідний шар. Приховані шари представлені наступною архітектурою: шар згортки, функція активації, шар зменшення розмірності. Схема представлена рисунку 2.

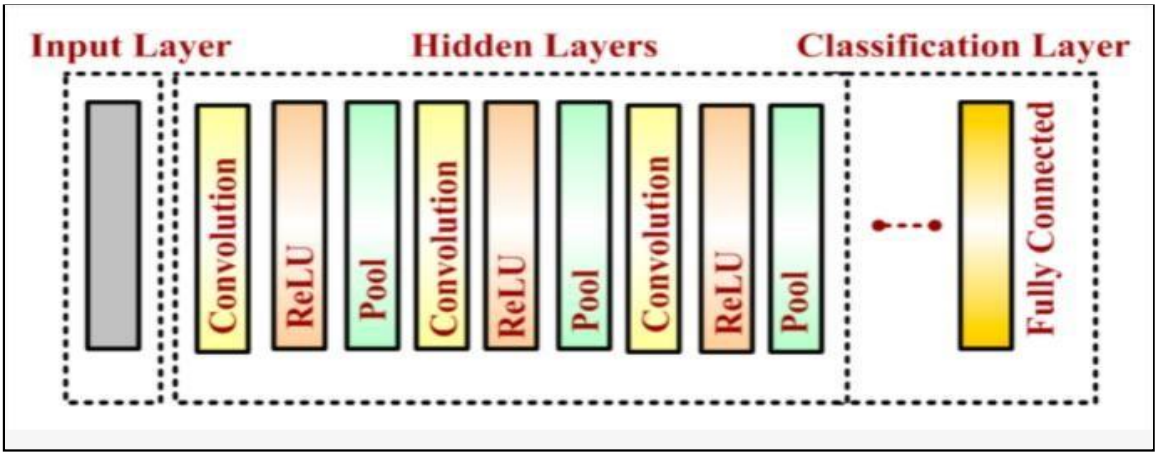


Рисунок 2 – Типова архітектура нейронної згорткової мережі

У вхідному шарі завантажуються дані, в ньому вони описуються (висота, ширина, кількість RGB каналів).

Прихований шар вважається основою згорткової мережі. У ньому відбувається виділення ознак рукописних цифр, у яких застосовуються функції згортки, об'єднання та активації.

Згортковий використовується для отримання ознак зображення. Зображення вхідного шару з розміром $n \times n$ та ядро розміром $m \times m$, згортаються в наступний шар розміром $L \times L$ за формулою:

$$L = n - m + 1. \quad (1)$$

Об'єднуючий шар додається серед двох згорткових, щоб зменшити розмірність вхідних даних зменшення обчислювальної складності. Поєднання дозволяє передавати вибрані значення на наступний рівень, залишаючи непотрібні значення позаду. Об'єднуючий шар також допомагає у виборі функцій та управлінні переоснащенням. Операція об'єднання виконується незалежно. Він працює, витягуючи тільки одне вихідне значення з мозаїчних підобластей вхідних зображень, що не перекриваються. Приклад представлений рисунку 3.

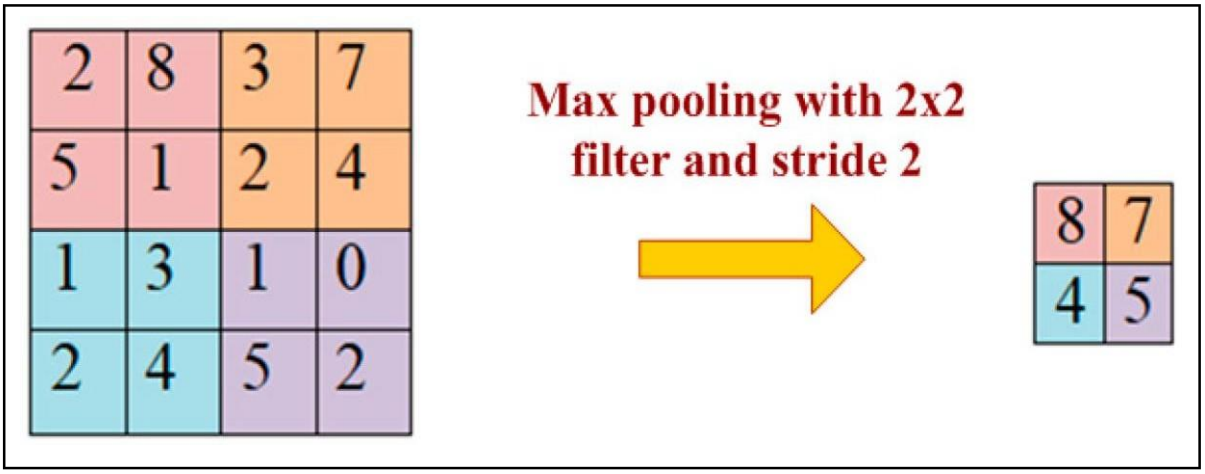


Рисунок 3 – Максимальне поєднання з ядром 2 × 2 та кроком 2

1.2 Основні методи розпізнавання образів та класифікації об'єктів

Класифікація – тобто визначення типу об'єкта, кожного виділеного сегмента окремо. Ці завдання становлять виявлення об'єктів. Таким чином, виявлення об'єктів – це процес пошуку та класифікації об'єктів у зображенні. До вирішення завдань детектування та класифікації застосовуються різні підходи: статистичні, спеціально розроблені теорії ключових точок, генетичні алгоритми. Іншим методом є машинне навчання, що часто використовує в роботі додатково один із попередніх. Нейронні мережі є найперспективнішим методом для детектування та класифікації об'єктів на зображенні. Для виконання класифікації, завдань розпізнавання, виявлення об'єктів на зображенні використовуються технологія нейронних згорткових мереж. Згорткові нейронні мережі є важливими інструментами для глибокого навчання та особливо корисні для класифікації зображень, виявлення об'єктів та завдань розпізнавання.

Одним з наївних підходів на основі згорткових нейронних мереж може бути використання як ядра канонічних зображень класів, які необхідно знайти на зображенні, і подальше використання ковзного вікна для обчислення згортки. Для зменшення кількості розглянутих рамок виділяють два основних паралельні підходи, що розвиваються [2].

Двоетапні методи, вони ж «методи, засновані на регіонах» – підхід, поділений на два етапи. У першому етапі селективним пошуком чи з допомогою спеціального шару нейронної мережі виділяються регіони інтересу – області , з високою ймовірністю містять у собі об'єкти. На другому етапі обрані регіони розглядаються класифікатором для визначення належності вихідним класам та регресором, що уточнює розташування обмежуючих рамок.

Одноетапні методи – підхід, що не використовує окремий алгоритм для генерації регіонів, натомість передбачаючи координати певної кількості рамок, що обмежують, з різними характеристиками, такими, як результати

класифікації і ступінь впевненості і надалі коригуючи місцезнаходження рамок.

Розглянемо більш докладно ці методи.

1.2.1 Двоетапні методи

R-CNN.

Перші моделі інтуїтивно починають із пошуку області, а потім виконують класифікацію. У R-CNN метод вибіркового пошуку, розроблений Дж.Р.Р. Уйлінгс та ін. (2012) є альтернативою повному пошуку на зображенні для фіксації розташування об'єкта. Він ініціалізує невеликі області зображення та поєднує їх у ієрархічну групу. Таким чином, остання група є блоком, що містить все зображення. Виявлені області поєднуються відповідно до різних колірних просторів і показників подібності. Результатом є кілька речень регіонів, які можуть містити об'єкт шляхом злиття невеликих регіонів (рис.4).

Рисунок 4 – Візуалізація пропозицій галузі алгоритму

Модель R-CNN (Р. Гіршик та ін., 2014) поєднує метод вибіркового пошуку для виявлення пропозицій регіонів і глибоке навчання для виявлення об'єкта в цих регіонах. Розмір кожної пропозиції регіону змінюється, щоб

відповідати вхідним даним CNN, з яких ми витягуємо вектор ознак із 4096 вимірами. Вектор ознак передається до декількох класифікаторів для отримання ймовірностей приналежності до кожного класу. Кожен із цих класів має класифікатор, навчений робити висновки про можливість виявлення цього об'єкта для заданого вектора ознак [3]. Цей вектор також передає лінійний регресор, щоб адаптувати форми рамки, що обмежує, для пропозиції регіону і, таким чином, зменшити помилки локалізації (рис.5).

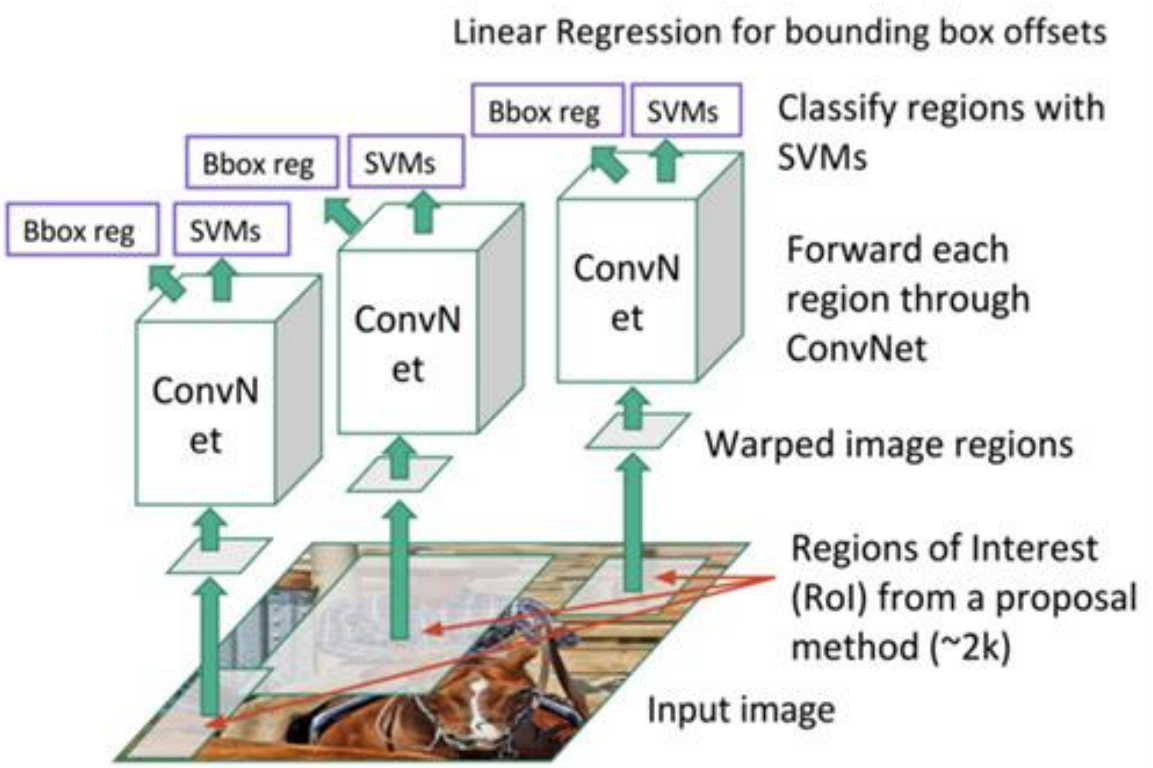


Рисунок 5 – Регіональна мережа згортання (R-CNN)

Перерахуємо недоліки R-CNN:

– навчання мережі, як і раніше, займає величезну кількість часу, так як вам доведеться класифікувати 2000 пропозицій регіонів для кожного зображення;
– його не можна реалізувати в режимі реального часу, тому що для кожного тестового зображення потрібно близько 47 секунд;

– алгоритм вибіркового пошуку є фіксованим алгоритмом. Тож цьому етапі навчання немає. Це може призвести до поганих пропозицій регіонів-кандидатів.

Fast R-CNN.

Мета швидкої мережі згортання на основі регіонів (Fast R-CNN), розробленої Р. Гіршиком (2015), полягає в тому, щоб скоротити витрати часу, пов'язані з великою кількістю моделей, необхідних для аналізу всіх пропозицій регіонів.

Основна CNN з кількома шарами згортки приймає все зображення в якості вхідних даних замість використання CNN для кожної пропозиції регіону (R-CNN). Області інтересів виявляються за допомогою методу вибіркового пошуку, що застосовується до створених карт об'єктів. Формально розмір карт об'єктів зменшується з використанням шару пула RoI, щоб отримати допустиму область інтересів з фіксованою висотою та шириною як гіперпараметри. Кожен шар області інтересу передає пов'язані повні шари, створюючи вектор ознак. Вектор використовується для прогнозування об'єкта, що спостерігається, за допомогою класифікатора softmax і для адаптації локалізації обмежувальної рамки за допомогою лінійного регресора.

Все зображення передає модель CNN визначення області інтересу на картах об'єктів. Кожна область відокремлена за допомогою шару пулу RoI, і він живить пов'язані повністю шари. Цей вектор використовується класифікатором softmax для виявлення об'єкта та лінійним регресором для зміни координат рамки, що обмежує.

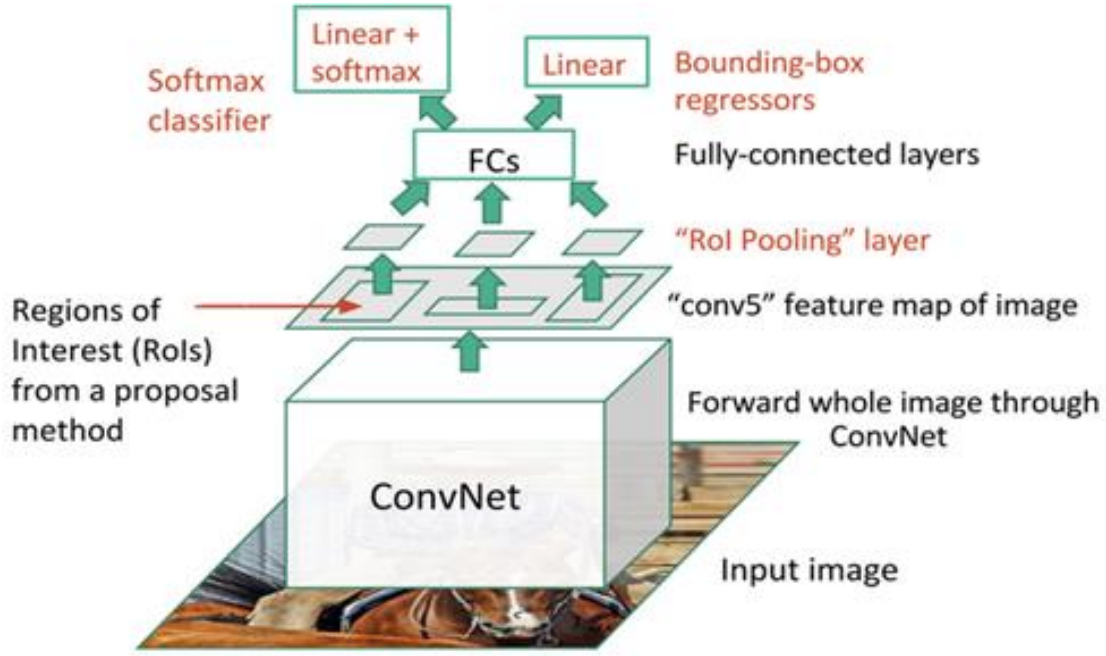


Рисунок 6 – Мережна мережа Fast R-CNN

Faster R-CNN [4].

Пропозиції регіонів, виявлені за допомогою методу вибіркового пошуку, як і раніше, були необхідні в попередній моделі, яка вимагала значних обчислювальних ресурсів. С. Рен та ін. (2016) представили мережу пропозицій регіонів для прямого створення пропозицій регіонів, прогнозування обмежувальних рамок та виявлення об'єктів. Швидша мережа сітки на основі регіонів являє собою комбінацію між RPN і моделлю Fast R-CNN.

Модель CNN приймає як вхідні дані все зображення і створює карти характеристик. Вікно розміром 3x3 ковзає по всіх картах об'єктів і виводить вектор ознак, пов'язаний з двома повністю пов'язаними шарами, одна для блокової регресії та одна для блокової класифікації. Пропозиції кількох регіонів передбачаються повністю пов'язаними верствами. Фіксується максимум k областей, тому вихідні шари регресії блоків мають розмір 4k (координати блоків, їх висота і ширина), а вихідні дані шару класифікації блоків мають розмір 2k («об'єктивність» балів щоб виявити об'єкт чи ні в

коробці).

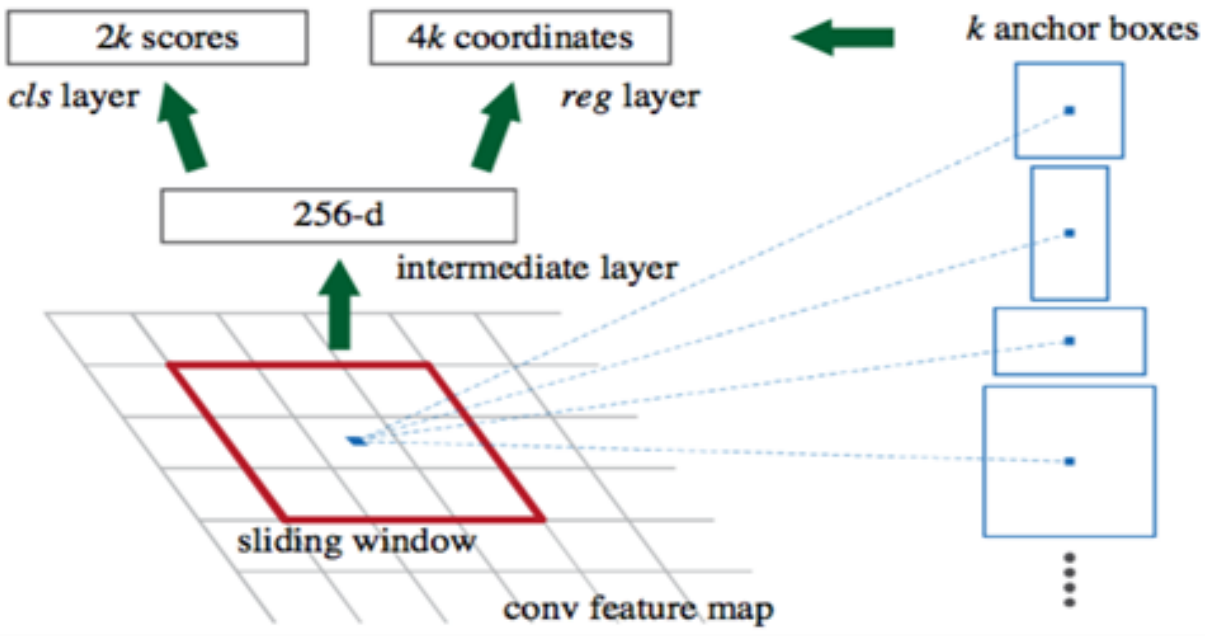


Рисунок 7 – Виявлення блоків прив'язки для одного вікна 3x3

Коли блоки прив'язки виявлені, вони вибираються шляхом застосування порога до показника об'єктивності, щоб залишити тільки відповідні блоки. Ці блоки прив'язки та карти об'єктів, обчислені вихідною моделлю CNN, подають модель Fast R-CNN.

Швидше R-CNN використовує RPN, щоб уникнути методу вибіркового пошуку, прискорити процеси навчання та тестування та підвищити продуктивність. RPN використовує попередньо вивчену модель набору даних ImageNet для класифікації і точно налаштовує набір даних PASCAL VOC. Потім згенеровані пропозиції регіонів з якірними полями використовують для навчання Fast R-CNN. Цей процес є ітеративним.

Все зображення передає модель CNN для створення блоків прив'язки як пропозиції області з упевненістю, що вона містить об'єкт.

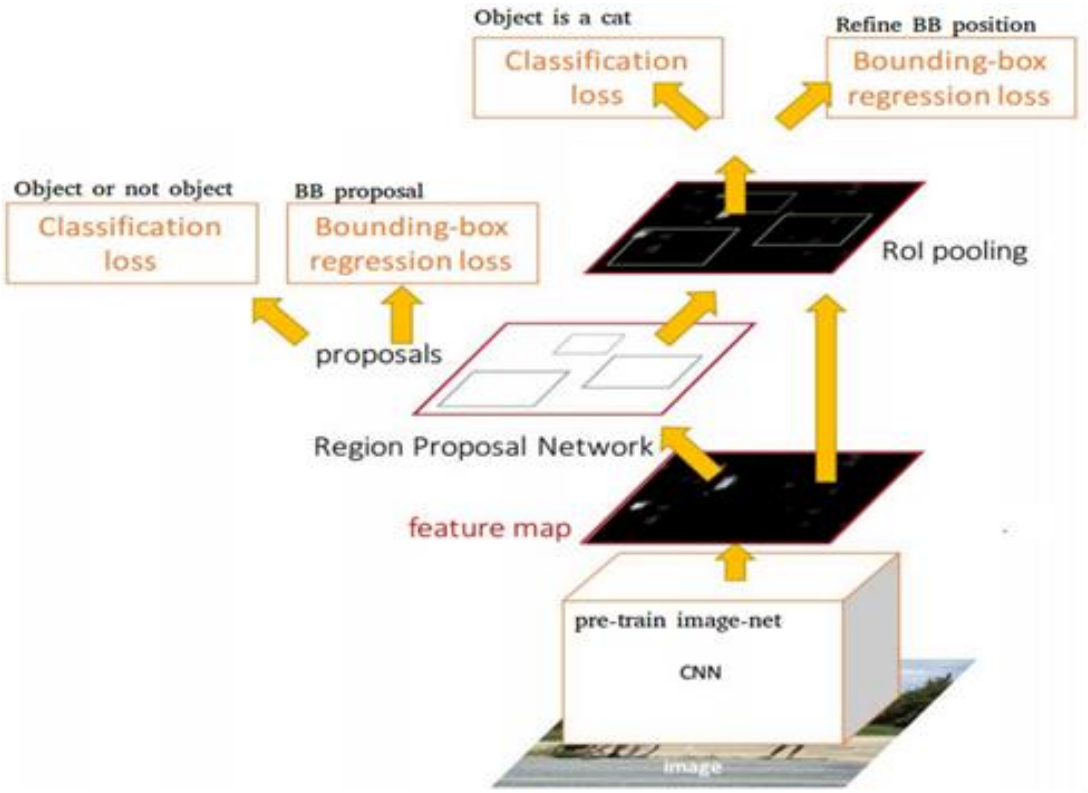


Рисунок 8 – Розпізнавання мережею Faster R-CNN

Використовується Fast R-CNN, що приймає як вхідні дані карти об'єктів і пропозиції регіонів. Для кожної скриньки обчислюються ймовірності виявлення кожного об'єкта та корекція розташування скриньки.

Region-based Fully Convolutional Network [4].

Методології Fast та Faster R-CNN полягають у виявленні пропозицій регіонів та розпізнаванні об'єкта у кожному регіоні. Регіональна повністю згорткова мережа являє собою модель тільки зі згортковими шарами, що забезпечує повне зворотне поширення для навчання та логічного виведення. Автори об'єднали два основних кроки в одну модель, щоб одночасно враховувати виявлення об'єкта (інваріант розташування) та його положення (варіант розташування).

Модель ResNet-101 приймає вихідне зображення як вхідні дані. Останній шар виводить карти об'єктів, кожна з яких спеціалізується на виявленні категорії у будь-якому місці. Наприклад, одна карта ознак

спеціалізується на виявленні кішки, інша на банані і так далі. Такі карти об'єктів називаються картами оцінки з урахуванням становища, оскільки враховують просторову локалізацію конкретного об'єкта. Він складається з k*k*(C+1) карток оцінок, де k – розмір картки оцінок, а C – кількість класів. Усі ці карти утворюють банк окулярів. По суті ми створюємо патчі, які можуть розпізнавати частину об'єкта. Наприклад, за k=3 ми можемо розпізнати 3x3 частини об'єкта.

Паралельно потрібно запустити RPN для створення області інтересу. Нарешті, ми поділяємо кожну область інтересу на осередки та звіряємо їх із банком результатів. Якщо активовано достатню кількість цих частин, то патч голосує так, я розпізнав об'єкт.

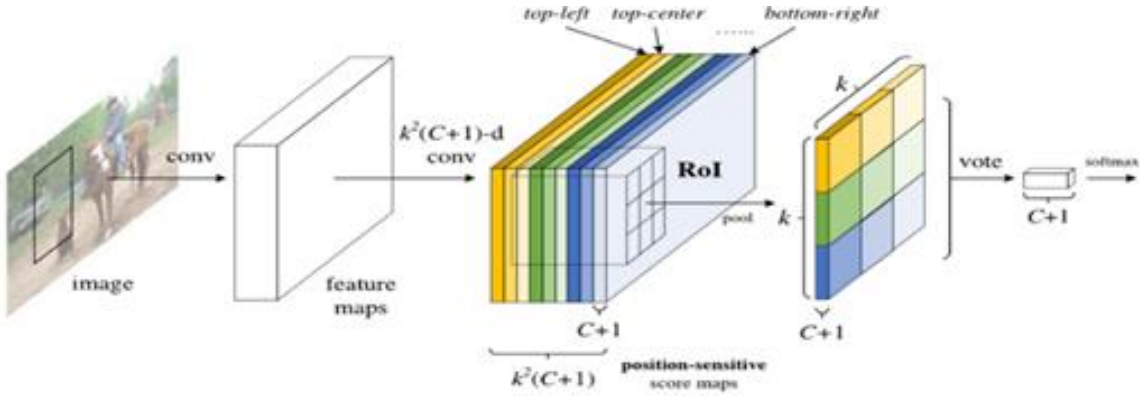


Рисунок 9 – Розпізнавання за допомогою моделі ResNet

Вхідне зображення передає модель ResNet для створення карток об'єктів. Модель RPN визначає область інтересів, і кожної області обчислюється оцінка, щоб визначити найбільш ймовірний об'єкт, якщо він є.

Mask Region Convolutional Network [4].

Ще одне розширення моделі Faster R-CNN доданої паралельної гілки до виявлення рамки, що обмежує, щоб передбачити маску об'єкта. Маска об'єкта – це його сегментація пікселів на зображенні. Ця модель перевершує сучасну в чотирьох задачах COCO: сегментація екземпляра, виявлення

рамки, що обмежує, виявлення об'єкта і виявлення ключової точки.

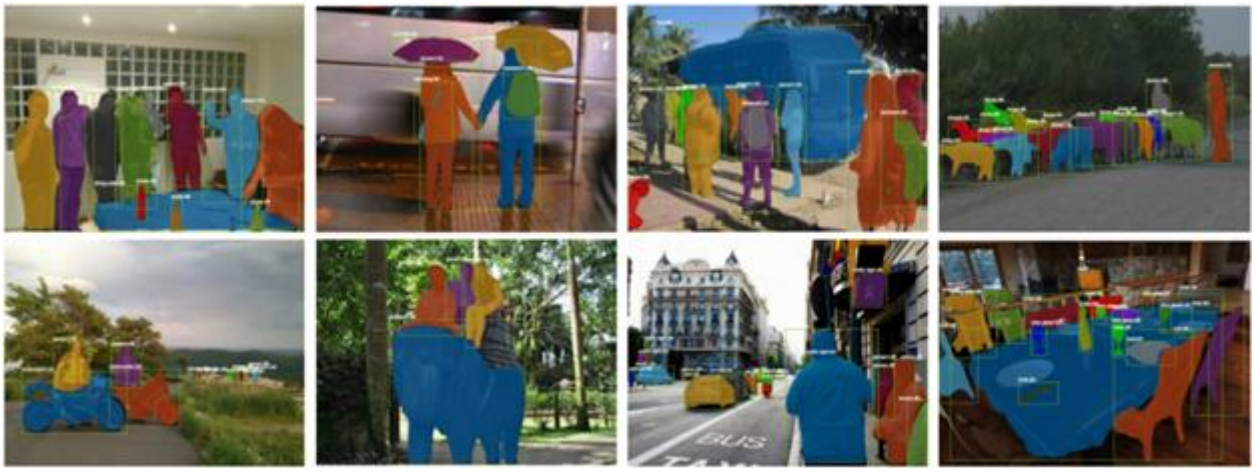


Рисунок 10 – Приклади застосування Mask R-CNN у тестовому наборі даних

Мережа на основі області маски використовує більш швидкий конвеєр R-CNN з трьома вихідними гілками для кожного об'єкта-кандидата: мітка класу, зміщення обмежуючої рамки та маска об'єкта. Він використовує мережу регіональних пропозицій для створення пропозицій обмежуючої рамки і одночасно створює три результати для кожної області, що цікавить.

Початковий шар RoIPool, що використовується у Faster R-CNN, замінюється шаром RoIAlign. Він видаляє квантування координат вихідної області інтересу та обчислює точні значення розташування. Шар RoIAlign забезпечує масштабну еквівалентність та трансляційну еквівалентність пропозиціям регіону.

Модель приймає зображення як вхідні дані і передає мережу ResNeXt зі 101 шаром. Ця модель схожа на ResNet, але кожен залишковий блок розрізається на більш легкі перетворення, які поєднуються для додавання розрідженості в блок. Модель виявляє області інтересу, які обробляються за допомогою рівня RoIAlign. Одна гілка мережі пов'язана з повним шаром для обчислення координат обмежують прямокутників і ймовірностей, пов'язаних з об'єктами. Інша гілка пов'язана з двома шарами згортки, останній обчислює

маску виявленого об'єкта.

Підсумовуються три функції втрат, пов'язані з кожним розв'язуваним завданням. Ця сума зведена до мінімуму і дає відмінні результати, оскільки розв'язання задачі сегментації покращує локалізацію та, отже, класифікацію.

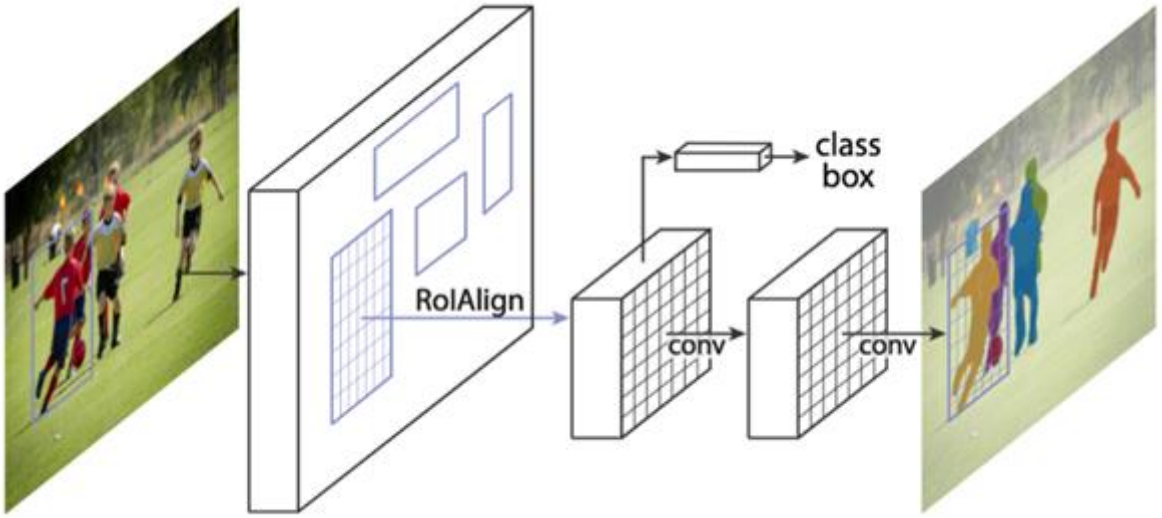


Рисунок 11 – Середовище Mask R-CNN для сегментації екземплярів

1.2.2 Одноетапні методи

Сімейство алгоритмів R-CNN використовує передбачення регіонів, що дозволяє забезпечувати хорошу точність, але може бути дуже повільним для деяких сфер, таких як безпілотне керування автомобілем. Можна виділити ще одне сімейство алгоритмів, що паралельно розвиваються, для детекції зображень, яке не використовує регіони – сімейство алгоритмів швидкої детекції.

You Only Look Once [5].

Модель YOLO (J. Redmon et al., 2016) безпосередньо передбачає обмежувальні рамки та ймовірності класів за допомогою однієї мережі в одній оцінці. Простота моделі YOLO дозволяє робити прогнози у реальному часі.

Спочатку модель приймає зображення як вхідні дані. Він поділяє його на сітку SxS. Кожен осередок цієї сітки передбачає B обмежують прямокутників з показником достовірності. Ця впевненість є просто ймовірністю виявлення об'єкта, помножену на IoU між передбаченим і наземним полем істинності.

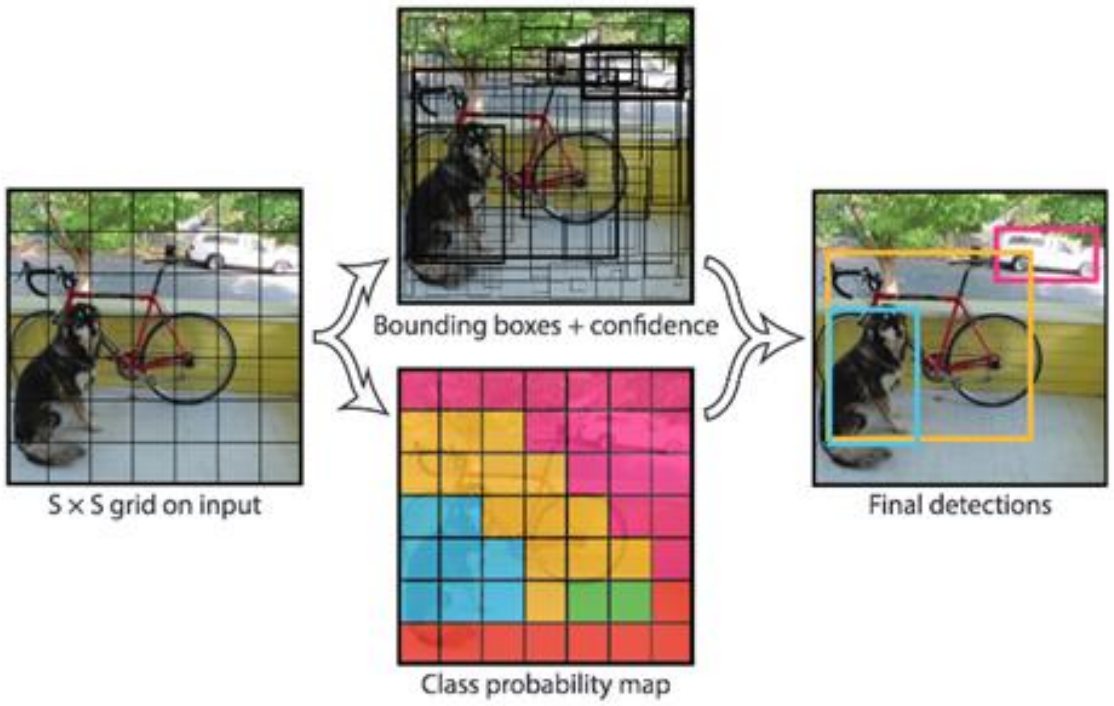


Рисунок 12 – Приклад застосування моделі YOLO

Використовувана CNN натхненна моделлю GoogLeNet, в якій представлені початкові модулі. Мережа має 24 згорткових шари, за якими слідують 2 повнозв'язкові шари. Шари скорочення з фільтрами 1x1⁴, за якими йдуть згорткові шари 3x3, замінюють початкові початкові модулі. Модель Fast YOLO – це легша версія, в якій всього 9 згорткових шарів і менше фільтрів. Більшість згорткових шарів попередньо навчені з використанням набору даних ImageNet із класифікацією.

Останній шар виводить тензор S * S * (C + B * 5), що відповідає прогнозам для кожної комірки сітки. C – кількість ймовірностей для кожного

класу. B – фіксована кількість блоків прив'язки на комірку, кожен з цих блоків пов'язаний з 4 координатами (координати центру блоку, ширина та висота) та довірчим значенням.

У попередніх моделях передбачені рамки, що обмежують, часто містили об'єкт. Однак модель YOLO передбачає велику кількість рамок, що обмежують. Таким чином, є багато рамок, що обмежують, без будь-якого об'єкта. Метод максимального придушення застосовується в кінці мережі. Він полягає в об'єднанні обмежуючих рамок одного і того ж об'єкта в одну, що сильно перекриваються.

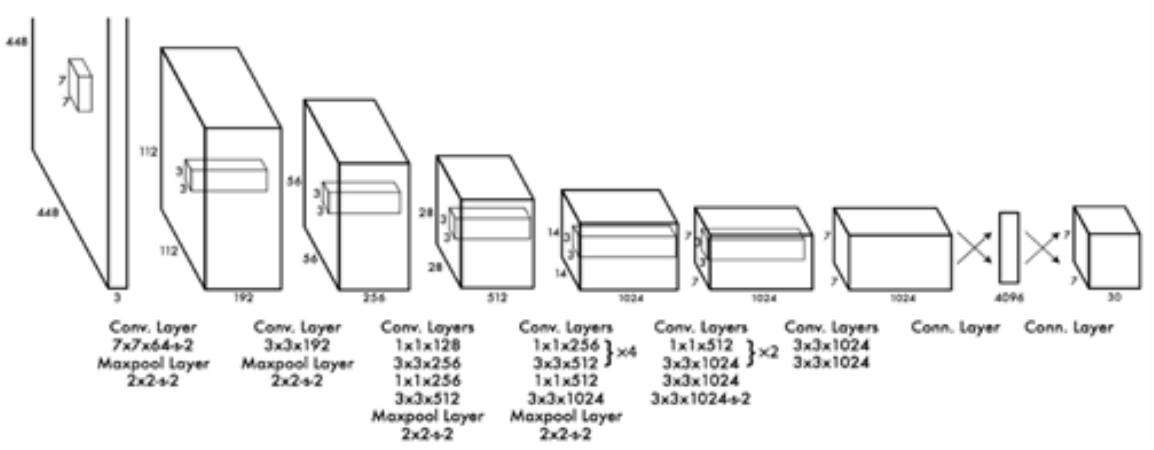


Рисунок 13 – Архітектура YOLO: 24 згорткових шари та 2 повнозв'язкові шари

Плюси методу YOLO:

- швидкість: цей алгоритм підвищує швидкість виявлення, оскільки може прогнозувати об'єкти у режимі реального часу;
- висока точність: YOLO – це метод прогнозування, який забезпечує точні результати з мінімальними помилками фону;
- YOLO може узагальнювати зображення без навантаження пам'яті обробки.

Мінуси методу YOLO:

– YOLO страждає від значно більшої кількості помилок локалізації та має проблеми з ідентифікацією найближчих предметів.

Single-Shot Detector.

Подібно до моделі YOLO, W. Liu et al. (2016) розробили одноразовий детектор для одночасного прогнозування всіх обмежувальних рамок та ймовірностей класів за допомогою наскрізної архітектури CNN.

Як вхідні дані модель приймає зображення, яке проходить через кілька згорткових шарів з різними розмірами фільтрів (10x10, 5x5 і 3x3). Карти об'єктів зі згорткових шарів у різних положеннях мережі використовуються для прогнозування рамок, що обмежують. Вони обробляються спеціальними шарами згортками з фільтрами 3x3, званими додатковими шарами об'єктів, для створення набору обмежуючих рамок, подібних до якірних рамок Fast R-CNN.

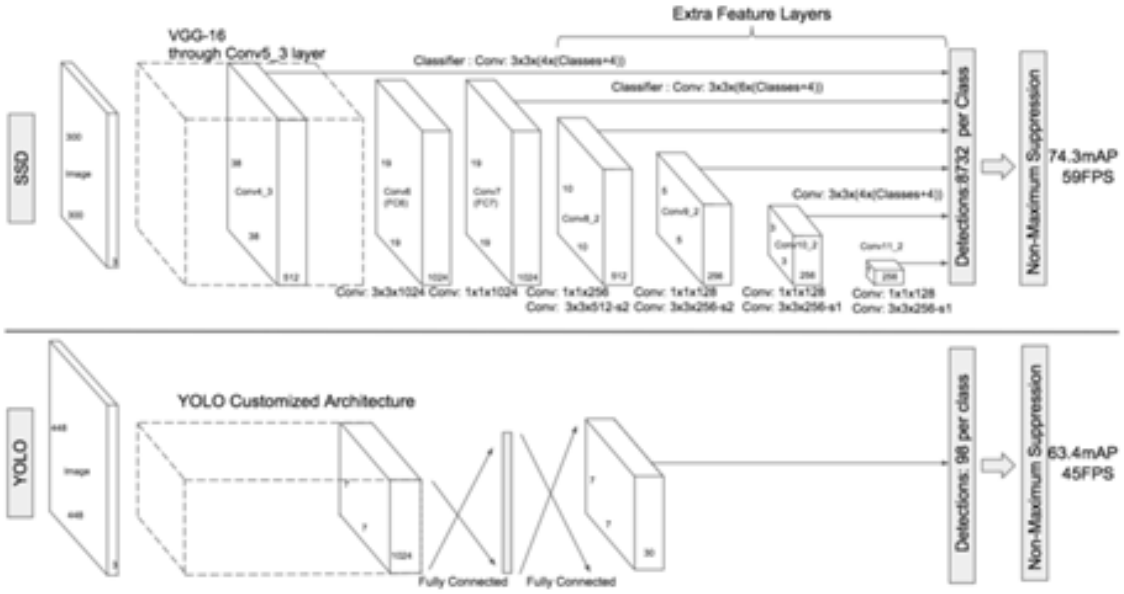


Рисунок 14 – Порівняння архітектур SSD та YOLO

Кожен «ящик» має 4 параметри: координати центру, ширину та висоту. У той самий час він створює вектор ймовірностей, що відповідає довірі до

кожного класу об'єктів.

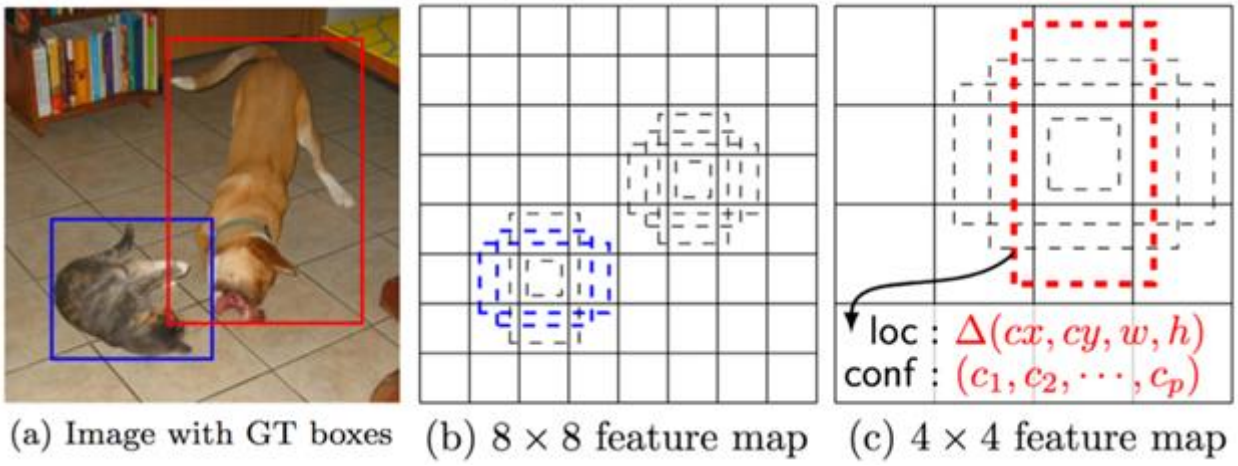


Рисунок 15 – Приклад роботи методу SSD

(a) Модель бере зображення та його рамки, що обмежують

(b) Невеликі набори блоків із різним співвідношенням сторін фіксуються іншою картою ознак

(c) Під час навчання локалізація ящиків змінюється, щоб максимально відповідати дійсності

Метод немаксимального придушення також використовується в кінці моделі SSD, щоб зберегти найбільш релевантні рамки, що обмежують. Потім використовується Hard Negative Mining, оскільки все ще прогнозується багато негативних полів. Він полягає у виборі лише частини цих блоків під час навчання. Ящики впорядковані за достовірністю, а вершина вибирається залежно від співвідношення між негативним та позитивним значенням, яке не перевищує 1/3 [5].

Мінуси методу SSD:

– ступінь точності SSD трохи знижується при ідентифікації дрібніших речей. Якщо модель дуже велика, швидкість може значно впасти.

1.3 Порівняння алгоритмів розпізнавання образів

Найбільш популярним зразком є набір даних Microsoft COCO. Різні моделі зазвичай оцінюються відповідно до показника середньої точності. Далі ми порівняємо найкращі алгоритми виявлення об'єктів у реальному часі. Важливо, що вибір алгоритму залежить від варіанта використання та застосування; різні алгоритми чудово справляються з різними завданнями (наприклад, Beta R-CNN показує найкращі результати виявлення пішоходів).

Найкращим алгоритмом виявлення об'єктів у реальному часі в 2022 році є YOLOV7, за яким слідує Vision Transformer, такий як Swin і DualSwin, PP-YOLOE, YOLOR, YOLOV4 і EfficientDet.

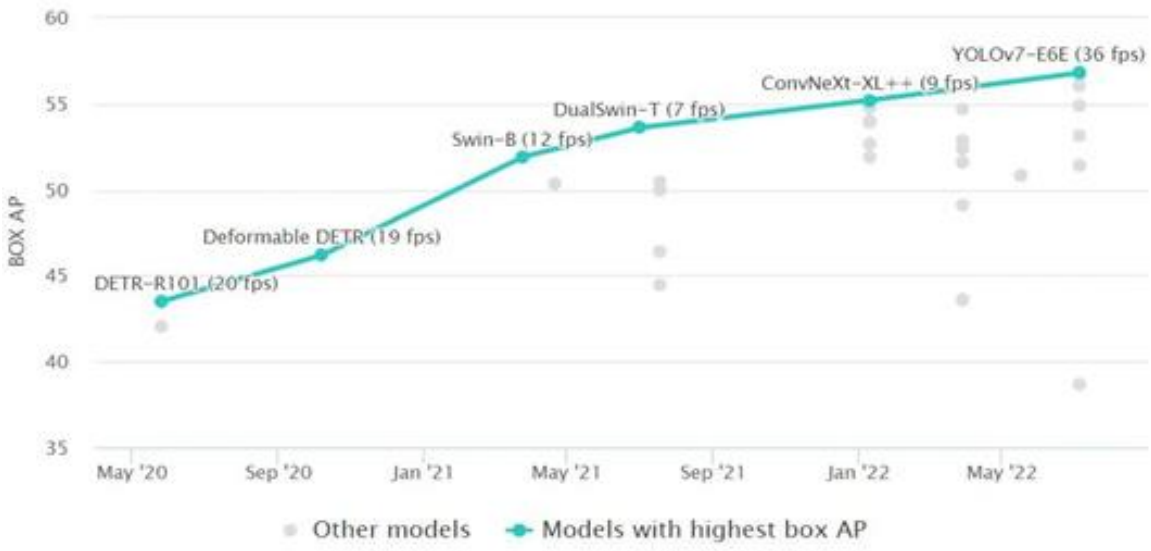


Рисунок 16 – Графік коефіцієнта точності та швидкості різні алгоритмів

Виявлення об'єктів у реальному часі в тесті COCO: найсучасніше за середньою точністю (AP), найшвидший алгоритм виявлення об'єктів у реальному часі (час виведення). Крім того, у наборі даних MS COCO важливим показником тесту є час виведення (мс/кадр, чим менше, тим краще) або кадрів на секунду (кадрів на секунду, чим вище, тим краще).

Швидкий прогрес у технології комп'ютерного зору дуже помітний у порівнянні часу логічного висновку. Ґрунтуючись на поточному часі виведення (чим менше, тим краще), YOLOV7 досягає 3,5 мс на кадр у порівнянні з YOLOV4 12 мс або популярним YOLOV3 29 мс. Зверніть увагу на те, як введення YOLO (одноетапний детектор) призвело до значного скорочення часу виведення порівняно з будь-якими раніше встановленими методами, такими як двоетапний метод Mask R-CNN (333 мс). З технічного погляду досить складно осмислено порівнювати різні архітектури та версії моделей.

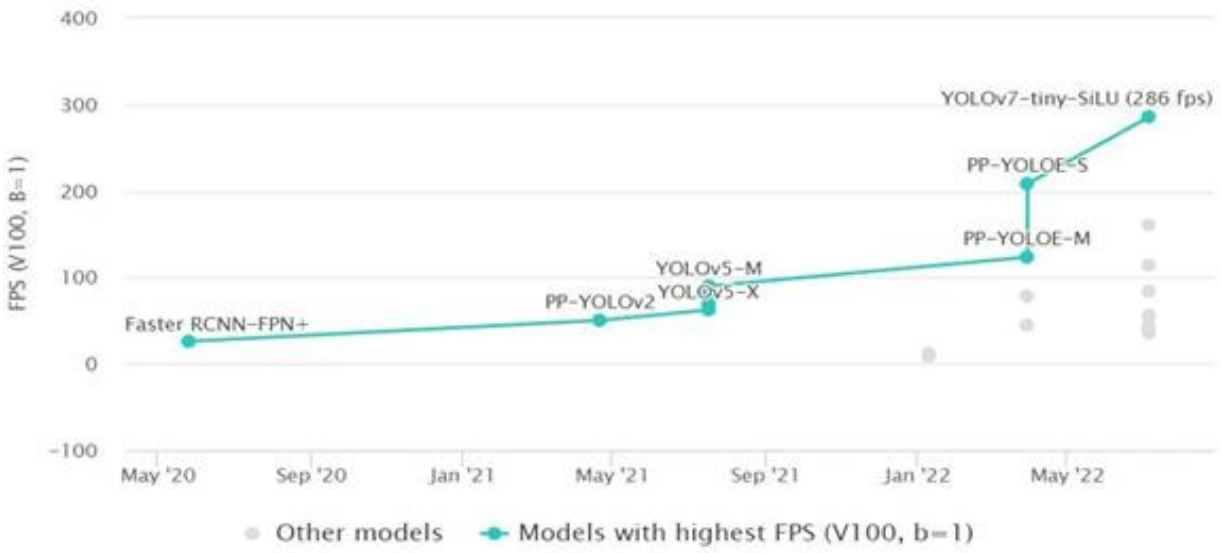


Рисунок 17 – Графік із порівняннями алгоритмів YOLO різних версій із алгоритмом Faster RCNN-FPN+

1.4 Порівняльний аналіз систем для розпізнавання математичних формул

«Microsoft Math Solver» [6].

Цей додаток використовує нейронну мережу для розпізнавання рівняння та подальшого пошуку його рішень на основі її передбачень. Тип нейронних мереж, що використовуються, не вказаний розробником.

Програма існує для різних платформ: iOs, android, web-додаток. У випадку, коли користувач використовує мобільний додаток, зображення надходить із камери смартфона і далі обробляється. Також завдання можна ввести в рукописному режимі. Приклад роботи програми представлено рисунку 18.

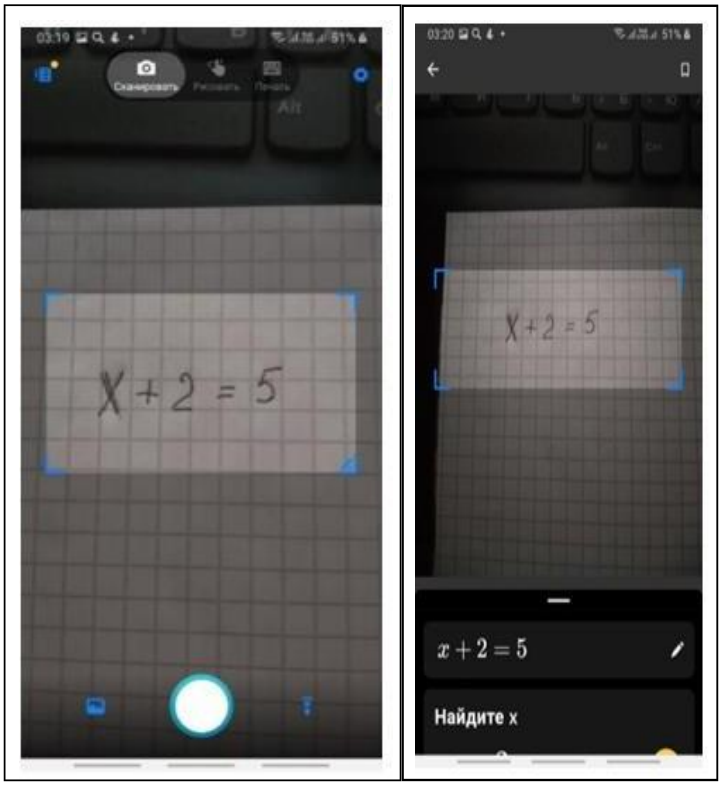


Рисунок 18 – MathSolver: сканування рівняння і отримання результату

«PhotoMath» [7].

Цей мобільний додаток має такі функції: сканування рівняння; відправлення рівняння до хмарного сховища; видати результат. У хмарному сховищі рівняння розпізнається високоточною нейронною мережею та ведеться пошук розв'язання. Тип нейронних мереж, що використовуються, не вказаний розробником. Програма не може працювати без підключення до Інтернету. Приклад роботи представлений рисунку 19.

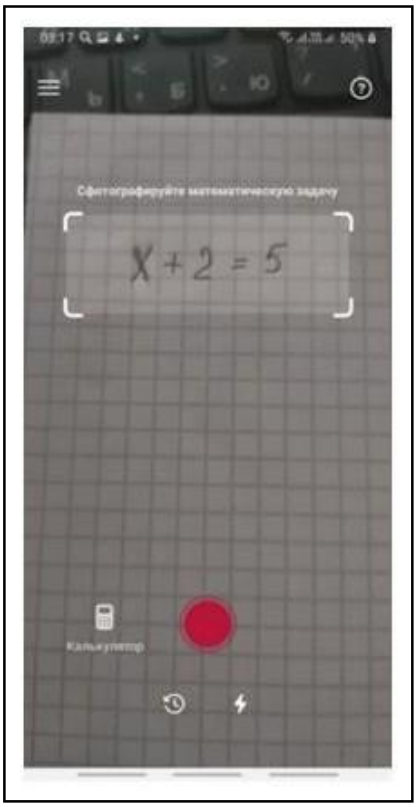


Рисунок 19 – PhotoMath. Скан рукописного рівняння з аркуша паперу

«OneNote» [8].

OneNote є настільною програмою, в якій присутня функція перекладу рукописного введення математичних рівнянь у друкований текст.

У перетворенні рівнянь OneNote використовує оптичне розпізнавання символів. Оптичне розпізнавання символів – це процес перекладу тексту зображення у текстовий формат. Приклад роботи програми представлений рисунку 20.

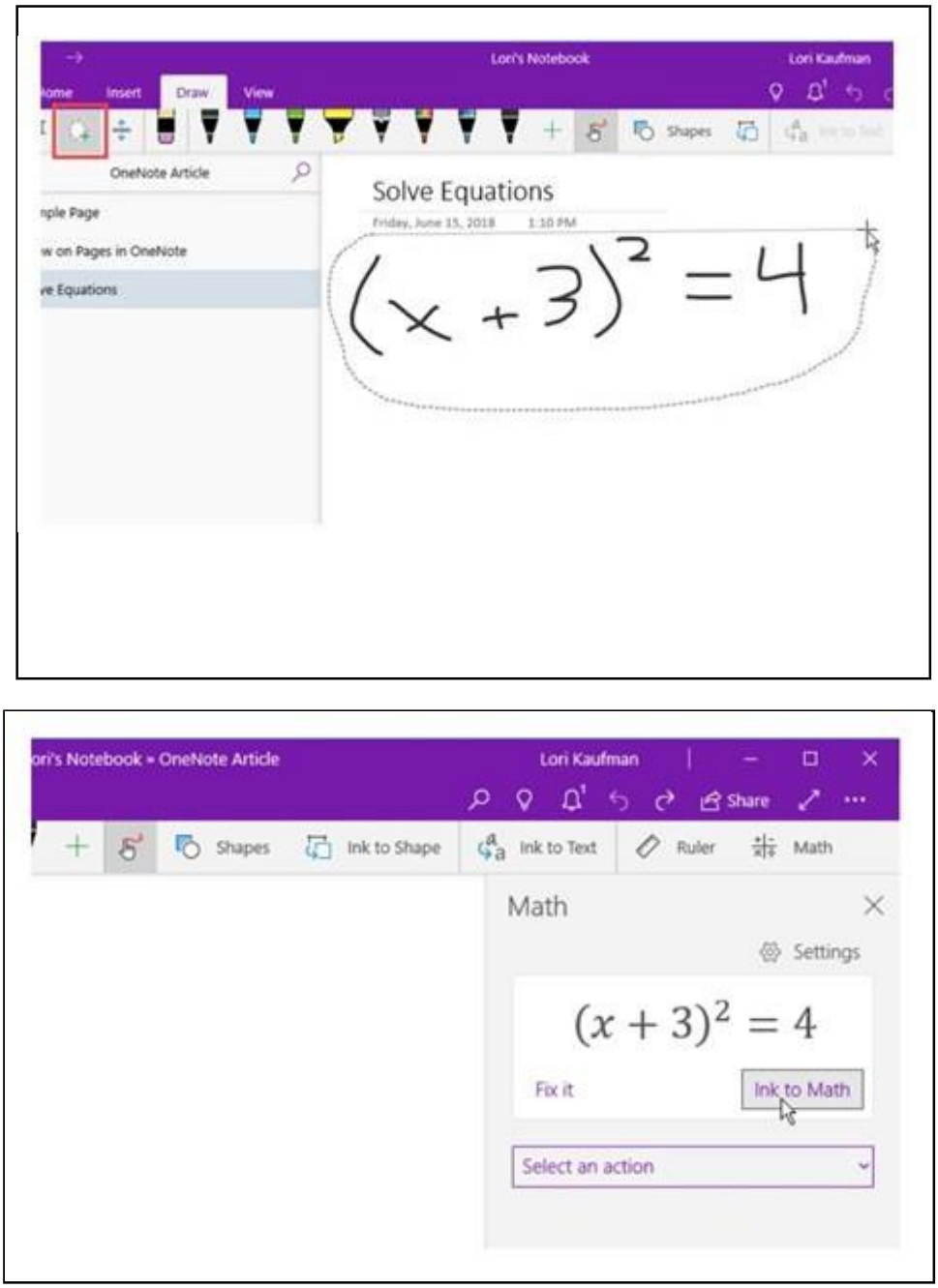


Рисунок 20 – Робота програми OneNote

Проведений огляд аналогів показав, що ці додатки є дуже актуальними. Вони реалізовані у форматі мобільних та веб-додатків.

1.5 Постановка задачі

Метою магістерської кваліфікаційної роботи є розробка програми для

розпізнавання математичних формул з використанням згорткових нейронних мереж. Для досягнення поставленої мети необхідно вирішити такі завдання:

- підготувати два навчальні набори даних;
- реалізувати дві нейронні мережі, провести їх навчання та тестування;
- спроектувати та реалізувати додаток для розпізнавання математичних формул;
- провести тестування реалізованої програми.

2 СТВОРЕННЯ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ РОЗПІЗНАВАННЯ ФОРМУЛ ТА СИМВОЛІВ

2.1 Топологія нейронної мережі

Згорткові нейронні мережі дуже сильні у сприйнятті формул і символів способами, які допомагають в автоматичному вилученні функцій та роблять CNN найбільш підходящою технікою для вирішення завдань цього класу.

Наведемо алгоритм створення нейронної мережі.

Попередня обробка.

Кодування даних.

Побудова моделі. Після кодування даних зображення та мітки готові для встановлення у модель. Модель складається з вилучення ознак за допомогою згортки та двійкової класифікації. Згортка і максимальний пул проводиться для вилучення особливостей зображення, згорткові фільтри застосовуються до зображення, за яким слідує шар максимального пула, за яким слідує ще один шар згортки з фільтрами. У результаті отримуємо зображення для вирівнювання. Вирівнюючий шар згладить зображення в ряд значень, які будуть зіставлені щільному шару з нейронів, з'єднаних з вихідним шаром нейронів [9]. Схема дії нейронної мережі представлена рисунку 21.

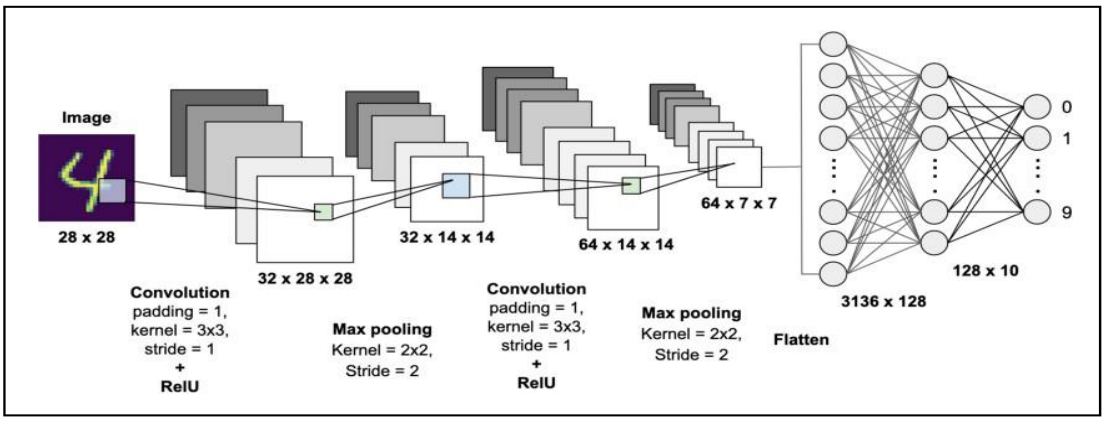


Рисунок 21 – Схема дії згорткової нейронної мережі

2.2 Розпізнавання формули

Спочатку відбувається підготовка зображення. Зокрема зображення бінаризується. Потім відбувається пошук зв'язкових компонентів у зображенні. Занадто маленькі за розміром забираються. Далі кожен символ готується до розпізнавання, і подається на розпізнавання нейронної мережі. Результати розпізнавання нейронної мережі збираються, і відбувається постобробка символів. На даному етапі якраз і збирається формула. Далі розпізнана формула перетворюється на потрібний формат і виводиться користувачеві.

Розглянемо детальніше кожен етап.

Підготовка зображення.

На цьому етапі відбувається попереднє покращення вихідного зображення. Воно бінаризується і з нього забирається зайвий шум.

Алгоритм сегментування.

Виділити символи з фону.

Після того, як були виділені всі області зв'язності, деякі з них відкидаються, зокрема, ті, які занадто маленькі за розміром (шум).

Потім для кожного символу визначається його положення, межі, точка центру. Ця інформація буде використана надалі.

Підготовка символу до розпізнавання.

А, щоб розпізнати зображення із символами з допомогою нейронної мережі, їх слід підготувати, тобто перетворити на потрібний формат.

Процес підготовки символу до розпізнавання можна описати так.

Символ вирізається по краях вихідного зображення.

Зображення перетворюється на квадратне по більшому краю.

Розміри отриманого зображення перетворюються на потрібний формат.

По краям додається порожнє простір і розмір збільшується до необхідної кількості пікселів.

Описаний процес представлений рисунку 22.

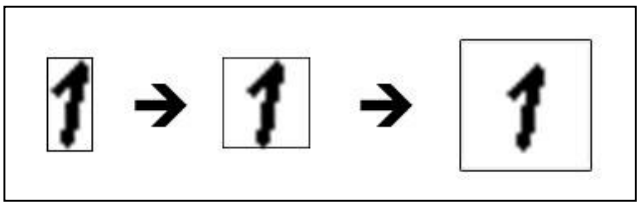


Рисунок 22 – Перетворення символу

Отримане зображення перетворюється на одновимірний масив елемента) і нормується, тобто значення від 0 до 255 перетворюються на значення в інтервалі від 0 до 1.

Розпізнавання символів.

На даному етапі відбувається розпізнавання кожного окремого символу за допомогою нейронної мережі.

Визначення положення символу у формулі.

Після того, як символи розпізнано, проводиться їх додаткова обробка.

Символи сортуються ліворуч, і далі оцінюється їх приналежність до ступеня або до нижнього індексу. Для цього обчислюється кут між центрами поточного та попереднього символу.

Якщо кут більше 30 і менше 60 градусів, це ступінь.

Якщо кут більший від -60 і менше від -30 градусів, то це нижній індекс.

Візуально це можна побачити рисунку 23.

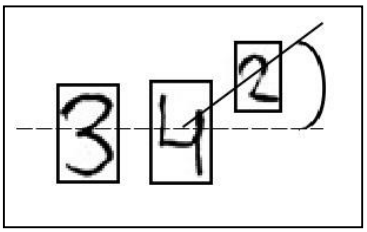


Рисунок 23 – Визначення положення за кутом

Далі обробляються всі символи, розпізнані символом «мінус». І якщо

два мінуси йдуть поспіль, причому вони знаходяться один під одним, то два дані символу перетворюються на один – «рівно» [11].

З'єднання окремих символів у формулу.

Після того, як були опрацьовані всі символи, необхідно їх зібрати в одну формулу.

Символи збираються послідовно, зліва направо. До кожного символу визначається положення у формулі.

3 ПРОЕКТУВАННЯ СИСТЕМИ РОЗПІЗНАВАННЯ

Проектування програмного забезпечення – процес створення проекту програмного забезпечення.

Метою проектування є визначення внутрішніх властивостей системи та деталізації її зовнішніх (видимих) властивостей на основі виданих замовником вимог до програмного забезпечення (вихідні умови завдання). Ці вимоги аналізуються.

У процесі проектування створюється проектна документація, куди входять текстові описи, діаграми, моделі майбутньої програми. Тут допомагає мова UML. UML – графічна мова для візуалізації, опису параметрів, конструювання та документування різних систем (програм, зокрема). Діаграми створюються за допомогою спеціальних CASE засобів (набір інструментів та методів програмної інженерії для проектування програмного забезпечення, що допомагає забезпечити високу якість програм, відсутність помилок та простоту в обслуговуванні програмних продуктів). На основі технології UML будується єдина інформаційна модель. CASE засоби здатні генерувати код різними об'єктно-орієнтованими мовами [12].

3.1 Визначення вимог до системи

Вимоги до класів нейронної мережі.

До кінцевого набору класів повинні входити:

- цифри;
- літери латинського алфавіту верхнього і нижнього регістрів;
- математичні символи: +, -, =.

Функціональні вимоги до системи розпізнавання формул.

На підставі аналізу предметної області було визначено такі функціональні вимоги до системи, що розробляється:

- користувач повинен мати можливість вибрати та завантажити зображення;
- користувач повинен мати можливість отримати готовий результат;
- система повинна вимагати від користувача лише зображення та відкидати інші формати файлів;
- система повинна завантажувати зображення користувача;
- система повинна розпізнавати символи на зображенні і формувати їх у готову формулу у текстовому вигляді.

Нефункціональні вимоги.

Нефункціональні вимоги – опис властивостей або характеристик, які система повинна демонструвати, або обмеження, яких вона повинна дотримуватися, на відміну від поведінки системи, що спостерігається (зручність використання, безпека, надійність, розширюваність і т.д.).

Система має бути реалізована мовою Python.

Система має використовувати нейронні мережі для розпізнавання математичних формул.

3.2 Проектування діаграми варіантів використання системи

Для проектування програми була використана мова графічного опису об'єктно-орієнтованого програмування UML. Було побудовано модель взаємодії зовнішнього актора з програмною системою у вигляді діаграми варіантів використання.

Діаграма варіантів використання – діаграма, що описує, який функціонал програмної системи, що розробляється, доступний кожній групі користувачів.

Отримана діаграма зображено рисунку 24.

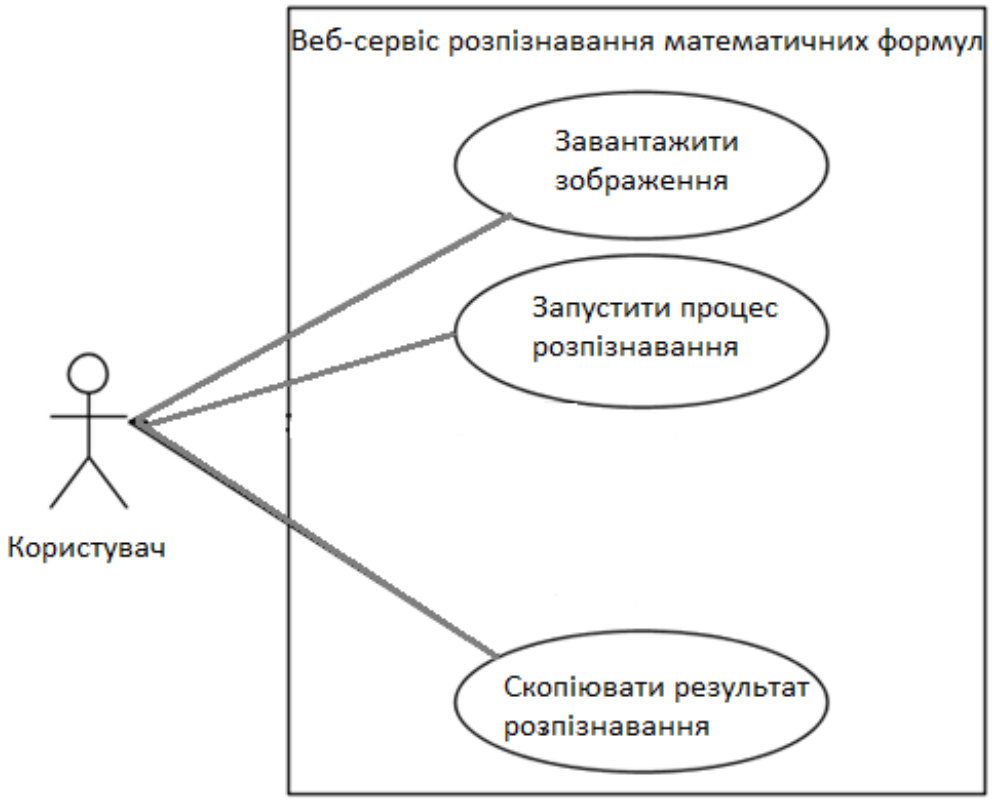


Рисунок 24 – Діаграма варіантів використання

На представленій діаграмі зображено 1 актор. Розглянемо детальніше.

Користувач – людина, що користується системою і використовує певні функції.

До кожного актора визначено свої варіанти використання.

Користувачеві доступні такі функції:

– завантажити зображення – користувач вибирає зображення для розпізнавання та завантажує його на сервер;
– запустити процес розпізнавання завантаженого файлу – користувач натискає кнопку «Розпізнати» і починається процес розпізнавання;
– скопіювати результат розпізнавання – користувач може скопіювати результат розпізнавання у спеціальних текстових полях.

3.3 Проектування діаграми діяльності системи

Діаграми діяльності можна використовувати на всіх етапах розробки програмного забезпечення та для різних цілей. І оскільки вони дуже схожі на блок-схеми, вони зазвичай популярніші за інші типи. Діаграма активності UML дозволяє детальніше візуалізувати конкретний випадок використання. Це діаграма поведінки, яка ілюструє потік діяльності через систему.

На основі вимог до системи було розроблено діаграму діяльності, представлену на рисунку 25.

Ця діаграма показує, як відбувається процес взаємодії користувача із системою.

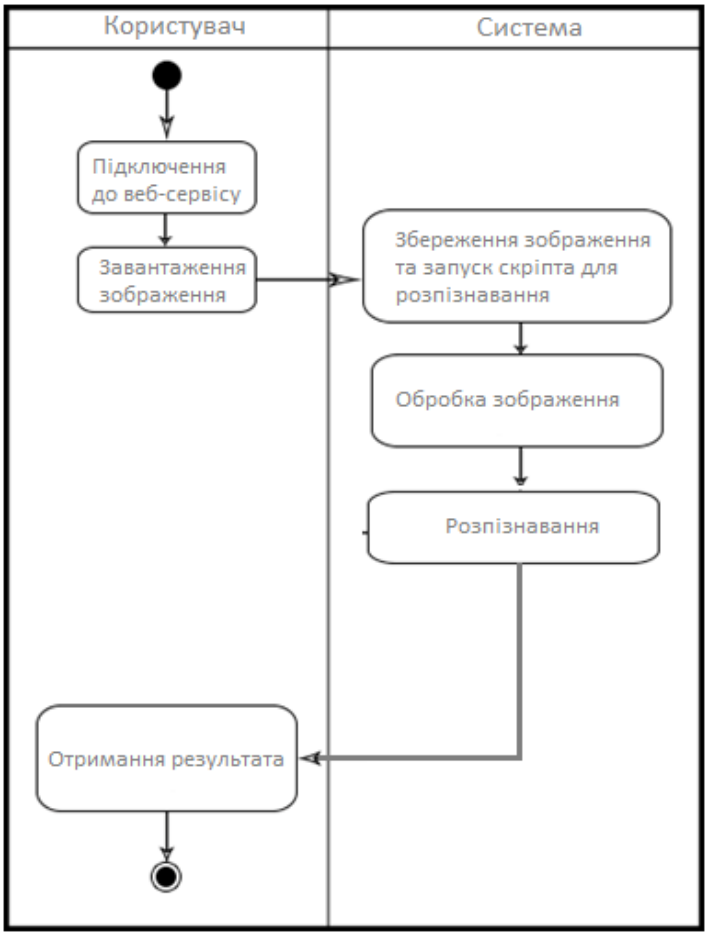


Рисунок 25 – Діаграма діяльності

3.4 Проектування діаграми структури системи

Діаграми компонентів UML дають концептуальну картину взаємодії між різними системами. Можуть бути як аспекти логічного, і фізичного моделювання. Більше того, компоненти автономні. Це елемент модульної системи UML, який можна замінити на альтернативний. Вони містять конструкції будь-якої складності та є автономними. Тільки через інтерфейси вкладені частини взаємодіють із іншими компонентами. Крім того, компоненти мають свої інтерфейси, але вони також можуть отримувати доступ до операцій та служб інших компонентів, використовуючи свої інтерфейси. На діаграмі компонентів інтерфейси також показують зв'язки та залежності в архітектурі програмного забезпечення.

У систему включені основні компоненти:

- Main;
- DetectFormula;
- DetectSymb.

Діаграма компонентів системи представлена рисунку 26.

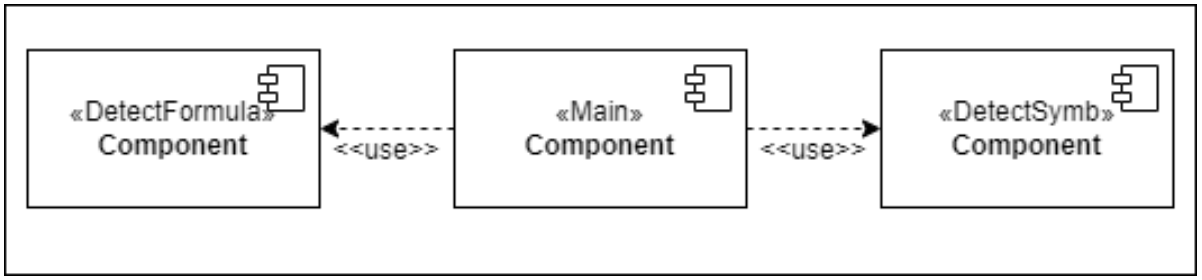


Рисунок 26 – Діаграма компонентів

Main – головний модуль, який ініціює взаємодію з користувачем та роботу системи загалом. У цьому модулі прописано функцію перекладу умовних позначень у загальноприйняті. Написано взаємодію з телеграм-ботом, обробка помилок та основний алгоритм обробки зображень та підключення нейронних мереж.

DetectFormula – це компонент системи, який виявляє цілі формули. Також мною були додані команди, за допомогою яких виявлені зображення «вирізуються» і поміщаються в папку з результатом для подальшого використання другою нейронною мережею.

DetectSymb – компонент системи, який детектує символи у формулі, яка була виявлена першою нейронною мережею, має в своєму розпорядженні в потрібному порядку, за допомогою додатково написаних функцій. Також є функція, яка очищає результат від зайвої інформації, наприклад, подвійне виявлення одного і того ж символу або детекція крапки з комою в кінці виразу, та функція визначення ступеня у виразі.

3.5 Проектування графічного інтерфейсу користувача

Графічний інтерфейс користувача надає візуальне подання програми або системи, що робить взаємодію з нею інтуїтивнішим і зручнішим для користувача. GUI включає елементи, такі як вікна, кнопки, поля введення та інші графічні елементи, які дозволяють користувачам виконувати різні операції без необхідності вводити команди вручну.

Найважливіші властивості GUI-інтерфейсу – це можливість безпосереднього маніпулювання, підтримка миші або покажчика, використання графіки та наявність області для функцій та даних програми [13].

Система розпізнавання рукописних математичних формул буде веб-сервіс. За допомогою цього сервісу користувач матиме можливість завантажити зображення, розпізнати на ньому формули та отримати готовий результат.

Інтерфейс веб-сервісу має бути простим та зрозумілим, але при цьому повинен повністю забезпечувати необхідну функціональність.

Інтерфейс веб-сервісу складається з 3 основних частин: шапка сайту, основна секція та підвал. Основна секція містить поле для завантаження

зображення користувачем і результат розпізнавання. Результат виводиться як таблиці, у якій містяться: вихідне зображення, результат у відповідному полі.

Макет графічного інтерфейсу користувача зображено рисунку 27.

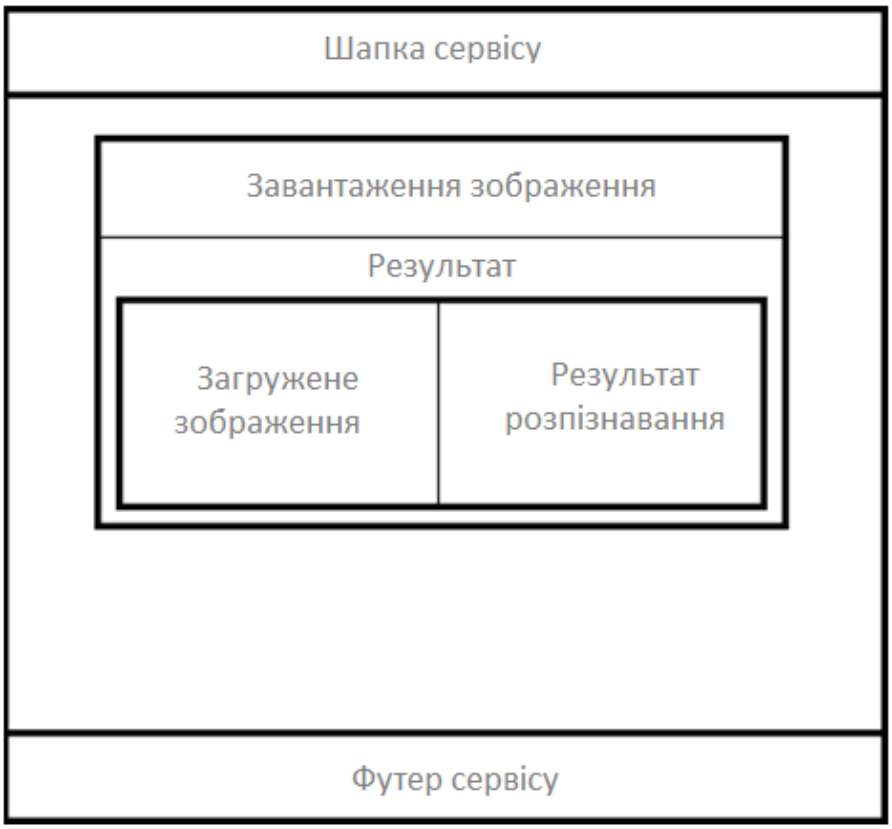


Рисунок 27 – Макет графічного інтерфейсу

4 РЕАЛІЗАЦІЯ СИСТЕМИ

4.1 Вибір програмних засобів реалізації

Використана мова програмування – Python з версією 3.9.

Для мови програмування Python існує велика кількість готових бібліотек для роботи з нейронними мережами. Використання цих бібліотек покликане значною мірою спростити завдання розробки нейронних мереж. Далі у цьому розділі будуть розглянуті найпопулярніші з них.

TensorFlow [15].

Дуже популярна бібліотека Google. Документація для цієї бібліотеки добре опрацьована. Також дуже багато статей у мережі Інтернет та прикладів нейронних мереж для даної бібліотеки. Для представлення нейронної мережі тут використовуються графи, а зберігання даних багатомірні масиви – тензоры.

Theano[16].

Інша не менш популярна бібліотека для Python. Бібліотека отримала свою назву на честь імені дружини давньогрецького філософа та математика Піфагора – Феано (або Теано). Так само, як і в інших бібліотеках, є інтеграція з бібліотекою NumPy []. Є можливість проведення обчислень як на CPU, так і на GPU.

Keras[17].

Бібліотека, а якщо точніше, фреймворк для створення нейронних мереж. Keras є надбудовою над TensorFlow та Theano і може використовувати будь-яку з цих двох бібліотек для проведення обчислень. З усіх бібліотек ця найбільш зручна та проста для розуміння. Нові шари тут додаються лише за допомогою однієї функції. Хороша документація та безліч готових прикладів – все це є великою перевагою для Keras.

Для виконання роботи було підключено бібліотеки, наведені нижче.

Google Colaboratory [18].

Хмарне середовище для роботи мовою Python у браузері.

PyCharm [19].

Середа на Python, що використовується для розробки та налагодження проекту. Версія середовища – community edition 2022.3.3.

Roboflow [20].

Пакет для роботи з веб-сервісом Roboflow, що дозволяє виводити список інформації про робочі області, проекти, версії; завантажувати зображення у свої проекти; візуалізувати та зберігати прогнози, які були зроблені для моделі.

Matplotlib 3.2.2 [21].

Бібліотека для створення статистичних, анімованих та інтерактивних візуалізацій на Python.

Numpy 1.18.5 [22].

Бібліотека за допомогою багатовимірних масивів, матриць, високорівневих математичних функцій для операцій із масивами.

Opencv-python 4.1.1 [23].

Бібліотека для комп'ютерного зору, призначена для роботи із зображеннями. Використовується для завантаження та коректного відображення зображення у системі.

Pillow 7.1.2 [24].

Бібліотека для роботи з обробкою зображення.

PyYAML 5.3.1 [25].

Бібліотека для роботи із файлами формату yaml. Файл із розширенням .yaml використовується для налаштування набору даних.

Torch 1.7.0. [26].

Бібліотека використовується для розробки та навчаннянейронної мережі, що базується на моделі глибокого навчання.

Torchvision 0.8.1 [27].

Бібліотека з популярними наборами даних, архітектурами моделей та загальними перетвореннями зображень для комп'ютерного зору.

Tqdm 4.64.0 [28].

Бібліотека для відображення рядка прогресу.

Tensorboard 2.4.1 [29].

Інструмент для забезпечення візуалізації метрик, отриманих у результаті навчання нейронної мережі.

Pandas 1.1.4 [30].

Бібліотека аналізу даних, необхідна для структурованої візуалізації даних у процесі навчання.

Seaborn 0.11.0 [31].

Бібліотека для візуалізації даних. Використовується для відображення результатів навчання.

4.2 Підготовка та робота з навчальною вибіркою

Початковий набір даних.

Для того, щоб нейронна мережа визначала елементи ланцюжка математичних символів, потрібно розділити роботу на два етапи:

- розпізнавання повноцінної формули;
- детекція кожного символу із знайденої області.

Для роботи алгоритму використовується два, окремо розмічені, набори даних: формули та математичні символи. У наборах є лише друковані символи. Спочатку в наборі даних з формулами було 321 зображення і 1642 розмічені області, в наборі даних із символами відповідно 544 зображення і 6664 області. Так само обидва набори були почищені від невірних та зайвих даних, таких як «other_class» – 1781 область.

Для розмітки всіх даних використовувалася платформа Roboflow [32].

Приклад інтерфейсу платформи представлений рисунку 28.

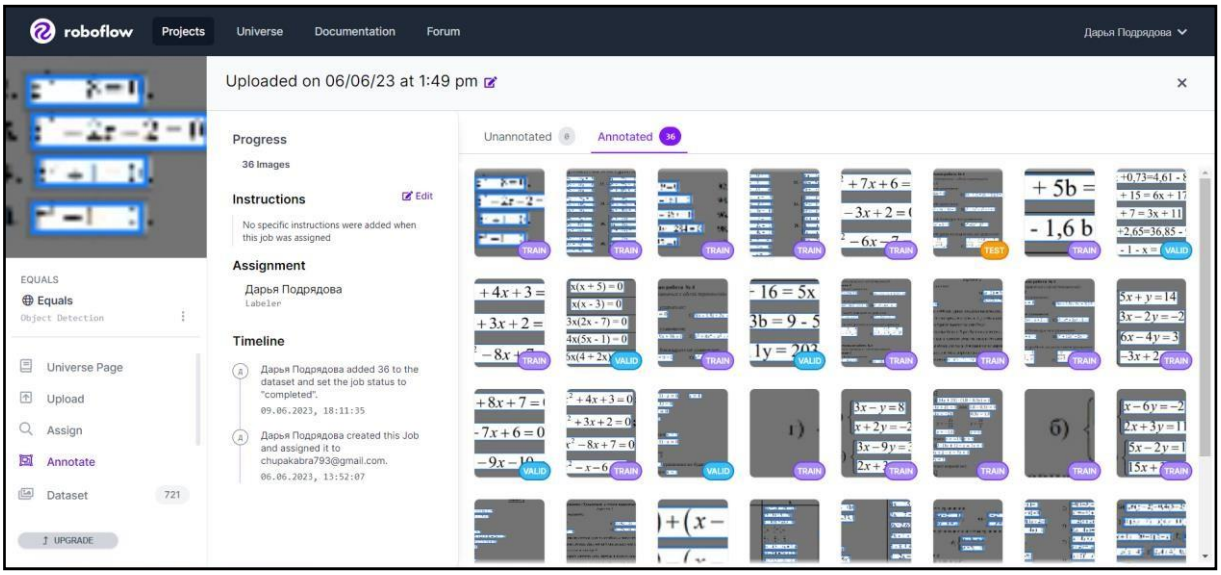


Рисунок 28 – Інтерфейс платформи Roboflow

Збір даних.

Набір даних із цілими формулами.

Дані для навчального набору даних були зібрані вручну та з відкритого джерела – платформи для розмітки Roboflow [33].

Приклад аркуша з лінійними рівняннями, створеними вручну, представлений рисунку 29.

а) x +3y = 1;	в) 2x – 5y = 7;
б) y – 2x + 6 = 0;	г) 5x + 3y -2 = 0.

Рисунок 29 – Приклад аркуша з рівняннями

З джерела було взято зображення з чотирма різними класами рівнянь: тригонометрія, межі, інтеграли та диференціали.

Всі запозичені картинки були перевірені на якість та достовірність розмітки. Приклад зображення з інтегральним виразом представлений

рисунку 30.

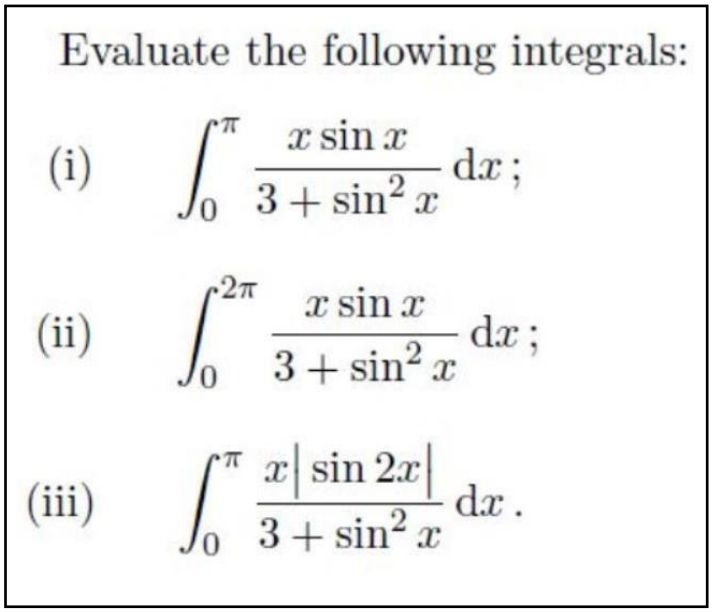


Рисунок 30 – Приклад зображення інтегрального виразу з відкритого джерела

Підсумковий навчальний набір даних складається з виразів із тригонометрією, диференціалами, інтегралами, межами та лінійними рівняннями. Загальна кількість зображень становила 721 штука.

Набір даних із окремими символами.

З джерела з готовими наборами даних було взято зображення з різними класами символів:

- числа 0-9;
- деякі змінні;
- синус, косинус, тангенс, котангенс;
- математичні оператори (+, -, *, /, =)

Приклад зображення з вихідного набору даних представлено рисунку 31.

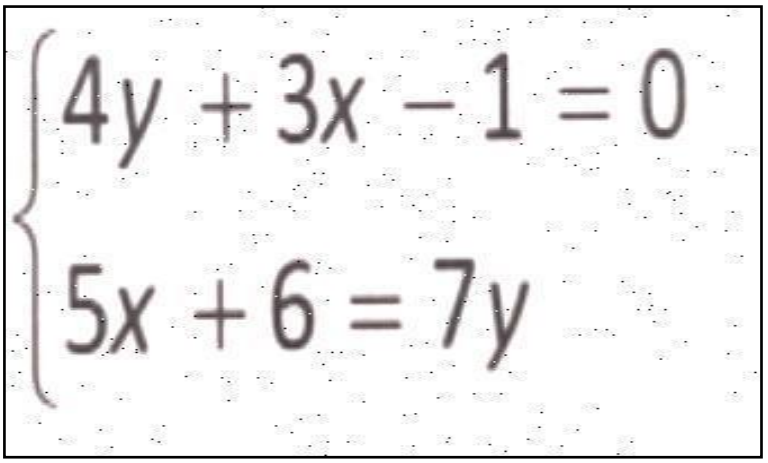


Рисунок 31– Приклад зображення з набору даних із символами

Для набору даних з окремими символами також було додано зображення. Приклад зображення з друкованими символами, яке додавали додатково представлений на рисунку 32.

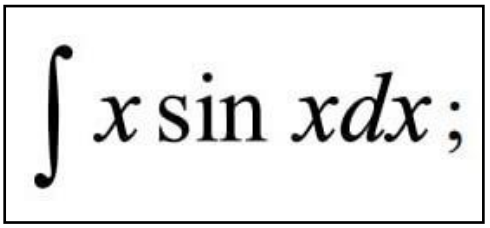


Рисунок 32 – Приклад розмітки друкованих символів

Загальна кількість зображень у наборі даних для нейронної мережі з окремими символами становила 899 штук.

Розмітка даних.

Розмітка набору даних із друкованими символами проводилася через виділення кожного символу у свою рамку та призначення йому відповідного класу. Приклад показаний рисунку 33.

Рисунок 33 – Приклад розмітки рукописних символів

На кожному зображенні є кілька формул або символів. Майже всі зображення чорно-білого кольору.

Файл даних для навчання.

Після підготовки, набір даних вивантажується з платформи у вигляді пакета файлів, що включає: папку із зображеннями, папку з файлами текстового формату з параметрами розмітки та файлом конфігурації.

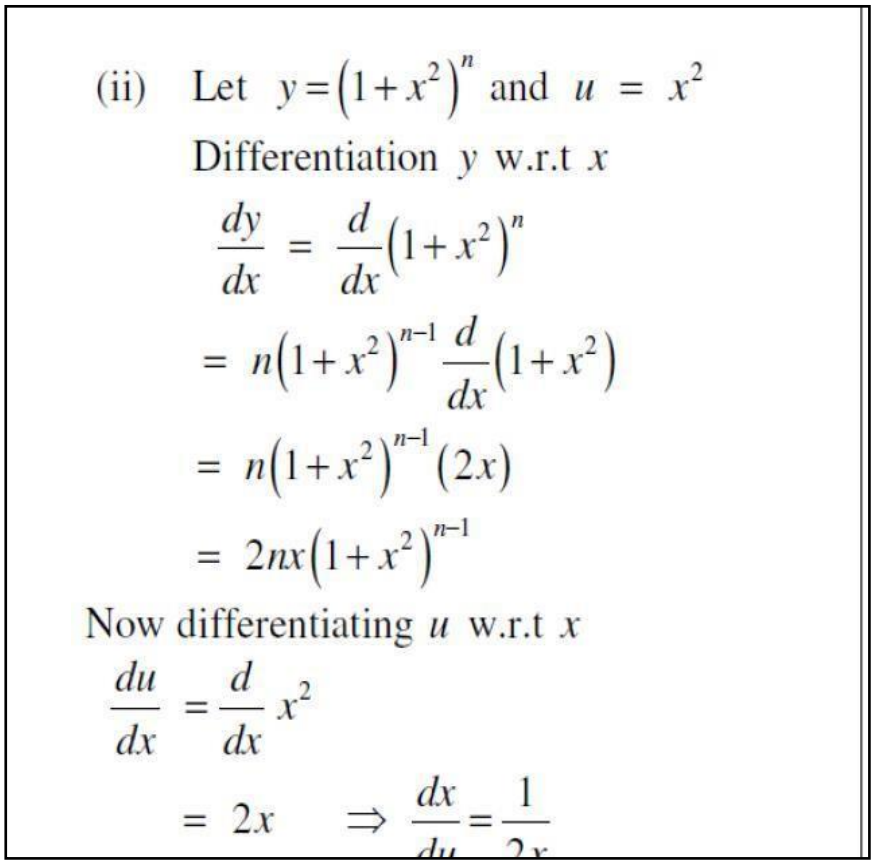
Приклад файлу з параметрами навчання представлений рисунку 34 і саме зображення рисунку 35.

DIFF2_png.rf.870222929072193e8af05044a5dc98bc.txt – Блокнот

Файл Изменить Просмотр

```
0 0.55078125 0.03671875 0.1578125 0.0734375
0 0.615625 0.12421875 0.2859375 0.0875
0 0.6734375 0.2171875 0.38125 0.07890625
0 0.62265625 0.3046875 0.29140625 0.08984375
0 0.64453125 0.38515625 0.32734375 0.08984375
0 0.69296875 0.49140625 0.41640625 0.0859375
0 0.62421875 0.5828125 0.26875 0.09296875
0 0.73828125 0.67890625 0.50625 0.1
0 0.61640625 0.7921875 0.27578125 0.08984375
0 0.6453125 0.91953125 0.31796875 0.1
```

Рисунок 34 – Файл розмітки



(ii) Let $y=\left(1+x^2\right)^n$ and $u = x^2$

Differentiation y w.r.t x

$$\frac{dy}{dx} = \frac{d}{dx}\left(1+x^2\right)^n$$

$$= n\left(1+x^2\right)^{n-1}\frac{d}{dx}\left(1+x^2\right)$$

$$= n\left(1+x^2\right)^{n-1}(2x)$$

$$= 2nx\left(1+x^2\right)^{n-1}$$

Now differentiating u w.r.t x

$$\frac{du}{dx} = \frac{d}{dx}x^2$$

$$= 2x \quad \Rightarrow \frac{dx}{du} = \frac{1}{2x}$$

Рисунок 35 – Зображення до файлу розмітки

На рисунку 34 першій позиції кожного рядка написано номер назви класу у загальному списку. Цей список позначений у файлі data.yaml. Приклад файлу показано рисунку 36.

data.yaml

```yaml
names:
- Differentiation
- Integration
- Limits
- Linear
- Trigonometry
nc: 5
roboflow:
  license: CC BY 4.0
  project: equals-pzmaq
  url: https://universe.roboflow.com/equals/equals-pzmaq/dataset/2
  version: 2
  workspace: equals
test: ../test/images
train: /content/gdrive/MyDrive/yolov5/datasets/Equals-2/train/images
val: /content/gdrive/MyDrive/yolov5/datasets/Equals-2/valid/images

```

Рисунок 36 – Файл конфігурації

4.3 Реалізація нейронної мережі

Для реалізації системи було обрано архітектуру YOLO. Виявлення об'єктів у YOLO виконується як завдання регресії та надає ймовірності класів виявлених зображень. Цей алгоритм використовується нейронними мережами для виявлення об'єктів у режимі реального часу. Також він популярний через його швидкість і точність. YOLO використовується для детекції автомобільного трафіку, автомобільних знаків, тварин тощо.

Алгоритм YOLO працює, розділяючи зображення на N сіток, кожна з яких має область однакового розміру $S \times S$. Кожна з цих N сіток відповідає за виявлення та локалізацію об'єкта, що міститься в ній.

Відповідно, ці сітки пророкують координати рамки, що обмежує, щодо координат їх осередків, а також мітку об'єкта і ймовірністьприсутності об'єкта в осередку. Але цей процес призводить до великої кількості прогнозів, що повторюються, через те, що кілька осередків прогнозують один і той же об'єкт з різними прогнозами обмежуючої рамки. Щоб вирішити цю

проблему, детектор використовує Non Maximal Suppression – YOLO пригнічує всі рамки, що обмежують, з більш низькими показниками ймовірності. YOLO досягає цього, спочатку переглядаючи оцінки ймовірності, пов'язані з кожним рішенням, та вибираючи найбільшу з них. Після цього він пригнічує рамки, що обмежують, що мають найбільше перетин над об'єднанням, з поточною обмежувальною рамкою з високою ймовірністю.

Архітектура моделі YOLOV5s.

Для навчання на даних користувача та реалізації детекції була обрана передбачена модель YOLOV5s. Згідно з дослідженням, у якому наводиться порівняння швидкостей навчання, найвищою швидкістю та найменшими витратами по пам'яті має YOLOV5s. Тому було ухвалено рішення навчати цю модель. Графік представлений рисунку 37.

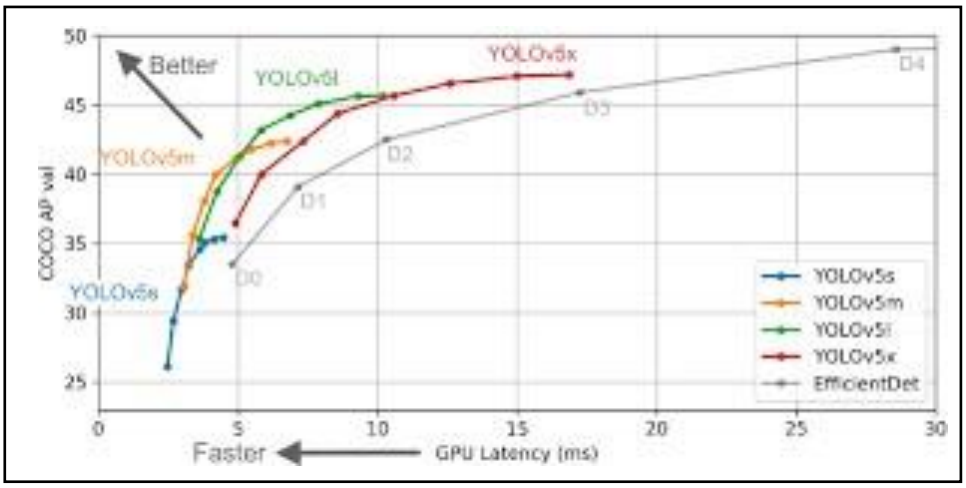


Рисунок 37 – Графік швидкості моделей навчання

Наведемо YAML-файл, який описує архітектуру моделі:

```
# Parameters
nc: 80 # number of classes
depth_multiple: 0.33 # model depth multiple width_multiple:
0.50 # layer channel multiple anchors:
```

```
- [10,13, 16,30, 33,23] # P3/8
- [30,61, 62,45, 59,119] # P4/16
- [116,90, 156,198, 373,326] # P5/32

# YOLOv5 v6.0 backbone backbone:
# [from, number, module, args]
[[-1, 1, Conv, [64, 6, 2, 2]], # 0-P1/2
[-1, 1, Conv, [128, 3, 2]], # 1-P2/4
[-1, 3, C3, [128]],
[-1, 1, Conv, [256, 3, 2]], # 3-P3/8
[-1, 6, C3, [256]],
[-1, 1, Conv, [512, 3, 2]], # 5-P4/16
[-1, 9, C3, [512]],
[-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
[-1, 3, C3, [1024]],
[-1, 1, SPPF, [1024, 5]], # 9
]

# YOLOv5 v6.0 head head:
[[-1, 1, Conv, [512, 1, 1]],
[-1, 1, nn.Upsample, [None, 2, 'nearest']],
[[-1, 6], 1, Concat, [1]], # cat backbone P4
[-1, 3, C3, [512, False]], # 13
[-1, 1, Conv, [256, 1, 1]],
[-1, 1, nn.Upsample, [None, 2, 'nearest']],
[[-1, 4], 1, Concat, [1]], # cat backbone P3
[-1, 3, C3, [256, False]], # 17 (P3/8-small)
[-1, 1, Conv, [256, 3, 2]],
[[-1, 14], 1, Concat, [1]], # cat head P4
[-1, 3, C3, [512, False]], # 20 (P4/16-medium)
[-1, 1, Conv, [512, 3, 2]],
[[-1, 10], 1, Concat, [1]], # cat head P5
[-1, 3, C3, [1024, False]], # 23 (P5/32-large)
[[17, 20, 23], 1, Detect, [nc, anchors]], # Detect(P3, P4,
P5)
]
```

Навчання моделей YOLOV5s.

Навчання проводилося окремо для двох різних завдань:

– детектування та класифікація формул;

– детектування та класифікація символів.

Тому необхідно було навчити дві нейронні мережі з однаковою топологією та різними наборами даних.

Навчання проходило у хмарному сервісі Google Colaboratory. У YOLOv5s використані: оптимізатор – SGD, функції активації – SiLU та Sigmond, функція втрат – FocalLoss.

Над обома наборами даних було проведено аугментація – штучне збільшення набору даних рахунок зміни кута нахилу чи якості зображення.

Навчання нейронної мережі для визначення формул.

Для навчання моделі розпізнавання формул було використано приблизно 1300 зображень з формулами. У відсотковому співвідношенні – 84% від основного набору даних.

За результатами експериментів навчання було обрано модель, навчену на 50 епохах з наступними метриками.

Значення: mAP – 0,983, precision – 0,961, recall – 0,950.

Графічне подання зміни метрик зображено рисунку 38.

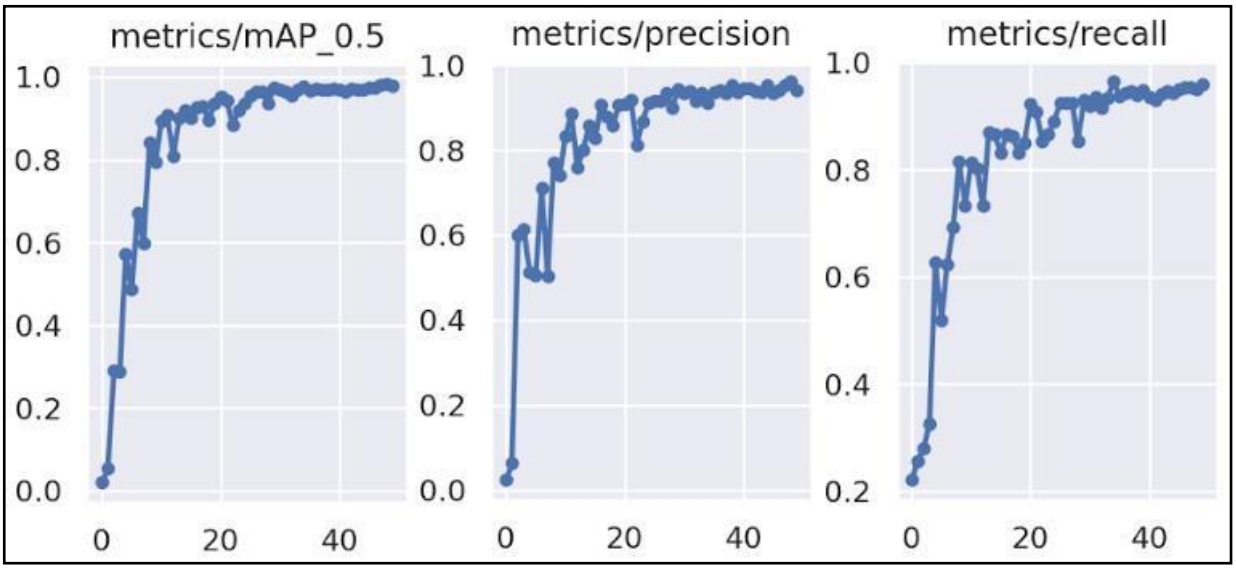


Рисунок 38 – Графічна вистава метрик

Навчання нейронної мережі для визначення символів.

Для навчання моделі розпізнавання формул було використано приблизно 2200 зображень із символами або групою символів, наприклад: sin, cos, tg, ln і тд. У відсотковому співвідношенні – 93% від основного набору даних.

За результатами експериментів навчання було обрано модель, навчену на 100 епохах з наступними метриками.

Значення: mAP – 0,898, precision – 0,905, recall – 0,843.

Графічне подання зміни метрик зображено рисунку 39.

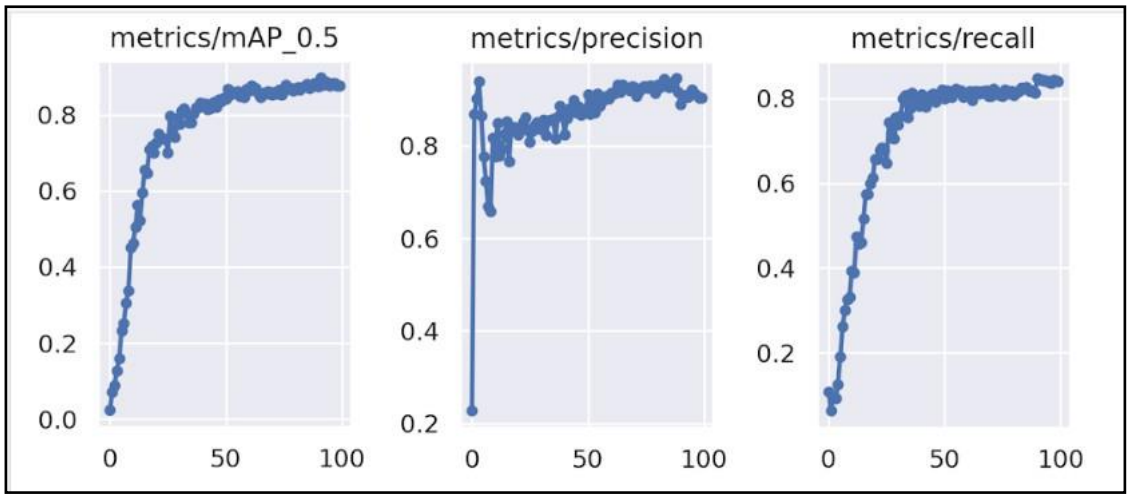


Рисунок 39 – Графічна вистава метрик

4.3 Інтерфейс взаємодії з користувачем

Для реалізації веб-складової були використані технології:

- HTML5, CSS3 для верстки;
- Javascript, jQuery 3.x для інтерактивності та посилки Ajax-запитів;
- PHP 7 на стороні сервера для роботи сервісу та для перехоплення та обробки Ajax-запитів, а також для запуску Python-скрипту для розпізнавання формул.

Для того, щоб процес розпізнавання формул був максимально зручним,

був розроблений веб-сервіс. Він дозволяє розпізнавати онлайн формули, не встановлюючи для цього окремих програм.

Веб-сервіс розміщується на хмарних серверах AWS (Amazon Web Services). На сервері встановлено дистрибутив Ubuntu 16.04 LTS.

Попередньо на сервер були встановлені всі необхідні бібліотеки для мови Python. Також було встановлено XAMPP.

Для веб-сервісу, згідно з макетом, було намальовано та зверстано шаблон на HTML5 та CSS3. Для додавання інтерактивності було використано javascript-бібліотеку jQuery.

Результат представлений рисунку 40.

Рисунок 40 – Шаблон веб-сервісу

Під час того, як користувач перебуває в очікуванні результату розпізнавання, він бачить gif-зображення, що імітує завантаження (рисунок 41).

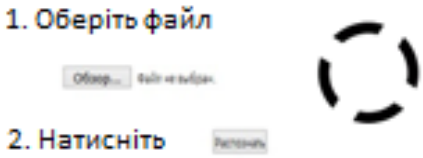


Рисунок 41 – Імітація процесу розпізнавання

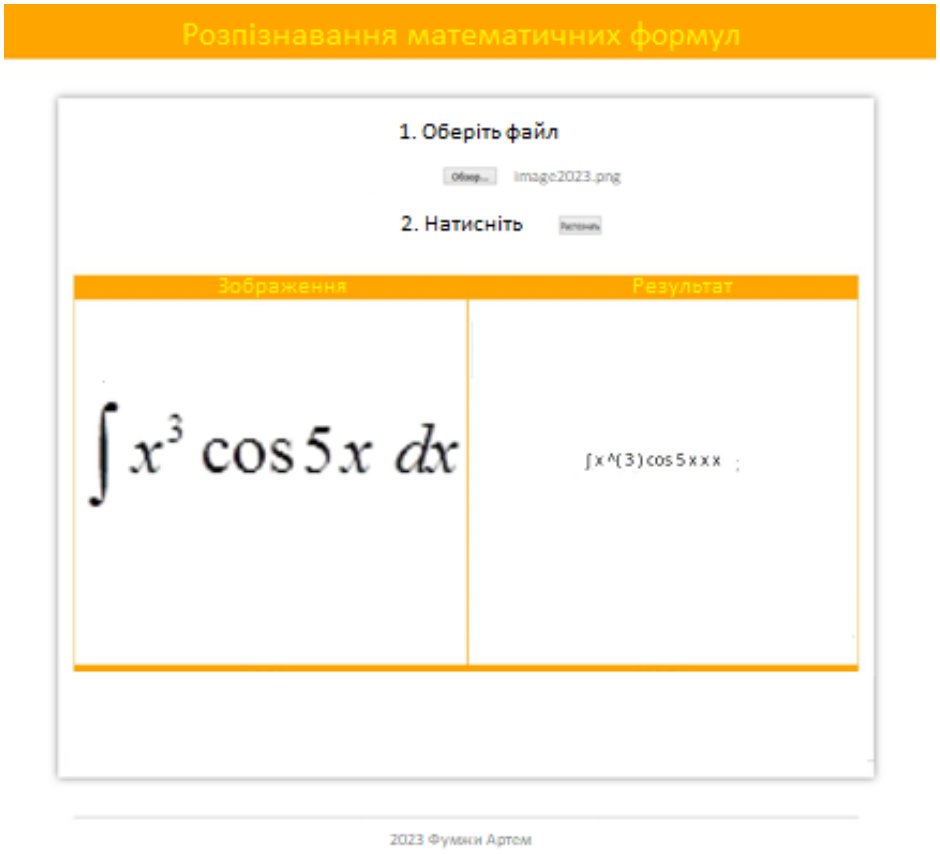


Рисунок 42 – Результат розпізнавання

Налаштування сервера AWS.

Спочатку на сервері були встановлені всі необхідні бібліотеки для Python.

Потім для роботи веб-сервісу було встановлено XAMPP (кросплатформове складання веб-сервера). Разом з XAMPP було встановлено PHP, MySQL.

Для зручної роботи над проектом було створено репозиторій на bit-

bucket.org. Таким чином, всі локальні зміни заливались спочатку в репозиторій, потім звідти на хмару.

Розглянемо принцип роботи веб-сервісу.

Коротко про схему роботи веб-сервісу. Користувач завантажує зображення. Зображення за допомогою Ajax передається на сервер, де його обробляє PHP скрипт. Зображення завантажується на сервер, а посилання на нього передається до Python-скрипту для розпізнавання, запуск якого також відбувається в PHP-скрипті.

Далі результат розпізнавання обробляється і передається користувачу.

Також було реалізовано функції обробки результатів детекції нейронних мереж.

У модулі детекції символів, крім підключеної нейронної мережі, додатково були реалізовані функція видалення зайвих рамок детектування і функція виявлення ступеня в результаті.

```
def delete_double_frame(list): b = []
s = list[0].get('mid_y')
for i in range(1, len(list)):
if   list[i].get('x1s')  -    list[i-1].get('x1s')     <
3:#  andlist[i].get('x1s') not in b:
b.append(list[i].get('x1s'))
if len(b) == 1:
for i in range(len(list)):
if list[i].get('x1s') == b[0]: list.pop(i)
break
else:
pp = len (list)
for i in range(pp-len(b)): for j in range(len(b)):
if list[i].get('x1s') == b[j]:
# print(list[i].get('x1s')) list.pop(i)
# print(list)
```

Функція виявлення ступеня:

```
def get_power(list):
s = list[0].get('mid_y')
for i in range(1,len(list)): s += list[i].get('mid_y')
mid_y = s/len(list) c = []
for i in range(len(list)):
if list[i].get('mid_y') < mid_y*0.9: c.append(i)
for i in range(len(c)):
if i == 0 or (i != 0 and c[i] - c[i-1] > 1):
list.insert(c[i],  {'x1s':  list[c[i]].get('x1s')  -  1,
'mid_y':
list[c[i]].get('mid_y'), 'class': '^('})
c[i]+= 1
for j in range(i+1,len(c)): c[j] += 1
if i == len(c)-1 or
(i != len(c)-1 та c[i+1] - c[i] > 1): if i == len(c)-1:
c[i] -= 1
list.insert(c[i] + 2,{'x1s': list[c[i] + 1].get('x1s') - 1,
'mid_y': list[c[i] + 1].get('mid_y'), 'class': ')'}) else:
list.insert(c[i]+1, {'x1s': list[c[i]+1].get('x1s') - 1,
'mid_y': list[c[i]+1].get('mid_y'), 'class': ')'})
for j in range(i + 1, len(c)): c[j] += 1
st = []
for i in range(len(list)): st.append(list[i].get('class'))
return st
```

За допомогою цих функцій вдалося досягти більш правильних, розпізнаваних та зрозумілих результатів.

5 ТЕСТУВАННЯ СИСТЕМИ

У світі програмного забезпечення якість продукту грає ключову роль успіху будь-якої організації. Тестування є одним з основних етапів розробки програмного забезпечення, який допомагає забезпечити високу якість продукту.

Функціональне тестування – це процес перевірки різних функцій програмного забезпечення виявлення помилок і переконання у тому, що вони працюють відповідно до специфікацій. Зазвичай до етапів функціонального тестування можна віднести такі види та рівні тестування:

- юніт-тестування;
- інтеграційне тестування;
- системне тестування;
- регресійне тестування;
- тестування прийнятності для користувача.

Нефункціональне тестування стосується аспектів програмного забезпечення, які не пов'язані з конкретними функціями, включаючи такі аспекти:

- тестування продуктивності;
- тестування безпеки;
- тестування сумісності;
- тестування надійності;
- тестування юзабіліті та GUI;
- тестування локалізації та інтерналізації.

5.1 Функціональне тестування системи

Було проведене функціональне тестування програми.

Функціональне тестування перевіряє відповідність вимогою системи, що проектується.

Тест №1. Мета: перевірка запуску.

Очікуваний результат: веб-сервіс повинен завантажуватися та виводити шаблон у браузері.

Висновок: очікуваний та отриманий результат збігаються, мета досягнута.

Тест №2. Мета: перевірити завантаження зображення та запуск процесу розпізнавання.

Очікуваний результат: веб-сервіс повинен завантажити обране зображення та запустити процес розпізнавання.

Отримані результати зображені на рисунках у додатку. Висновок: очікуваний та отриманий результат збігаються, мета досягнута.

Тест №3. Мета: перевірити процес розпізнавання символів на зображення.

Очікуваний результат: при натисканні на кнопку «Розпізнати», програма має розпочати процес розпізнавання символів на зображенні та після закінчення процесу, видати результат користувачеві.

Висновок: очікуваний та отриманий результат збігаються, мета досягла нуту.

Виходячи з результатів функціонального тестування розроблена система відповідає функціональним вимогам.

5.2 Тестування нейронної мережі

У цьому розділі продемонстровано роботу нейронних мереж, навчених на наборах даних для детекції формул та символів окремо. Модель навчається з результатами mAP: 0,983 – для формул та 0,898 – для символів. Метрика mAP означає середню точність всім класів набору даних. Для формул було виділено 5 класів (інтеграли, лінійні рівняння, диференціальні рівняння, тригонометричні вирази, межі) та для символів – 23 (числа 0–9, знаки: /, *, +, -, =, знак інтеграла, дужки, змінні: x, y, z, t, d).

На визначальній рамці є два значення: назва класу і значення метрики mAP. Приклад рамки представлений рисунку 43.

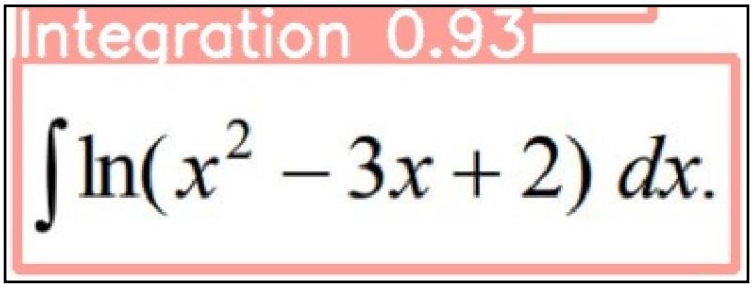


Рисунок 43 – Приклад означальної рамки

Приклад тестування нейронної мережі виявлення формул представлений рисунку 44.

Рисунок 44 – Приклад детекції формул

На рисунках 45 і 46 наведено приклад роботи нейронної мережі для виявлення символів.

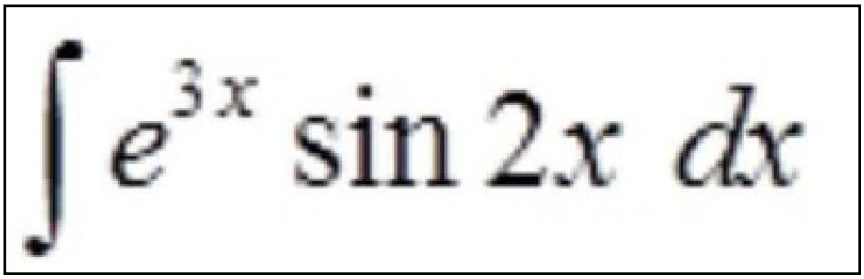


Рисунок 45 – Фрагмент початкового зображення з формулою

Рисунок 46 – Результат роботи нейронної мережі

Експеримент над навчанням нейронних мереж.

Для того, щоб отримати значення оптимальної кількості епох для навчання нейронних мереж, було проведено експеримент. Навчання кожної нейронної мережі проводилося кілька разів з однаковим значенням параметрів, крім кількості епох, воно змінювалося з 50 до 150. Результати експериментів представлені на таблицях 1 та 2.

В результаті експериментів, виявлено, що найефективнішим було навчання: нейронна мережа для детекції формул – 50 епох, нейронна мережа для детекції символів – 100 епох.

Таблиця 1 – Результати експериментів навчання нейронної мережі для формул

Кількість епох	Час навчання/хв.	mAP	Precision	Recall
150	31,8	0,985	0,962	0,955
100	20,9	0,982	0,958	0,954
75	16,3	0,983	0,968	0,934
50	10,8	0,983	0,961	0,950

Таблиця 2 – Результати експериментів навчання нейронної мережі для символів

Кількість епох	Час навчання/хв.	mAP	Precision	Recall
150	82,3	0,881	0,936	0,833
100	57,3	0,898	0,905	0,843
75	42,8	0,863	0,903	0,820
50	28,5	0,850	0,876	0,817

ВИСНОВОК

У ході роботи було виконано такі завдання:

- проведено аналіз систем-аналогів для розпізнавання математичних формул з використанням згорткових нейронних мереж;
- підготовлено два навчальні набори даних;
- реалізовано дві нейронні мережі, проведено їх навчання та тестування;
- спроектовано та реалізовано додаток для розпізнавання математичних формул;
- проведено тестування реалізованої програми.

Подальший напрямок розробки представлений нижче:

- додати обробку дробів;
- додати обробку диференціальних рівнянь (лапки, посимвольно);
- додати обробку тригонометричних виразів (градуси, радіани);
- додати обробку певних інтегралів;
- додати до набору даних зображення з різним розташуванням виразів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Що таке нейронна мережа? [Електронний ресурс] URL: https://aws.amazon.com/ru/what-is/neural-network/ (дата звернення: 10.11.2023 р.).
2. Метрики якості моделей бінарної класифікації. [Електронний ресурс] URL: https://loginom.ru/blog/classification-quality (дата звернення: 12.11.2023 р.).
3. Завдання знаходження об'єктів на зображенні. [Електронний ресурс] URL: http://neerc.ifmo.ru/wiki (Дата звернення: 15.11.2023 р.).
4. YOLOV5 compared to Faster RCNN. Who wins? [Електронний ресурс]. URL: https://towardsdatascience.com/yolov5-compared-to-faster-rcnn- who-wins-a771cd6c9fb4/ (дата звернення: 15.11.2023 р.).
5. V7 – YOLO: Real-Time Object Detection Explained. [Електронний ресурс] URL: https://www.v7labs.com/blog/yolo-object-detection (Дата звернення: 20.11.2023 р.).
6. Microsoft Math Solver. [Електронний ресурс] URL: https://math.microsoft.com/ua (дата звернення: 15.11.2023 р.).
7. PhotoMath. [Електронний ресурс] URL: https://photomath.com/ru/ (дата звернення: 15.11.2023 р.).
8. OneNote. [Електронний ресурс] URL: https://www.microsoft.com/ru-ua/microsoft-365/onenote/digital-note-taking-app (дата звернення: 17.11.2023 р.).
9. Smart Engines – Що таке OCR? Навіщо потрібне оптичне розпізнавання символів у сучасному світі мобільних технологій? [Електронний ресурс] URL: https://smartengines.ru/blog/chto-znachit-raspoz- navanie-ocr/ (дата звернення: 17.11.2023 р.).
10. Gadgetshelp.com – Чому ви повинні перейти з OneNote 2016 на OneNote для Windows 10. [Електронний ресурс] URL: -One- note-Dlia-Windows-10 (дата звернення: 17.11.2023 р.).

11. Ahlawat S., Choudhary A., Nayyar A., Singh S., Yoon B. Improved handwritten digital recognition using convolutional neural networks (Cnn) // Sensors (Switzerland), 2020. – vol. 20, no. 12. – 14 с.
12. Вігерс Карл. Розробка вимог до програмного забезпечення. / / К: Видавничо-торговельний будинок Російська редакція, 2004. – 576 с.
13. Bhagyashree PM, Likhitha LK, Rajesh DS Handwritten Digit Recognition Using Deep Learning.//International Journal of Scientific Research в Science and Technology, Jul. 2021. – 153 – 158 pp.
14. How to Develop a CNN for MNIST Handwritten Digit Classification. [Електронний ресурс] URL: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/. (Дата звернення: 20.11.2023 р.).
15. TensorFlow – MNIST. [Електронний ресурс] URL: https://www.tensorflow.org/datasets/catalog/mnist (Дата звернення: 20.11.2023 р.).
16. SpringerOpen – Convolutional neural networks: an overview and application in radiology. [Електронний ресурс] URL: https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9 (дата звернення: 20.11.2023 р.).
17. IBM – Convolutional Neural Networks. [Електронний ресурс] URL: https://www.ibm.com/cloud/learn/convolutional-neural-networks#toc-what-are-c-MWGVhUiG9 (дата звернення: 20.11.2023 р.).
18. Roboflow. [Електронний ресурс] URL: https://app.roboflow.com (дата звернення: 20.11.2023 р.).
19. Mathleaks_filtered Computer Vision Project. [Електронний ресурс] URL: https://universe.roboflow.com/mike-robot/mathleaks_filtered (дата звернення: 20.11.2023 р.).
20. Mathematical Expression Detection Computer Vision Project. [Електронний ресурс] URL: https://universe.roboflow.com/equation-detection- cadby/mathematical-expression-detection (дата звернення:

20.11.2023 р.).

21. Google Colaboratory. [Електронний ресурс]. URL: https://colab.research.google.com/ (дата звернення: 20.11.2023 р.).
22. PyCharm. [Електронний ресурс]. URL: https://www.jetbrains.com/ru-ru/pycharm/ (Дата звернення: 20.11.2023 р.).
23. Roboflow. [Електронний ресурс]. URL: https://app.roboflow.com (дата звернення: 20.11.2023).
24. Matplotlib. [Електронний ресурс]. URL: https://matplotlib.org/ (дата звернення: 22.11.2023).
25. Numpy. [Електронний ресурс]. URL: https://numpy.org/ (дата звернення: 22.11.2023 р.).
26. OpenCV. [Електронний ресурс]. URL: https://pypi.org/project/opencv-python/ (дата звернення: 22.11.2023 р.).
27. Pillow. [Електронний ресурс]. URL: https://pypi.org/project/Pillow/ (дата звернення: 22.11.2023 р.).
28. PyYAML. [Електронний ресурс]. URL: https://pyyaml.org/wiki/PyYAMLDocumentation (дата звернення: 22.11.2023 р.).
29. Torch. [Електронний ресурс]. URL: https://pytorch.org/ (дата звернення: 22.11.2023 р.).
30. Torchvision. [Електронний ресурс]. URL: https://pytorch.org/ (дата звернення: 22.11.2023).
31. Tqdm. [Електронний ресурс]. URL: https://tqdm.github.io/ (дата звернення: 22.11.2023 р.).
32. Roboflow – Dataset Health Check. [Електронний ресурс] URL: https://app.roboflow.com/equals/equals-pzmaq/health (дата звернення: 22.11.2023 р.).
33. Roboflow – Dataset Health Check. [Електронний ресурс] URL: https://app.roboflow.com/equals/gg-s7vld/health (дата звернення: 15.11.2023 р.).

34. Tensorboard. [Електронний ресурс]. URL: https://www.tensorflow.org/tensorboard/get_started (Дата звернення: 22.11.2023 р.).
35. Pandas. [Електронний ресурс]. URL: https://pandas.pydata.org/ (дата звернення: 22.11.2023 р.).